DISSERTAÇÃO DE MESTRADO Nº 341

ESTRATÉGIAS PARA COMBINAÇÃO DE TÉCNICAS DE BOOSTING E SUPPORT VECTOR MACHINES

Thiago Turchetti Maia

DATA DA DEFESA: 13.03.2003

Estratégias para Combinação de Boosting e Support Vector Machines

Thiago Turchetti Maia

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Engenharia Elétrica

Universidade Federal de Minas Gerais

Fevereiro 2003

"ESTRATÉGIAS PARA COMBINAÇÃO DE TÉCNICAS DE BOOSTING E SUPPORT VECTOR MACHINES"

THIAGO TURCHETTI MAIA

Dissertação de Mestrado submetida à banca examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como parte dos requisitos necessários à obtenção do grau de *Mestre em Engenharia Elétrica*.

Aprovada em 13 de março de 2003.

Por:

Antônio de Pádua Braga – Ph.D.
Prof. DELT/EEUFMG – orientador

André C. P. L. F. Carvalho – Dr.
Prof. ICMC/USP – co-orientador

Carmen Déa Moraes Pataro – Dra.
Profa. DELT/EEUFMG

Eduardo Mazoni Andrade Marçal Mendes – Dr.
Prof. DEE/EEUFMG

Sueli Mingoti – Dra.
Profa. Depto. Estatística/ICEx/UFMG

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 O Problema de Aprendizado Computacional

O problema de ensinar uma máquina como adquirir novo conhecimento tem sido explorado desde a introdução do computador moderno. O primeiro modelo de neurônio artificial foi criado por McCulloch e Pitts em 1943 [MP43], em seu trabalho clássico que descreve o que deveria ser a unidade básica de uma rede de neurônios artificiais inspirada no cérebro humano. O foco desse trabalho foi muito mais sobre a capacidade computacional do modelo proposto, do que em métodos pragmáticos para aquisição de conhecimento. Mais tarde em 1949, o aprendizado destas redes foi explorado por Hebb [Heb49], que propôs um modelo de ajuste de pesos nas entradas de cada neurônio. Este modelo se baseou na teoria de que neurônios biológicos aprendem baseados no reforço de suas ligações sinápticas para outros neurônios excitados. Widrow e Hoff [WH60] refinaram ainda mais as idéias de Hebb em 1960, incorporando o método do gradiente descendente para minimizar o erro de saída de cada neurônio. A primeira tentativa de abordar o problema de reconhecimento de padrões a partir de redes de neurônios foi feita por Rosenblatt em 1958 [Ros58], onde ele introduziu o modelo *Perceptron*. O Perceptron é uma simples rede capaz de resolver problemas de classificação linearmente separáveis no qual o espaço de saída é dividido em regiões complementares que correspondem a categorias.

Apesar desta abordagem conexionista ter recebido grande atenção devido ao trabalho de Rosenblatt, a comunidade de aprendizado de máquina foi quase completamente desencorajada por Minsky e Papert em 1969 [MP69] a continuar seu desenvolvimento. Minsky e Papert argumentaram que a limitação de não ser capaz de resolver problemas não-linearmente separáveis era muito grave para ser ignorada, além de apontarem outros problemas como o crescimento explosivo do

consumo de recursos no tempo e espaço, além da falta de uma regra de aprendizado para redes multi-camadas. Salvo raras exceções, a década de 1970 foi marcada pela falta de interesse da comunidade na área. Apenas após o trabalho de Hopfield em 1982 [Hop82] e o desenvolvimento do algoritmo *backpropagation* em 1986 [RHW86] que redes de neurônios artificiais [1] novamente ganharam interesse geral. Hopfield explorou as relações entre redes associativas recorrentes e modelos físicos, o que iniciou uma série de outros trabalhos que utilizaram teorias da Física para descrever o comportamento de máquinas de aprendizado [Hop82]. Finalmente, muita atenção foi dada ao trabalho de Rumelhart et al. [RHW86], que criaram um algoritmo, o *backpropagation*, para treinar redes com múltiplas camadas baseado no ajuste de pesos de cada camada de acordo com a retro-propagação de erros de saída.

Esse ressurgimento da escola conexionista na década de 1980 foi seguido por uma nova onda de pesquisas no início da década de 1990 que exploraram novas teorias e algoritmos focados não só na construção e treinamento de máquinas de aprendizado poderosas, mas também na reorganização do ambiente ao seu redor visando o aumento de seu desempenho [2]. Vapnik [Vap95] se refere a este período como o retorno às origens da teoria de aprendizado estatístico. Trabalhos pioneiros como os de Munro [Mun92], Schapire [Sch92], Drucker et al. [DSS93], e Cachin [Cac94] serviram de base para outros como Breiman [Bre94], que introduziram o conceito de *Bagging* para o treinamento e combinação de múltiplas cópias de um algoritmo de aprendizado, e finalmente Freund e Schapire, que em seguida introduziram a famosa família de algoritmos AdaBoost. O termo *Boosting* foi apresentado por Schapire em 1990 [Sch90], no trabalho que mostrou pela primeira vez que qualquer algoritmo de tempo polinomial que gera hipóteses cujos erros são arbitrariamente melhores que uma hipótese aleatória pode ser transformado em um algoritmo de tempo polinomial com erro arbitrariamente pequeno.

Em paralelo com o desenvolvimento de modelos de aprendizado e metodologias para a melhoria do desempenho de algoritmos já existentes, este retorno às origens também motivou a criação de diversas novas máquinas de aprendizado. Entre elas, destacam-se as de Vapnik e colaboradores que ficaram sendo conhecidas como *Support Vector Machines* (SVMs), ou máquinas de vetores de suporte. Baseados em outros trabalhos de Vapnik e Chervonenkis de 1965 [VC71], nos quais

[1] Redes de neurônios artificiais são geralmente chamadas pela comunidade de aprendizado de máquina brasileira de *redes neurais artificiais*, ou ainda *redes neuronais artificiais*.

[2] Note que o termo *desempenho* é usado por todo o texto sem especificação de desempenho no tempo ou no espaço. Quando utilizado assim de forma tão genérica, o texto refere-se ao *desempenho global* composto da combinação de diversas medidas de desempenho passíveis de serem aplicadas a máquinas ou algoritmos de aprendizado.

métodos de hiperplanos de separação ótimos foram desenvolvidos, Vapnik et al. criaram a idéia de máquina de vetores de suporte em 1992 [Vap95]. Desde então, o crescente interesse da comunidade de aprendizado de máquina motivou o desenvolvimento de diversos algoritmos de treinamento especializados na resolução dos problemas de otimização com restrições convexos e quadráticos sobre os quais SVMs são formuladas. Novas técnicas, como *chunking* e *decomposition*, inspiraram o trabalho de John Platt [Pla98a, Pla98b], cujo algoritmo *Sequential Minimal Optimization* (SMO), permitiu com que o uso de SVMs fosse popularizado devido à sua simplicidade de implementação e sua drástica redução no consumo de recursos computacionais em comparação à outros algoritmos.

1.2 Objetivos e Motivação

Dos diversos tipos de problema resolvidos por diferentes algoritmos de aprendizado, neste trabalho foram abordados problemas de classificação binária. Dois motivos justificam tal escolha. Primeiro, problemas de classificação binária são um dos mais fundamentais do aprendizado de máquina, onde casos mais sofisticados muitas vezes são resolvidos a partir de extensões deste caso mais simples. Muitos modelos e algoritmos, originalmente desenvolvidos para lidar com problemas de classificação binária, foram adaptados para estender sua aplicabilidade para problemas de classificação multi-classe (n-ários), classificação multi-rótulo, e até mesmo problemas de regressão. Além disso, problemas de classificação binária são o habitat natural dos dois modelos de aprendizado combinados neste trabalho, Boosting e SVMs, apesar de também haver diversas extensões já propostas para ambos. Portanto, a utilização das versões originais e mais fundamentais destes dois modelos foi proposta para validar a idéia de utilizá-los juntos em máquinas híbridas de aprendizado.

É formalizada agora a definição de problema de classificação binária, assumido ao longo de todo o texto. Considere um problema de classificação com um domínio de entrada n-dimensional e um domínio de saída binário, ambos contendo l instâncias tal que existe um conjunto de dados $(x_1, y_1), \ldots, (x_l, y_l)$, onde cada padrão x_i pertence ao domínio $X = \mathbb{R}^n$, cada rótulo y_i pertence ao domínio $Y = \{-1, +1\}$ e cada padrão x_i é classificado pelo rótulo y_i correspondente. Neste trabalho foram selecionadas 22 bases de dados com as quais os quatro algoritmos híbridos propostos mais adiante, além da versão original do SMO como referência, foram testados. Dos 22 bancos de dados selecionados, 8 correspondem a problemas reais de biologia, 2 a problemas reais de bioinformática e genômica, um a um problema real de física, 9 foram sinteticamente gerados a partir de entradas bi-dimensionais e 2 foram sinteticamente gerados com entradas de 20 dimensões.

Uma descrição detalhada destas bases de dados e seus problemas correspondentes é dada no Capítulo 5.

Das diversas abordagens de aprendizado de máquina que poderiam ser utilizadas para problemas de classificação binária, certamente há um maior interesse naquelas que produzem modelos mais precisos com o menor tempo de execução possível. Um dos mais interessantes algoritmos de aprendizado desenvolvidos recentemente é o SMO, proposto por John Platt [Pla98a]. Além de basear-se no grande poder de expressão conferido por SVMs, Platt conseguiu aproveitar-se de diversas propriedades do problema convexo e quadrático de otimização com restrições responsável por seu treinamento e criou um algoritmo eficiente que geralmente é mais rápido e mais preciso que muitos outros algoritmos de aprendizado.

Seria muita inocência tentar propor um algoritmo que é geralmente mais preciso e mais rápido que o SMO. Mesmo assim, da mesma forma que Platt aproveitou-se das propriedades específicas do problema de otimização de SVMs, sua abordagem pode ser estendendida para criar novos algoritmos que explorem propriedades específicas de problemas de aprendizado, ocasionalmente apresentando um desempenho melhor que o SMO em termos de precisão e tempo de execução. Para alcançar tal objetivo, o SMO foi utilizado como base dos algoritmos híbridos e combinado a um algoritmo de Boosting, de forma que o forte fundamentalismo teórico de Boosting pudesse melhorar seu desempenho.

De acordo com Freund [Fre95], um algoritmo de Boosting é um algoritmo de aprendizado que utiliza um segundo algoritmo de aprendizado como subrotina. A partir da repetida execução de um dado algoritmo de aprendizado fraco sobre uma distribuição de dados de treinamento, algoritmos de Boosting geram uma hipótese forte que é composta pela combinação de todas as hipóteses fracas geradas através de votos ponderados. Os primeiros algoritmos de Boosting foram originalmente apresentados por Schapire [Sch90] e Freund [Fre95]. Mais recentemente, Freund e Schapire introduziram **AdaBoost**, uma família de algoritmos de Boosting genéricos que resolveram muitas das dificuldades práticas de algoritmos antecessores [FS95, FS96a].

Portanto, o objetivo mais importante deste trabalho é a criação de uma nova classe de algoritmos que se baseiam nos pontos mais fortes dos modelos aprendizado por traz de Boosting e SVMs, aproveitando-se de propriedades específicas de problemas de aprendizado para superar o eficiente algoritmo SMO proposto por Platt.

1.3 Contribuições Deste Trabalho

Neste trabalho técnicas de Boosting e SVMs foram combinadas para criar novos algoritmos híbridos de aprendizado que consistentemente apresentam melhor desempenho que algoritmos tradicionais baseados em SVMs, como SMO, em problemas de classificação binária, tanto em termos de tempo quanto em termos de acurácia. Apesar de SVMs serem construídas para criarem hiperplanos de separação ótimos em tarefas de classificação binária, foi mostrado que diversos fatores como polarização de dados, funções de kernel mal-escolhidas, e parâmetros de regularização e kernel desajustados, são capazes de degradar o desempenho de algoritmos tradicionais de SVMs, que portanto pode ser melhorado através de algoritmos de Boosting. Foram combinados dois dos mais conhecidos algoritmos de Boosting e de treinamento de SVMs, o AdaBoost.M1 de Freund e Schapire [FS95] e o *Sequential Minimal Optimization* (SMO) de Platt [Pla98a]. Sua interação foi abordada em diferentes níveis, desde o uso de SVMs como algoritmos de aprendizado caixa-preta, até a combinação e eliminação de heurísticas particulares do SMO e de mecanismos de seleção probabilística do AdaBoost.M1, portanto propondo quatro novos algoritmos híbridos com diferentes graus de acoplamento.

A mais importante contribuição deste trabalho é provar ser viável a integração de SVMs com Boosting para a criação de algoritmos mais eficientes que outros algoritmos de aprendizado poderosos como SMO, tanto em termos de tempo de execução quanto em termos de acurácia. A literatura de Boosting adverte que algoritmos de classificação muito complexos, quando usados como algoritmos gerados de hipóteses fracas, não são capazes de produzir boas hipóteses fortes através de um algoritmo de Boosting. Uma vez que SVMs são das máquinas de aprendizado mais sofisticadas já propostas, contradizer o fato de que estejam sujeitas a esta restrição de desempenho é de grande importância para desenvolvimentos futuros de máquinas de aprendizado, ainda mais avançadas, a partir da incorporação de Boosting e princípios de vetores de suporte.

Finalmente, além de dar os primeiros passos em direção a algoritmos híbridos mais eficientes, outra importante contribuição deste trabalho é a construção de novos algoritmos que já apresentam desempenho melhor que o SMO em determinados problemas de classificação. Enquanto são discutidos resultados e conclusões deste trabalho, são ressaltadas diversas propriedades comuns a todos os conjuntos de dados com os quais os algoritmos propostos tiveram melhor desempenho. Além disso, um dos maiores problemas de SVMs é o ajuste de parâmetros de regularização e do kernel durante a fase de treinamento. Apesar da simplicidade dos procedimentos de ajuste utilizados, os bons patamares de desempenho obtidos fazem com que seja possível que esta abordagem híbrida possa compensar por pequenos desajustes

nos parâmetros da SVM e de seu kernel.

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma. No Capítulo 2 são apresentados alguns conceitos de Boosting, assim como a família de algoritmos AdaBoost, de Freund e Schapire [FS95, Sch02], mais especificamente seu membro para classificação binária, AdaBoost.M1. No Capítulo 3 são apresentados conceitos e proposições de máquinas de vetores de suporte, onde atenção especial é dada ao algoritmo *Sequential Minimal Optimization* (SMO) desenvolvido por John Platt [Pla98a].

A partir dos conceitos introduzidos nos Capítulos 2 e 3, o Capítulo 4 propõe a criação de algoritmos híbridos que combinam SVMs e técnicas de Boosting na criação de máquinas de aprendizado mais eficientes. O Capítulo 5 descreve e discute todos os experimentos executados e os resultados obtidos. Finalmente, o Capítulo 6 apresenta as conclusões alcançadas a partir deste trabalho.

Capítulo 2

Boosting

Este capítulo descreve a versão genérica do AdaBoost.M1, o algoritmo de Boosting escolhido para ser acoplado a SVMs neste trabalho. O AdaBoost.M1 foi o primeiro membro da família AdaBoost, introduzida por Freund e Schapire em 1995 [FS95]. Desde então, diversas extensões e modificações desta versão original do AdaBoost tem sido propostas por seus próprios criadores, por exemplo em [FS96a, FS96b, FS97, FS99, Sch99] e mais recentemente em [Sch02].

O pseudo-código para uma versão generalizada do AdaBoost, como primeiro apresentada por Schapire e Singer [SS99] e mais tarde por Schapire [Sch02], é apresentada na Figura 2.1. O algoritmo recebe como entrada um conjunto de treinamento $(x_1, y_1), \ldots, (x_m, y_m)$ com m elementos, onde $x_i \in X$ e $y_i \in Y = \{-1, +1\}$. A distribuição D_t é inicializada com uma distribuição uniforme, onde $D_1(i) = 1/m$. Em cada uma das $t = 1, \ldots, T$ etapas, AdaBoost executa um algoritmo de aprendizado para construir uma hipótese fraca h_t baseada no status atual da distribuição D_t. O objetivo deste algoritmo é minimizar seu próprio erro de treinamento ϵ_t. Uma vez que h_t foi construída e avaliada, AdaBoost escolhe um parâmetro $\alpha_t \in \mathbb{R}$ para medir a relevância de cada hipótese h_t.

Para uma versão de classificação binária como AdaBoost.M1, ϵ_t é calculado da seguinte forma:

$$\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]. \tag{2.1}$$

Também assumindo o caso de domínio de saída binário tal que $h_t \in \{-1, +1\}$, o peso associado a cada hipótese é:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{2.2}$$

De acordo com os valores de α_t no final de cada rodada, D_t é atualizada e normalizada com o objetivo de focar a atenção do algoritmo nos exemplos mais

Dado: $(x_1, y_1), \ldots, (x_m, y_m)$, onde $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Inicialize: $D_1(i) = 1/m$
Para $t = 1, \ldots, T$:

- Utilize o gerador de hipóteses fracas a partir da distribuição D_t.
- Armazene a hipótese fraca $h_t : X \rightarrow \mathbb{R}$.
- Escolha: $\alpha_t \in \mathbb{R}$
- Atualize:

$$D_{t+1}(i) = \frac{D_t(i)\, e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

onde Z_t é um fator de normalização (escolhido de tal forma que D_{t+1} seja uma distribuição).

Compute a hipótese forte:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Figura 2.1: Pseudo-código de uma versão generalizada do AdaBoost.

difíceis de serem classificados. Após a T-ésima rodada, a hipótese forte é computada através de um voto ponderado onde o peso de cada hipótese fraca h_t é dado por α_t.

De um ponto de vista pragmático, AdaBoost tem diversas vantagens como algoritmo de Boosting. Ele é relativamente rápido, simples e de fácil codificação. Além disso, não possui nenhum parâmetro a ser ajustado a não ser o número de hipóteses a ser gerado, T.

Capítulo 3

Support Vector Machines

3.1 Formulação da SVM Utilizada

Dentre as muitas formulações possíveis de máquinas de vetores de suporte, neste trabalho foi escolhida uma máquina que utiliza o princípio de funções de kernel para induzir o aprendizado em espaços de Hilbert, como descrito por Cristianini e Shawe-Taylor [CST00] e Haykin [Hay94]. Considere o seguinte problema de otimização para a formulação desta SVM, onde é mostrado o caso onde a flexibilização da margem é dada pela norma-1 do vetor de pesos:

$$\begin{array}{ll} \text{minimize}_{\xi,w,b} & \langle w \cdot w \rangle + C \sum_{i=1}^{l} \xi_i, \\ \text{sujeito a} & y_i \left(\langle w \cdot x_i \rangle + b \right) \geq 1 - \xi_i, i = 1, \ldots, l, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{array} \tag{3.1}$$

onde w é o vetor de pesos, x é o vetor de entrada, C é o parâmetro de regularização entre o erro de treinamento e a margem, e ξ é o vetor responsável pela flexibilização do hiperplano de separação.

O Lagrangiano para o problema de otimização descrito na Equação (3.1) é:

$$L_P \left(w, b, \xi, \alpha, r \right) = \frac{1}{2} \langle w \cdot w \rangle + 2 \sum_{i=1}^{l} \xi_i - \sum_{i=1}^{l} \alpha_i [y_i \left(\langle w \cdot x_i \rangle + b \right) - 1 + \xi_i] - \sum_{i=1}^{l} r_i \xi_i, \tag{3.2}$$

onde $\alpha_i \geq 0$ e $r_i \geq 0$. São então calculadas as derivadas parciais para L_P:

$$\frac{\partial L_P(w, b, \xi, \alpha, r)}{\partial w} = w - \sum_{i=1}^{l} y_i \alpha_i x_i = 0, \tag{3.3}$$

$$\frac{\partial L_P(w, b, \xi, \alpha, r)}{\partial \xi_i} = C - \alpha_i - r_i = 0, \tag{3.4}$$

$$\frac{\partial L_P(w, b, \xi, \alpha, r)}{\partial b} = \sum_{i=1}^{l} y_i \alpha_i = 0. \tag{3.5}$$

Ao substituir estas relações na forma primal, a seguinte forma dual do problema é obtida:

$$L_D(w, b, \xi, \alpha, r) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle. \tag{3.6}$$

As condições de complementaridade de Karush-Kuhn-Tucker para o problema são portanto:

$$\begin{array}{ll} \alpha_i [y_i (\langle w \cdot x_i \rangle + b) - 1 + \xi_i] = 0. & i = 1, \ldots, l, \\ \xi_i (\alpha_i - C) = 0, & i = 1, \ldots, l. \end{array} \tag{3.7}$$

A proposição 1 a seguir formaliza todo o problema.

Proposition 1. *Considere a classificação de um exemplo de treinamento:*

$$S = ((x_1, y_1), \ldots, (x_l, y_l)),$$

através do espaço implicitamente induzido pelo kernel $K(x, z)$, e suponha que os parâmetros α' resolvem o seguinte problema de otimização quadrática:

$$\begin{array}{ll} \textit{maximize} & L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(x_i, x_j), \\ \textit{sujeito a} & \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & C \geq \alpha_i \geq 0, i = 1, \ldots, l. \end{array}$$

Considere $f(x) = \sum_{i=1}^{l} y_i \alpha_i' K(x_i, x) + b'$, onde b' é escolhido tal que $y_i f(x_i) = 1$ para qualquer i com $C > \alpha_i' > 0$. Esta regra de decisão, dada por sign $(f(x))$, é equivalente ao hiperplano no espaço induzido implicitamente definido pelo kernel $K(x, z)$ que é resolvido pelo problema de otimização (3.1), onde as variáveis de folga (ξ) são definidas relativas à margem geométrica:

$$\gamma = \left(\sum_{j \in \{sv\}} y_i y_j \alpha_i' \alpha_j' K(x_i, x_j) \right)^{-\frac{1}{2}}.$$

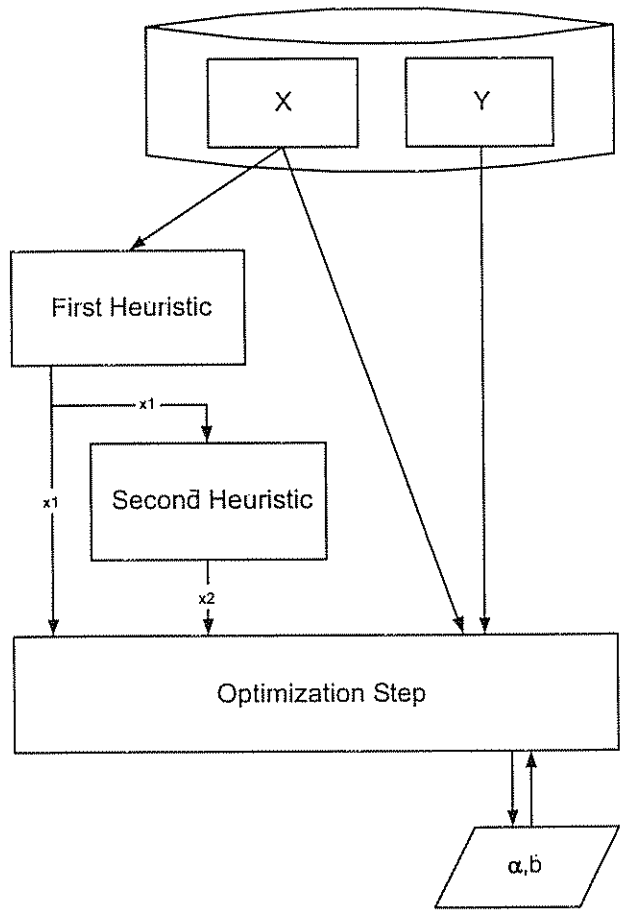


Figura 3.1: Desenho esquemático do funcionamento do algoritmo SMO.

3.2 Sequential Minimal Optimization

Nesta seção é descrito um dos algoritmos chave deste trabalho, o chamado *Sequential Minimal Optimization* (SMO), desenvolvido por John Platt [Pla98a, Pla98b]. Platt estendeu as idéias de Osuna et al. [OFG97] sobre decomposição de problemas de otimização quadrática (QP) em uma série de problemas menores com objetivo de diminuir a quantidade de recursos necessária pelo algoritmo. Platt levou o conceito ao extremo, onde o problema QP original é quebrado em problemas menores com apenas dois multiplicadores de Lagrange, que é o número mínimo necessário para que os subproblemas possam respeitar as condições de complementaridade de KKT. Uma das grandes vantagens do SMO é que a resolução de seus subproblemas, ao contrário dos diversos métodos numéricos para resolução de problemas de otimização, é feita analiticamente, garantindo desempenho em termos de tempo de

execução e evitando o acúmulo de erros numéricos.

O corpo do algoritmo SMO pode ser divido em três partes distintas: o método para resolução analítica dos pequenos problemas de otimização com dois multiplicadores de Lagrange, as heurísticas de seleção de quais dois multiplicadores de Lagrange serão escolhidos para serem otimizados a cada iteração e um método para calcular o valor de polarização (*threshold*) da SVM. Apesar dos detalhes dos métodos de resolução analítica de problemas QP e valores de polarização estarem além do escopo deste texto, a seguir são descritas as duas heurísticas de seleção dois multiplicadores de Lagrange, α_1 e α_2, mais relevantes neste trabalho:

Primeira heurística de seleção A primeira heurística de seleção compõe o laço mais externo do algoritmo SMO, que percorre todos os pontos a procura daqueles que violam as condições de KKT. Quando encontrados, estes pontos são selecionados para otimização e passados a segunda heurística de seleção.

Segunda heurística de seleção A segunda heurística de seleção leva em consideração a escolha feita para primeira e procura encontrar seu par, outro multiplicador de Lagrange, que proporcionará a maior contribuição do passo de otimização em direção à solução final. Para tal, esta heurística aproveita o cache de erros armazenado pelo algoritmo para avaliar qual escolha de multiplicadores de Lagrange produz a maior diferença entre os erros associados aos dois pontos em questão. Se esta seleção inicial falhar em produzir uma diferença significativa entre os erros dos pontos, por exemplo quando não há mais pontos que violam as condições de KKT, o algoritmo abandona esta premissa e passa a vasculhar cada um dos multiplicadores individualmente.

Note que todos as verificações das condições de KKT são feitas dentro de uma determinada margem de tolerância. De acordo com Platt [Pla98a], esta margem de tolerância inibe a incidência de erros numéricos, o que empiricamente favorece a convergência do algoritmo.

A Figura 3.2 traz o pseudo-código para o algoritmo SMO, como proposto originalmente por Platt [Pla98a], enquanto a Figura 3.1 traz seu desenho esquemático.

Figura 3.2: Pseudo-código do algoritmo *Sequential Minimal Optimization* (SMO).

---

target = desired output vector
point = training point matrix

```
procedure takeStep(i1,i2)
  if (i1 == i2) return 0
  alph1 = Lagrange multiplier for i1
  y1 = target[i1]
  E1 = SVM output on point[i1] - y1 (check in error cache)
  s = y1*y2
  Compute L, H                                                    10
  if (L == H)
    return 0
  k11 = kernel(point[i1],point[i1])
  k12 = kernel(point[i1],point[i2])
  k22 = kernel(point[i2],point[i2])
  eta = 2*k12-k11-k22
  if (eta < 0)
  {
    a2 = alph2 - y2*(E1-E2)/eta
    if (a2 < L) a2 = L                                            20
    else if (a2 > H) a2 = H
  }
  else
  {
    Lobj = objective function at a2=L
    Hobj = objective function at a2=H
    if (Lobj > Hobj+eps)
      a2 = L
    else if (Lobj < Hobj-eps)
      a2 = H                                                      30
    else
      a2 = alph2
  }
  if (|a2-alph2| < eps*(a2+alph2+eps))
    return 0
  a1 = alph1+s*(alph2-a2)
  Update threshold to reflect change in Lagrange multipliers
  Update weight vector to reflect change in a1 & a2, if linear SVM
  Update error cache using new Lagrange multipliers
  Store a1 in the alpha array                                     40
  Store a2 in the alpha array
  return 1
endprocedure

procedure examineExample(i2)
  y2 = target[i2]
```

```
  alph2 = Lagrange multiplier for i2
  E2 = SVM output on point[i2] - y2 (check in error cache)
  r2 = E2*y2
  if ((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))                    50
  {
    if (number of non-zero & non-C alpha > 1)
    {
      i1 = result of second choice heuristic
      if takeStep(i1,i2)
        return 1
    }
    loop over all non-zero and non-C alpha, starting at random point
    {
      i1 = identity of current alpha                                          60
      if takeStep(i1,i2)
        return 1
    }
    loop over all possible i1, starting at a random point
    {
      i1 = loop variable
      if takeStep(i1,i2)
        return 1
    }
  }                                                                           70
  return 0
endprocedure

main routine:
  initialize alpha array to all zero
  initialize threshold to zero
  numChanged = 0;
  examineAll = 1;
  while (numChanged > 0 | examineAll)
    {                                                                         80
      numChanged = 0;
      if (examineAll)
        loop I over all training examples
          numChanged += examineExample(I)
      else
        loop I over examples where alpha is not 0 & not C
          numChanged += examineExample(I)
      if (examineAll == 1)
        examineAll = 0
```

```
    else if (numChanged == 0)                                          90
      examineAll = 1
  }
```

Capítulo 4

Algoritmos Híbridos para Combinação de Boosting e SVMs

Este capítulo descreve os quatro algoritmos híbridos propostos para combinar Boosting e SVMs com diferentes níveis de acoplamento.

4.1 Integração Simples (SMO-B_α)

A forma mais simples e direta de integrar os algoritmos AdaBoost e SMO é utilizar suas proposições originais, onde o AdaBoost utiliza o SMO como seu gerador de hipóteses fracas. O diagrama esquemático para esta abordagem é mostrado na Figura 4.1. Mesmo que nenhuma modificação tenha sido introduzida ao AdaBoost ou SMO nesta versão, algumas observações são pertinentes. Primeiro, os dados do subconjunto de treinamento do SMO são selecionados através de um mecanismo de seleção com substituição, que pode inclusive duplicar instâncias do conjunto de treinamento. Segundo, cada hipótese fraca gerada pelo SMO deve ser armazenada pelo AdaBoost para ser mais tarde recriada na computação da hipótese forte. Para tal, foram armazenados, para cada SVM correspondente às hipóteses fracas, o vetor de multiplicadores de Lagrange α, o termo de polarização b e o mapeamento entre cada exemplo fornecido ao SMO e sua tupla original no conjunto de treinamento do AdaBoost. Note que não foi preciso armazenar quais vetores são considerados vetores de suporte em cada hipótese, uma vez que suas instâncias estão automaticamente associadas a multiplicadores de Lagrange com valores diferentes de zero.

O comportamento deste algoritmo híbrido é governado por dois parâmetros de regularização. O primeiro, h, determina o tamanho do subconjunto de dados de treinamento que é sorteado pelo mecanismo de sorteio probabilístico do AdaBoost para ser apresentado ao SMO. Este parâmetro ρ foi implementado como uma fração

da cardinalidade do conjunto original de treinamento. O segundo, T, é o número de hipóteses fracas que devem ser geradas e depois combinadas pelo AdaBoost em uma hipótese forte.

4.2 Seleção de Subconjuntos Modificada (SMO-B$_\beta$)

A primeira modificação proposta ao algoritmo **SMO-B$_\alpha$**, descrito na seção 4.1, consiste em uma alteração no mecanismo de seleção probabilística que não permita a seleção de duplicatas. Este mecanismo é referido como seleção sem repetição, onde o conceito de subconjunto matemático é reforçado tal que $\langle X', Y' \rangle \subseteq \langle X, Y \rangle$. Na literatura, este algoritmo é conhecido como seleção através de uma roleta probabilística [BLC00], que neste caso faz sorteios baseados em uma distribuição D. O desenho esquemático para esta versão modificada de algoritmo híbrido é apresentado na Figura 4.2.

Existem duas motivações para a modificação do seletor probabilístico original. Primeiramente, foi observado empiricamente que o mecanismo de atualização da distribuição de probabilidades do AdaBoost coloca uma grande ênfase em *outliers*. O efeito colateral indesejável da seleção com substituição para esta propriedade faz com que o conjunto de treinamento apresentado ao SMO tenda a conter repetidas duplicadas destes *outliers*, portanto polarizando excessivamente a geração de hipóteses fracas. Além disso, o SMO trata como exceção o caso onde dois multiplicadores de Lagrange correspondentes a vetores iguais são escolhidos para otimização, onde, de acordo com a formulação do método de resolução analítica do problema QP, uma divisão por zero é encontrada.

De acordo com a formulação de Boosting para problemas de classificação binária dada no Capítulo 2, AdaBoost requer um gerador de hipóteses fracas cujo erro médio de validação seja, no mínimo, melhor que uma hipótese aleatória, isto é, $\epsilon < 1/2$. Apesar de haver argumentos suficientes para modificar o mecanismo original de seleção probabilística do AdaBoost, mais um efeito colateral indesejado é encontrado quando uma das propriedades de convergência da função recursiva de atualização da distribuição de probabilidades é violada. O resultado da quebra desta propriedade é que, para casos de subconjuntos de dados degenerados, a restrição que demanda $\epsilon < 1/2$ também é quebrada. Note que se fosse permitida uma hipótese fraca com $\epsilon < 1/2$, seu peso correspondente ω atribuído pelo AdaBoost seria negativo, o que violaria o princípio de voto ponderado. A maioria das implementações de algoritmos de Boosting aborta sua execução quando uma hipótese como esta é encontrada. Para contornar o problema, entretanto, foi introduzido um teste de consistência para hipóteses fracas após serem geradas pelo SMO, antes de

serem consideradas pelo AdaBoost. Caso uma hipótese fraca seja gerada tal que $\epsilon < 1/2$, esta hipótese é descartada e o algoritmo procede para uma nova iteração através de um novo sorteio de um diferente subconjunto de treinamento. Note que, uma vez que um número arbitrário de hipóteses pode ser ignorado pelo algoritmo, após T iterações haverá T' hipóteses fracas válidas para computar a hipótese forte, onde $T' \leq T$.

4.3 Eliminação da Primeira Heurística de Seleção (SMO-B_γ)

Após serem introduzidas as duas estratégias de integração mais simples utilizadas no **SMO-B_α** e **SMO-B_β**, agora são introduzidas estratégias mais avançadas que não só propõe modificações unilaterais no AdaBoost ou SMO, mas também a combinação e eliminação de alguns de seus componentes. Os componentes de ambos algoritmos que compartilham alguma funcionalidade em comum são o mecanismo de seleção probabilística do AdaBoost, já considerando a modificação de seleção sem repetição introduzida em **SMO-B_β** e as heurísticas de seleção de multiplicadores de Lagrange do SMO. Em uma primeira tentativa, o mecanismo de seleção probabilística de um subconjunto de treinamento do AdaBoost e a primeira heurística de seleção do SMO foram eliminados. A cada iteração, o mesmo princípio de sorteio probabilístico foi utilizado para sortear a primeira instância do problema QP a ser otimizado. Esta instância, assim como na versão original que utilizada a primeira heurística de seleção, é alimentada à segunda heurística de seleção que desta vez examina todo o conjunto de treinamento, e não mais somente o subconjunto pré-selecionado, para escolher a segunda instância que maximiza a contribuição do passo de otimização dado em cada iteração do SMO.

Note que agora foi confiado ao mecanismo de atualização da distribuição D do AdaBoost não só a geração de subconjuntos de treinamento para construir boas hipóteses fracas, mas também a escolha do primeiro ponto do problema QP de dois multiplicadores de Lagrange a ser analiticamente resolvido. O princípio por traz da primeira heurística do SMO, que é responsável por garantir a convergência do algoritmo através da seleção de pontos que violem as condições de KKT, é portanto substituído pela premissa de que pontos que violem as condições KKT serão probabilisticamente selecionados de acordo com seu erro de treinamento ao longo das épocas de Boosting e da atualização da distribuição D. Assim como no **SMO-B_β**, o mesmo mecanismo de eliminação de hipóteses fracas inválidas é necessário para evitar casos onde $\omega < 0$. O desenho esquemático para o algoritmo com esta fusão entre componentes do AdaBoost e SMO é apresentado na Figura 4.3.

4.4 Eliminação da Segunda Heurística de Seleção (SMO-B_δ)

A abordagem utilizada no **SMO-B_γ** é agora estendida para ambas heurísticas de seleção do SMO. As duas heurísticas são, portanto, substituídas por uma versão modificada de um mecanismo de seleção probabilística, onde as duas instâncias correspondentes aos dois multiplicadores de Lagrange do problema QP a ser resolvido analiticamente são selecionados do conjunto original de treinamento de acordo com a distribuição D. O desenho esquemático para este algoritmo é apresentado na Figura 4.4.

Para o **SMO-B_δ**, ainda mais que no **SMO-B_γ**, é confiada ao algoritmo de Boosting a seleção de pontos que violem as condições de KKT e consequentemente façam com que o SMO convirja. Esta seleção corresponde à seleção probabilística de dois pontos do conjunto de treinamento, onde é esperado que os valores de probabilidade associados a cada ponto implicitamente enfatizem as instâncias mais difíceis de serem aprendidas, o que por sua vez correspondem, intuitivamente, aos pontos que mais violam as condições de KKT. Mais uma vez, assim como em **SMO-B_β** e **SMO-B_γ**, o mesmo mecanismo de eliminação de hipóteses fracas inválidas é necessário para evitar casos onde $\omega < 0$.

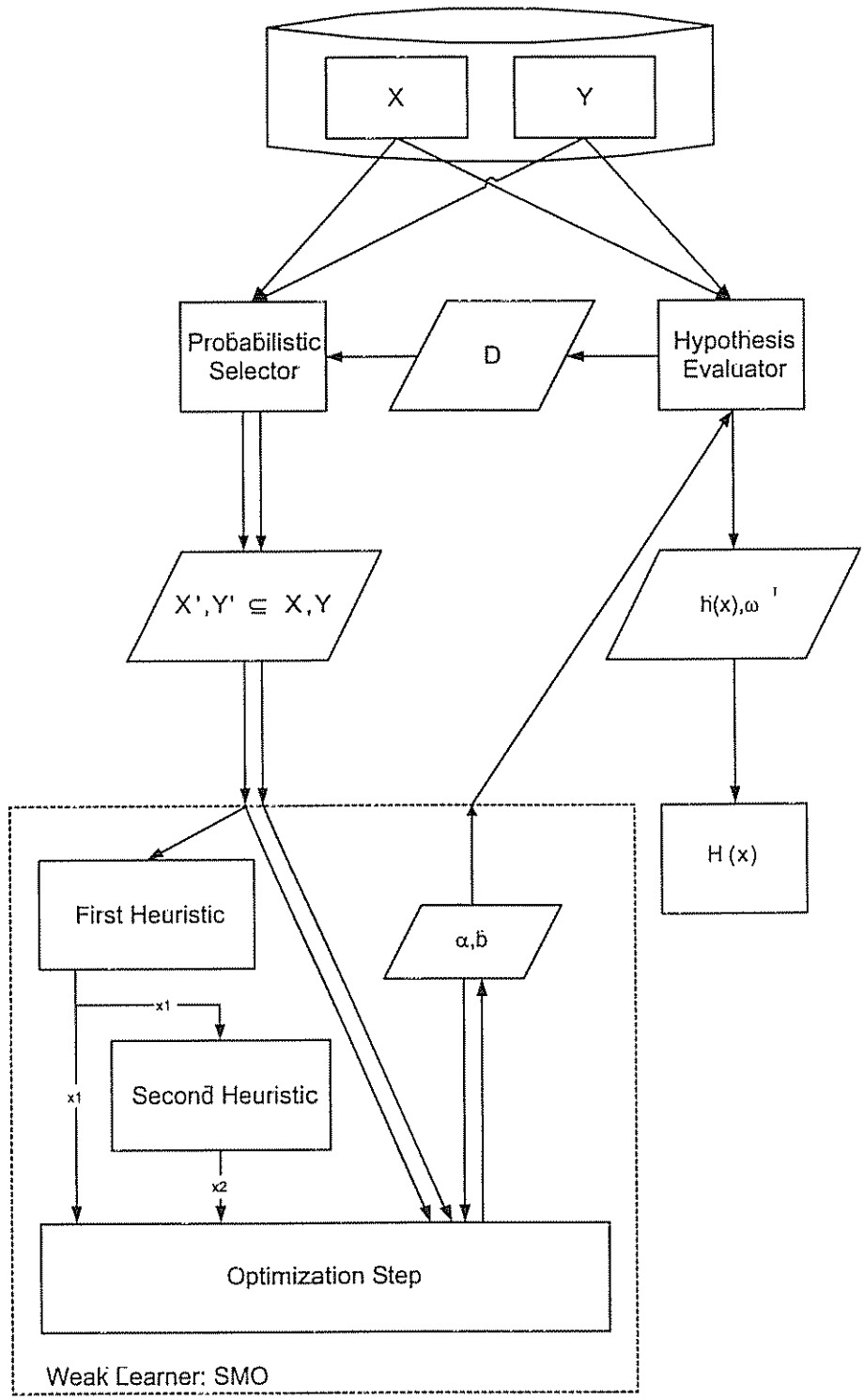


Figura 4.1: Desenho esquemático do algoritmo híbrido SMO-B_α.

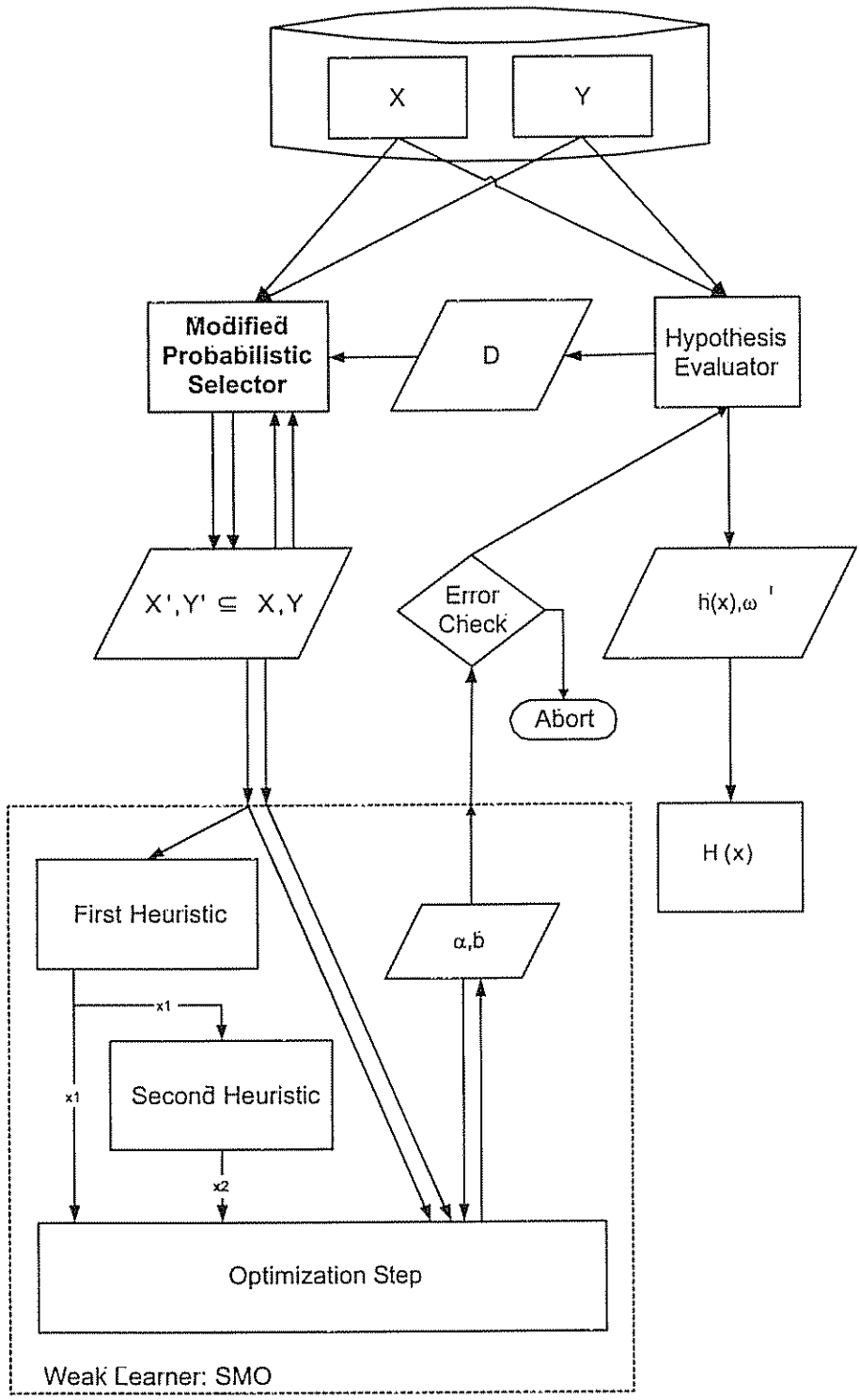


Figura 4.2: Desenho esquemático do algoritmo híbrido **SMO-B**$_\beta$.

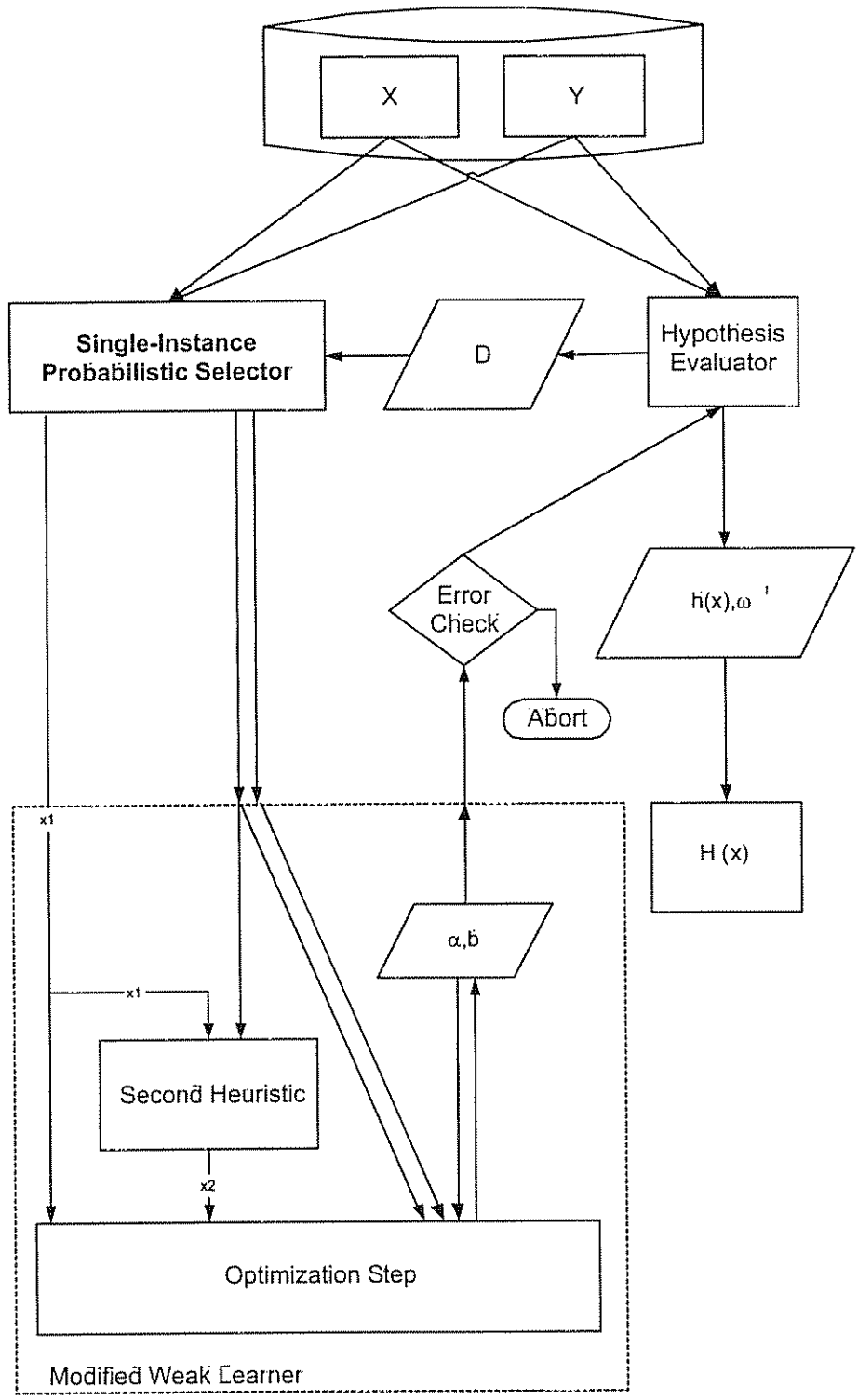


Figura 4.3: Desenho esquemático do algoritmo híbrido **SMO-B**$_\gamma$.

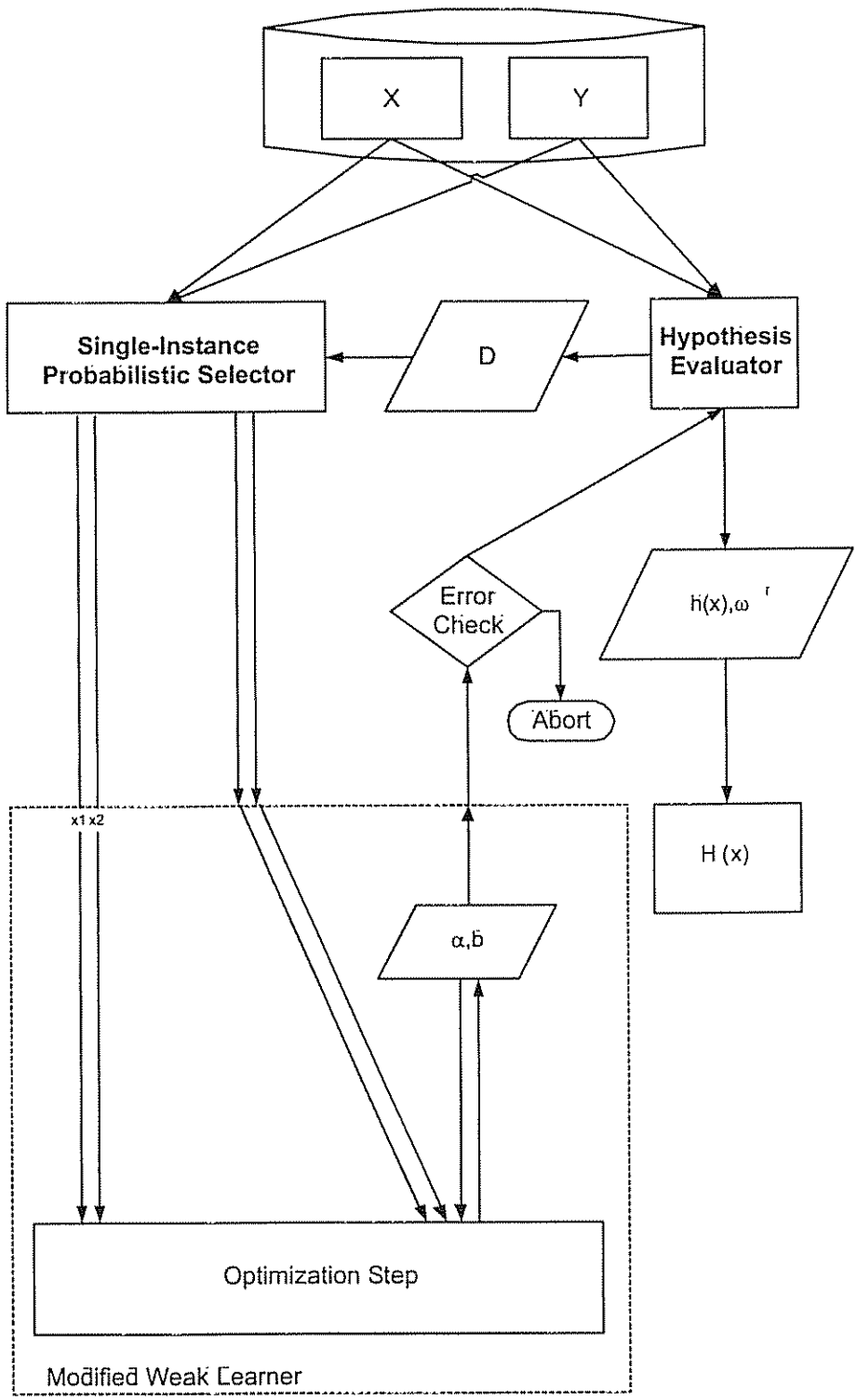


Figura 4.4: Desenho esquemático do algoritmo híbrido **SMO-B**$_\delta$.

Capítulo 5

Experimentos e Resultados

Este capítulo descreve a metodologia de todos os experimentos executados neste trabalho, bem como seus resultados e análises.

5.1 Descrições de Bases de Dados

Esta seção traz uma breve descrição das bases de dados utilizadas nos experimentos executados. Foram selecionados 22 diferentes bases de dados de fontes distintas, cada uma com suas características diferentes. Todos os conjuntos de dados têm saída binária em $\{-1, +1\}$ e vetores de entrada em $[-\infty, +\infty]$, portanto satisfazendo os requisitos necessários para serem utilizados em experimentos de classificação binária.

Pode-se classificar estas 22 bases de dados em quatro grupos de acordo com a origem de seus problemas:

Biologia Foram selecionados 8 bases de dados de problemas de biologia, muitos deles problemas de diagnóstico de doenças. Destas 8 bases de dados, 2 delas foram disponibilizadas por seus autores com conjuntos distintos de treinamento e teste, que são muito úteis para que sejam comparadas as capacidades de generalização de diferentes algoritmos. Muitas destas bases de dados são clássicas disponíveis no UCI Repository of Machine Learning Databases [BM98], tendo sido utilizadas em diversos trabalhos da literatura. Outras são frutos de trabalhos recentes que combinam avanços da biologia molecular com modernas técnicas de reconhecimento de padrões.

Genômica Há 2 bases de dados correspondentes a problemas de bioinformática. Um deles é baseado no diagnóstico de tecidos cancerosos a partir de amostras analisadas com *microarrays*, e outro consiste na determinação de sequências

promotoras de genes. A primeira base foi obtida de um estudo recente de classificação multi-classe de tipos de tecido a partir de SVMs, e a segunda foi obtida do UCI Repository of Machine Learning Databases [BM98].

Física Foi selecionada uma única base de dados correspondente a um problema que descreve um experimento com ondas de radar, também obtido do UCI Repository of Machine Learning Databases [BM98].

Sintéticos Bi-dimensionais Foram geradas 9 bases de dados sintéticas com entradas bi-dimensionais. Estas bases foram geradas a partir de três funções, cada uma parametrizada de 3 formas distintas correspondentes a diferentes graus de sobreposição entre classes.

Sintéticos Multi-dimensionais Foram, por fim, geradas 2 bases de dados sintéticas com entradas de 20 dimensões. Estas bases foram originalmente criadas por Leo Breiman em um estudo sobre polarização e variância de algoritmos de aprendizado [Bre96].

Com exceção das duas bases de dados com conjuntos de treinamento e validação disponibilizados separadamente, nenhuma das outras bases de dados possuiam tal distinção. Todos os experimentos com estas bases de dados foram executados a partir de conjuntos de treinamento e validação sorteados para cada repetição.

A tendência a favor da seleção de bases de dados para problemas de biologia e bioinformática foi intencional e é justificável. Estes problemas frequentemente apresentam desafios interessantes à máquinas de aprendizado, uma vez que geralmente contém grandes massas de dados organizadas em vetores com alta dimensionalidade. Portanto, o uso de máquinas avançadas, como SVMs, na sua resolução motivam o desenvolvimento de novas abordagens que possam ser ainda mais eficientes. Por exemplo, a habilidade de SVMs lidarem com grandes bases de dados fez com que fossem utilizadas em diversos problemas relacionados a dados de *microarray* e expressão de genes, como a classificação de genes de acordo com sua função [BGL+99, KK01] e a classificação de tipos de tecidos cancerosos [CDH+00, RTR+01].

5.2 Análises Preliminares

Antes de serem realizados experimentos usando algoritmos híbridos que combinam SVMs com Boosting, foram conduzidas análises preliminares sobre cada base de dados utilizando outras máquinas de aprendizado conhecidas. A primeira análise

realizada procurou determinar a separabilidade linear das bases dados, onde foram utilizadas simples máquinas lineares.

Em uma segunda análise, foi utilizada a versão tradicional do SMO para ajustar os parâmetros de regularização das SVMs bem como os parâmetros de seus kernels. O objetivo desta análise foi determinar valores para C, o parâmetro que regula o compromisso entre erro de treinamento e a margem, bem como os parâmetros do kernel RBF utilizado. Uma vez ajustados para cada base de dados neste passo de análise preliminar, estes parâmetros foram mantidos fixos para todos os experimentos subsequentes.

Note que para todos os resultados apresentados e discutidos nesta seção, foram transcritos em detalhe apenas os resultados uma das bases de dados, **bcw** (Wisconsin Breast Cancer), devido a limitações com relação ao tamanho do texto.

5.2.1 Discriminante Linear

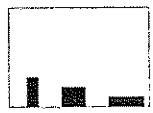
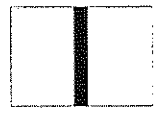
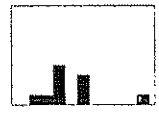
A primeira análise a qual todas as bases de dados selecionadas foram submetidas procurou verificar sua propriedade de separabilidade linear. Esta análise consistiu em tentar resolver cada problema de classificação com uma rede Perceptron com um único neurônio [Ros58].

De acordo com a teoria por trás do neurônio de McCulloch e Pitts [MP43], um único neurônio é capaz de corretamente classificar apenas problemas linearmente separáveis [Hay94, BLC00]. Uma vez que sua saída depende somente do produto escalar do vetor de pesos e do vetor de entrada, que pode ou não exceder um valor de polarização, esta máquina linear falha em descrever o que Duda, Hart e Stork chamaram de o mais simples problema não linear de todos, que é o resultado de uma função booleana bi-dimensional XOR. Este foi um dos argumentos que Minsky e Papert [MP69] usaram para delinear as deficiências do Perceptron em 1969.

Cada base de dados foi repetidamente analisada com a rede Perceptron por 10 vezes, cada vez com diferentes seleções de conjuntos de treinamento e validação correspondendo sempre a 70% e 30% do conjunto total de dados, respectivamente. O nodo Perceptron utilizado tinha um coeficiente de aprendizado $\eta = 0.01$, um erro de tolerância $tol = 0.1$ e um número máximo de épocas de treinamento $epoch_{max} = 100000$.

A Tabela 5.1 traz resultados do experimento para a base **bcw**. São mostrados a precisão do modelo (acurácia), o erro quadrático médio (MSE), o número de iterações gastos no treinamento, o tempo de treinamento e, para cada uma destas medidas, os histogramas correspondentes às 10 repetições do experimento com diferentes conjuntos de treinamento e validação.

Tabela 5.1: Resultados médios para uma rede Perceptron com um único neurônio após 10 rodadas de experimentos com conjuntos distintos de treinamento e validação.

Base	Acurácia	MSE	Iterações	Tempo de execução
				
bcw	96.24% ± 1.81%	0.0376 ± 0.0181	100000 ± 0	4.18 s ± 0.03 s

5.2.2 Ajuste de Parâmetros

O objetivo desta análise foi determinar o conjunto de parâmetros particulares das SVMs utilizadas que foram fixados nos experimentos com algoritmos híbridos. Estes parâmetros foram C, que regula o compromisso entre erro de treinamento e margem da SVM, e os dois parâmetros do kernel RBF utilizado, p_{sigma^2} e p_s. A função de base radial utilizada como kernel em todos os experimentos como pode ser definida como $\exp\left(-\frac{\|x-x_i\|^2}{2\sigma^2 s}\right)$, onde s é o parâmetro de escala linear e σ é o parâmetro de variância. Há dois importantes objetivos para realizar tal análise:

Ajuste de parâmetros. Os melhores conjuntos de parâmetros encontrados nesta análise para cada uma das 22 bases de dados foram mantidos fixos nos experimentos com algoritmos híbridos para consistência de resultados.

Referência de desempenho. A medida que os resultados de algoritmos híbridos são descritos, é interessante compará-los à versão padrão do SMO proposta por Platt [Pla98a]. Os parâmetros C, p_{σ^2} e p_s foram, portanto, preservados entre as execuções com SMO e os quatro algoritmos híbridos, evitando comparações de desempenho contaminadas por diferentes parâmetros da SVM.

A Tabela 5.2 traz os resultados para este experimento realizado sobre a base de dados **bcw** (Wisconsin Breast Cancer). São mostrados gráficos tri-dimensionais da taxa de classificação correta (acurácia), erro quadrático médio, taxa de vetores de suporte, norma dos hiperplanos de separação, número de iterações e tempo de execução. Cada grandeza é apresentada nos eixos z de cada gráfico, onde os eixos x e y correspondem a variações de p_{σ^2} e p_s. Por fim, para cada uma destas seis medidas, três gráficos independentes são mostrados para diferentes valores de C.

Além destes gráficos, a Tabela 5.2 traz também os dados e os histogramas particulares da configuração que resultou no melhor desempenho do algoritmo após

as 10 rodadas de experimentos. Os dois critérios utilizados para ordenar e selecionar os melhores parâmetros foram:

- Melhor acurácia;
- Menor tempo de execução.

Em casos de empate após a aplicação destes dois critérios, os conjuntos finalistas foram escolhidos aleatoriamente.

Tabela 5.2: Resultados médios para o SMO padrão após 10 rodadas de experimentos com conjuntos distintos de treinamento e validação.

bcw (gráficos da média de resultados para for $p_{\sigma^2} \times p_s$)

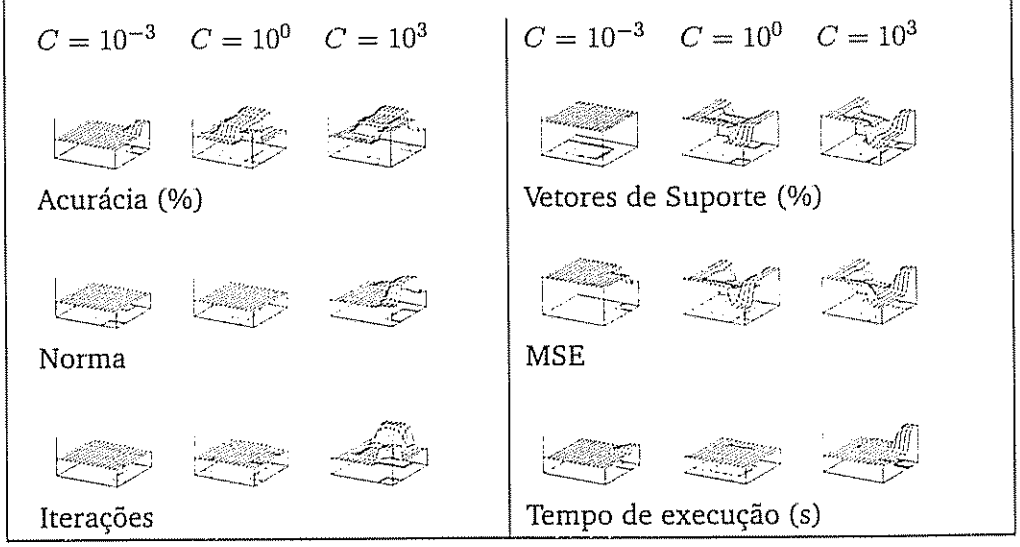
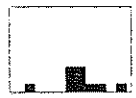
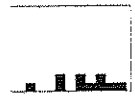
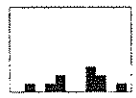
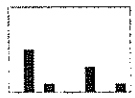


Tabela 5.2: (continuado)

bcw (histograma de experimentos com melhor desempenho)	
 Acurácia (%): 96.10% ± 0.97%	 Vetores de Suporte (%): 9.06% ± 1.43%
 Norma: 6075.9512 ± 584.4494	 MSE: 0.0390 ± 0.0097
 Iterações: 531 ± 209	 Tempo de execução (s): 5.46 s ± 2.40 s

5.3 Resultados Experimentais

Cada um destes experimentos foi executado com diferentes configurações de parâmetros que procuraram induzir diferentes padrões de comportamento nos algoritmos híbridos. Assim como na Seção 5.2, nestes experimentos foram observados os resultados dos algoritmos baseados na variação de dois parâmetros:

- T, o número de hipóteses fracas que o algoritmo de Boosting deve construir para depois combinar em uma hipótese forte através de votos ponderados, escolhido do conjunto $\{1, 100, 1000\}$;
- ρ, a fração da cardinalidade do conjunto de treinamento que determina a cardinalidade do subconjunto de treinamento selecionado através da distribuição de probabilidades mantida pelo algoritmo de Boosting, escolhida do conjunto $\{0.1, 0.4, 0.7, 1.0\}$.

Todas as 12 possíveis combinações destes dois conjuntos de parâmetros foram testadas sobre as 22 bases de dados com cada um dos quatro algoritmos híbridos. Na maioria dos casos, os experimentos foram repetidos 10 vezes para evitar anomalias estatísticas. Em poucos casos, devido a limitações de recursos computacionais, alguns experimentos foram executados menos de 10 vezes.

5.3.1 Resultados para SMO-B_α

O primeiro algoritmo híbrido proposto, SMO-B_α, é a forma mais direta e inocente de integrar Boosting e SVMs. Os resultados de sua execução para a base **bcw** estão transcritos na Tabela 5.3. Esta tabela apresenta inicialmente os resultados médios para as hipóteses finais computadas pelo algoritmo de Boosting. Além da descrição textual dos resultados, que mostra médias e desvios padrão correspondentes, gráficos tri-dimensionais são utilizados para descrever cada uma das medidas de desempenho de acordo com a variação dos dois parâmetros, T e ρ, nos eixos x e y respectivamente, onde T é mostrado em escala logarítmica.

Além disso, a Tabela 5.3 traz resultados médios para as hipóteses fracas criadas pelo algoritmo de Boosting. Como nos resultados das hipóteses fortes, estes resultados também são apresentados no formato de gráficos tri-dimensionais. Finalmente, a melhor configuração de parâmetros é destacada na Tabela 5.3, onde os três critérios utilizados para se determinar a melhor configuração são:

- Melhor acurácia;
- Menor tempo de execução;
- Maior número de hipóteses fracas válidas.

Em casos de empate após a aplicação destes três critérios, os conjuntos finalistas foram escolhidos aleatoriamente. Por fim, a Tabela 5.3 traz também os histogramas de dispersão para a execução desta melhor configuração de parâmetros do algoritmo.

Tabela 5.3: Média de resultados e melhor resultado para o algoritmo híbrido SMO-B_α após 10 rodadas de experimentos com conjuntos distintos de treinamento e validação.

bcw (média de resultados para hipóteses fortes)

T	ρ	Acurácia (%)	Hipóteses fracas válidas (%)	Tempo de execução (s)
1000	0.4	51.00% ± 16.81%	3.74% ± 1.84%	24.36 s ± 15.15 s
1	0.4	53.67% ± 14.99%	100.00% ± 0.00%	0.99 s ± 0.87 s
1	**0.1**	**87.71% ± 12.38%**	**100.00% ± 0.00%**	**0.03 s ± 0.03 s**
1000	0.1	50.14% ± 15.43%	4.90% ± 1.84%	1.13 s ± 0.12 s
1000	0.7	49.19% ± 15.41%	3.39% ± 1.70%	102.87 s ± 63.15 s
100	1	47.19% ± 15.17%	16.40% ± 10.86%	186.25 s ± 112.08 s
1	0.7	56.52% ± 13.98%	100.00% ± 0.00%	18.03 s ± 24.23 s
1000	1	44.71% ± 14.50%	1.60% ± 0.56%	178.40 s ± 98.95 s
100	0.7	50.05% ± 15.43%	30.20% ± 8.96%	127.54 s ± 118.44 s
100	0.4	53.86% ± 14.94%	35.30% ± 19.62%	21.25 s ± 12.50 s
1	1	47.29% ± 15.19%	100.00% ± 0.00%	228.99 s ± 271.93 s
100	0.1	49.19% ± 19.95%	55.10% ± 33.19%	0.59 s ± 0.34 s

bcw (média de resultados para hipóteses fortes para $T \times \rho$)

Acurácia (%)

Hip. fracas válidas (%)

Tempo de execução (s)

Tabela 5.3: (continuado)

bcw (média de resultados para hipóteses fracas)

T	ρ	Alpha	Taxa de erro (%)	Vetores de suporte (%)	Norma	Iterações do SMO
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	6.81% ± 2.54%	17973.7100 ± 17899.1636	301 ± 243
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	10.81% ± 2.51%	76494.4166 ± 57033.0937	1797 ± 2095
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	1.75% ± 1.04%	334.7604 ± 184.4323	16 ± 9
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	16.67% ± 5.35%	322822.2248 ± 248327.9780	8017 ± 7173
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	2.06% ± 1.12%	2562.8477 ± 1283.3616	57 ± 30
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	2.27% ± 0.70%	4831.3097 ± 3590.3131	102 ± 68
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.15% ± 0.06%	27.4609 ± 11.5640	2 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	1.37% ± 0.75%	4123.0772 ± 1374.9117	86 ± 37
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.22% ± 0.11%	251.2781 ± 161.4524	6 ± 3
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	0.25% ± 0.11%	395.3559 ± 203.2713	9 ± 5
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	0.14% ± 0.04%	403.1479 ± 178.5044	8 ± 3
1	**0.1**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**3.38% ± 0.84%**	**969.5497 ± 2283.8921**	**38 ± 50**

bcw (média de resultados para hipóteses fracas para $T \times \rho$)

Alpha

Taxa de erro (%)

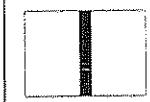
Vetores de suporte (%)

Norma

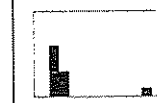
Iterações do SMO

bcw (histogramas de execuções para configuração com melhor desempenho)

Acurácia (%): 87.71% ± 12.38%



Hip. fracas válidas (%): 87.71% ± 12.38%



Tempo de execução (s): 0.03 s ± 0.03 s

5.3.2 Resultados para SMO-B_β

O segundo algoritmo híbrido proposto, SMO-B_β, é a evolução natural de SMO-B_α que procura resolver o problema de ignorar muitas hipóteses fracas. Os resultados para o SMO-B_β são mostrados na Tabela 5.4, que possui o mesmo formato que a Tabela 5.3. Os critérios de seleção das melhores configurações foram os mesmos descritos anteriormente na Seção 5.3.1 para SMO-B_α.

Tabela 5.4: Média de resultados e melhor resultado para o algoritmo híbrido SMO-B_β após 10 rodadas de experimentos com conjuntos distintos de treinamento e validação.

bcw (média de resultados para hipóteses fortes)

T	ρ	Acurácia (%)	Hipóteses fracas válidas (%)	Tempo de execução (s)
1000	0.4	97.62% ± 0.00%	3.70% ± 0.00%	905.42 s ± 0.00 s
1	**0.4**	**97.62% ± 0.00%**	**100.00% ± 0.00%**	**0.04 s ± 0.00 s**
1	0.1	96.19% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	96.67% ± 0.00%	99.90% ± 0.00%	14.31 s ± 0.00 s
1000	0.7	96.19% ± 0.00%	1.10% ± 0.00%	2805.17 s ± 0.00 s
100	1	96.67% ± 0.00%	100.00% ± 0.00%	444.31 s ± 0.00 s
1	0.7	97.14% ± 0.00%	100.00% ± 0.00%	2.12 s ± 0.00 s
1000	1	96.67% ± 0.00%	100.00% ± 0.00%	4491.92 s ± 0.00 s
100	0.7	97.14% ± 0.00%	10.00% ± 0.00%	285.85 s ± 0.00 s
100	0.4	95.24% ± 0.00%	14.00% ± 0.00%	100.29 s ± 0.00 s
1	1	96.67% ± 0.00%	100.00% ± 0.00%	5.04 s ± 0.00 s
100	0.1	97.14% ± 0.00%	100.00% ± 0.00%	1.20 s ± 0.00 s

bcw (média de resultados para hipóteses fortes para $T \times \rho$)

Acurácia (%)

Hip. fracas válidas (%)

Tempo de execução (s)

Tabela 5.4: (continuado)

bcw (média de resultados para hipóteses fracas)

T	ρ	Alpha	Taxa de erro (%)	Vetores de suporte (%)	Norma	Iterações do SMO
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	3.33% ± 0.00%	1666.2279 ± 0.0000	21 ± 0
1	0.7	2.0482 ± 0.0000	0.04% ± 0.00%	15.24% ± 0.00%	5772.3096 ± 0.0000	368 ± 0
100	0.1	9.8656 ± 0.0000	0.00% ± 0.00%	3.39% ± 0.00%	703.8516 ± 0.0000	36 ± 0
1	1	1.9346 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10022.3125 ± 0.0000	663 ± 0
100	0.4	0.0247 ± 0.0000	0.02% ± 0.00%	14.10% ± 0.00%	6115.5771 ± 0.0000	335 ± 0
100	0.7	-0.1583 ± 0.0000	0.04% ± 0.00%	19.17% ± 0.00%	7206.9525 ± 0.0000	459 ± 0
1000	0.1	9.3123 ± 0.0000	0.00% ± 0.00%	3.59% ± 0.00%	833.1179 ± 0.0000	39 ± 0
100	1	0.0193 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10056.9649 ± 0.0000	547 ± 0
1000	0.4	-0.1294 ± 0.0000	0.02% ± 0.00%	13.76% ± 0.00%	5602.5713 ± 0.0000	325 ± 0
1000	0.7	-0.3749 ± 0.0000	0.04% ± 0.00%	19.46% ± 0.00%	8896.0666 ± 0.0000	454 ± 0
1000	1	0.0019 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10054.5734 ± 0.0000	549 ± 0
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	2.86% ± 0.00%	187.4276 ± 0.0000	11 ± 0

bcw (média de resultados para hipóteses fracas para $T \times \rho$)

Alpha

Taxa de erro (%)

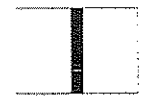
Vetores de suporte (%)

Norma

Iterações do SMO

bcw (histogramas de execuções para configuração com melhor desempenho)

Acurácia (%): 97.62% ± 0.00%



Hip. fracas válidas (%): 97.62% ± 0.00%

Tempo de execução (s): 0.04 s ± 0.00 s

5.3.3 Resultados para SMO-B$_\gamma$

SMO-B$_\gamma$ é o primeiro algoritmo híbrido proposto que tenta combinar componentes do SMO com componentes do AdaBoost.M1, ao invés de simplesmente tentar integrá-los como no **SMO-B$_\alpha$** e **SMO-B$_\beta$**. Os resultados para o **SMO-B$_\gamma$** são mostra-

dos na Tabela 5.5, que possui o mesmo formato que as Tabelas 5.3 e 5.4. Os critérios de seleção das melhores configurações foram os mesmos descritos anteriormente na Seção 5.3.1 para SMO-B_{α}.

Tabela 5.5: Média de resultados e melhor resultado para o algoritmo híbrido SMO-B_{γ} após 10 rodadas de experimentos com conjuntos distintos de treinamento e validação.

bcw (média de resultados para hipóteses fortes)

T	ρ	Acurácia (%)	Hipóteses fracas válidas (%)	Tempo de execução (s)
1000	0.4	96.29% ± 1.14%	18.15% ± 2.54%	35.43 s ± 4.12 s
1	0.4	90.24% ± 9.43%	100.00% ± 0.00%	0.05 s ± 0.01 s
1	0.1	95.52% ± 2.38%	100.00% ± 0.00%	0.03 s ± 0.01 s
1000	0.1	96.14% ± 1.12%	32.91% ± 4.01%	18.06 s ± 0.98 s
1000	**0.7**	**96.52% ± 1.17%**	**14.37% ± 3.00%**	**45.96 s ± 4.66 s**
100	1	96.19% ± 1.11%	21.40% ± 4.92%	5.31 s ± 0.60 s
1	0.7	91.19% ± 7.89%	100.00% ± 0.00%	0.06 s ± 0.01 s
1000	1	96.29% ± 0.85%	12.07% ± 4.28%	51.64 s ± 5.96 s
100	0.7	96.33% ± 1.04%	23.00% ± 3.69%	4.64 s ± 0.56 s
100	0.4	96.19% ± 1.02%	27.90% ± 5.56%	3.67 s ± 0.43 s
1	1	91.33% ± 8.41%	100.00% ± 0.00%	0.08 s ± 0.01 s
100	0.1	96.38% ± 1.19%	45.30% ± 7.84%	1.83 s ± 0.10 s

bcw (média de resultados para hipóteses fortes para $T \times \rho$)

Acurácia (%)

Hip. fracas válidas (%)

Tempo de execução (s)

Tabela 5.5: (continuado)

bcw (média de resultados para hipóteses fracas)

T	ρ	Alpha	Taxa de erro (%)	Vetores de suporte (%)	Norma
1	0.4	1.3624 ± 0.4054	0.19% ± 0.19%	15.76% ± 3.36%	399.5661 ± 122.6807
1	0.7	1.3731 ± 0.4324	0.19% ± 0.17%	26.38% ± 5.13%	526.2639 ± 156.5876
100	0.1	-0.0144 ± 0.0393	0.35% ± 0.08%	12.67% ± 0.67%	398.4358 ± 21.4161
1	1	1.3330 ± 0.4916	0.22% ± 0.22%	37.71% ± 5.19%	656.4435 ± 164.6508
100	0.4	-0.1293 ± 0.0438	0.15% ± 0.03%	25.65% ± 3.48%	626.1287 ± 79.9194
100	0.7	-0.2106 ± 0.0500	0.14% ± 0.03%	31.87% ± 4.38%	714.4174 ± 104.9769
1000	0.1	-0.0853 ± 0.0188	0.33% ± 0.05%	13.26% ± 0.83%	400.7593 ± 26.2831
100	1	-0.1961 ± 0.0672	0.12% ± 0.02%	35.89% ± 4.82%	756.3575 ± 114.6263
1000	0.4	-0.3065 ± 0.0576	0.16% ± 0.03%	25.54% ± 3.33%	639.6772 ± 71.2488
1000	**0.7**	**-0.3853 ± 0.0892**	**0.13% ± 0.02%**	**32.47% ± 3.74%**	**725.3301 ± 97.6372**
1000	1	-0.4039 ± 0.0785	0.13% ± 0.02%	35.92% ± 4.73%	789.2475 ± 96.5533
1	0.1	1.5457 ± 0.2571	0.12% ± 0.07%	5.43% ± 1.57%	194.0201 ± 111.1667

bcw (média de resultados para hipóteses fracas para $T \times \rho$)

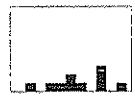
Alpha

Taxa de erro (%)

Vetores de suporte (%)

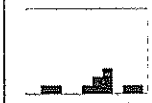
Norma

bcw (histogramas de execuções para configuração com melhor desempenho)



Acurácia (%): 96.52% ± 1.17%

Hip. fracas válidas (%): 0.10% ± 0.00%



Tempo de execução (s): 45.96 s ± 4.66 s

5.3.4 Resultados para SMO-B$_\delta$

Assim como SMO-B$_\beta$ sucedeu SMO-B$_\alpha$, SMO-B$_\delta$ é a evolução natural de SMO-B$_\gamma$, aumentando ainda mais o grau de acoplamento entre SMO e AdaBoost.M1. Os resultados para o SMO-B$_\gamma$ são mostrados na Tabela 5.6, que possui o mesmo formato

que as Tabelas 5.3 e 5.4, 5.5. Os critérios de seleção das melhores configurações foram os mesmos descritos anteriormente na Seção 5.3.1 para SMO-B_α.

Tabela 5.6: Média de resultados e melhor resultado para o algoritmo híbrido SMO-B_δ após 10 rodadas de experimentos com conjuntos distintos de treinamento e validação.

bcw (média de resultados para hipóteses fortes)

T	ρ	Acurácia (%)	Hipóteses fracas válidas (%)	Tempo de execução (s)
1000	0.4	96.14% ± 1.45%	70.54% ± 1.46%	68.12 s ± 0.91 s
1	0.4	92.67% ± 4.41%	100.00% ± 0.00%	0.03 s ± 0.01 s
1	0.1	91.86% ± 6.38%	100.00% ± 0.00%	0.02 s ± 0.01 s
1000	**0.1**	**96.29% ± 1.63%**	**69.17% ± 1.54%**	**17.62 s ± 0.16 s**
1000	0.7	96.00% ± 1.21%	71.67% ± 2.46%	118.28 s ± 2.35 s
100	1	96.14% ± 1.56%	80.40% ± 5.02%	16.08 s ± 0.38 s
1	0.7	92.14% ± 5.98%	100.00% ± 0.00%	0.03 s ± 0.01 s
1000	1	95.81% ± 1.24%	73.61% ± 1.94%	168.21 s ± 3.33 s
100	0.7	96.00% ± 1.45%	80.10% ± 3.21%	11.29 s ± 0.22 s
100	0.4	96.19% ± 1.52%	80.20% ± 2.60%	6.51 s ± 0.10 s
1	1	94.76% ± 3.47%	100.00% ± 0.00%	0.03 s ± 0.00 s
100	0.1	96.14% ± 1.35%	74.50% ± 3.50%	1.73 s ± 0.05 s

bcw (média de resultados para hipóteses fortes para $T \times \rho$)

Acurácia (%)

Hip. fracas válidas (%)

Tempo de execução (s)

Tabela 5.6: (continuado)

bcw (média de resultados para hipóteses fracas)

T	ρ	Alpha	Taxa de erro (%)	Vetores de suporte (%)	Norma
1	0.4	1.3973 ± 0.3459	0.16% ± 0.11%	7.71% ± 2.86%	211.5841 ± 34.3790
1	0.7	1.3892 ± 0.3426	0.17% ± 0.12%	10.00% ± 3.99%	276.8681 ± 123.6461
100	0.1	0.1214 ± 0.0182	0.92% ± 0.05%	11.65% ± 0.44%	447.1555 ± 56.9464
1	1	1.5918 ± 0.2485	0.10% ± 0.06%	13.24% ± 4.63%	340.5600 ± 95.3159
100	0.4	0.1355 ± 0.0149	0.86% ± 0.08%	27.08% ± 2.33%	900.5703 ± 129.9236
100	0.7	0.1459 ± 0.0226	0.86% ± 0.08%	34.93% ± 4.44%	1395.4959 ± 207.6325
1000	**0.1**	**0.0599 ± 0.0086**	**0.97% ± 0.03%**	**11.77% ± 0.58%**	**489.3496 ± 42.2209**
100	1	0.1546 ± 0.0249	0.83% ± 0.11%	39.98% ± 5.36%	2002.2801 ± 406.5583
1000	0.4	0.0646 ± 0.0079	0.96% ± 0.05%	23.46% ± 2.81%	825.2713 ± 116.4942
1000	0.7	0.0732 ± 0.0151	0.94% ± 0.06%	27.19% ± 3.75%	1253.2555 ± 238.3426
1000	1	0.0808 ± 0.0158	0.91% ± 0.08%	29.69% ± 4.23%	1721.3789 ± 362.4685
1	0.1	1.3876 ± 0.3919	0.18% ± 0.15%	5.05% ± 1.82%	130.3803 ± 49.4667

bcw (média de resultados para hipóteses fracas para $T \times \rho$)

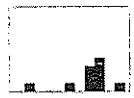
Alpha

Taxa de erro (%)

Vetores de suporte (%)

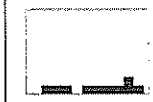
Norma

bcw (histogramas de execuções para configuração com melhor desempenho)



Acurácia (%): 96.29% ± 1.63%

Hip. fracas válidas (%): 0.10% ± 0.00%



Tempo de execução (s): 17.62 s ± 0.16 s

5.4 Sumário de Resultados

Esta seção traz tabelas com o sumário dos resultados de desempenho dos algoritmos propostos e da versão padrão do SMO para todos as 22 bases de dados selecionadas. A Tabela 5.7 compara resultados de desempenho em termos de acurácia, enquanto

a Tabela 5.8 compara tempo de execução.

Note que algumas células das Tabelas 5.7 e 5.8 foram deixadas em branco, uma vez que nem todas as bases de dados foram testadas com todos algoritmos por restrições de recursos computacionais.

Tabela 5.7: Sumário de resultados de acurácia para algoritmos híbridos.

Base de dados	SMO	SMO-B_α	SMO-B_β	SMO-B_γ	SMO-B_δ
bcw	96.10% ± 0.97%	87.71% ± 12.38%	97.62% ± 0.00%	96.52% ± 1.17%	96.29% ± 1.63%
cdges	80.48% ± 3.33%	70.24% ± 0.00%	77.38% ± 0.00%	77.38% ± 0.00%	83.33% ± 0.00%
$chess^0$	93.77% ± 1.38%	85.63% ± 8.01%	-	93.90% ± 1.49%	93.20% ± 1.68%
$chess^1$	85.37% ± 1.41%	78.60% ± 4.76%	-	84.57% ± 1.65%	82.40% ± 1.54%
$chess^2$	68.37% ± 1.26%	64.33% ± 3.91%	-	66.53% ± 2.17%	66.63% ± 2.40%
$gauss^0$	100.00% ± 0.00%	100.00% ± 0.00%	100.00% ± 0.00%	100.00% ± 0.00%	100.00% ± 0.00%
$gauss^1$	98.57% ± 0.68%	-	-	98.67% ± 0.00%	99.33% ± 0.00%
$gauss^2$	92.07% ± 1.05%	-	-	93.00% ± 0.00%	93.00% ± 0.00%
hepatitis	80.64% ± 4.61%	-	-	78.30% ± 6.51%	75.96% ± 6.32%
ionosphere	95.09% ± 1.83%	-	-	95.00% ± 0.95%	94.72% ± 2.74%
musk	93.01% ± 2.32%	-	-	93.01% ± 0.00%	93.71% ± 0.00%
pgs	81.25% ± 8.95%	-	87.50% ± 0.00%	88.12% ± 4.80%	82.50% ± 5.96%
pid	76.80% ± 2.51%	-	75.32% ± 0.00%	75.71% ± 2.02%	76.19% ± 2.28%
ringnorm	98.47% ± 0.70%	-	-	98.13% ± 0.92%	97.90% ± 0.56%
$spect^b$	76.36% ± 0.21%	-	-	77.81% ± 0.36%	78.50% ± 3.31%
$spect^r$	82.53% ± 0.00%	-	-	84.31% ± 2.88%	82.86% ± 1.65%
$spiral^0$	57.07% ± 2.63%	-	-	61.00% ± 0.00%	62.67% ± 0.00%
$spiral^1$	52.63% ± 1.39%	-	-	56.67% ± 0.00%	54.67% ± 0.00%
$spiral^2$	53.17% ± 1.21%	-	-	52.67% ± 0.00%	51.00% ± 0.00%
twonorm	97.97% ± 0.67%	-	-	97.50% ± 0.70%	96.67% ± 0.80%
wdbc	95.38% ± 1.37%	-	-	93.80% ± 2.01%	94.04% ± 1.47%
wpbc	79.83% ± 5.65%	-	-	75.67% ± 7.54%	77.67% ± 5.39%

Tabela 5.8: Sumário de resultados de tempo de execução para algoritmos híbridos.

Base de dados	SMO	SMO-B_α	SMO-B_β	SMO-B_γ	SMO-B_δ
bcw	5.46 s ± 2.40 s	0.03 s ± 0.03 s	0.04 s ± 0.00 s	45.96 s ± 4.66 s	17.62 s ± 0.16 s
cdges	158.45 s ± 21.21 s	28.82 s ± 0.00 s	97.81 s ± 0.00 s	13.37 s ± 0.00 s	282.34 s ± 0.00 s
chess^0	79.22 s ± 3.54 s	25.25 s ± 10.94 s	-	79.69 s ± 0.46 s	355.78 s ± 3.48 s
chess^1	79.74 s ± 17.01 s	9.84 s ± 9.01 s	-	751.56 s ± 17.27 s	57.81 s ± 1.61 s
chess^2	62.79 s ± 3.19 s	14.96 s ± 9.49 s	-	77.71 s ± 0.80 s	35.91 s ± 0.16 s
gauss^0	0.04 s ± 0.01 s	0.06 s ± 0.00 s	0.02 s ± 0.00 s	0.02 s ± 0.00 s	0.02 s ± 0.00 s
gauss^1	0.43 s ± 0.13 s	-	-	44.60 s ± 0.00 s	10.69 s ± 0.00 s
gauss^2	0.62 s ± 0.11 s	-	-	42.11 s ± 0.00 s	11.22 s ± 0.00 s
hepatitis	0.01 s ± 0.00 s	-	-	1.03 s ± 0.04 s	3.31 s ± 0.03 s
ionosphere	1.00 s ± 0.43 s	-	-	2.71 s ± 0.05 s	2.21 s ± 0.02 s
musk	8.87 s ± 3.86 s	-	-	17.95 s ± 0.00 s	165.83 s ± 0.00 s
pgs	0.19 s ± 0.08 s	-	5.11 s ± 0.00 s	10.81 s ± 0.13 s	3.61 s ± 0.03 s
pid	8.28 s ± 1.56 s	-	88.99 s ± 0.00 s	9.73 s ± 0.13 s	117.75 s ± 0.77 s
ringnorm	21.23 s ± 5.62 s	-	-	328.96 s ± 6.99 s	283.86 s ± 3.70 s
spect^b	0.18 s ± 0.03 s	-	-	3.49 s ± 0.04 s	0.07 s ± 0.04 s
spect^r	0.27 s ± 0.02 s	-	-	0.05 s ± 0.00 s	0.17 s ± 0.03 s
spiral^0	38.72 s ± 27.51 s	-	-	127.82 s ± 0.00 s	353.34 s ± 0.00 s
spiral^1	5.96 s ± 0.85 s	-	-	13.11 s ± 0.00 s	64.38 s ± 0.00 s
spiral^2	171.59 s ± 2.66 s	-	-	49156.51 s ± 0.00 s	1.24 s ± 0.00 s
twonorm	0.72 s ± 0.03 s	-	-	9.20 s ± 0.36 s	46.22 s ± 0.16 s
wdbc	2.98 s ± 0.58 s	-	-	34.99 s ± 1.41 s	80.04 s ± 0.97 s
wpbc	4.84 s ± 3.30 s	-	-	0.21 s ± 0.01 s	5.01 s ± 0.04 s

Capítulo 6

Conclusão

Tendo em vista os resultados discutidos no Capítulo 5, a conclusão mais fundamental que este trabalho possibilita é de que é possível combinar técnicas de Boosting e SVMs em algoritmos híbridos que frequentemente apresentam desempenho superior a outras máquinas de aprendizado sofisticadas, tais como as próprias formulações tradicionais de SVMs. Esta conclusão contradiz uma importante ressalva da literatura de Boosting. Muitos autores consideram, dentre eles Schapire [Sch02], que algoritmos muito complexos quando utilizados como geradores de hipóteses fracas, não apresentam bons resultados na hipótese forte computada por um algoritmo de Boosting. Esta restrição apresentou um risco considerável para esta pesquisa, uma vez que SVMs são das máquinas de aprendizado mais sofisticadas conhecidas na literatura [CST00]. De acordo com os resultados do Capítulo 5, o algoritmo inicial proposto, **SMO-B$_\alpha$**, falhou em produzir resultados melhores que a versão original do SMO, como já havia sido previsto por outros autores. No entanto, contradizendo Schapire e outros, a partir da análise de sua execução foi possível localizar as fontes do problema e eliminá-las na versão subsequente, **SMO-B$_\beta$**.

Outro ponto vem da teoria de SVMs. Uma vez que são formuladas como problemas de otimização, o hiperplano de separação alcançado por SVMs é considerado ótimo uma vez que seu algoritmo de treinamento converge. No entanto, foi observado que os algoritmos híbridos propostos apresentaram desempenho ainda melhor que a versão original do SMO, o que indica que a formulação de SVMs está sujeita a erros provenientes da falta de exemplos de treinamento suficientes ou ainda do desajuste de parâmetros da máquina e do kernel utilizados. Os resultados obtidos indicam que em sua junção com Boosting nos algoritmos híbridos, esta dificuldade de alcançar bons resultados com parâmetros desajustados pode ter sido parcialmente superada.

Esta questão nos leva a outra conclusão. Independentemente se acoplado

ou não com Boosting em algoritmos híbridos, um dos maiores problemas práticos de utilizar SVMs e métodos baseados em kernels em problemas de aprendizado está na dificuldade de ajustar seus parâmetros. Nos experimentos realizados foram examinados três casos onde a dificuldade de ajuste de parâmetros impediu que quaisquer dos algoritmos testados apresentassem desempenho satisfatório, especificamente nas bases de dados **spiral**^{0}, **spiral**^{1}e **spiral**^{2}. Em um experimento isolado, os parâmetros de uma SVM foram ajustados com muito mais rigor que nos experimentos originais, aproximadamente com precisão 6 ordens de magnitude maior. Como resultado, o SMO conseguiu um desempenho de acurácia próximo a 100%, realçando a suceptibilidade de SVMs ao ajuste de parâmetros. A conclusão, portanto, é que apesar desta abordagem híbrida conseguir compensar pequenos desvios nos parâmetros de SVMs, este ajuste continua sendo um de seus maiores problemas, acopladas a Boosting ou não.

A conclusão final deste trabalho, e talvez a mais importante, está relacionada aos promissores resultados obtidos nos experimentos. Fica claro que apesar do processo de desenvolvimento do trabalho ter abrangido uma grande quantidade de opções, algoritmos e bases de dados, uma abordagem mais vertical pode ser utilizada agora para explorar em mais detalhe cada um dos pontos brevemente ressaltados aqui. A conclusão é, portanto, que uma exploração mais a fundo de estratégias para combinação de Boosting e SVMs tem um grande potencial para produzir máquinas de aprendizado ainda mais eficientes, tanto em termos de acurácia quanto em termos de tempo de execução.

Bibliografia

[BGL+99] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, Jr. M. Ares, and D. Haussler. Support vector machine classification of microarray gene expression data. Technical report, University of California, Santa Cruz, 1999.

[BLC00] A. P. Braga, T. B. Ludermir, and A. F. Carvalho. *Redes Neurais Artificiais: Teoria e Aplicações*. LTC, 2000.

[BM98] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

[Bre94] Leo Breiman. Bagging predictors. Technical report, University of California at Berkeley, 1994.

[Bre96] Leo Breiman. Bias, variance and arcing classifiers. Technical report, University of California at Berkeley, April 1996.

[Cac94] Christian Cachin. Pedagogical pattern selection strategies. *Neural Networks*, 7(1):175–181, 1994.

[CDH+00] J. Cai, A. Dayanik, N. Hasan, T. Terauchi, and H. Yu. Supervised machine learning algorithms for classification of cancer tissue types using microarray gene expression data. Technical report, Columbia University, 2000.

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[DSS93] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:705 – 719, 1993.

[Fre95] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 2(121):256–285, September 1995.

[FS95] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*. LNCS, March 1995.

[FS96a] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[FS96b] Y. Freund and R. E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the 9th Annual Conference on Computer Learning Theory*, pages 325–332. ACM Press, New York, NY, 1996.

[FS97] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[FS99] Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, September 1999. Appearing in Japanese, translation by Naoki Abe.

[Hay94] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company, Inc., 866 Third Avenue, New York, New York 10022, 1994.

[Heb49] D. O. Hebb. *The Organization of Behavior*. Wiley, 1949.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective properties. In *Proceedings of the National Academy of Sciences*, 79, pages 2554–2558, 1982.

[KK01] Michihiro Kuramochi and George Karypis. Gene classification using expression profiles: A feasibility study. In *IEEE International Conference on Bioinformatics and Biomedical Egineering*, pages 191–200, 2001.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP69] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[Mun92] P. W. Munro. Repeat until bored: A pattern selection strategy. *Advances in neural information processing systems*, 4:1001–1008, 1992.

[OFG97] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII – Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, 1997.

[Pla98a] John C. Platt. *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. MIT Press, 1998.

[Pla98b] John C. Platt. Sequential minimal optimization: a fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65:386–408, 1958.

[RTR+01] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C. H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, , and T. R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *PNAS*, 98(26):15149–15154, 2001.

[Sch90] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[Sch92] R. E. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. MIT Press, 1992.

[Sch99] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[Sch02] R. E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[SS99] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, December 1999.

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[VC71] V. N. Vapnik and A. J. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[WH60] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *Institute of Radio Engineers, Western Electronic Show and Convention*, 1960.

Strategies for Combining Boosting and Support Vector Machines

by

Thiago Turchetti Maia

B.Sc., Universidade Federal de Minas Gerais, Brazil, 1999.

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Electrical Engineering

Universidade Federal de Minas Gerais

February 2003

Abstract

Support Vector Machines are known in literature to be one of the most efficient learning models for tackling classification problems, mainly due to their strong theoretical foundation and their training based on solving a convex constrained quadratic optimization problem. Among the many algorithms for training SVMs by solving this problem, the Sequential Minimal Optimization (SMO) algorithm is of outstanding performance thanks to its approach that breaks this problem up into many smaller optimization problems, a technique otherwise known as chunking, and uses an analytical method to solve each small chunk, as opposed to traditional numerical algorithms.

Meanwhile, Boosting techniques have received a great deal of attention lately from the machine learning community. Many Boosting algorithms work by using an ordinary classifier algorithm to produce different weak hypotheses for the learning problem, which are later combined into a single strong hypothesis through majority voting. The most well known Boosting algorithms in literature are the AdaBoost family members, which greatly increased Boosting's appeal by solving many of the practical problems of earlier algorithms.

In this work we combine Boosting with Support Vector Machines, namely the Adaboost.M1 and SMO algorithms, to create new hybrid algorithms that outperform the standard SMO version in selected contexts. We achieve this integration with different degrees of coupling, where the four algorithms proposed start from the most straight forward black-box integration strategy to deep modifications and mergers between AdaBoost and SMO components.

Although our tests show that our proposed algorithms exhibited better performance for most problems experimented with, there were some particular ones where SMO still performed better. Nevertheless, it is possible to identify trends of behavior bound to specific properties of the problem being solved, where one may hence apply the proposed algorithms in cases where it is known to succeed. Finally, besides introducing this new class of algorithms and paving the way for the further development of algorithms, we also show that part of the difficulties of using SVMs

in practice, namely the tuning of appropriate machine and kernel parameters, may be partially compensated by the hybrid approach, therefore increasing the applicability of the hybrid algorithms.

Contents

List of Tables

List of Figures

I dedicate this work to my family, for their endless unconditional support that eased all my striving moments along the road.

Acknowledgments

I would like to thank all my family and friends who, at all times, gave me all the support I needed to overcome the many obstacles along this long road.

I would like to thank Vetta Technologies and its crew for providing me with a resourceful infrastructure where I executed all experiments in this work.

I would like to thank my supervisor, Prof. Dr. Antônio de Pádua Braga, from whom I have received full support to pursue this endeavoring path and encouragement to pursue new ones yet to come.

Thiago Turchetti Maia
Belo Horizonte, February 2003

Chapter 1

Introduction

1.1 The Learning Problem

The problem of teaching a machine how to acquire knowledge has been explored ever since modern computer architectures came to be. The first model of an artificial neuron is due to McCulloch and Pitts back in 1943 [MP43], in their classical work of describing what was to be the basic unit of a network of neurons that would resemble the human brain. The focus of their work was much less in the methods for inducing knowledge into such networks, but instead in their computing abilities [BLC00]. Later in 1949, the learning of these networks of neurons was explored by Hebb [Heb49], which proposed a model based on the adjustment of weights in neuron's inputs. This model stood on the theory that biological neurons learn based on the reinforcement of synaptic links to other excited neurons. Widrow and Hoff [WH60] further refined Hebb's ideas in 1960, incorporating the method of gradient descent to minimize the output error of each neuron. The first attempt to tackle pattern recognition problems with neural networks is due to Rosenblatt in 1958 [Ros58] in which he introduced the *Perceptron* model. The Perceptron is a simple neural network able to solve linearly separable classification problems in which the output space is divided into complementary regions that correspond to each category. A very brief description of the Perceptron is given in section 5.2.1, where we use its restriction of only solving linearly separable problems to our advantage.

Though this connectionist approach [1] received great attention after the work of Rosenblatt, the machine learning community was discouraged to pursue it any further by Minsky and Papert in 1969 [MP69]. Minsky and Papert argued that the

[1] Other names that have been used to name this field include *connectionism, parallel distributed processing, neural computation, adaptive networks,* and *collective computation* [RN95].

limitation of not being able to solve non-linearly separable problems was too great to be ignored, along with other restrictions such as explosive growth in space and time, and the lack of a learning rule to multi-layered networks. The 1970's were hence marked by the lack of interest from the community in this area, except for a few scattered researchers. It was not until the works of Hopfield in 1982 [Hop82] and the development of the *back-propagation* algorithm in 1986 [RHW86] that artificial neural networks regained general interest. Hopfield explored the relation between recurrent associative networks and physical models, which paved the way for using Physics theories to describe the behavior of learning machines [Hop82]. Finally, much attention was drawn to the work of Rumelhart et al. [RHW86], where they managed to create an algorithm for training multi-layered neural networks, *back-propagation*, based on the adjusting of weights in each layer according to the retro-propagation of their output errors.

This rebirth from ashes of the connectionist school in the 1980's was followed by a new wave of research starting in the 1990's that explored new theories and algorithms focused not only is constructing and training powerful learning machines, but also in reorganizing the environment around them in order to improve their performance [2]. Vapnik [Vap95] refers to this period as the return to the origins of statistical learning theory. Pioneer works such as those from Munro [Mun92], Schapire [Sch92], Drucker et al. [DSS93], and Cachin [Cac94] served as basis for others like Breiman [Bre94], who introduced the Bagging method for training and combining multiple copies of a learning algorithm, and also Freund and Schapire, which later introduced the famous AdaBoost family of algorithms. The term *Boosting* itself is due to Schapire in 1990 [Sch90], in the famous paper where he first showed that any polynomial-time learning algorithm that generated hypotheses whose error was marginally better than random could be transformed into a polynomial-time learning algorithm with arbitrarily small error.

In parallel with the development of learning models and methodologies for improving the performance of previous algorithms, this return to the origins also motivated the creation of several new learning machines. Among these machines, we highlight those of Vapnik and co-workers which became known as Support Vector Machines. Relying on the previous 1965 work from Vapnik and Chervonenkis [VC71], who developed the method of the optimal separating hyperplane, Vapnik et al. created the idea of Support Vector Machines in 1992 [Vap95]. Since then, the

[2]Notice that the term *performance* has been used throughout the text without specifying either *time performance*, *space performance*, or any other reference of performance measure. Whenever it is used as such a general concept, we mean the *overall performance* consisting of the combination of all relevant measures that may be applied to a given machine or algorithm.

growing interest of the machine learning community motivated the development of several specialized training algorithms for solving the convex constrained quadratic optimization problem with which SVMs are formulated. New techniques such as chunking and decomposition inspired the work of John Platt [Pla98a, Pla98b], whose groundbreaking Sequential Minimal Optimization (SMO) algorithm allowed for a great popularization of SVMs due to its simplicity of implementation and drastic reduction of required computational resources.

1.2 Aims and Motivations

Out of the many learning problems tackled by different machine learning algorithms, in this work we chose to work with binary classification problems. The reason for this choice is twofold. First, binary classification problems are about the most fundamental problems in machine learning, where more sophisticated learning problems are often presented as simple extensions of the binary classification case. Many learning models and algorithms, originally designed to deal with binary classification, have been adapted to extend their applicability to multi-class classification, multi-label classification, and even regression problems. Although we limit ourselves to investigating only binary classification databases, we already foresee the extension of this work to other problem domains, as outlined in Chapter 7. Second, binary classification problems are the natural habitat of the two learning machines combined in this work, namely Boosting and Support Vector Machines, although many extensions have been proposed for both. Therefore, we are able to validate the idea of combining them together while still using their most straight forward original versions.

We now formalize the definition of binary classification learning problem, assumed throughout the text. Consider the classification problem having an n-dimensional input domain and a binary output domain, both containing l instances such that we have a dataset $(x_1, y_1), \ldots, (x_l, y_l)$, where each pattern x_i belongs to domain $X = \mathbb{R}^n$, each label y_i belongs to domain $Y = \{-1, +1\}$, and each pattern x_i is classified by its corresponding label y_i. In this work we selected 22 databases with which we experimented each of the algorithms proposed as well as the standard version of SMO. Out of the 22 datasets studied, 8 corresponded to real-world biology problems, 2 to real-world genomics problems, one to a real-world physics problem, 9 were synthetically generated with bi-dimensional inputs, and 2 were synthetically generated with 20-dimensional inputs. A thorough description of these datasets and their corresponding problems is given in Chapter 5.

Out of the many machine learning approaches possible of being used in bi-

nary classification problems, we are certainly interested in those which gives us the most accurate models in the shortest execution time. One of the most outstanding machine learning algorithms recently proposed is SMO, by John Platt [Pla98a]. While relying on the great expression power given by Support Vector Machines, Platt managed to take advantage of several particular properties of the convex constrained quadratic optimization problems, responsible for their training, in creating an efficient algorithm which is generally be both faster and more accurate than many other learning algorithms.

It would be extremely naive if we attempted to propose a new algorithm that would generally both faster and more accurate than SMO. Nonetheless, as Platt took advantage of properties of the optimization problem at hand, we may extend his approach by deriving a new algorithm that explores specific properties of the learning problems at hand, eventually even outperforming SMO in both accuracy and execution time. To achieve this goal, we use SMO as the basis of our new algorithms, along with a Boosting algorithm which is expected to use its strong theoretical background to enhance SMO's performance.

According to Freund [Fre95], a Boosting algorithm is a learning algorithm that uses as a subroutine a different learning algorithm. By repeatedly running a given weak learning algorithm on a maintained distribution of the training data, Boosting algorithms generate a so-called *strong* hypothesis, which is a single classifier achieved by combining the weak hypotheses through majority voting. The first effective Boosting algorithms were presented by Schapire [Sch90] and Freund [Fre95]. More recently, Freund and Schapire introduced **AdaBoost**, which is a generic family of Boosting algorithms [FS95, FS96a] that solved many of the practical difficulties of earlier algorithms.

Therefore, the most important goal of this work is to create a new class of algorithms that rely on the strongest features provided by both Boosting and Support Vector Machines, taking advantage of specific properties of learning problems when attempting to outperform Platt's efficient Sequential Minimal Optimization algorithm.

1.3 Contributions of This Work

In this work we combine Boosting techniques together with Support Vector Machines, deriving new robust hybrid learning algorithms that consistently outperform standard SVMs in particular classification problems, both in accuracy and time. Even though SVMs are constructed as to yield an optimal separating hyperplane for binary classification tasks, we show that various factors such as data biasing, im-

proper kernel functions, and unfit regularization and kernel parameters are capable of downgrading the performance of standard SVMs, which may hence be boosted using Boosting algorithms. We put together two of the most known Boosting and SVM training algorithms, namely Freund and Schapire's [FS95] AdaBoost.M1, and Platt's [Pla98a] Sequential Minimal Optimization (SMO). We approach the integration of AdaBoost.M1 and SMO at different levels, starting from using SVMs as black-box weak learners, to combining the heuristics inherent to SMO with the probabilistic selection mechanisms found in AdaBoost.M1, therefore proposing four hybrid algorithms with different degrees of coupling and modifications added to AdaBoost.M1 and SMO.

One of the most important contribution of this work is to prove that the integration of Support Vector Machines with Boosting yields algorithms that, for a set of problems with specific properties, are more efficient than other major learning algorithms, such as SMO, both in terms of accuracy and execution time. It is known in the machine learning literature that that overly complex classifying algorithms used as weak learners may fail to produce good strong hypotheses through a Boosting algorithm. Since SVMs are one of the most sophisticated learning machines currently known, contradicting the fact that they are bound to this restriction is of great relevancy to the future development of better learning machines that incorporate Boosting and Support Vector principles.

Finally, aside from setting the first steps toward more efficient hybrid implementations, another important contribution of this work is the actual construction of new learning algorithms that already perform better than SMO for selected learning problems. As we discuss results and conclusions in Chapters 5 and 6, we identify several common properties in the datasets over which the algorithms proposed obtained better results, thus segmenting the applicability of these algorithms to contexts where they are already known to have excelled. Also, one of the main problems of using Support Vector Machines is the tuning of machine and kernel parameters involved in the training process. Despite the simplicity of our tuning procedures and since good performance standards were still able to be sustained, we found that this hybrid approach may compensate for slightly unfit SVM and kernel parameters.

1.4 Dissertation Outline

This dissertation is organized as follows. In Chapter 2 we review Boosting, the concepts behind it, and its application as a general tool for achieving better learning machines. We start with the description of early works that originated the concept of Boosting, including the classic ones from Munro [Mun92] and Cachin [Cac94].

Later, we introduce some of the principles of Valiant's Probably Approximately Correct (PAC) model of learning [Val84], together with important concepts such as weak learnability. Finally, we thoroughly review the AdaBoost family of algorithms by Freund and Schapire [FS95, Sch02], including their applicability, formulation, and bound analyses.

Chapter 3 introduces the concepts and propositions of Support Vector Machines designed for solving classification problems. We examine concepts of generalization theory for machine learning, such as empirical risk minimization, Vapnik-Chervonenkis theory [Vap82], structural risk minimization [Bur98], inner-product kernel functions and learning in kernel-induced feature spaces. Finally, we examine formulations of different machines, evolving from simple linear SVMs into non-linear and non-linear soft-margin SVMs, as well as different approaches used to train them, including the Sequential Minimal Optimization (SMO) algorithm by John Platt [Pla98a].

Using all the groundwork laid down in Chapters 2 and 3, Chapter 4 proposes the creation of hybrid algorithms that combine Support Vector Machines together with Boosting techniques in order to create better learning machines that benefit from desirable properties of both. We start the chapter by describing the motivations that inspired such hybrid algorithms, followed by the description of the new hybrid algorithms proposed.

In Chapter 5 we describe and discuss all experiments carried out and results obtained. We first describe each database used in detail, followed by the results of preliminary analyses performed over all databases using linear discriminants and the standard SMO algorithm. Finally, we describe results for the hybrid algorithms proposed in Chapter 4, along with their in-depth discussion.

Chapter 6 summarizes the results obtained from this work, and highlights the many conclusions we reached from them. Chapter 7 looks into the future and suggests new paths toward the continuity of this work. Each of the ideas discussed has the potential to not only increase the applicability of the algorithms developed up to now, but also to lead to new algorithms with better generalization and faster execution times.

For the sake of good aesthetics, Appendix A brings the complete results for all experiments and analyses commented throughout the text. Although smalls subsets of these results are repeated in Chapter 5, we chose to banish these complete result descriptions to the Appendices, so as not to clutter the main body of text. For reference on all time measures contained in the results, Appendix B brings remarks about program instrumentation and execution environment.

Chapter 2

Boosting

This chapter brings a review of Boosting, the concepts behind it, and its application as a general tool for achieving better learning machines. Many of the ideas introduced here will be revisited in Chapter 4, where we combine them with Support Vector Machines to create new hybrid learning algorithms. We start with the description of early works that originated the concept of Boosting in Section 2.1, including the classic works from Munro [Mun92] and Cachin [Cac94]. In Section 2.2, we introduce some of the principles of Valiant's Probably Approximately Correct (PAC) model of learning [Val84], together with important concepts such as weak learnability. The PAC model is later revisited in Chapter 3, where it is used to back up yet another fundamental theory of machine learning due to Vapnik and Chervonenkis [VC71]. Section 2.3 brings a thorough description of the AdaBoost family of algorithms by Freund and Schapire [FS95, Sch02], including their applicability, formulation, and bound analyses. Section 2.4 concludes the chapter describing some of latest trends in Boosting, including an interesting parallel with Support Vector Machines inspired by Rätsch et al. [Rät01].

2.1 Pedagogical Strategies for Learning Machines

In 1992, Munro introduced the term *pedagogical* to refer to a strategy applied in improving the performance of learning machines [Mun92]. Later in 1994, Cachin adopted the same names and analogies between human Pedagogy and this concept of studying and enhancing the mechanisms of learning machines and algorithms [Cac94]. These two authors and their respective papers pioneered the study of optimizing the learning of known machines, which, in those cases, were multi-layer Perceptron neural networks being trained with the back-propagation algorithm. The

focus of their work was the customized selection of input patterns[1] to be presented to training algorithms. For most training algorithms, patterns in the training set are usually presented equally, sometimes according to a predefined scheme, most of the time ignoring the current status of the machine and the amount of learning effectively undertaken. These selection strategies work by attempting to determine an order in which to present patterns to the learner, therefore trying not only to save computational time often wasted on already-learned patterns, but also trying to avoid biasing subsets of patterns which in turn may cause the overfitting of the training set.

There are two types of strategies to use when selecting training patterns, one deterministic and another stochastic. Munro [Mun92] describes a deterministic strategy where a pattern is repeated until its training error falls below a certain threshold value. We shall give more attention to the stochastic strategies described by Cachin [Cac94], once they are the ancestors of today's modern Boosting techniques. Stochastic strategies are implemented by drawing a pattern from a training set using a distribution value which is maintained by the training algorithm in order to reflect the current status of the learning machine. They often start with a uniform distribution, where the probability of drawing any of the patterns from the set is the same. As training occurs, the common property between all strategies is to increase the probability value of patterns with higher output error, thus favoring patterns which are harder to be learned. Specifically in Cachin's research, this procedure takes place at every training step, which is consistent with the use of on-line back-propagation where weights are updated after each pattern presentation [Cac94]. Notice how important it is to still consider all patterns in the training set during probabilistic selection, even if some of them have much lower probability of being selected than others. Since back-propagation minimizes the error function of the network locally [Hay94], individual patterns may be forgotten if not re-presented from time to time during learning, which hence would lead to underfitting the training set.

The following sections shorty describe Cachin's selection strategies explored in his paper [Cac94]. Some of them have been used in recent works with advanced learning machines, whereas others, though more naive, still are perfectly good examples of the ideas behind boosting [2] standard learning algorithms. Furthermore, we shall see later that many advanced Boosting algorithms use recurrent error func-

[1] In order to keep naming consistency with Cachin's paper, we stick to the term *pattern* to denote what is otherwise known as an input vector, an individual instance or sample of a data-set.

[2] The term *boosting* is used throughout the text in its lowercase form to denote the gerund of the verb to boost, as opposed to the noun *Boosting* used so far.

tions not unlike the first two strategies just described. For a complete description of all strategies along with their benchmark on different learning tasks, please refer to [Cac94]. Amazingly enough, Cachin shows how some of these strategies frequently outperform the standard deterministic selection mechanism from back-propagation, even though a theoretical background is not available to back them up.

2.1.1 Error-Dependent Presentation Probability

One of the simplest forms of maintaining the probability value p_k of selecting a given pattern k is to make it proportional to its output error Δ_k. This ensures that patterns with greater error are more often selected, and vice-versa. Different functions may be selected to implement this proportionality, for instance $p_k \propto \Delta_k$, $p_k \propto (\Delta_k)^2$, $p_k \propto (\Delta_k)^4$, $p_k \propto e^{\Delta_k}$, or $p_k \propto 10^{\Delta_k}$.

Notice that the values p_k for all k in the training set must be normalized after each update to ensure that they represent a proper probability distribution, that is, each p_k must be divided by the sum of all $p_k \forall k$.

2.1.2 Error-Dependent Repetition

This strategy is similar to Error-Dependent Presentation Probability, except that each pattern is repeatedly presented according to a deterministic schedule. All patterns are initially selected, with each Δ_k being saved as well as its maximum value Δ_{max}. Each pattern k is then repeated if $p_k > \frac{i\Delta_{max}}{W}$, where i iterates in $0 \leq i \leq W$. This procedure is repeated until the network error falls below an arbitrary threshold. As before, different functions for Δ_k may be used for updating p_k.

This approach is very similar to Error-Dependent Presentation Probability, except for two main differences. First, it is guaranteed to present all patterns at least once to the learner, hence avoiding underfitting. Second, due to its deterministic nature, higher-error patterns will be orderly presented to the learner with no interference of lower-error ones.

2.1.3 Card-File System

This strategy consists of keeping a predetermined order to present patterns to the learner, represented by the stack of indices of all P patterns, $\{i_1, \ldots, i_P\}$. Whenever a pattern k is selected from the top of the stack $\{i_k, \ldots, i_P\}$, it is presented to the learner and repositioned in the queue on a position proportional to its output error. This simple mechanism ensures that harder patterns are more often revisited by the learning algorithm than patterns already mastered.

Notice that this strategy has two flavors. First, one may deterministically pop the top of the stack to select the next pattern to be presented. Second, one may use a probability value proportional to patterns' positions on the stack to draw the selection.

2.1.4 Training Set Partition

For a training set with P patterns, we divide this set into N partitions $\{S_1, \ldots, S_N\}$. We start training by feeding the learner with S_1, where all patterns in the subset are uniformly drawn from it. We then add each new partition S_n incrementally to the set of possible selection choices $S_1 \cup S_2 \cup \ldots \cup S_N$, with the option of presenting only S_n for some time before reconsidering the union of all previously-selected partitions.

2.1.5 Repeat Until Learned

This is the strategy originally used by Munro [Mun92]. It consists of repeating a given pattern until its error falls below a threshold β. The value of β depends on the learning problem at hand, where it is suggested to be proportional to the mean squared error of all examples in the training set. Notice that even though other strategies previously described did not necessarily converge at all times, this particular one has the potential to be easily kept from converging when confronted with higher-error patterns.

2.2 The PAC Framework

In this section we give an overview of Valiant's Probably Approximately Correct (PAC) model, or *distribution free* learning model. The PAC framework serves not only as a basis for the Boosting algorithms described in this chapter, but also for the Vapnik-Chervonenkis theory presented in Chapter 3. Valiant [Val84] introduced this model back in 1984, where he defines the accuracy of a learning algorithm to be $1 - \epsilon$ and its reliability to be $1 - \delta$ if it generates a hypothesis whose accuracy is at least $1 - \epsilon$ with a probability of at least $1 - \delta$. Haussler et al. [HKLW91] showed that increasing the reliability of any learning algorithm is relatively easy through testing the hypothesis generated over an independent set of examples to validate its accuracy. It is straightforward to see that by running this algorithm at least $O\left(\log\left(\frac{1}{\delta_2}\right) / (1 - \delta_1)\right)$ times, its reliability will increase from $1 - \delta_1$ to $1 - \delta_2$ [Fre95].

Increasing accuracy, on the other hand, is much harder. Two variants of the PAC model were introduced by Kearns and Valiant [KV94] to address the problem.

The first, the *strong* PAC model, takes the required accuracy $1 - \epsilon$ as an input, and is required to generate hypotheses with error smaller than ϵ with polynomial resources in $1/\epsilon$. The second, the *weak* PAC model, is required to generate hypotheses with error ϵ only marginally smaller than $1/2$, that is, hypotheses marginally better than a random guess. Although Kearns and Valiant [KV94] proved that monotone boolean functions may be learned with the weak model, but not with the strong model, Schapire [Sch90] proved that weak and strong PAC learning are equivalent in the distribution free case [Fre95].

Schapire [Sch90] was responsible for presenting the first algorithm referred to as Boosting, which consisted of a method for generating hypotheses of arbitrary accuracy using time and space resources polynomial in $1/\epsilon$. The main idea behind it consisted of creating different *weak* hypotheses generated using different distributions of instances, and then combining them into a single and more accurate *strong* hypothesis using the principle of weighted majority voting, detailed in section 2.2.1. These ideas are the same behind the AdaBoost family of algorithms, described ahead in Section 2.3. In the remaining of this section we explore the historical and basic theories that led to the development of such advanced algorithms.

2.2.1 The Majority-Vote Game

In this section we briefly describe majority voting, one of the core concepts behind Boosting learning algorithms. We do so by first presenting the majority-vote game, thoroughly explored by Freund [Fre95], and later using it to enhance learning machines in Section 2.2.2. Weighted majority voting was introduced by the multiplicative weight-update rule of Littlestone and Warmuth [LW94]. Before them, many other relevant works leveraged the idea of using multiple expert predictions to tackle one same learning problem, such as the classic from Cesa-Bianchi et al. [CBFH$^+$93], the models proposed by Vovk [Vov90], Freund et al. [FSSW97], Kivinen and Warmuth [KW94], and those from Haussler, Kivinen and Warmuth [HKW95]. The mapping of the problem to game theory, as we describe it below and as used by Freund [Fre95], is due to Chung [Chu94], though it has also been more recently explored by Schapire [Sch01].

Let there be two players, a *weightor* G and a *chooser* C. Consider a space $\langle X, \Sigma, V \rangle$ over which the game is played, where X is the training set of input samples, Σ is a σ-algebra over X, and V is a probability measure function. Σ is defined as a nonempty collection of subsets of X such that the empty set is in Σ, for any $A \in \Sigma$ then $\overline{A} \in \Sigma$, and finally if A_n is a sequence of N elements of Σ then $\bigcup_{i=1}^{N} A_i \in \Sigma$ [Vap95]. The probability of any set $A \in \Sigma$ is given by $V(A)$. A real-valued parameter $0 < \gamma \leq 1/2$ is fixed prior to starting the game, which proceeds in

two-step iterations.

On the first step, player G selects a weight measure W to be used on X, which is a probability measure on $\langle X, \Sigma \rangle$ denoted by $W(A)$ where $A \in \Sigma$. On the second step, player C selects a set $U \in \Sigma$ such that $W(U) \geq 1/2 + \gamma$, and flags all instances $u \in U$.

These two steps are repeated until player G chooses to stop, when it is then rewarded with all instances $x \in X$ such that each x has been flagged in more than half of the iterations played. Let this subset of X be the reward set R such that $R \subset X$, and its complement $\overline{R}$ be the loss set. The value of the reward and loss sets are hence given by $V(R)$ and $V(\overline{R})$, respectively. The goal of the weightor G is to maximize $V(R)$, whereas the goal of the chooser C is to maximize $V(\overline{R})$.

The question about the game in which we are interested is whether there is a general strategy that guarantees a large reward for the weightor, regardless of the probability space used. Freund [Fre95] proves this strategy to exist so that for any probability space $\langle X, \Sigma, V \rangle$, any ϵ and any $\gamma > 0$, the weightor is guaranteed to receive a reward $V(R) > 1 - \epsilon$ after at most $\frac{1}{2}\left(\frac{1}{\gamma}\right)^2 \ln \frac{1}{\epsilon}$ iterations. The proof of the theoretical bounds for this strategy is presented by Freund [Fre95], and is left out of this text due to size constraints. The concept of majority voting is later used in Section 2.2.2 for the actual boosting of learning algorithms.

2.2.2 Weak Learnability Using Majority Voting

In this section we describe the connection between boosting a learning algorithm and the majority-vote game presented in Section 2.2.1. Parts of this description are also presented by Freund [Fre95]. We start by defining a minimal framework of distribution free concept learning. A *concept* is defined as a binary mapping from an input domain X to $\{-1, +1\}$, where for a concept c and an instance $x \in X$, $c(x)$ denotes its label in $\{-1, +1\}$. A *concept class* C is a collection of concepts.

The learner knows C, but its job is to learn an approximation of a concept $c \in C$. We assume that the learner has access to a source S, which returns instances $x \in X$ drawn randomly and independently according to a given distribution D, along with their corresponding label $c(x)$. The learner is given access to S, from which it draws different instances and creates a hypothesis h. A hypothesis is nothing but an algorithm that takes any arbitrary input $x' \in X$ and outputs its prediction of what $c(x')$ should be. We indicate the probability of h correctly mapping c as $P(h(x') = c(x'))$ over D, which we refer to as accuracy. Analogously, the error of h over D is $P(h(x') \neq c(x'))$. If the error is less than ϵ, we say h is ϵ-good for c and D.

A given learning algorithm A has a uniform sample complexity $m(\epsilon, \delta)$ if for all $\epsilon > 0$, $\delta < 1$, all D, and $c \in C$, given ϵ and δ as parameters, A makes at most $m(\epsilon, \delta)$ calls to S and outputs a hypothesis h with probability of at least $1-\delta$ of being ϵ-good for c and D. Suppose there exists real values $0 \leq \epsilon_0 < 1/2$ and $0 \leq \delta_0 < 1$ such that each hypothesis h generated from m_0 examples has error at most ϵ_0 with probability of at least $1 - \delta_0$. The upper bound in sample size for A to achieve this accuracy is m_0. The parameters ϵ_0 and δ_0 measure the difference between A and a perfect learning algorithm that always generates hypotheses with no error for a given concept. We then find it useful to define the parameters $\gamma = 1/2 - \epsilon_0$ and $\lambda = 1 - \delta_0$, which measure how far a learning algorithm is from complete uselessness, in this context where the accuracy of learning algorithms are required to be arbitrarily better than $1/2$.

We then use the principle of majority voting to combine hypotheses generated from A using different distributions, where the majority votes for input samples are expected to be more accurate than those of each individual hypothesis. We start by creating instances of the learning algorithm that will be boosted, which from now on we shall refer to as *weak hypotheses*. Let D be a initial uniform probability distribution with probability values $1/|X|$ assigned to each $x \in X$. In order to create a weak hypothesis, we first draw a subset U of size m from X by calling S m times using D. We then present U to A in order to create a new weak hypothesis h_t, where t is the index of the hypothesis. We expect h_t to have error smaller than $1/2 - \gamma$ for D. Finally, we give the learner a feedback from the Boosting algorithm by normalizing the distribution D after updating it with an increment on the probability values of all instances x for which $h_t(x)$. These steps are repeated T times, thus generating T weak hypothesis. Finally, we use majority voting to combine the outputs of these hypotheses when evaluating new input instances in X.

The problem of updating D is equivalent to the majority-vote game described in Section 2.2.1. The *value* of an instance corresponds to its probability value assigned by the target distribution, which is uniform in this case. The *weight* of an instance corresponds to its probability assigned by the Boosting algorithm. The decision to *flag* an instance corresponds to the learner generating a correct hypothesis for that point. The *reward set* corresponds to the set on which the majority vote was correct, and the *loss* corresponds to the probability of the majority vote being mistaken, measured with respect to the target distribution.

The theoretical proofs of bounds for this generalized way of boosting learning algorithms with majority voting, as described in this section, is given by Freund [Fre95]. Due to size constraints, we save these proofs for Section 2.3, where we present them for AdaBoost.

2.2.3 Horse Racing Gambling

In one of the first papers from Freund and Schapire about Boosting [FS95], where they also introduce the later-described AdaBoost algorithm, an interesting fictitious problem of a horse racing gambler is presented together with the description of a general learning framework. The analysis of this framework and the algorithms proposed with it provide useful insight for the understanding of the evolution from boosting weak learners by majority voting, as presented in Section 2.2.2, to modern Boosting algorithms such as AdaBoost, to be examined ahead in Section 2.3.

The story is that of a gambler that, frustrated by steady losses in horse racing bets, decides to recruit a group of other gamblers to bet for him. The problem is therefore the on-line allocation of resources among these gamblers, corresponding to the sharing of the total amount waged at each race. Consider an agent A with $\{1, \ldots, N\}$ allocation strategies to choose from. At each time $t | t \in \{1, \ldots, T\}$, A decides on a distribution $\boldsymbol{p}^t$ over the strategies, that is, $\sum_{i=1}^{N} p_i^t = 1$ and $p_i^t \geq 0$ for all strategies. Each strategy i suffers a loss $\ell_i^t \in [0, 1]$, where the average loss of A, referred to as *mixture loss*, is $\sum_{i=1}^{N} p_i^t \ell_i^t = \boldsymbol{p}^t \cdot \boldsymbol{\ell}^t$.

The goal of A is to minimize its loss relative to the best strategy, that is, its net loss:

$$L_A - \min_i L_i, \tag{2.1}$$

where the total cumulative loss of A in T runs is:

$$L_A = \sum_{t=1}^{T} \boldsymbol{p}^t \cdot \boldsymbol{\ell}^t, \tag{2.2}$$

and the cumulative loss of strategy i is:

$$L_i = \sum_{t=1}^{T} \ell_i^t. \tag{2.3}$$

We may now introduce $\boldsymbol{Hedge}\,(\beta)$, the algorithm presented by Freund and Schapire [FS95] to solve the problem of on-line allocation of resources. It maintains a weight vector whose value at time t is $\boldsymbol{w}^t = \{w_1^t, \ldots, w_N^t\}$, where the initial values are set as to describe a uniform distribution $w_i^1 = 1/N$. Despite this initial setting, these values need not necessarily sum to one at other times, since the distribution vector is updated after its normalization:

$$\boldsymbol{p}^t = \frac{\boldsymbol{w}^t}{\sum_{i=1}^{N} w_i^t}. \tag{2.4}$$

The weight vector is updated using the results in the loss vector and multiplicative rule below, which derives from [LW94]:

$$w_i^{t+1} = w_i^t \cdot U_\beta\left(\ell_i^t\right), \tag{2.5}$$

where $U_\beta : [0,1] \rightarrow [0,1]$ may be any function parametrized by $\beta \in [0,1]$ so that:

$$\beta^r \leq U_\beta(r) \leq 1 - (1-\beta)\, r. \tag{2.6}$$

Littlestone and Warmuth [LW94] assert that a satisfactory $U_\beta(r)$ always exists for any given $\beta \in [0,1]$ and $r \in [0,1]$. We shall now examine the upper bound for $\boldsymbol{Hedge}(\beta)$ on $\sum_{i=1}^{N} w_i^{T+1}$, as Littlestone and Warmuth did for their weighted majority voting algorithm, since its dictates the upper bound on the loss of the algorithm.

Freund and Schapire [FS95] prove that for any sequence of loss vectors $\ell^1, \ldots, \ell^T$, the following inequality holds:

$$L_{\boldsymbol{Hedge}(\beta)} \leq \frac{-\ln\left(sum_{i=1}^{N} w_i^{T+1}\right)}{1-\beta} \tag{2.7}$$

Furthermore, Freund and Schapire also prove that for any sequence of loss vectors $\ell^1, \ldots, \ell^T$ and for any $i \in \{1, \ldots, N\}$, this inequality becomes:

$$L_{\boldsymbol{Hedge}(\beta)} \leq \frac{-\ln\left(w_i^1\right) - L_i \ln \beta}{1-\beta}, \tag{2.8}$$

which means that $\boldsymbol{Hedge}(\beta)$ is not much worse than the best strategy i for the sequence. The two key factors that govern its behavior are the choices of β and the initial weight w_i^1 for each strategy. If we maintain $w_i^1 = 1/N$ as suggested before, this bound becomes:

$$L_{\boldsymbol{Hedge}(\beta)} \leq \frac{\min_i L_i \ln(1/\beta) + \ln N}{1-\beta} \tag{2.9}$$

Finally, we must now specify β to maximally exploit any prior knowledge we may have about the problem being solved. Suppose we know in advance that for the N strategies we have, there is an upper bound $\tilde{L}$. Freund and Schapire [FS95] demonstrate how to transform Equation (2.9) into:

$$L_{\boldsymbol{Hedge}(\beta)} \leq \min_i L_i + \sqrt{2\tilde{L} \ln N} + \ln N, \tag{2.10}$$

for $\beta = \frac{1}{1+\sqrt{2\ln N/\tilde{L}}}$.

In general, if T is known ahead of time, we may use $\tilde{L} = T$ as an upper bound on the cumulative loss of each strategy i. Since $\tilde{L} \leq T$, we have that the worst case rate of convergence for the algorithm is $O\left(\sqrt{(\ln N)/T}\right)$. However, as $\tilde{L}$ tends to zero, this rate significantly improves to $O((\ln N)/T)$ [FS95].

Given: $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize: $D_1(i) = 1/m$
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \rightarrow \mathbb{R}$.
- Choose: $\alpha_t \in \mathbb{R}$
- Update:

$$D_{t+1}(i) = \frac{D_t(i)\, e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Figure 2.1: Pseudo-code for a generalized version of AdaBoost.

2.3 The AdaBoost Family of Algorithms

Now that we have described the basics of the PAC framework as well as some of the historical developments of Boosting algorithms, we may describe that which became the most well-known family of boosters, AdaBoost. The first AdaBoost algorithm was introduced by Freund and Schapire [FS95] in 1995 using the binary learning framework described in Section 2.2.2. It emerged as a natural evolution of the resource allocation algorithm described in Section 2.2.3. Since then, Freund and Schapire published many papers describing their algorithms along with performance comparisons with other learning schemes, such as [FS96a, FS96b, FS97, FS99b, Sch99a], and more recently [Sch02].

The pseudo-code for a generalized version of AdaBoost, as first given by Schapire and Singer [SS99] and later by Schapire [Sch02], is presented in Figure 2.1. The algorithm takes as input a training set $(x_1, y_1), \ldots, (x_m, y_m)$ with m instances, where $x_i \in X$ and $y_i \in Y = \{-1, +1\}$. The distribution D_t is first initialized to a uniform distribution where $D_1(i) = 1/m$. In each of the $t = 1, \ldots, T$ rounds,

AdaBoost calls a weak learning algorithm to construct *weak* hypotheses h_t based on the current status of the distribution D_t. The objective of this weak learner, or base learner, is to minimize its training error ϵ_t. Once h_t is built and evaluated, AdaBoost chooses parameter $\alpha_t \in \mathbb{R}$ which measures how relevant each hypothesis h_t is.

Even though we have not explicitly defined how to compute ϵ_t in this generalized version of the algorithm, if we assume a binary output domain where $h_t \in \{-1, +1\}$, we have:

$$\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]. \tag{2.11}$$

Also strictly for $h_t \in \{-1, +1\}$, we may define the weight of each hypothesis as:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{2.12}$$

Based on α_t at the end of each round, D_t is updated and normalized in an attempt to focus the attention of the algorithm on those examples which are harder to learn. The normalization factor Z_t is chosen so that D_{t+1} will be a distribution, that is, so that the summation of all probability values in D_{t+1} is one. After the T-th and final round, the *strong* hypothesis is computed by the weighted majority vote of the T weak hypotheses where α_t is the weight assigned to each h_t.

Pragmatically, AdaBoost has many advantages as a Boosting algorithm. It is relatively fast, simple, and easy to program. Also, it has no parameters to tune except for the number of rounds T, and also requires no prior knowledge about the weak learner being used. Nevertheless, there are some requirements that must be met in order for it to function correctly. AdaBoost will fail to work if there is not sufficient training data or if the restriction on quality of the weak learner, dependent on output domain of the problem, is not met [Sch02]. According to Dietterich [Die00], Boosting seems to be specially susceptible to noise. Moreover, Boosting may also fail to perform well given overly complex base classifiers, which is one of the issues addressed in Chapter 4, or too weak base classifiers [Sch02].

The following sections discuss specializations of the AdaBoost algorithm. The first two, AdaBoost.M1 and AdaBoost.M2, have been introduced by Freund and Schapire [FS96a] in 1996. AdaBoost.M1 is the best-known version of the family, and it is often wrongly referred to as simply AdaBoost in literature. AdaBoost.M2 is an extension of AdaBoost.M1 that works with more sophisticated base learners. Later in 1999, Schapire and Singer [SS99] proposed three other versions of AdaBoost, AdaBoost.MH, AdaBoost.MO, and AdaBoost.MR, all still consistent with the general version presented in Figure 2.1, though each one with either differentiated input and output domains or customized loss functions.

2.3.1 AdaBoost.M1

AdaBoost.M1 is certainly one of the best-known Boosting algorithms in literature, mainly due to its relative simplicity and applicability to a wide range of problems and weak learners. The algorithm, presented in Figure 2.2, is very similar to the generalized version presented in Figure 2.1. There are two differences worth commenting though. First, the output domains of the learning problems being solved are not restricted to $\{-1, +1\}$, but instead may be a finite set of k labels $\{1, \ldots, k\}$. Second, even though h_t is not restricted to a binary output, the error measure takes the same form as Equation (2.11), that is, $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.

The important theoretical property of AdaBoost.M1 is stated in Theorem 1, which is presented [FS96a] and proved by Freund and Schapire [FS95, FS97].

Theorem 1. *Suppose the weak learning algorithm, when called by AdaBoost.M1, generates hypotheses with errors $\epsilon_1, \ldots, \epsilon_T$, where ϵ_t is as defined in Figure 2.2. Assume each $\epsilon_t \leq 1/2$, and let $\gamma_t = 1/2 - \epsilon_t$. Then the following upper bound holds on the error of the final hypothesis H:*

$$\frac{|\{i : H(x_i) \neq y_i\}|}{m} \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq e^{-2\sum_{t=1}^{T} \gamma_t^2}.$$

Theorem 1 shows that a weak learner with consistent error rates marginally better than $1/k$, where k is the number of finite labels in the problem's output domain, may lead to strong hypotheses whose training error drop exponentially fast. For binary problems, this means that the weak learner must only be arbitrarily better than a random guess. Though this may be considered as a strength of the algorithm at times, this requirement may be hard to meet for large values of k.

2.3.2 AdaBoost.M2

This extension of AdaBoost.M1 attempts to overcome the limitation of the latter's weak hypotheses by improving the communication between the booster, AdaBoost.M2 itself, and the weak learner. Each hypothesis returned by the weak learner now may output a set of "plausible"[3] labels, instead of a single one. Furthermore, each label also contains a degree of plausibility, which corresponds to its confidence rating. Therefore, the output of these hypotheses is no longer in $\{1, \ldots, k\}$, but instead a vector in $[0, 1]^k$, where values toward 1 or 0 mean greater or lesser plausibility, respectively.

[3] Freund and Schapire [FS96a] emphasize the use of the term *plausible* instead of *probable*, so as not to be mistaken for probability values.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$, $y_i \in Y = \{1, \ldots, k\}$
Initialize: $D_1(i) = 1/m$
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \rightarrow Y$.
- Calculate the error of h_t: $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
  If $\epsilon_t > 1/2$, set $T = t - 1$ and abort loop.
- Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ and $\alpha_t = \log \frac{1}{\beta_t}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise,} \end{cases}$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \arg\max_{y \in Y} \sum_{t: h_t(x) = y} \alpha_t.$$

Figure 2.2: Pseudo-code for AdaBoost.M1.

Now that the expressive power of weak hypotheses is increased, this information must be taken into account accordingly. In order to do that, we replace the original error measure computed with respect to a distribution over examples by a more sophisticated *pseudo-loss* function computed with respect to a distribution over the set of all pairs of examples and incorrect labels. Consider a mislabel (i, y), where i is the index of the training example and y is the associated incorrect label. Let B be the set of all mislabels $B = \{(i, y) : i \in \{1, \ldots, m\}, y \neq y_i\}$. The pseudo-loss function is defined as:

$$\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y) \left(1 - h_t(x_i, y_i) + h_t(x_i, y)\right). \tag{2.13}$$

The complete derivation of Equation (2.13) is given by Freund and Schapire [FS97].

Theorem 2. *Suppose the weak learning algorithm, when called by AdaBoost.M2, generates hypotheses with pseudo-losses $\epsilon_1, \ldots, \epsilon_T$, where ϵ_t is as defined in Figure 2.3. Let $\gamma_t = 1/2 - \epsilon_t$. Then the following upper bound holds on the error of the final hypothesis H for a problem with k output labels:*

$$\frac{|\{i : H(x_i) \neq y_i\}|}{m} \leq (k-1) \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq (k-1)\, e^{-2 \sum_{t=1}^{T} \gamma_t^2}.$$

Theorem 2 states the bound on the training error for AdaBoost.M2. Notice that weak hypotheses must have pseudo-loss less than $1/2$, regardless of the number of classes in the problem. Also, even though weak hypotheses output sophisticated pseudo-loss measures, the final strong hypothesis still outputs its prediction over the finite set of labels defined by the problem's output domain. Finally, notice that due to their error measure and pseudo-loss function, respectively, AdaBoost.M1 and AdaBoost.M2 are equivalent for binary problems, differing only in handling problems whose output domain contains more than two labels [FS96a].

2.3.3 AdaBoost.MH

Besides being able to deal with multiclass problems, AdaBoost has also been extended to deal with multilabel predictions, that is, where instances may belong to any number of classes. Let Υ be a finite set of labels $\Upsilon = \{1, \ldots, k\}$, such that $k = |\Upsilon|$. Instead of the traditional classification setting where labeled examples are pairs (x, y) such that $x \in X$ and $y \in Y$, where X and Y are the input and output domains, respectively, we now have pairs (x, Y), where $Y \subseteq \Upsilon$. Notice that the traditional setting for single-label classification is the mere case where $|Y| = 1$.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$, $y_i \in Y = \{1, \ldots, k\}$
Let $B = \{(i, y) : i \in \{1, \ldots, m\}, y \neq y_i\}$.
Initialize: $D_1(i, y) = 1/|B|$ for $(i, y) \in B$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \times Y \to [0, 1]$.
- Calculate the pseudo-loss of h_t:

$$\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y) \left(1 - h_t(x_i, y_i) + h_t(x_i, y)\right)$$

- Set $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$ and $\alpha_t = \log \frac{1}{\beta_t}$.
- Update:

$$D_{t+1}(i, y) = \frac{D_t(i)}{Z_t} \cdot \beta_t^{(1/2)(1 + h_t(x_i, y_i) - h_t(x_i, y))}$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t h_t(x, y).$$

Figure 2.3: Pseudo-code for AdaBoost.M2.

The AdaBoost.MH algorithm presented in Figure 2.4 incorporates this multilabel scenario together with a different loss function called *Hamming loss*. Hamming loss assumes the multilabel prediction is reduced to the prediction of all and only all correct labels, that is, $H : X \rightarrow 2^{\Upsilon}$. With respect to a distribution D, the loss is:

$$\frac{1}{k} E_{(x,Y)\sim D}[|h(x) \Delta Y|] \tag{2.14}$$

where Δ denotes symmetric difference.

In order to minimize Hamming loss, we may split the problem into k orthogonal binary problems. The Hamming loss may then be seen as the average of the error rate of h on all k problems. For $Y \subseteq \Upsilon$, we define $Y[\ell]$ for $\ell \in \Upsilon$:

$$Y[\ell] = \begin{cases} +1 & \text{if } \ell \in Y \\ -1 & \text{if } \ell \notin Y \end{cases} \tag{2.15}$$

Finally, we identify any function $H : X \rightarrow 2^{\Upsilon}$ with a corresponding function $H : X \times \Upsilon \rightarrow \{-1, +1\}$ as defined by $H(x, \ell) = H(x)[\ell]$.

In Figure 2.4 we present a more abstract form of AdaBoost.MH, meaning, a form with no definition of output domain of h_t, hence also missing the definition for α_t. Schapire and Singer [SS99] analyze the case where $h_t \in \{-1, +1\}$, where they also provide interesting insights about comparing the use of AdaBoost.MH in single-label problems against other simpler versions of AdaBoost.

2.3.4 AdaBoost.MO

AdaBoost.MH mapped a single-label problem into a multilabel problem by transforming each pair (x, y) in a multilabel observation $(x, Y) | y \in Y$. However, a more sophisticated and effective mapping consists of transforming (x, y) into $(x, \lambda(y))$, where λ is a one-to-one mapping $\lambda : \Upsilon \rightarrow 2^{\Upsilon'}$ and $k' = |\Upsilon'|$. Notice that the label set Υ' need not be equal to Υ.

This coding scheme proposed by Dietterich and Bakiri [DB95] was used by Schapire and Singer [SS99] to create AdaBoost.MO, presented in Figure 2.5. Similar schemes with the same purpose were also used by Allwein et al. [ASS00] and Schapire [Sch97]. There are two ways to evaluate AdaBoost.MO's final hypothesis. The first, referred to as *Variant 1*, consists of choosing the output label $y \in Y$ whose mapped output code $\lambda(y)$ has the shortest Hamming distance to $H(x)$. The second, *Variant 2*, predicts the label y whose pair (x, y) was given the smallest weight under the final distribution, that is, the same approach taken by other AdaBoost versions.

Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$, where $x_i \in X$, $Y_i \subseteq \Upsilon = \{1, \ldots, k\}$
Initialize: $D_1(i, \ell) = 1/m \cdot |Y|$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \times Y \to \mathbb{R}$.
- Choose: $\alpha_t \in \mathbb{R}$
- Update:

$$D_{t+1}(i, \ell) = \frac{D_t(i, \ell)\, e^{-\alpha_t y_i[\ell] h_t(x_i, \ell)}}{Z_t}$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x, \ell) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x, \ell)\right).$$

Figure 2.4: Pseudo-code for AdaBoost.MH.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$, $y_i \in Y = \{1, \ldots, k\}$
Given: a mapping $\lambda : Y \to 2^{Y'}$

- Run AdaBoost.MH on relabeled data: $(x_1, \lambda(y_1)), \ldots, (x_m, \lambda(y_m))$.
- Get back final hypothesis H of form $H(x, y') = sign(f(x, y'))$ where $f(x, y') = \sum_{t=1}^{T} \alpha_t h_t(x, y')$.
- Output modified final hypothesis:

$$\text{(Variant 1)} \quad H_1(x) = \arg\min_{y \in Y} |\lambda(y)\, \Delta H(x)|$$
$$\text{(Variant 2)} \quad H_2(x) = \arg\min_{y \in Y} \sum_{y' \in Y'} e^{-\lambda(y)[y'] f(x, y')}$$

Figure 2.5: Pseudo-code for AdaBoost.MO.

2.3.5 AdaBoost.MR

All AdaBoost instances so far dealt with problems whose instances were exactly identified with one or more finite labels. A variation of this problem is that where the goal is to find a hypothesis which ranks labels according to their confidence ratings. To solve this problem and create AdaBoost.MR, Schapire and Singer [SS99] used an approach close to that used by Freund et al. [FISS98], presented in Figure 2.6.

The final hypothesis has the form $H : X \times \Upsilon \rightarrow \mathbb{R}$, where the association of each label $\ell \in \Upsilon$ with each instance $x \in X$ has a degree of confidence. With respect to (x, Y), we only care about the relative ordering of the *crucial pairs* ℓ_0 and ℓ_1, for which $\ell_0 \notin Y_i$ and $\ell_1 \in Y$. Our goal is to find a hypothesis with the smallest number of misorderings so that labels in Y are ranked above labels not in Y. This goal may be measured quantitatively by a ranking loss function, described as follows with respect to a distribution D:

$$E_{(x,Y)\sim D}[\frac{|\{(\ell_0, \ell_1) \in (\Upsilon - Y) \times Y : H(x, \ell_1) \leq H(x, \ell_0)\}|}{|Y||\Upsilon - Y|}]. \tag{2.16}$$

Finally, notice that in AdaBoost.MR, D_t is now maintained over $\{1, \ldots, m\} \times Y \times Y$, and that the weight increases and decreases are computed as a function of the difference in ratings $h_t(x_i, \ell_0) - h_t(x_i, \ell_1)$.

2.3.6 Analysis of Error Bounds

Given that our main interest for AdaBoost in this work is for dealing with binary problems, we now take the time to analyze the training and generalization errors for the generalized version of AdaBoost presented in Figure 2.1. Notice that this version is equivalent to AdaBoost.M1 from Figure 2.2 if the latter is adjusted to a binary output domain instead of a finite set of labels with arbitrary number of elements.

We start by analyzing the training error, since Schapire [Sch02] considered it to be the most basic theoretical property of AdaBoost. The following theorem proved by Schapire and Singer [SS99], which is itself the generalization of another theorem from Freund and Schapire [FS95, FS97], states the bounds on the training error of the final classifier. Other works by Schapire [Sch99b, Sch99c] and Freund and Schapire [FS99a] also bring interesting analyses of the theory of AdaBoost as well as its convergence with game theory.

Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$, where $x_i \in X$, $Y_i \subseteq \Upsilon = \{1, \ldots, k\}$

Initialize: $D_1(i, \ell_0, \ell_1) = \begin{cases} 1/(m \cdot |Y_i| \cdot |\Upsilon - Y_i|) & \text{if } \ell_0 \notin Y_i \text{ and } \ell_1 \in Y_i \\ 0 & \text{otherwise.} \end{cases}$

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \times Y \to \mathbb{R}$.
- Choose: $\alpha_t \in \mathbb{R}$
- Update:

$$D_{t+1}(i, \ell_0, \ell_1) = \frac{D_t(i, \ell_0, \ell_1)\, e^{\frac{1}{2}\alpha_t (h_t(x_i, \ell_0) - h_t(x_i, \ell_1))}}{Z_t},$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x, \ell) = \sum_{t=1}^{T} \alpha_t h_t(x, \ell).$$

Figure 2.6: Pseudo-code for AdaBoost.MR.

Theorem 3. *Assuming the notation of Figure 2.1, the following bound holds on the training error of H:*

$$\frac{1}{m}|\{i : H(x_i) \neq y_i\}| \leq \frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)} = \prod_{t=1}^{T} Z_t.$$

Proof. We prove the theorem by unraveling the recursive definition of the update rule.

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \\ &= \frac{e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)}}{m \prod_{t=1}^{T} Z_t} \end{aligned} \tag{2.17}$$

Moreover, if $H(x_i) \neq y_i$ then we have that $y_i \sum_{t=1}^{T} \alpha_t y_i h_t(x_i) \leq 0$, which also implies that $e^{-\sum_{t=1}^{T} \alpha_t y_i h_t(x_i)} \geq 1$. Thus,

$$[H(x_i) \neq y_i] \leq e^{-\sum_{t=1}^{T} \alpha_t y_i h_t(x_i)}. \tag{2.18}$$

Combining Equations (2.17) and (2.18) gives the stated bound on the training error:

$$\begin{aligned} \frac{1}{m}\sum_{i=1}^{m}[H(x_i) \neq y_i] &\leq \frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)} \\ &= \sum_{i=1}^{m} \left(\prod_{t=1}^{T} Z_t\right) D_{T+1}(i) \\ &= \prod_{t=1}^{T} Z_t \end{aligned} \tag{2.19}$$

□

Theorem 3 suggests that a custom selection of α_t and h_t can minimize Z_t:

$$Z_t = \sum_{i=1}^{m} D_t(i)\, e^{-\alpha_t y_i h_t(x_i)}. \tag{2.20}$$

For binary classifiers, we use the definition of α_t given in Equation (2.12), which yields the following bounds on the training error:

$$\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T}[2\sqrt{\epsilon_t(1-\epsilon_t)}] = \prod_{t=1}^{T}\sqrt{1-4\gamma_t^2} \leq e^{-2\sum_{t=1}^{T}\gamma_t^2}, \tag{2.21}$$

where $\gamma_t = 1/2 - \epsilon_t$. Therefore, if the weak learner is marginally better than random, that is, if $\gamma_t \geq \gamma$ for some $\gamma > 0$, then the training error drops exponentially fast in T since the bound in Equation (2.21) is at most $e^{-2T\gamma^2}$.

Now that we have dissected the behavior of AdaBoost with respect to its training error, we proceed to analyze its generalization error, which is much more relevant when evaluating and boosting learning machines. One of the most popular methods for doing this analysis is Vapnik's method of structural risk minimization

[Vap82], which we later use in Chapter 3 to explain the foundations of Support Vector Machines. This method was also previously used by Schapire et al. [SFBL97, SFBL98]. We state Vapnik's theorem below, in the context given by Freund and Schapire [FS97].

Theorem 4. *Let X be a class of binary functions over some domain X. Let d be the Vapnik-Chervonenkis dimension of H. Let P be a distribution over the pairs $X \times \{0,1\}$. For $h \in H$, we define the generalization error of h with respect to D to be:*

$$\epsilon_g(h) = Pr_{(x,y)\sim D}[h(x) \neq y].$$

Let $S = \{(x_1, Y_1), \ldots, (x_m, Y_m)\}$ be the training set with m independent random samples drawn from $X \times \{0,1\}$ according to D. Define the empirical error of h with respect to the set S to be:

$$\hat{\epsilon}(h) = \frac{|\{i : h(x_i) \neq y_i\}|}{m}.$$

Then for any $\delta > 0$ we have that:

$$Pr\lfloor \exists h \in H : |\hat{\epsilon}(h) - \epsilon_g(h)| > 2\sqrt{\frac{d\left(\ln \frac{2m}{d} + 1\right) + \ln \frac{9}{\delta}}{m}} \rfloor \leq \delta$$

where the probability is computed with respect to the random choice of the sample S.

Let $\theta : \mathbb{R} \rightarrow \{0,1\}$ be defined by:

$$\theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{2.22}$$

and, for any class functions H, let $\Theta_T(H)$ be the class of all functions defined as a linear threshold of T functions in H:

$$\Theta_T(H) = \left\{ \theta\left(\sum_{t=1}^{T} a_t h_t - b \right) : b, a_1, \ldots, a_T \in \mathbb{R}; h_1, \ldots, h_T \in H \right\}. \tag{2.23}$$

Hence, if all hypotheses generated by the weak learner belong to some class H, then the final strong hypothesis, after T rounds of boosting, belongs $\Theta_T(H)$. Thus, Theorem 5 provides an upper bound on the VC-dimension of the class of final hypotheses generated of AdaBoost in terms of the class of weak hypotheses.

Theorem 5. *Let H be a class of binary functions of VC-dimension $d \geq 2$. Then the VC-dimension of $\Theta_T(H)$ is at most $2(d+1)(T+1)\log_2(eT+e)$.*

Therefore, if the hypotheses generated by the weak learner are chosen from a class of VC-dimension $d \geq 2$, then the final hypothesis generated by AdaBoost after T iterations belong to a class of VC-dimension at most $2(d+1)(T+1)\log_2(eT+e)$.

Proof. We use a result about the VC-dimension of computation networks proved by Baum and Haussler [BH89]. We can view the final hypothesis output by AdaBoost as a function that is computed by a two-layer feed-forward network where the computation units of the first layer are the weak hypotheses and the computation unit of the second layer is the linear threshold function which combines weak hypotheses. The VC-dimension of the set of linear threshold functions over $\mathbb{R}^T$ is $T+1$ [WD81]. Thus the sum over all computation units of the VC-dimensions of the classes of functions associated with each unit is $Td + (T+1) < (T+1)(d+1)$. Baum and Haussler's Theorem 1 [BH89] implies that the number of different functions that can be realized by $h \in \Theta_T(H)$ when the domain is restricted to a set of size m is at most $((T+1)em/(T+1)(d+1))^{(T+1)(d+1)}$. If $d \geq 2$, $T \geq 1$ and we set $m = \lceil 2(T+1)(d+1)\log_2(eT+e) \rceil$, then the number of realizable functions is smaller than 2^m, which implies that the VC-dimension of $\Theta_T(H)$ is smaller than m. □

Alternatively, another way of analyzing the generalization of AdaBoost was proposed by Schapire et al. [FISS98]. Following the research by Bartlett [Bar98], they derived a different analysis in terms of the margins of the training examples. The margin of example (x, y) is defined as:

$$\text{margin}_f(x, y) = \frac{y \sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} |\alpha_t|} \tag{2.24}$$

This margin is a real value in $[-1, +1]$, and is positive if and only if H correctly classifies the example. Also, the magnitude of the margin may be interpreted as a measure of confidence in the prediction [Sch02]. Schapire et al. [FISS98] proved that larger margins on the training set translate into superior upper bounds on the generalization error, which is at most:

$$\hat{\Pr}[\text{margin}_f(x, y) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right) \tag{2.25}$$

for any $\theta > 0$ with high probability [4]. Interestingly enough, note that this bound is entirely independent of T, that is, the number of boosting rounds run by AdaBoost. Schapire et al. proved that Boosting is aggressive at reducing the margin, since it concentrates on examples with smallest margin magnitudes.

[4] $\tilde{O}$ corresponds to the *soft-O* complexity notation, which disregards both logarithmic and constant factors.

2.4 Advances on the Boosting Front

Although there have been many advances in Boosting over its brief existence in literature, Boosting is yet a very incipient area of research. Several new fields have received plenty of attention from the machine learning community, and new groundbreaking discoveries still happen from time to time. The next few sections explore some the newly developed concepts related to Boosting, as well as their practical applications in different problems.

2.4.1 Relation to Support Vector Machines

Aside this text, it is rare to find work combining Boosting with Support Vector Machines. One such example is the interesting paper by Rätsch et al. [RMSM02] in which they built a framework for converting algorithms based on SVM principles into new Boosting algorithms. Another example is by Freund and Schapire [FS99b], where they point out the differences and similarities between SVMs and Boosting. These authors conclude that though SVMs and Boosting methods differ, both work well in very high-dimensional feature spaces. One can think of Boosting as a Support Vector approach in high-dimensional feature spaces spanned by the weak hypotheses. Similarly, one can think of an SVM as a Boosting approach in high-dimensional space.

Despite the insightful conclusions obtained by Rätsch et al., their interest in the combination of Boosting and SVMs is rather different than the approach explored in this work, where we used SVMs as customized weak learners for well-studied Boosting algorithms. Nonetheless, yet another remark from Rätsch et al. is worthwhile pointing out, which is the similarity between Boosting and SVMs. According to them, it has become "common folklore" to mistake the internal workings of Boosting and SVMs as essentially the same, except for the way they measure the margin, Boosting with a 1-norm [5] and SVMs with the 2-norm.

SVMs and Boosting use totally unique strategies to handle high or even infinite dimensional spaces. First, SVMs use the 2-norm to implicitly compute scalar products in the feature space with the aid of an internal kernel. No other norm may be expressed in terms of scalar products. Boosting, on the other hand, performs the computation explicitly in the feature space, where a 1-norm is used to induce sparseness on the separating hyperplane solution to protect it from from the high or even infinite dimensionality of the feature space [RMSM02]. Freund and Schapire [FS99b] draw yet another analogy between SVMs and Boosting, where they associate the SVM approach with solving a quadratic programming (QP) problem, and

[5] Please recall that $||x||_1 = \sum_i |x_i|$.

the Boosting approach with a greedy search based on linear programming.

2.4.2 Research and Applications

Though not directly inserted in the context of the discussions regarding Boosting presented in this chapter, it is worth mentioning a few authors whose works not only are innovative, but also may lead to some of the future work described in Chapter 7.

Several authors worked on interesting problems of boosting text classification machines, which differ significantly from standard problems in Euclidean input spaces. For instance, the BoosTexter system from Schapire and Singer [SS00], the RankBoost algorithm from Iyer et al. [ILS+00], and a text filtering scheme from Schapire et at. [SSS98]. Abney et al. [ASS99], though not directly classifying chunks of text, used Boosting techniques in a natural language tagging application. Collins et al. [CSS00] give a unified account of Boosting and logistic regression in which each learning problem is cast in terms of the optimization of Bregman distances.

Freund et al. [FMS01] present an interesting new algorithm, not at all inspired on AdaBoost, with which they show how computing the weighted average of hypotheses, instead of simply selecting the best hypothesis from a set, can protect against overfitting. Rosset et al. [RZH02] study Boosting from a new perspective, where they show that it approximately minimizes its loss criterion with an L_1 constraint, and as this constraint diminishes in a separable problem, the solution converges to a "L_1-optimal" separating hyperplane. Finally, Rätsch [Rät01] proposes a statistical learning theory framework for analyzing Boosting methods with which he explores means of improving the robustness of Boosting algorithms via mathematical optimization techniques.

Chapter 3

Support Vector Machines

This chapter introduces the concepts and propositions of Support Vector Machines designed for solving classification problems, starting with a simple linear machine and gradually evolving into an advanced non-linear soft-margin machine. In Section 3.1 we examine concepts of generalization theory for machine learning, such as empirical risk minimization, Vapnik-Chervonenkis theory [Vap82], and structural risk minimization [Bur98]. In Section 3.2 we use a trivial linear discriminant to introduce a basic learning framework later used to describe Support Vector Machines from the bottom up. Section 3.3 introduces the concepts of inner-product kernel functions and learning in kernel-induced feature spaces, which are some of the key features behind the representation power of SVMs. Finally, in Section 3.4 we examine formulations of different machines, evolving from simple linear SVMs into non-linear and non-linear soft-margin SVMs. Concluding the chapter, in Section 3.5 we describe different approaches used to train these learning machines, starting with reviews of simple algorithms and later focusing on the Sequential Minimal Optimization (SMO) algorithm by John Platt [Pla98a].

3.1 Generalization Theory

There are many advanced learning methodologies based on learning in high-dimensional feature spaces. One of them, based on kernel-induced feature spaces, further discussed in Section 3.3, offers great expressive power to learning machines. Even though this is a most welcome feature for solving difficult learning problems, this power must also be controlled to avoid undesirable phenomena such as overfitting.

One of the learning theories able to guarantee generalization bounds on learning machines is that of Vapnik and Chervonenkis [Vap82]. Not only this is the most appropriate theory for describing Support Vector Machines, it has also his-

torically motivated them. Vapnik [Vap95] divides learning theory into four parts:

- Theory of consistency of learning processes.
- Non-asymptotic theory of the rate of convergence of learning processes.
- Theory of controlling the generalization ability of learning processes.
- Theory of constructing learning algorithms.

In this section we shall examine concepts regarding the first three parts, where we establish the foundations to explore the forth part when we discuss the intricacies of Support Vector Machines.

Even though we do not address this issue here, other learning theories such as Bayesian analysis may be used to describe SVMs. A brief outline of such analysis is given by Cristianini and Shawe-Taylor [CST00].

3.1.1 PAC Learning Revisited

We now explore Valiant's Probably Approximately Correct (PAC) model, or *distribution free* learning model, under a new light. We examined both weak and strong PAC learning in Section 2.2 in the context of Boosting, using them to explain the motivations behind majority voting and the principles of Boosting algorithms. We shall now revisit some of these issues and introduce new concepts that together serve as basis for the Vapnik-Chervonenkis theory, which we will use later to describe Support Vector Machines.

According to Cristianini and Shawe-Taylor [CST00], one of the key assumptions of the PAC model is that training and testing data in a learning procedure are generated independently and identically (*i.i.d.*) according to an unknown yet fixed distribution D. There are refinements of the model that consider the distribution D changing over time, or even cases where the learning algorithm may influence the selection of data. These refinements will not be considered in this discussion, in which we assume the i.i.d. case.

Since data samples are drawn according to D, we may introduce an error measure for a generic binary classification function f to define the probability of misclassification of a random example:

$$\mathrm{err}_D(f) = D\{(x, y) : f(x \neq y)\}, \tag{3.1}$$

where $x \in X$ and $y \in Y = \{-1, +1\}$. We refer to this error measure as *risk functional*. The purpose of this analysis is to assert bounds on the error of the hypothesis, for instance in terms of the number of training examples required to

obtain a particular level of error. This concept is known as the *sample complexity* of the learning problem.

Consider the selection mechanism of a hypothesis function f from a class F, such that the selection must be based on a set $S = ((x_1, y_1), \ldots, (x_N, y_N))$ of N training examples chosen i.i.d. according to D. We may use the PAC model to examine the generalization bounds of this selection mechanism, the learning algorithm itself, through the bound $\epsilon = \epsilon(N, F, \delta)$, where δ is a parameter specified by the algorithm. This bound asserts that with probability of at least $1 - \delta$ over the training sets in S, the generalization error of f is bounded as follows:

$$\mathrm{err}_D(f) \leq \epsilon(N, F, \delta), \tag{3.2}$$

or, in other words, is probably approximately correct [CST00]. Put differently, we may state this inequality in terms of the small probability that the training set gives rise to a hypothesis function with large error:

$$D^n\{S : \mathrm{err}_D(f) > \epsilon(N, F, \delta)\} < \delta. \tag{3.3}$$

3.1.2 Empirical Risk Minimization

We may define the risk functional given in Section 3.1.1 more formally, where the hypothesis function derived by the learner is $f(x, w) \rightarrow y$, and where $w \in W$ is a parameter for the learning algorithm chosen from the set of all possible parameters. We then define a generic loss function $L(y, f(x, w))$, where L is implemented according to the domain Y, which contains all possible y. For instance, given a binary classification problem, a simple example of loss function would be:

$$L(y, f(x, w)) = \begin{cases} 0 & \text{if } y = f(x, w) \\ 1 & \text{if } y \neq f(x, w). \end{cases} \tag{3.4}$$

Many other examples of loss functions may be used, for example in regression problems where Y is continuous. Finally, we may state the generic form of the risk functional as described by Almeida [ABB01b] and Burges [Bur98]:

$$R(w) = \frac{1}{2} \int L(y, f(x, w)) \, p_{xy}(x, y), \tag{3.5}$$

where $p_{xy}(x, y)$ is the probability density function that describes D. Hence, the job of the learning algorithm is to find the minimum value of $R(w)$, which is equivalent to finding $f(x, w')$ where w' is the optimal parameter set.

If $p_{xy}(x, y)$ is know, minimizing $R(w)$ becomes relatively easy. When this density function is unknown, though, its solution is no longer trivial as it becomes an ill-posed problem [ABB01b].

The inductive principle of empirical risk minimization is a method for estimating the minimization of the functional risk in terms of a loss function. Now that we have defined the functional risk in Equation (3.5), we may also define the empirical functional risk as:

$$R_{emp}(w) = \frac{1}{2N} \sum_{i=1}^{N} L(y_i, f(x_i, w)) \tag{3.6}$$

Note that there are no probability distributions in Equation (3.6). $R_{emp}(w)$ is fixed for a particular choice of w and for a particular training set $\langle x_i, y_i \rangle$ [Bur98].

Assume that w_0 and w_1 are the optimal parameter sets for the empirical functional risk and functional risk, respectively. We have that $R_{emp}(w_0)$ and $R(w_1)$ will converge to the same value as the number of samples N tends to infinity [ABB01b]. Moreover, for a given w' and some value η such that $0 \leq \eta \leq 1$, according to Vapnik [Vap95], the following bound holds with probability $1 - \eta$:

$$R(w') \leq R_{emp}(w') + \sqrt{\frac{h(\log(2l/h) + 1) - \log(\eta/4)}{l}}, \tag{3.7}$$

where h is the Vapnik-Chervonenkis dimension of the class F. Burges [Bur98] refers to inequality (3.7) as *risk bound*, and to the second term on its right hand side as *VC confidence*.

3.1.3 Vapnik-Chervonenkis Theory

Given a finite set of hypotheses, we may obtain a bound in the form of the inequality (3.2). Assume the selection of f is consistent with the training examples in S. The probability that all N independent examples are consistent with a hypothesis f such that $\text{err}_D(f) > \epsilon$ is bounded by:

$$D^N\{S : f \text{ consistent and } \text{err}_D(f) > \epsilon\} \leq (1 - \epsilon)^N \leq e^{-\epsilon N} \tag{3.8}$$

We have now that the probability of one single hypothesis in F being consistent with S, even if all $|F|$ hypotheses have large error, is at most $|F|e^{-\epsilon N}$ by the union bound on the probability that one of several events occurs [CST00]. We may now bound the probability of any consistent hypothesis f_S, as in inequality (3.3):

$$D^N\{S : f \text{ consistent and } \text{err}_D(f_S) > \epsilon\} < |F|e^{-\epsilon N} \tag{3.9}$$

In order to keep $|F|e^{-\epsilon N} < \delta$, we have that:

$$\epsilon = \epsilon(N, F, \delta) = \frac{1}{N} \ln \frac{|F|}{\delta}. \tag{3.10}$$

Equation (3.10) demonstrates how the complexity of the function class F, measured by its cardinality, has a direct effect on the error bound, where a $|F|$ too large may lead to overfitting. This result shows that a property relating the true error to the empirical error holds for all hypotheses in F, for which reason it is said to demonstrate *uniform convergence*. According to Cristianini and Shawe-Taylor [CST00], learning theory relies on bounding the difference between empirical and true estimates of error uniformly over the set of hypotheses and conditions that can arise, where the major contribution of the theory by Vapnik and Chervonenkis is to extend such analysis to infinite sets of hypotheses.

The bounding over an infinite set of functions is done by bounding the probability of inequality (3.3) by twice the probability of having zero error on the training examples but, high error on a second random sample $\hat{S}$:

$$\begin{aligned} &D^N\{S : \exists f \in F : \mathrm{err}_S(f) = 0, \mathrm{err}_S(f) > \epsilon\} \leq \\ &2D^{2N}\{S\hat{S} : \exists f \in F : \mathrm{err}_S(f) = 0, \mathrm{err}_{\hat{S}}(f) > \epsilon N/2\}. \end{aligned} \tag{3.11}$$

This bound is an application of Chernoff bounds, provided that $N > 2/\epsilon$ [CST00]. In order to obtain a union bound on the overall probability of the right hand side of inequality (3.11), we define what Vapnik [Vap95] called one of the three milestones of learning theory, the *growth function*:

$$B_F(N) = \max_{(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)\in X^N} |\{(f(\boldsymbol{x}_1),\ldots,f(\boldsymbol{x}_N)) : f \in F\}|. \tag{3.12}$$

It is interesting to observe that this function may not exceed 2^N since the sets over which the maximum is searched are all subsets of the set of binary sequences of length N. A set of points $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)$ is said to be *shattered* by F if:

$$\{(h(\boldsymbol{x}_1),\ldots,h(\boldsymbol{x}_N)) : f \in F\} = \{-1,+1\}^N. \tag{3.13}$$

The growth function is equal to 2^N for all N if there are sets of any size which can be shattered. Finally, consider the case where the largest size of shattered set is h, and where the growth function can be bounded as follows for $N > h$:

$$B_F(N) \leq \left(\frac{eN}{h}\right)^h, \tag{3.14}$$

giving polynomial growth with exponent h. The value d is known as the Vapnik-Chervonenkis (VC) dimension of the class F. The VC dimension is a measure of the richness or flexibility of the function class, and it is ofter referred to as its capacity [CST00]. We may now rewrite inequality (3.3) as:

$$D^n\{S : \exists f \in F : \mathrm{err}_S(f) = 0, \mathrm{err}_D(f) > \epsilon\} < 2\left(\frac{2eN}{h}\right)^N 2^{-\epsilon N/2}, \tag{3.15}$$

which results in a PAC bound for any consistent hypothesis f_S:

$$\mathrm{err}_D(f) \leq \epsilon(N, F, \delta) = \frac{1}{N}\left(N \log \frac{2eN}{h} + \log \frac{2}{\delta}\right). \tag{3.16}$$

This demonstrates the *fundamental theorem of learning* due to Vapnik and Chervonenkis [VC71].

Theorem 6. *(Vapnik and Chervonenkis) Let F be a hypothesis space having VC dimension h. For any probability distribution D on $X \times \{-1, +1\}$, with probability $1-\delta$ over N random examples S, any hypothesis $f \in F$ that is consistent with S has error no more than:*

$$err_D(f) \leq \epsilon(N, F, \delta) = \frac{1}{N}\left(h \log \frac{2eN}{h} + \log \frac{2}{\delta}\right),$$

provided $h \leq N$ and $N > 2/\epsilon$.

VC theory not only provides a distribution free bound on the generalization of a consistent hypothesis, but also shows that the bound is tight up to log factors. This property is shown in the following theorem.

Theorem 7. *Let F be a hypothesis space with finite VC dimension $h \geq 1$. Then for any learning algorithm there exist distributions such that with probability at least δ over N random examples, the error of the hypothesis f returned by the algorithm is at least:*

$$\max\left(\frac{h-1}{32N}, \frac{1}{N}\ln\frac{1}{\delta}\right).$$

3.1.4 Structural Risk Minimization

The VC theory outlined so far applies only when hypotheses are consistent with the training data. This theory may be adapted to allow for a number of errors on the training set by counting the permutations which leave no more errors on the left hand side. The resulting bound on the generalization error is given in the following theorem.

Theorem 8. *Let F be a hypothesis space having VC dimension h. For any probability distribution D on $X \times \{-1, +1\}$, with probability $1-\delta$ over N random examples S, any hypothesis $f \in F$ that makes k errors on thee training set S has error no more than:*

$$err_D(f) \leq \epsilon(N, F, \delta) = \frac{2k}{N} + \frac{4}{N}\left(h \log \frac{2eN}{h} + \log \frac{4}{\delta}\right),$$

provided $h \leq N$.

This theorem suggests that a learning algorithm for a hypothesis class F must minimize the number of training errors, since everything else in the bound has to be fixed by the choice of F. This is the inductive principle known as empirical risk minimization, discussed in Section 3.1.2, since it seeks to empirically minimize the value of the risk functional.

Let us extend the application of Theorem 8 to a nested sequence of hypothesis classes $F_1 \subset F_2 \subset \ldots \subset F_M$ by using δ/M, hence making the probability of any one of the bounds failing to hold to be less than δ. If an ideal hypothesis f_i with minimum training error is searched in each class F_i, then the number of errors k_i over the fixed training set S will satisfy the monotonic increasing inequality $k_1 \geq k_2 \geq \ldots \geq k_M$. Similarly, the VC dimension for each class H_i will also satisfy a monotonic increasing inequality $h_1 \geq h_2 \geq \ldots \geq h_M$. The bound of Theorem 8 may be used to choose the hypothesis h_i for which the bound is minimal, that is, the reduction in the number of errors outweighs the increase in capacity. Building a learning machine may therefore start with the selection of most appropriate hypothesis class H. This induction strategy, due to Vapnik [Vap82], is known as *structural risk minimization*.

3.2 Linear Learning Machines

In order to describe advanced non-linear learning machines such as Support Vector Machines, in this section we first explore the basic principles of linear learning machines. These simple concepts form the very basis of many supervised learning algorithms, and historically have initiated and motivated the field. Simple models such as the McCulloch and Pitts [MP43] model of an artificial neuron are perfect examples of this pioneer work, which was later followed by names such as Hebb [Heb49], Rosenblatt [Ros58], and Widrow and Hoff [WH60].

We start our discussion with a simple display of a linear classification model, equivalent to that known as the McCulloch and Pitts neuron. Later, we extend this model to its dual representation, which will be used in the description of SVMs themselves.

3.2.1 Linear Classification

Consider a binary classification problem with a real input domain $X \in \mathbb{R}^n$, and an output domain $Y \in \{-1, +1\}$. We shall use a classification function $f : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ such that the input $\boldsymbol{x}$ is assigned to the positive class if $f(\boldsymbol{x}) \geq 0$, and to the negative class otherwise. Considering that f is a linear function of $\boldsymbol{x} \in X$, we may

define f as:

$$\begin{aligned} f(\boldsymbol{x}) &= \langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b \\ &= \sum_{i=1}^{n} w_i x_i + b, \end{aligned} \tag{3.17}$$

where $(\boldsymbol{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters governing f. In the context of the McCulloch and Pitts neuron, $\boldsymbol{w}$ is known as the weight vector, and b as the threshold value. Since the output solely depends upon an inner product between two vectors exceeding or not the threshold value, this learning machine may only describe a linearly separable problem [DHS01, Hay94, BLC00]. Geometrically, f may be interpreted as defining a separating hyperplane of dimension $n-1$ between classes in a n-dimensional space.

Among the many possible ways to determine the optimal values for $(\boldsymbol{w}, b)$, that is, for training the learning machine, the first iterative algorithm to do so was Rosenblatt's Perceptron [Ros58]. The Perceptron algorithm works by starting with arbitrary initial values for the weight vector and the threshold value. Then, following repeated iterations, these values are adjusted based on the error values defined as the difference between the output value of the machine and the ideal value expected.

3.2.2 Dual Representation

Strang [Str86] exemplifies duality in a three-dimensional space, where he states that *the minimum distance to the points on a line* is equal to *the maximum distance to planes through that line*. We refer to the former representation of the same concept as its primal form, and to the latter as its dual form. Dual representations are most useful in optimization problems, where one often attempts to minimize a primal form while at the same time attempting to maximize a dual form. If the problem obeys the strong duality theorem, presented by Cristianini and Shawe-Taylor [CST00], both forms will equal once their optimum value are found, that is, there will be no duality gap on the solution.

Consider the linear classification machine and its training algorithm, the Perceptron, described in Section 3.2.1. Cristianini and Shawe-Taylor [CST00] demonstrate that, without loss of generality, if the initial weight vector and threshold value are the zero vector and zero, respectively, the final hypothesis output by the algorithm will be a linear combination of the training points:

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i, \tag{3.18}$$

where $\hat{\boldsymbol{w}}$ is the augmented weight vector $\boldsymbol{w}$ which incorporates the original threshold value b, and l is the number of iterations run by the algorithm. We may now

rewrite the decision function in Equation (3.17) using its dual representation:

$$\begin{aligned} f'(x) &= \langle w \cdot x \rangle + b \\ &= \langle \sum_{i=1}^{l} \alpha_i y_i x_i \cdot x \rangle + b \\ &= \sum_{i=1}^{l} \alpha_i y_i \langle x_i \cdot x \rangle + b \end{aligned} \tag{3.19}$$

Besides writing the decision function in its dual form, Cristianini and Shawe-Taylor [CST00] also describe the primal and dual forms of the Perceptron algorithm pseudo-code, along with an interesting analysis of bounds from learning theory.

3.3 Kernel-Induced Feature Spaces

This section describes what Cristianini and Shawe-Taylor [CST00] described as one of the main building blocks of Support Vector Machines, which is the use of kernel-induced feature spaces for translating the original input space. Haykin [Hay94] also highlights the concept, where he states that the basic idea of an SVM is summarized by the non-linear mapping of an input vector into a high-dimensional feature space that is hidden from both the input and output, where later an optimal separating hyperplane is sought.

3.3.1 Mapping into Feature Spaces

The difficulty of a learning task varies with the complexity of the target function to be learned, which in turn depends on how this function is represented. A common strategy in machine learning is the preprocessing of data, where its representation is changed as follows:

$$x = (x_1, \ldots, x_n) \rightarrow \phi(x) = (\phi_1(x), \ldots, \phi_N(x)). \tag{3.20}$$

It is equivalent to say that the input space X was mapped into a new space $F = \{\phi(x) | x \in X\}$. The space F is referred to as *feature space*, where the process of finding the most suitable representation for the data is known as *feature selection*. A suitable representation is characterized for making learning easier than in the original input space, for instance by mapping a non-linear problem into a linear one.

There are several approaches for feature selection. Common techniques seek, for instance, eliminating features which are irrelevant to the problem. Others, such as principal component analysis, attempt to map the original input space a into feature space with less dimensions, thus trying to avoid the so-called *curse of dimensionality* [CST00, Hay94, BLC00].

Consider the linear learning machine described in Section 3.2.1. We may rewrite Equation (3.17) incorporating mapping into a feature space:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi_i(\boldsymbol{x}) + b. \tag{3.21}$$

As shown in Section 3.2.2, a important property of linear learning machines is the ability to represent them in their dual form. Hence, we may rewrite Equation (3.19) incorporating mapping into a feature space, which is analogous to the dual form of Equation (3.21):

$$f'(\boldsymbol{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b. \tag{3.22}$$

We now define the *kernel* function, which is a way of directly computing the internal product $\langle \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \rangle$. Using a kernel function, we are thus able to transform a linear learning machine into a non-linear learning machine. A kernel is a function K, such that for all $\boldsymbol{x}, \boldsymbol{z} \in X$:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z}) \rangle. \tag{3.23}$$

There are interesting remarks about using a kernel function in learning machines. First, a consequence of the dual representation is that the dimension of the feature space need not affect the computation of the kernel. Furthermore, we do not need to known the underlying feature map to be able to learn in the feature space. Also, since feature vectors are not explicitly represented, potential computational problems inherent in evaluating the feature map are automatically overcome.

3.3.2 Building Kernels

Since the number of operations required to compute the inner product evaluated by a kernel function is not necessarily proportional to the number of features, using kernel functions as feature maps is often very interesting in terms of computational resources. The first step in order to use such mapping is to determine the most appropriate kernel function. Even though it may seem more intuitive to build a kernel function starting with building its corresponding feature space, in practice, the approach taken is to define the kernel function directly, hence implicitly defining the feature space. We now state the necessary conditions to ensure that a given function K is a kernel for some feature space.

First of all, a kernel function $K(\boldsymbol{x}, \boldsymbol{z})$ must be symmetric, that is:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z}) \rangle = \langle \boldsymbol{\phi}(\boldsymbol{z}) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \rangle = K(\boldsymbol{z}, \boldsymbol{x}) \tag{3.24}$$

Also, it should satisfy the inequalities that follow from the Cauchy-Schwarz inequality [1]:

$$\begin{aligned} K(\boldsymbol{x},\boldsymbol{z})^2 &= \langle \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z}) \rangle^2 \leq ||\boldsymbol{\phi}(\boldsymbol{x})||^2 ||\boldsymbol{\phi}(\boldsymbol{z})||^2 \\ &= \langle \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \rangle \langle \boldsymbol{\phi}(\boldsymbol{z}) \cdot \boldsymbol{\phi}(\boldsymbol{z}) \rangle^2 = K(\boldsymbol{x},\boldsymbol{x}) K(\boldsymbol{z},\boldsymbol{z}) \end{aligned} \tag{3.25}$$

Finally, it may obey Mercer's theorem, which provides a way for characterizing a kernel function, described as follows. Consider a finite input space $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_1\}$, and suppose $K(\boldsymbol{x},\boldsymbol{z})$ is a symmetric function on X. Consider the matrix $\boldsymbol{K}$:

$$\boldsymbol{k} = \left(K(\boldsymbol{x}_i,\boldsymbol{x}_j)\right)_{i,j=1}^{n}. \tag{3.26}$$

We have that $\boldsymbol{K}$ is symmetric and there is an orthogonal matrix $\boldsymbol{V}$ such that $\boldsymbol{K} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'$, where $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues λ_k of $\boldsymbol{K}$ and the corresponding eigenvectors $\boldsymbol{v}_t = (v_{ti})_{i=1}^{n}$, which are the columns of $\boldsymbol{V}$. Assuming all eigenvalues are non-negative, consider the feature mapping:

$$\phi : \boldsymbol{x}_i \rightarrow \left(\sqrt{\lambda_t} v_{ti}\right)_{t=1}^{n} \in \mathbb{R}^n, i = 1, \ldots, n. \tag{3.27}$$

We have now that:

$$\langle \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle = \sum_{t=1}^{n} \lambda_t v_{ti} v_{tj} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'_{ij} = \boldsymbol{K}_{ij} = K(\boldsymbol{x}_i,\boldsymbol{x}_j), \tag{3.28}$$

which implies that $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is a kernel function with the corresponding feature mapping ϕ. The proof that the eigenvalues λ_i must be non-negative is presented by Cristianini and Shawe-Taylor [CST00], given by contradiction based on their effect on the geometry of the space.

We may generalize an inner product in a Hilbert space by introducing weights α_i for each dimension, hence allowing infinite dimension in the feature vectors:

$$\langle \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z}) \rangle = \sum_{i=1}^{\infty} \lambda_i \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{z}) = K(\boldsymbol{x},\boldsymbol{z}) \tag{3.29}$$

According to Mercer's theorem, a continuous symmetric function $K(\boldsymbol{x},\boldsymbol{z})$ may be represented as:

$$\sum_{i=1}^{\infty} \lambda_i \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{z}), \tag{3.30}$$

[1] Please notice that during the text, whenever we write $||\boldsymbol{x}||$, we mean the 2-norm, or Euclidean norm, of vector $\boldsymbol{x}$. In selected contexts, we use $||\boldsymbol{x}||_1$ to refer to the 1-norm of vector $\boldsymbol{x}$.

where all λ_i are non-negative, which is equivalent to $K(\boldsymbol{x}, \boldsymbol{z})$ being an inner product in the feature space $F \supseteq \psi(X)$, where F is the l_2 space of all sequences $\Psi = (\psi_1, \ldots, \psi_i, \ldots)$ for which $\sum_{i=1}^{\infty} \lambda_i \psi_i^2 < \infty$.

This will induce a space defined by the feature vector, and as a consequence a linear decision function. We represent both primal and dual forms of this decision function as follows, where the term on left hand side is the primal and the term on the right hand side is the dual:

$$f(\boldsymbol{x}) = \sum_{i=1}^{\infty} \lambda_i \psi_i \phi_i(\boldsymbol{x}) + b = \sum_{j=1}^{l} \alpha_j y_j K(\boldsymbol{x}, \boldsymbol{x}_j) + b \tag{3.31}$$

Although we do not address the issue in detail here, the contribution from functional analysis comes from the study of the eigenvalue problem for integral equations of the form:

$$\int_K K(\boldsymbol{x}, \boldsymbol{z}) \phi(\boldsymbol{z}) d\boldsymbol{z} = \lambda \phi(\boldsymbol{x}), \tag{3.32}$$

where $K(\boldsymbol{x}, \boldsymbol{z})$ is a bounded, symmetric, and positive kernel function and X is a compact space. The details of this analysis are presented by Cristianini and Shawe-Taylor [CST00], along with many other interesting considerations about building kernels that are beyond the scope of this text, such as reproducing kernel Hilbert spaces, building kernels from kernels, and building kernels from features.

We next conclude the discussion by stating Mercer's theorem.

Theorem 9. *(Mercer) Let X be a compact subset of $\mathbb{R}^n$. Suppose K is a continuous symmetric function such that the integral operator $T_K : L_2(X) \rightarrow L_2(X)$,*

$$(T_K f)(\cdot) = \int_K K(\cdot, \boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x},$$

is positive, that is,

$$\int_{X \times X} K(\boldsymbol{x}, \boldsymbol{z}) f(\boldsymbol{x}) f(\boldsymbol{z}) d\boldsymbol{x} d\boldsymbol{z} \geq 0,$$

for all $f \in L_2(X)$. Then we can expand $K(\boldsymbol{x}, \boldsymbol{z})$ in a uniformly convergent series (on $X \times X$) in terms of T_k's eigen-functions $\phi_j \in L_2(X)$, normalized in such a way that $||\phi_j||_{L_2} = 1$, and positive associated eigenvalues $\alpha_j > 0$,

$$K(\boldsymbol{x}, \boldsymbol{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\boldsymbol{x}) \phi_j(\boldsymbol{z}).$$

Name	$K(\boldsymbol{x}, \boldsymbol{x}_i), i = 1, \ldots, N$	Comments
Polynomial	$(\boldsymbol{x} \cdot \boldsymbol{x}_i + 1)^p$	Power p is a parameter determined a priori.
Radial-basis function (RBF)	$\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}{2\sigma^2}\right)$	Width σ^2 is a parameter determined a priori.
Two-layer Perceptron	$\tanh(\beta_0 \boldsymbol{x} \cdot \boldsymbol{x}_i + \beta_1)$	Mercer's theorem is satisfied only for some values of β_0 and β_1.

Table 3.1: Summary of commonly used inner-product kernel functions.

3.3.3 Examples of Kernels

There are many possible functions that satisfy all requirements necessary to be used as kernels. One of the steps of building a Support Vector Machine, as will be described ahead in Section 3.4, is the selection of the most appropriate kernel for the problem in hand. Table 3.3.3 presents some of the most commonly used kernels in literature [Bur98, ABB01b, CST00].

There are many interesting works about creating customized kernels for problems whose input spaces are non-Euclidean. Some of these are referenced in Section 7.1, where we suggest their use among others in future follow-ups of this work.

3.4 Building Support Vector Machines

In this section we combine the concepts introduced in the previous sections of this chapter to build learning machines known as Support Vector Machines. We start by describing the principles of building a simple SVM for linearly separable problems. Then, using this basic machine as a basis, we incorporate inner-product kernels into the model to make learning possible on the corresponding induced feature spaces. As we shall see later, this learning machine based on finding a separating hyperplane in an induced feature space has the ability to tackle problems with non-linear input spaces. Completing our description of classification machines, we add slack variables to this separating hyperplane in order to cope with noisy data. Though the main focus of this chapter is to describe Support Vector Machines for classification problems, we conclude the section by briefly describing the use of SVMs in regression problems.

3.4.1 Linear Support Vector Machines

Consider the linear learning machine and the binary classification problem described in Section 3.2. Assume that the machine is trained on linearly separable data. Considering an input space with n dimensions, the geometrical interpretation for the decision function $f(\boldsymbol{x}) = \langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b$ is a separating hyperplane with $n-1$ dimensions. Notice that the points $\boldsymbol{x}$ lying on this hyperplane imply that $f(\boldsymbol{x}) = 0$, where $\boldsymbol{w}$ is normal to the hyperplane, $|b|/||\boldsymbol{x}||$ is the perpendicular distance from the hyperplane to the origin, and $||\boldsymbol{x}||$ is the Euclidean norm of $\boldsymbol{w}$. Let us define γ_+ as the shortest distance from the separating hyperplane to the closest positive example, similarly for γ_- and the closest negative example. We define the important concept of *geometric margin* of a separating hyperplane as $\gamma = \gamma_+ + \gamma_-$. For a linearly separable case, the support vector machine attempts to find the separating hyperplane with the largest margin possible.

Consider the margin of the decision function output as the *functional margin*. Cristianini and Shawe-Taylor [CST00] prove that the geometric margin is the functional margin of a normalized weight vector. Therefore, we may optimize the geometric margin by fixing the functional margin to 1 and minimizing the Euclidean norm of the weight vector. Consider a positive example $\boldsymbol{x}^+$ and a negative example $\boldsymbol{x}^-$. If the functional margin is fixed to 1, we have that:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}^+ \rangle + b = +1, \tag{3.33}$$

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}^- \rangle + b = -1. \tag{3.34}$$

We now compute the geometric margin γ by normalizing $\boldsymbol{w}$:

$$\begin{aligned} \gamma &= \frac{1}{2}\left(\left\langle \frac{\boldsymbol{w}}{||\boldsymbol{w}||} \cdot \boldsymbol{x}^+ \right\rangle - \left\langle \frac{\boldsymbol{w}}{||\boldsymbol{w}||} \cdot \boldsymbol{x}^- \right\rangle \right) \\ &= \frac{1}{2||\boldsymbol{w}||} \left(\langle \boldsymbol{w} \cdot \boldsymbol{x}^+ \rangle - \langle \boldsymbol{w} \cdot \boldsymbol{x}^- \rangle \right) \\ &= \frac{1}{||\boldsymbol{w}||}. \end{aligned} \tag{3.35}$$

Therefore, we achieve the maximum margin γ by minimizing the norm of the weight vector $||\boldsymbol{w}||$. Burges [Bur98] proves the same result by determining the maximum margin between two parallel hyperplanes that contain the closest positive and negative examples. The formulation of the problem is given as follows.

Proposition 1. *Given a linearly separable training sample:*

$$S = \left((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l) \right),$$

the hyperplane $(\boldsymbol{w}, b)$ *that solves the optimization problem:*

$$\begin{aligned} &\textit{minimize}_{\boldsymbol{w},b} && \langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle, \\ &\textit{subject to} && y_i \left(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \right) \geq 1, \\ & && i = 1, \ldots, l, \end{aligned}$$

realizes the maximal margin hyperplane with geometric margin $\gamma = 1/||\boldsymbol{w}||$.

In order to maximize the margin, we choose the approach of using the Lagrangian of the problem:

$$L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{l} \alpha_i [y_i (\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1], \tag{3.36}$$

where $\alpha_i \geq 0$ are the Lagrange multipliers. Aside from minimizing this primal form, given that this is a convex quadratic programming (QP) problem, we may also solve it by maximizing its dual form. We first differentiate L_P with respect to $\boldsymbol{w}$ and b:

$$\frac{\partial L_P(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{x}_i = \mathbf{0}, \tag{3.37}$$

$$\frac{\partial L_P(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = \sum_{i=1}^{l} y_i \alpha_i = 0. \tag{3.38}$$

Since these are equality constraints in the dual formulation, we can substitute them into the primal form from Equation (3.36) to obtain the dual form:

$$\begin{aligned} L_D(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \tfrac{1}{2}||\boldsymbol{w}||^2 - \textstyle\sum_{i=1}^{l} \alpha_i [y_i (\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1], \\ &= \tfrac{1}{2} \textstyle\sum_{i\,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle - \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle + \sum_{i=1}^{l} \alpha_i, \\ &= \textstyle\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle. \end{aligned} \tag{3.39}$$

Recall that after differentiating the primal form of the Lagrangian of the linear SVM in Equation (3.37), we may write the weight vector as:

$$\boldsymbol{w} = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{x}_i. \tag{3.40}$$

The formulation of the dual problem uses this definition of $\boldsymbol{w}$, and is given after Proposition 1.

Proposition 2. *Given a linearly separable training sample:*

$$S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l)),$$

and suppose the parameters $\boldsymbol{\alpha}'$ *solve the following quadratic optimization (QP) problem:*

$$\begin{aligned} &\textit{maximize} && L_D(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle, \\ &\textit{subject to} && \textstyle\sum_{i=1}^{l} y_i \alpha_i = 0, \\ &&& \alpha_i \geq 0, i = 1, \ldots, l. \end{aligned}$$

Then the weight vector $w' = \sum_{i=1}^{l} y_i \alpha_i' x_i$ *realizes the maximal margin hyperplane with geometric margin:*

$$\gamma = \frac{1}{||w||}.$$

Since the threshold value b is not present in the dual problem, we compute b' using the primal constraints:

$$b' = -\frac{\max_{y_i=-1}(\langle w' \cdot x_i \rangle) + \min_{y_i=1}(\langle w' \cdot x_i \rangle)}{2} \tag{3.41}$$

We now describe the Karush-Kuhn-Tucker (KKT) conditions for constrained optimization problems. All KKT conditions are satisfied at the solution of the constrained optimization problems for Support Vector Machines, since all constraints in this context are linear [Bur98]. The KKT conditions for the primal form L_P from Equation (3.36) are given below:

$$\frac{\partial L_P(w_v, b, \alpha)}{\partial w_v} = w_v - \sum_{i=1}^{l} \alpha_i y_i x_{iv} = 0 \quad v = 1, \ldots, n \tag{3.42}$$

$$\frac{\partial L_P(w_v, b, \alpha)}{\partial b} = -\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{3.43}$$

$$y_i(\langle x_i \cdot w \rangle + b) - 1 \geq 0 \quad i = 1, \ldots, l \tag{3.44}$$

$$\alpha_i \geq 0 \quad \forall i \tag{3.45}$$

$$\alpha_i(y_i(\langle w \cdot x_i \rangle + b) - 1) = 0 \quad \forall i \tag{3.46}$$

According to Burges [Bur98], who in turn relies on the results from Fletcher [Fle87], solving the SVM problem is equivalent to finding a solution to the KKT conditions, which are necessary and sufficient for w, b, and α to be a solution. For a complete reference of the subject, Bazaraa [BSS92] brings a thorough survey of most concepts used so far to model SVMs, such as constrained convex optimization problems, KKT conditions, Lagrangian duality, as well as several algorithms for solving optimization problems.

3.4.2 Non-Linear Support Vector Machines

After introducing the basic Support Vector Machine from Section 3.4.1 for solving linearly separable problems, we now move to the Support Vector Machine that Cristianini and Shawe-Taylor [CST00] considered the easiest to understand and yet the basis of many other more complex SVMs. This classifier relies on the same principles of achieving the maximal margin hyperplane as its linear predecessor, only it does so in a kernel-induced feature space, hence also allowing for non-linear problems.

Consider an optimal solution α', (w', b) for the dual problem in Proposition 2. According to the KKT complementarity condition in Equation (3.46), we have that:

$$\alpha_i'[y_i\left(\langle w' \cdot x_i\rangle + b'\right) - 1] = 0, \forall i. \tag{3.47}$$

This implies that the non-zero values of α_i are only those whose inputs x_i have functional margin one and therefore lie closest to the hyperplane. These input vectors are referred to as *support vectors*, since they are the ones that determine the solution of the machine. If we consider the dual representation of the separating hyperplane, we may write the optimal separating hyperplane disregarding non-support vectors:

$$\begin{array}{l} \sum_{i=1}^{l} y_i\alpha_i'\langle x_i \cdot x\rangle + b' \\ \sum_{i\in\{support\ vectors\}}^{l} y_i\alpha_i'\langle x_i \cdot x\rangle + b'. \end{array} \tag{3.48}$$

Also after the KKT complementarity condition in Equation (3.46), we may write the square of the Euclidean norm of the weight vector alternatively:

$$\begin{array}{rl} \langle w' \cdot w'\rangle & = \sum_{i,j=1}^{l} y_i y_j \alpha_i'\alpha_j'\langle x_i \cdot x_j\rangle \\ & = \sum_{j\in\{sv\}} y_j\alpha_j' \sum_{i\in\{sv\}} y_i\alpha_i'\langle x_i \cdot x_j\rangle \\ & = \sum_{j\in\{sv\}} \alpha_j'\left(1 - y_j b'\right) \\ & = \sum_{j\in\{sv\}} \alpha_j'. \end{array} \tag{3.49}$$

We may now extend Proposition 2 by writing the solution's geometric margin, first defined in Equation (3.35), in terms of the Lagrange multipliers:

$$\begin{array}{rl} \gamma & = \frac{1}{\|w\|} \\ & = \left(\sum_{j\in\{sv\}} \alpha_j'\right)^{-\frac{1}{2}} \end{array} \tag{3.50}$$

Consider a kernel function $K(x_i, x_j)$. If we replace the internal product of input vectors $\langle x_i \cdot x_j\rangle$ from Proposition 2 with this kernel function, we have that the separating hyperplane is sough in the feature space defined by $K(x_i, x_j)$. Notice that the geometric margin of the solution, defined solely in terms of the Lagrange multipliers, is not altered, despite the change in space. We synthesize these results on the following proposition.

Proposition 3. *Given a training sample:*

$$S = \left((x_1, y_1), \ldots, (x_l, y_l)\right),$$

where S is linearly separable in the feature space implicitly defined by the kernel $K(x_i, x_j)$, suppose the parameters α' and b' solve the following quadratic optimization

(QP) problem:

$$\begin{array}{ll} \textit{maximize} & L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j), \\ \textit{subject to} & \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & \alpha_i \geq 0, i = 1, \ldots, l. \end{array}$$

Then the decision rule given by sign $(f(\boldsymbol{x}))$, where $f(\boldsymbol{x}) = \sum_{i=1}^{l} y_i \alpha_i' K(\boldsymbol{x}_i, \boldsymbol{x}) + b'$, is equivalent to the maximal margin hyperplane in the feature space implicitly defined by the kernel $K(\boldsymbol{x}, \boldsymbol{z})$, and this hyperplane has geometric margin:

$$\gamma = \left(\sum_{j \in \{sv\}} \alpha_j' \right)^{-\frac{1}{2}}.$$

Notice that since the kernel $K(\boldsymbol{x}, \boldsymbol{z})$ must satisfy Mercer's theorem, the optimization problem is convex [CST00]. Hence, one of the properties required for a kernel function to define a feature space is that the maximal margin optimization problem has a unique solution, that is, that there is duality gap.

We now consider the expected generalization error for the Support Vector Machine described so far in this section. It is interesting to notice the effect that the number of support vectors of a given solution has over its expected generalization, where the fewer support vectors, the better generalization. According to Cristianini and Shawe-Taylor [CST00], this is consistent with the principle known as *Ockham's Razor*, which may be interpreted in this context as favoring more compact representations for classification functions as opposed to more complex ones[2]. We next present a theorem given by these same authors on SVM generalization bounds, after Theorem 6 from Vapnik and Chervonenkis.

Theorem 10. *Consider the thresholding real-valued linear functions L with unit weight vectors on an inner product space X. For any probability distribution D on $X \times \{-1, +1\}$, with probability $1 - \delta$ over l random examples S, the maximal margin hyperplane has error no more than:*

$$err_D(f) \leq \frac{1}{l - d} \left(d \log \frac{el}{d} + \log l\delta \right),$$

where d is the number of support vectors.

[2]Ockham's Razor is the principle proposed by William of Ockham in the fourteenth century: *Pluralitas non est ponenda sine neccesitate*, which translates to "entities should not be multiplied unnecessarily".

3.4.3 Soft Margin Optimization

Even though the non-linear maximal margin classifier described in Section 3.4.2 is the basis for many more advanced Support Vector Machines, its direct applicability suffers from its lack of ability to handle noisy data, which is almost always the case in real-world problems. Since the maximal margin classifiers outputs hypothesis with zero training error, the machine's behavior is dictated by its training data, which in turn may include outliers and idiosyncrasies that jeopardize the quality of the final solution. We now explore the concept of *margin distributions*, where we add slack variables to the maximal margin classifier to allow for its margin constraints to be violated.

Recall from Proposition 1 the primal optimization problem for the maximal margin case. We introduce slack variables that allow the margin constraints to be violated in order to optimize the margin slack vector:

$$\begin{array}{ll} \text{minimize}_{w,b} & \langle w \cdot w \rangle, \\ \text{subject to} & y_i \left(\langle w_i \cdot x_i \rangle + b \right) \geq 1 - \xi_i, i = 1, \ldots, l, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{array} \tag{3.51}$$

Cristianini and Shawe-Taylor [CST00] demonstrate the bounds on the generalization error in terms of the 2-norm of the margin slack vector, also called 2-norm *soft margin*, which contains ξ_i scaled by the norm of the weight vector. Their result suggests an optimal choice for C in the optimization problem, as follows.

$$\begin{array}{ll} \text{minimize}_{\xi,w,b} & \langle w \cdot w \rangle + C \sum_{i=1}^{l} \xi_i^2, \\ \text{subject to} & y_i \left(\langle w \cdot x_i \rangle + b \right) \geq 1 - \xi_i, i = 1, \ldots, l, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{array} \tag{3.52}$$

The same approach may be adapted for the 1-norm case, where the optimization problem minimizes another combination of the norm of weights and the 1-norm of slack variables. Once there is a value of C that corresponds to the optimal choice of $||w||$, C will give the optimal bound as it will correspond to achieving the minimum of $||\xi_i||_1$ with the given value of $||w||$.

$$\begin{array}{ll} \text{minimize}_{\xi,w,b} & \langle w \cdot w \rangle + C \sum_{i=1}^{l} \xi_i, \\ \text{subject to} & y_i \left(\langle w \cdot x_i \rangle + b \right) \geq 1 - \xi_i, i = 1, \ldots, l, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{array} \tag{3.53}$$

Consider the case for 2-norm soft margins. We have that the primal Lagrangian for the problem in Equation (3.52) is:

$$L_P(w, b, \xi, \alpha) = \frac{1}{2}\langle w \cdot w \rangle + \frac{C}{2} \sum i = 1^l \xi_i^2 - \sum_{i=1}^{l} \alpha_i [y_i \left(\langle w_i \cdot x_i \rangle + b \right) - 1 + \xi_i]. \tag{3.54}$$

If we calculate the partial derivatives for L_P we have:

$$\frac{\partial L_P(w,b,\xi,\alpha)}{\partial w} = w - \sum_{i=1}^{l} y_i \alpha_i x_i = 0, \tag{3.55}$$

$$\frac{\partial L_P(w,b,\xi,\alpha)}{\partial \xi} = C\xi - \alpha = 0, \tag{3.56}$$

$$\frac{\partial L_P(w,b,\xi,\alpha)}{\partial b} = \sum_{i=1}^{l} y_i \alpha_i = 0. \tag{3.57}$$

Since these are equality constraints in the dual formulation, we can substitute them into the primal form from Equation (3.54) to obtain the dual form:

$$L_D(w,b,\xi,\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle - \frac{1}{2C} \langle x_i \cdot x_j \rangle. \tag{3.58}$$

Hence, maximizing the dual representation above is equivalent to Proposition 4, where δ_{ij} is the Kronecker δ defined to be 1 if $i = j$ and 0, and the corresponding KKT complementarity conditions are:

$$\alpha_i [y_i (\langle w \cdot x_i \rangle + b) - 1 + \xi_i] = 0, i = 1, \ldots, l. \tag{3.59}$$

Proposition 4. *Consider classifying a training sample:*

$$S = ((x_1, y_1), \ldots, (x_l, y_l)),$$

using the feature space implicitly defined by the kernel $K(x,z)$, and suppose the parameters α' solve the following quadratic optimization (QP) problem:

$$\begin{array}{ll} \textit{maximize} & L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \left(K(x_i, x_j) + \frac{1}{C} \delta_{ij} \right), \\ \textit{subject to} & \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & \alpha_i \geq 0, i = 1, \ldots, l. \end{array}$$

Let $f(x) = \sum_{i=1}^{l} y_i \alpha_i' K(x_i, x) + b'$, where b' is chosen so that $y_i f(x_i) = 1 - \alpha_i'/C$ for any i with $\alpha_i' \neq 0$. Then the decision rule given by sign $(f(x))$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(x,z)$ which solves the optimization problem (3.52), where the slack variables are defined relative to the geometric margin:

$$\gamma = \left(\sum_{j \in \{sv\}} \alpha_j' - \frac{1}{C} \langle \alpha' \cdot \alpha' \rangle \right)^{-\frac{1}{2}}.$$

Proof. The value of b' is chosen using the relation $\alpha_i = C\xi_i$ and by reference to the primal constraints which by the KKT complementarity conditions:

$$\alpha_i[y_i\left(\langle w_i \cdot x_i\rangle + b\right) - 1 + \xi_i] = 0, i = 1, \ldots, l,$$

must be equalities for non-zero α_i. It remains to compute the norm of w' which defines the size of the geometric margin.

$$\begin{aligned}\langle w' \cdot w'\rangle &= \sum_{i,j=1}^{l} y_i y_j \alpha_i' \alpha_j' K\left(x_i \cdot x_j\right) \\ &= \sum_{j\in\{sv\}} y_j \alpha_j' \sum_{i\in\{sv\}} y_i \alpha_i' K\left(x_i \cdot x_j\right) \\ &= \sum_{j\in\{sv\}} \alpha_j' \left(1 - \xi_i' - y_j b'\right) \\ &= \sum_{j\in\{sv\}} \alpha_j' - \sum_{j\in\{sv\}} \alpha_j' \xi_j' \\ &= \sum_{j\in\{sv\}} \alpha_j' - \frac{1}{C}\langle \alpha' \cdot \alpha'\rangle.\end{aligned}$$

□

According to Cristianini and Shawe-Taylor [CST00], this quadratic optimization problem is equivalent to a simple change in the function:

$$K'\left(x, z\right) = K\left(x, z\right) + \frac{1}{C}\delta_x\left(z\right). \tag{3.60}$$

Consider now the case for 1-norm soft margin. We have that the Lagrangian for the 1-norm soft margin optimization problem is:

$$L_P\left(w, b, \xi, \alpha, r\right) = \frac{1}{2}\langle w \cdot w\rangle + 2\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i[y_i\left(\langle w \cdot x_i\rangle + b\right) - 1 + \xi_i] - \sum_{i=1}^{l} r_i\xi_i, \tag{3.61}$$

with $\alpha_i \geq 0$ and $r_i \geq 0$. We then calculate the partial derivatives for L_P:

$$\frac{\partial L_P\left(w, b, \xi, \alpha, r\right)}{\partial w} = w - \sum_{i=1}^{l} y_i\alpha_i x_i = 0, \tag{3.62}$$

$$\frac{\partial L_P\left(w, b, \xi, \alpha, r\right)}{\partial \xi_i} = C - \alpha_i - r_i = 0, \tag{3.63}$$

$$\frac{\partial L_P\left(w, b, \xi, \alpha, r\right)}{\partial b} = \sum_{i=1}^{l} y_i\alpha_i = 0. \tag{3.64}$$

We substitute these relations into the primal form from Equation (3.61) to obtain the dual form:

$$L_D\left(w, b, \xi, \alpha, r\right) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j\rangle, \tag{3.65}$$

which is identical to that for the maximal margin, except that the constraints $C - \alpha_i - r_i = 0$ and $r_i \geq 0$ enforce $\alpha_i \leq C$, while $\xi_i \neq 0$ only if $r_i = 0$ and therefore $\alpha_i = C$. The corresponding KKT complementarity conditions are therefore:

$$\begin{array}{ll} \alpha_i[y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 + \xi_i] = 0, & i = 1, \ldots, l, \\ \xi_i(\alpha_i - C) = 0, & i = 1, \ldots, l. \end{array} \tag{3.66}$$

Notice that these KKT conditions imply that there may only be non-zero slack variables if $\alpha_i = C$. Points with non-zero slack variables have geometric margin less than $1/||\boldsymbol{w}||$, where points for which $0 < \alpha_i < C$ lie at distance $1/||\boldsymbol{w}||$ from the separating hyperplane. Proposition 5 formalizes the problem.

Proposition 5. *Consider classifying a training sample:*

$$S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l)),$$

using the feature space implicitly defined by the kernel $K(\boldsymbol{x}, \boldsymbol{z})$, and suppose the parameters $\boldsymbol{\alpha}'$ solve the following quadratic optimization problem:

$$\begin{array}{ll} \textit{maximize} & L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j), \\ \textit{subject to} & \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & C \geq \alpha_i \geq 0, i = 1, \ldots, l. \end{array}$$

Let $f(\boldsymbol{x}) = \sum_{i=1}^{l} y_i \alpha_i' K(\boldsymbol{x}_i, \boldsymbol{x}) + b'$, where b' is chosen so that $y_i f(\boldsymbol{x}_i) = 1$ for any i with $C > \alpha_i' > 0$. Then the decision rule given by sign $(f(\boldsymbol{x}))$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\boldsymbol{x}, \boldsymbol{z})$ which solves the optimization problem (3.53), where the slack variables are defined relative to the geometric margin:

$$\gamma = \left(\sum_{j \in \{sv\}} y_i y_j \alpha_i' \alpha_j' K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right)^{-\frac{1}{2}}.$$

Proof. The value of b' is chosen using the KKT complementarity which imply that if $C > \alpha_i' > 0$ both $\xi_i' = 0$ and:

$$\alpha_i[y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 + \xi_i] = 0.$$

The norm of $\boldsymbol{w}'$ is clearly given by the expression:

$$\begin{array}{rl} \langle \boldsymbol{w}' \cdot \boldsymbol{w}' \rangle & = \sum_{i,j=1}^{l} y_i y_j \alpha_i' \alpha_j' K(\boldsymbol{x}_i \cdot \boldsymbol{x}_j) \\ & = \sum_{j \in \{sv\}} \sum_{i \in \{sv\}} y_i y_j \alpha_i' \alpha_j' K(\boldsymbol{x}_i \cdot \boldsymbol{x}_j). \end{array}$$

□

Notice that all α_i are upper bounded by C, which brings up the name *box constraint* since the vector α must lie inside a box with length C in the positive orthant. C is hence a regularization parameter that adjusts the trade-off between accuracy and generalization. Another interesting outcome is that this constraint also ensures that the feasible region is bounded and hence the primal always has a non-empty feasible region. The concepts around box constraints with 1-norm soft margins are explicitly explored by the Sequential Minimization Optimization algorithm, later described in Section 3.5.3.

Pontil and Verri [PV98] explored the effects of the regularization parameter C over the solution of the SVM. They concluded that some support vectors whose Lagrange multipliers are smaller than C, referred to as *margin vectors*, play a distinct role in the composition of the solution. Given the optimal separating hyperplane described by a decision surface, this hyperplane can be written as a sum of two orthogonal terms, the first depending only on the margin vectors, the second proportional to the regularization parameter. Therefore, this result from Pontil and Verri enables us to better predict the effects of changes in C over the solution.

3.4.4 Support Vector Regression

Even though the focus of this work is the use of Support Vector Machines in classification problems, one of the possible future works to follow is the use of the same hybrid techniques developed here in regression problems, as described in Section 7.7. The approach of using support vectors, so far only described for classification problems, can be adapted to regression problems as well. This adapted approach is based on the same concepts of learning a non-linear function with a linear learning machine in a kernel-induced feature space. For complete references on using SVMs in regression problems, please refer to Burges [Bur98] or Cristianini and Shawe-Taylor [CST00].

Like the classification approach, in regression we seek to optimize the generalization bounds according to a loss function that ignores errors within a certain distance of the correct value. This loss function is referred to as ϵ-insensitive loss function, where there are variables ξ_i that measure the error for each of the training points, and $\xi_i = 0$ if vector i is within the so-called ϵ-insensitive band. The are many choices for this function, for instance the linear ϵ-insensitive loss and the quadratic ϵ-insensitive loss, which correspond to the 1-norm and 2-norm of the loss vector, respectively.

Lets consider first the quadratic ϵ-insensitive loss case. We skip the description of the function itself and directly describe the formulation of the corresponding

primal optimization problem:

$$\begin{array}{ll} \text{minimize} & \|w\|^2 + C\sum_{i=1}^{l}\left(\xi_i^2 + \hat{\xi}_i^2\right), \\ \text{subject to} & (\langle w \cdot x_i\rangle + b) - y_i \leq \epsilon + \xi_i, i = 1, \ldots, l, \\ & y_i - (\langle w \cdot x_i\rangle + b) \leq \epsilon + \hat{\xi}_i, i = 1, \ldots, l, \\ & \xi, \hat{\xi}_i \geq 0, i = 1, \ldots, l, \end{array} \tag{3.67}$$

where two slack variables are introduced, one for exceeding the target value by more than ϵ, and another for being more than ϵ below the target. The dual problem is derived using the same approach as in the classification case, only now taking into account that $\xi_i\hat{\xi}_i = 0$ and therefore that $\alpha_i\hat{\alpha}_i$ holds for the corresponding Lagrange multipliers:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{l} y_i\left(\hat{\alpha}_i - \alpha_i\right) - \epsilon\sum_{i=1}^{l}\left(\hat{\alpha}_i + \alpha_i\right) \\ & \quad - \frac{1}{2}\sum_{i,j=1}^{l}\left(\hat{\alpha}_i - \alpha_i\right)\left(\hat{\alpha}_i - \alpha_j\right)\left(\langle x_i \cdot x_j\rangle + \frac{1}{C}\delta_{ij}\right) \\ \text{subject to} & \sum_{i=1}^{l}\left(\hat{\alpha}_i - \alpha_i\right) = 0, \\ & \hat{\alpha}_i \geq 0, \alpha_i \geq 0, i = 1, \ldots, l, \end{array} \tag{3.68}$$

where the KKT complementarity conditions are:

$$\begin{array}{ll} \alpha_i\left(\langle w \cdot x_i\rangle + b - y_i - \epsilon - \xi_i\right) = 0, & i = 1, \ldots, l, \\ \hat{\alpha}_i\left(y_i - \langle w \cdot x_i\rangle - b - \epsilon - \hat{\xi}_i\right) = 0, & i = 1, \ldots, l, \\ \xi_i\hat{\xi}_i = 0, \alpha_i\hat{\alpha}_i, & i = 1, \ldots, l. \end{array} \tag{3.69}$$

Consider now the linear ϵ-insensitive loss case. We may write the the corresponding primal optimization problem as follows:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\left(\xi_i + \hat{\xi}_i\right), \\ \text{subject to} & (\langle w \cdot x_i\rangle + b) - y_i \leq \epsilon + \xi_i, i = 1, \ldots, l, \\ & y_i - (\langle w \cdot x_i\rangle + b) \leq \epsilon + \hat{\xi}_i, i = 1, \ldots, l, \\ & \xi, \hat{\xi}_i \geq 0, i = 1, \ldots, l. \end{array} \tag{3.70}$$

The corresponding dual problem is:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{l} y_i\left(\hat{\alpha}_i - \alpha_i\right) - \epsilon\sum_{i=1}^{l}\left(\hat{\alpha}_i + \alpha_i\right) \\ & \quad - \frac{1}{2}\sum_{i,j=1}^{l}\left(\hat{\alpha}_i - \alpha_i\right)\left(\hat{\alpha}_j - \alpha_j\right)\langle x_i \cdot x_j\rangle \\ \text{subject to} & \sum_{i=1}^{l}\left(\hat{\alpha}_i - \alpha_i\right) = 0, \\ & \hat{\alpha}_i \leq C, \alpha_i \geq 0, i = 1, \ldots, l, \end{array} \tag{3.71}$$

where the KKT complementarity conditions are:

$$\begin{array}{ll} \alpha_i\left(\langle w \cdot x_i\rangle + b - y_i - \epsilon - \xi_i\right) = 0, & i = 1, \ldots, l, \\ \hat{\alpha}_i\left(y_i - \langle w \cdot x_i\rangle - b - \epsilon - \hat{\xi}_i\right) = 0, & i = 1, \ldots, l, \\ \xi_i\hat{\xi}_i = 0, \alpha_i\hat{\alpha}_i, & i = 1, \ldots, l. \\ \left(\alpha_i - C\right)\xi_i = 0, \left(\hat{\alpha}_i - C\right)\hat{\xi}_i = 0, & i = 1, \ldots, l. \end{array} \tag{3.72}$$

Notice that if we replace $\langle x_i \cdot x_j \rangle$ by $K(x_i, x_j)$ in the dual forms for both the quadratic and linear ϵ-insensitive loss cases above, we have machines that learn in the corresponding kernel induced feature space. This is the same technique used before to map input spaces to kernel induced feature spaces in classification problems.

There are different approaches for support vector regression other than using ϵ-insensitive loss functions. One these approaches explores the case where $\epsilon = 0$ for the quadratic loss function, which corresponds to least squares regression with a weight decay factor. This approach is referred to as ridge regression, and it is thoroughly described by many authors including Cristianini and Shawe-Taylor [CST00].

3.5 Training Methods

In previous sections, we described how the problem of training a Support Vector Machine may be reduced to a constrained optimization problem. Solving constrained optimization problems has been widely studied in literature, specially for the convex case. In this section we describe methods for training SVMs, starting with well known generic optimization methods, shifting to enhanced methods that take advantage of specific SVM properties.

Despite having a wide range of methods for training SVMs from optimization literature, the problem has particular characteristics that suggest the use of customized approaches. For instance, large training sets impose serious resource restrictions due to the quadratic complexity of the problem in space. The dimensionality of the kernel matrix grows with the square of the training set size, thus potentially generating a substantial number of variables to be optimized. Furthermore, SVM problems are rather sparse, therefore computational resource consumption and performance may be substantially improved if the algorithms being used are able to deal efficiently with sparseness.

3.5.1 General Techniques

We shall now describe some techniques based on generic optimization methods for solving the problem of training classification Support Vector Machines. Consider the problem stated in Proposition 5, where $C = \infty$ gives the hard margin case, an adaptation of the kernel as described in Equation (3.60) gives 2-norm soft margin optimization, and $C < \infty$ gives the 1-norm soft margin case. We also have that the Karush-Kuhn-Tucker complementarity conditions for the 2-norm and 1-norm soft margin cases are given by Equations (3.59) and (3.66), respectively. Recall that the

convexity of the problem ensures that a solution can always be found efficiently, where satisfying all KKT conditions is necessary and sufficient for finding a solution.

There are many numerical approaches to this problem, where algorithms iteratively increase the value of the dual objective function without leaving the feasible region, until a stop criterion is met. Cristianini and Shawe-Taylor [CST00] describe three different stopping criteria derived from the properties of the problem, as we describe next.

1. One of the simplest stop criterion possible is monitoring the growth of the dual objective function. Since there are no local maxima in the problem, the function will reach its maximum value at the solution. Therefore, one may monitor the fractional rate of increase between iterations to fall below a given threshold, and hence stop the iterative training.

2. Since meeting all KKT conditions is necessary and sufficient for a solution, these conditions provide a natural stopping criteria. For the 1-norm soft margin case, we have that:

$$\begin{array}{c} 0 \leq \alpha_i \leq C, \\ y_i f(\boldsymbol{x}_i) \quad \begin{cases} \geq 1 & \text{for points with } \alpha_i = 0, \\ = 1 & \text{for points with } 0 < \alpha_i < C, \\ \leq 1 & \text{for points with } \alpha_i = C. \end{cases} \end{array} \tag{3.73}$$

   For the 2-norm soft margin case, we have that:

$$\begin{array}{c} \alpha_i \geq 0, \\ y_i f(\boldsymbol{x}_i) \quad \begin{cases} \geq 1 & \text{for points with } \alpha_i = 0, \\ = 1 - \alpha_i / C & \text{for points with } \alpha_i > 0, \end{cases} \end{array} \tag{3.74}$$

   where slack variables are implicitly defined by $\xi_i = \alpha_i / C$.

3. Considering that the duality gap at the solution is zero in convex quadratic optimization problems, one may monitor progress by assessing the difference between the primal and dual objective functions. The following ratio may be computed and checked against a threshold value as it converges toward zero:

$$\frac{L_P - L_D}{L_P - 1}, \tag{3.75}$$

   where L_P is the primal objective function and L_D is the dual objective function.

The simplest method for achieving a numerical solution for a convex optimization problem is using gradient ascent, also known as steepest ascent algorithm. It works by iteratively shifting the solution toward the direction of the steepest ascent by steps s of fixed length, factored using a learning rate η, similarly to many neural network training algorithms [BLC00, Hay94]. The update of Lagrange multipliers at time t is given by:

$$\alpha_i^{t+1} = \alpha_i^t + \eta_i s_i. \tag{3.76}$$

Though very simple to implement, this approach suffers from the general problems associated with naive gradient ascent, such as convergence speed and oscillation before convergence. Cristianini and Shawe-Taylor [CST00] introduce the topic of training SVMs with a thorough description of this method, which is also explored by Almeida et al. [ABB01b].

Another iterative method similar to gradient ascent is that of projected conjugate gradients (PCG). Although the concepts behind both methods are essentially the same, where the solution is achieved by climbing the convex region up to its maximum, the direction of each step is computed in a more sophisticated and efficient way.

Yet another iterative algorithm was proposed by Mangasarian and Musicant [MM98, MM99], called Successive Over Relaxation (SOR). It consists of relaxing some of the equality constraints in the problem formulation, leaving it with only inequality constraints. One of the interesting properties of SOR is its ability to deal with massive datasets [MM99, BMM99]. According to Cristianini and Shawe-Taylor [CST00], SOR is equivalent to stochastic gradient ascent combined with sample selection heuristics such as those described by Platt [Pla98a].

3.5.2 Decomposition and Chunking

One of the drawbacks of the iterative algorithms described in Section 3.5.1 is their computational resource requirements, more specifically the necessity to store the data in the form of a kernel matrix, thus implying quadratic complexity in space. New algorithms emerged attempting to make Support Vector Machine training efficient even for large data sets. One of the techniques derived from this effort is the so-called *active set* or *working set* method in optimization, which consists of discarding inactive constraints to simplify the problem, where selection heuristics are used to determine which constraints are active and which are not.

One of these heuristics is called *chunking*, where the algorithm starts by training the SVM using an arbitrary subset of the data, also called active or working set. Chunking is analogous to the *divide and conquer* strategy for algorithms, where a

problem is broken down into smaller more tractable sub-problems. The SVM is repeatedly trained with new data subsets, where each subset of chosen as to keep the support vectors from the previous iterations together with the points that most violate KKT conditions. The algorithm iterates until a stop criterion is found, where the values of the Lagrange multipliers yield the final solution. This algorithm does not change the complexity requirements of SVM training, where the kernel matrix must only be stored in memory for the working set currently being processed.

Inspired by chunking techniques, Joachims [Joa98a] introduced SVM^{light}, which selects the Lagrange multipliers of its active set positioned on the steepest feasible descent direction, determined based on a first order approximation of the objective function. Chunking techniques do not have theoretical convergence proofs, though in practice, they are remarkably efficient for dealing with large data sets.

Another more advanced heuristic based on chunking is sometimes referred to as *decomposition*, introduced by Osuna et al. [OFG97]. Instead of focusing in finding the optimal data set over which the optimizer must be executed, decomposition attempts to optimize the global problem by processing small chunks of data at a time. Osuna et al. proved that the algorithm asymptotically converges if at least one point violating KKT conditions is added to the working set. Osuna et al. also suggest that the number of Lagrange multipliers should be fixed, where for each multiplier added to the working set, another has to be removed.

3.5.3 Sequential Minimal Optimization

We now describe one the key algorithms used in this work, the so-called *Sequential Minimal Optimization* (SMO). SMO is due to John Platt [Pla98a, Pla98b], who extended the ideas by Osuna et al. [OFG97] about decomposing a quadratic programming (QP) optimization problem into a series of smaller problems in order to decrease the resources needed to execute the algorithm. Platt took this concept to the extreme, where the original problem is divided in the smallest possible QP problems, which allows them to be solved analytically rather than iteratively. This way, SMO requires a linear amount of memory and a linear-to-quadratic time in the training set size, whereas other chunking methods, such as projected conjugate gradients chunking, require linear-to-cubic times [Pla98a]. The efficiency of SMO compared to previous SVM training algorithms is certainly one of the factors that triggered the recent interest that SVMs have received in different application domains, often outperforming many other traditional techniques.

Recall the problem stated in Proposition 5, where $C = \infty$ gives the hard margin case, an adaptation of the kernel as described in Equation (3.60) gives 2-norm soft margin optimization, and $C < \infty$ gives the 1-norm soft margin case. The KKT

conditions for both 2-norm and 1-norm soft margin cases are given by Equations (3.59) and (3.66), respectively. We have that Lagrange multipliers α_i and α_j must obey a linear equality constraint. Therefore, it follows that the smallest possible QP sub-problem derived from the original problem must optimize two Lagrange multipliers at a time. The main advantage of SMO is that solving the problem for two Lagrange multipliers can be done analytically, thus avoiding expensive numerical iterative algorithms. Also, since the maximum number of multipliers in the working set is two, only a two-by-two matrix is required to store the kernel matrix.

The SMO algorithm may be broken down into three parts: an analytical method for solving the optimization problem for two Lagrange multipliers, a selection heuristic to choose which two Lagrange multipliers to optimize, and a method for computing the threshold value of the SVM. We now describe each one of these components.

Consider the linear constraint relating any two Lagrange multipliers α_1 and α_2 chosen to be optimized such that $\sum_{i=1}^{l} \alpha_i y_i = 0$ holds. It follows that α_1 and α_2 are constrained in a box defined by $0 \leq \alpha_1, \alpha_2 \leq C$ according to this condition, where α_1 and α_2 must lie in a diagonal line such that:

$$\alpha_1 y_1 + \alpha_2 y_2 = \text{constant}. \tag{3.77}$$

This geometric display offers another means for explaining the minimum number of Lagrange multipliers to be selected, where the algorithm would not be able to fulfill all constraints if only one multiplier was to be updated at a time.

Without loss of generality, the algorithm first computes the second Lagrange multiplier α_2 and computes the ends of the diagonal line segment in terms of α_2. If $y_1 \neq y_2$, then the following bounds apply to α_2:

$$L = \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right), H = \min\left(C, C + \alpha_2^{old} - \alpha_1^{old}\right). \tag{3.78}$$

If $y_1 = y_2$, then these bounds become:

$$L = \max\left(0, \alpha_1^{old} + \alpha_2^{old} - C\right), H = \min\left(C, \alpha_1^{old} + \alpha_2^{old}\right). \tag{3.79}$$

Let E_i be the difference between the machine's output $f(\boldsymbol{x})$ and the target classification on training points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, such that:

$$E_i = f(\boldsymbol{x}_i) - y_i = \left(\sum_{j=1}^{l} \alpha_j y_j K(\boldsymbol{x}_j, \boldsymbol{x}_i) + b\right) - y_i, i = 1, 2, \tag{3.80}$$

where $\phi(\cdot)$ is the mapping to the feature space corresponding to the kernel function $K(\cdot, \cdot)$. Consider also the second derivative of the objective function along the

diagonal line:

$$\eta = K(\boldsymbol{x}_1, \boldsymbol{x}_1) + K(\boldsymbol{x}_2, \boldsymbol{x}_2) - 2K(\boldsymbol{x}_1, \boldsymbol{x}_2) = ||\phi(\boldsymbol{x}_1) - \phi(\boldsymbol{x}_2)||^2. \tag{3.81}$$

We must now compute the location of the constrained maximum of the objective function while allowing only two Lagrange multipliers to change. Under normal circumstances, there will be a maximum lying along the direction of the linear equality constraint and $\eta < 0$. In this case, we have that the following unconstrained maximum:

$$\alpha_2^{new} = \alpha_2^{old} - \frac{y_2(E_1 - E_2)}{\eta}. \tag{3.82}$$

Next, we find the constrained maximum by clipping the unconstrained maximum to the ends of the line segment:

$$\alpha_2^{new,clipped} = \begin{cases} H & \text{if } \alpha_2^{new} \geq H \\ \alpha_2^{new} & \text{if } L < \alpha_2^{new} < H \\ L & \text{if } \alpha_2^{new} \leq L \end{cases}. \tag{3.83}$$

We now compute the first Lagrange multiplier, α_1, based on $\alpha_2^{new,clipped}$:

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 \left(\alpha_2^{old} - \alpha_2^{new,clipped}\right). \tag{3.84}$$

The second derivative η will not be negative under normal circumstances, though SMO does not fail should this happen. Notice that η may be equal to zero if more than one training point has the same input vector.

Cristianini and Shawe-Taylor [CST00] present the analytic solution for two Lagrange multipliers as a theorem, which they prove based on the partial derivative of the objective function with respect to α_2 and the geometric properties of the problem.

Next, we describe the selection heuristics used by SMO to determine the pairs of Lagrange multipliers to optimize at each iteration. According to Osuna et al. [OFG97], a chunking algorithm will asymptotically converge if points that violate KKT conditions are always added to the working set. It follows that the algorithm tends to converge faster if the points that most violate KKT conditions are selected, since they have larger contributions to the gap between the current and optimal solutions [CST00]. Although always selecting the two points that most violate KKT conditions ensures that the algorithm will converge with less iterations, evaluating these conditions for all points is computationally expensive. Therefore, selection heuristics must be used to select two points with a reasonable contribution toward the solution, though yet computationally cheap to be sought.

Platt uses two nested heuristics for selecting Lagrange multipliers, where the first heuristic selects α_1, and the second selects α_2 based on the first choice.

First Choice Heuristic The first choice heuristic provides the outer loop of the SMO algorithm, which scans through all points determining those that violate KKT conditions. When found, these points are selected for optimization and handled over to the second choice heuristic. In order to speed up training, the algorithm only scans points whose Lagrange multipliers α_i satisfy $0 < \alpha_i < C$, that is, those points not on the boundary of the feasible solution. Only after all such points do not violate KKT conditions, the algorithm resumes scanning through the complete list of points, including those on the boundary of the feasible solution.

Second Choice Heuristic The selection of the second Lagrange multiplier must take into consideration the selection of the first, where the pair that yields the greater contribution toward the solution is ideal. In order to quickly determine this pair, SMO evaluates the step size $|E_1 - E_2|$ and selects the pair with its largest value, where error values $E_i \; \forall i \mid 0 < \alpha_i < C$ are cached by the algorithm to avoid repeated evaluations of the kernel function. It follows that if E_1 is negative, SMO chooses an example with minimum error E_2, and if E_1 is positive, it choose an example with maximum E_2. If this choice of α_2 fails to deliver a significant increase in the dual objective, SMO attempts scans through every non-bound example, and if this new choice still fails to provide a relevant contribution, SMO searches through the complete set.

Notice that all condition checks against KKT conditions are made within a margin ϵ. According to Platt [Pla98a], this loose checking of KKT conditions inhibit the incidence of numerical errors, empirically shortening the convergence time depending on its value.

The error cache used in the heuristics is kept for all values E_i whose Lagrange multipliers lie within the feasible region, that is, $0 < \alpha_i < C$. When a non-bound point is selected for optimization, that is, a point not lying over the bounds of the feasible region, its error value is set to zero. After each optimization step, the error values for every non-bound multipliers α_k are updated as follows:

$$\begin{aligned} E_k^{new} = E_k^{old} &+ y_1 \left(\alpha_1^{new} - \alpha_1^{old}\right) K\left(\boldsymbol{x}_1, \boldsymbol{x}_k\right) \\ &+ y_2 \left(\alpha_2^{new,clipped} - \alpha_2^{old}\right) K\left(\boldsymbol{x}_2, \boldsymbol{x}_k\right) + b^{old} - b^{new}, \end{aligned} \tag{3.85}$$

where b is the threshold value of the SVM. Finally, after solving for all Lagrange multipliers in the optimization problem, we must determine how to compute b. After every step, b is re-evaluated so that KKT conditions are fulfilled for both optimized

example selections. A threshold value b_1 is valid when α_1^{new} is not at the bounds, since it forces the SVM to output y_1 given the input vector $\boldsymbol{x}_1$:

$$b_1 = E_1 + y_1 \left(\alpha_1^{new} - \alpha_1^{old}\right) K\left(\boldsymbol{x}_1, \boldsymbol{x}_1\right) + y_2 \left(\alpha_2^{new,clipped} - \alpha_2^{old}\right) K\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) + b^{old} \tag{3.86}$$

Similarly, a threshold value b_2 is valid when α_2^{new} is not at the bounds , since it forces the SVM to output y_2 given the input vector $\boldsymbol{x}_2$:

$$b_2 = E_2 + y_1 \left(\alpha_1^{new} - \alpha_1^{old}\right) K\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) + y_2 \left(\alpha_2^{new,clipped} - \alpha_2^{old}\right) K\left(\boldsymbol{x}_2, \boldsymbol{x}_2\right) + b^{old} \tag{3.87}$$

Whenever both b_1 and b_2 are valid, we have that $b_1 = b_2$. When both α_1 and α_2 lie over the bounds and $L \geq H$, the interval between b_1 and b_2 provides infinite possible thresholds that are consistent the KKT, where SMO sets b to the halfway point $b_1 + \frac{b_2 - b_1}{2}$.

Figure 3.5.3 brings the pseudo-code for the SMO algorithm as originally portrayed by Platt [Pla98a].

```
target = desired output vector
point = training point matrix

procedure takeStep(i1,i2)
  if (i1 == i2) return 0
  alph1 = Lagrange multiplier for i1
  y1 = target[i1]
  E1 = SVM output on point[i1] - y1 (check in error cache)
  s = y1*y2
  Compute L, H                                                    10
  if (L == H)
    return 0
  k11 = kernel(point[i1],point[i1])
  k12 = kernel(point[i1],point[i2])
  k22 = kernel(point[i2],point[i2])
  eta = 2*k12-k11-k22
  if (eta < 0)
  {
    a2 = alph2 - y2*(E1-E2)/eta
    if (a2 < L) a2 = L                                            20
    else if (a2 > H) a2 = H
  }
  else
```

```
  {
    Lobj = objective function at a2=L
    Hobj = objective function at a2=H
    if (Lobj > Hobj+eps)
      a2 = L
    else if (Lobj < Hobj-eps)
      a2 = H                                                    30
    else
      a2 = alph2
  }
  if (|a2-alph2| < eps*(a2+alph2+eps))
    return 0
  a1 = alph1+s*(alph2-a2)
  Update threshold to reflect change in Lagrange multipliers
  Update weight vector to reflect change in a1 & a2, if linear SVM
  Update error cache using new Lagrange multipliers
  Store a1 in the alpha array                                   40
  Store a2 in the alpha array
  return 1
endprocedure

procedure examineExample(i2)
  y2 = target[i2]
  alph2 = Lagrange multiplier for i2
  E2 = SVM output on point[i2] - y2 (check in error cache)
  r2 = E2*y2
  if ((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))      50
  {
    if (number of non-zero & non-C alpha > 1)
    {
      i1 = result of second choice heuristic
      if takeStep(i1,i2)
        return 1
    }
    loop over all non-zero and non-C alpha, starting at random point
    {
      i1 = identity of current alpha                            60
      if takeStep(i1,i2)
        return 1
    }
    loop over all possible i1, starting at a random point
    {
      i1 = loop variable
```

```
      if takeStep(i1,i2)
        return 1
    }
  }                                                          70
  return 0
endprocedure

main routine:
  initialize alpha array to all zero
  initialize threshold to zero
  numChanged = 0;
  examineAll = 1;
  while (numChanged > 0 | examineAll)
    {                                                        80
      numChanged = 0;
      if (examineAll)
        loop I over all training examples
          numChanged += examineExample(I)
      else
        loop I over examples where alpha is not 0 & not C
          numChanged += examineExample(I)
      if (examineAll == 1)
        examineAll = 0
      else if (numChanged == 0)                              90
        examineAll = 1
    }
```

Figure 3.1: Pseudo-code for Platt's Sequential Minimal Optimization algorithm.

Chapter 4

Hybrid Algorithms Combining Boosting and Support Vector Machines

Using all the groundwork laid down in Chapters 2 and 3, this chapter describes the use of hybrid algorithms that combine Support Vector Machines together with Boosting techniques in order to create better learning machines that benefit from desirable properties of both. As we shall see, these two learning approaches have many individual strengths and weaknesses, where we expect that hybrid combinations will use these strengths to implicitly overcome some of each other's weaknesses. We start the chapter describing the motivations that inspired such hybrid algorithms in Section 4.1, followed by a brief modular review of our two building blocks, AdaBoost.M1 and SMO, in Section 4.2. We conclude the chapter with Sections 4.3 and 4.4, where we describe the new hybrid algorithms and their complexity analyses, respectively.

4.1 Motivations and Previous Works

In Chapter 3, we have seen that training a Support Vector Machines for binary classification problems consists of solving a quadratic programming (QP) problem, where different algorithms may be used to find the optimal separating hyperplane that maximizes the margin to its closest training points. Despite the fact that these solutions are found to be optimal in the context of the optimization model being used, solutions by SVMs are far from being optimal in the general context of the learning problem. Even though SVM principles are guaranteed to provide good generaliza-

tion when compared to other machine learning approaches, it is clear that their solution is not perfect for either the lack of enough training data to fully characterize the problem, a failure or unfitness of the SVM model and its formalization, or both. SVMs may also fail to perform well in the unfortunate case of having poor kernel selection for the problem in hand, or even in noisy problems where there are excessive outliers.

Similarly to Support Vector Machines, Boosting also offers theoretical guarantees that the final solution will lower its generalization error, this time toward a bound given by a probabilistic expression dependent on the properties of the problem, as described in Chapter 2. Nonetheless, Boosting algorithms also fail to provide a perfect solution to learning problems, even more so than SVMs. According to Freund and Schapire [FS99b], most of the work of applying SVMs or Boosting to a specific learning problem come down to selecting a kernel function or a weak learner, respectively. Schapire [Sch02] states that the performance of Boosting on a particular problem is clearly dependent on the data and weak learner. According to him and consistent with theory, Boosting may fail to perform well if given insufficient data, noisy data, too complex or too poor weak classifiers.

We shall extend the analysis of differences between Boosting and Support Vector Machines, as we previously started in Section 2.4.1, now based on Freund and Schapire [FS99b]. According to them, some of the most distinguished differences are:

Different weight vector norms. SVMs use the 2-norm, or Euclidean norm, for the weight vector, where Boosting uses the 1-norm. Although the difference between these norms may seem irrelevant in low-dimensional spaces, SVM and Boosting work with very high-dimensional spaces, which implies possibly large differences between margins.

Different searching approaches. SVM training is mostly done by solving a QP program, where the optimal solution is sough in the induced feature space by the means of the kernel function. Boosting, on the other hand employs greedy search to iteratively walk in the solution space.

So, given that neither SVMs nor Boosting are perfect silver-bullet machine learning approaches despite their very attractive theoretical properties, we can devise new ways for combining both together, trying to take advantage of their differences to make up for each one's individual inefficiencies. It would be interesting if a hybrid algorithm were able to combine features such as SVM's ability to handle noise well, in which Boosting fails, and Boosting's ability to overcome poor weak learners, where SVMs suffer with bad kernel selections and parameter tuning.

This hybrid approach has been tried before in literature. Onoda [ORM00] describes a similar approach where a boosted SVM was compared to the classical K Nearest Neighbors algorithm, and to RBF and MLP neural networks. The problem discussed was a multi-dimensional classifier applied to a monitoring system for household electric appliances. Although the author did not present any outstanding results, nor did he explore the learning aspects or the algorithms used in training the boosted SVM, the paper did prove feasible the concept of applying Boosting techniques to a Support Vector Machine.

Other related works also validate the idea of using instance pre-selection schemes in order to enhance the performance of Support Vector Machines, for instance those from Almeida et al. [ABB00, ABB01a]. These authors first used an a priori cluster selection strategy to accelerate the training of classification SVMs [ABB00]. Later, the same authors created a new training algorithm based on gradient ascent and error-dependent repetition techniques described in Sections 3.5.1 and 2.1.2, respectively, in which KKT conditions need not be evaluated. Interestingly, they report generalization errors equal to or better than those from SMO for a couple of problems [ABB01a].

4.2 Combining AdaBoost and SMO

Motivated by previous results of combining Support Vector Machines with Boosting and similar pattern selection mechanisms, in this work we attempted to combine the AdaBoost algorithm, more specifically AdaBoost.M1, with Platt's Sequential Minimal Optimization training algorithm. As we next present in Section 4.3, there are different strategies to accomplish this integration with different degrees of coupling. Before that, we shall re-examine some of the internal working details of both AdaBoost.M1 and SMO.

Consider the schematic drawing in Figure 4.1 for the AdaBoost.M1 algorithm first presented in Section 2.3.1. We may divide the several steps that compose the algorithm into separate modules for individual analysis. We first have a repository of training data where all input patterns in X are available with their corresponding labels in Y. There is a probabilistic selection mechanism based on distribution D, which is first initialized to a uniform distribution at time 0. This mechanism selects a set $\langle x, y \rangle^n$ of size n such that $\langle x, y \rangle \in \langle X, Y \rangle$, where each tuple $\langle x, y \rangle$ is drawn from $\langle X, Y \rangle$ regardless if it has been already selected, a method we refer to as selection *with replacement*. This extended set of the training set is the data effective fed to the weak learning algorithm, also referred to as base classifier. This weak learner outputs a weak hypothesis h_t at time t, which is passed to a hypothesis evaluator. In

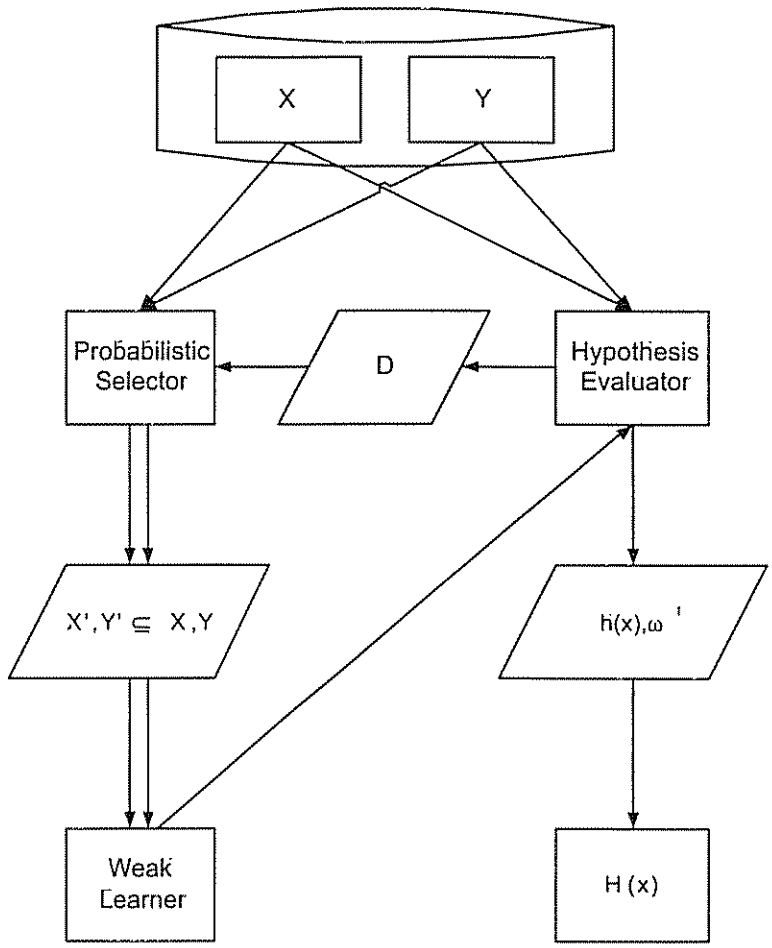


Figure 4.1: Schematic drawing of the internal workings of AdaBoost.M1.

turn, this evaluator assess the quality of the hypothesis and translates its measure to a weight value ω_t (previously addressed as α_t in Chapter 2), which is stored with h_t and used to update the distribution D. Finally, when all T iterations are completed, the final hypothesis H is output by combining all weak hypothesis h_t using a majority vote weighted by ω_t.

Consider now the schematic drawing in Figure 4.2, corresponding to Platt's SMO algorithm described in Section 3.5.3 and transcribed in Figure 3.5.3. As with AdaBoost.M1, we have a repository of training data where all input patterns in X are available with their corresponding labels in Y. In SMO, the role of the probabilistic selector from AdaBoost is fulfilled by two selection heuristics that determine which two instances from the complete training set will be chosen to undergo the optimization step. This optimization, in turn, is the analytical solution of the

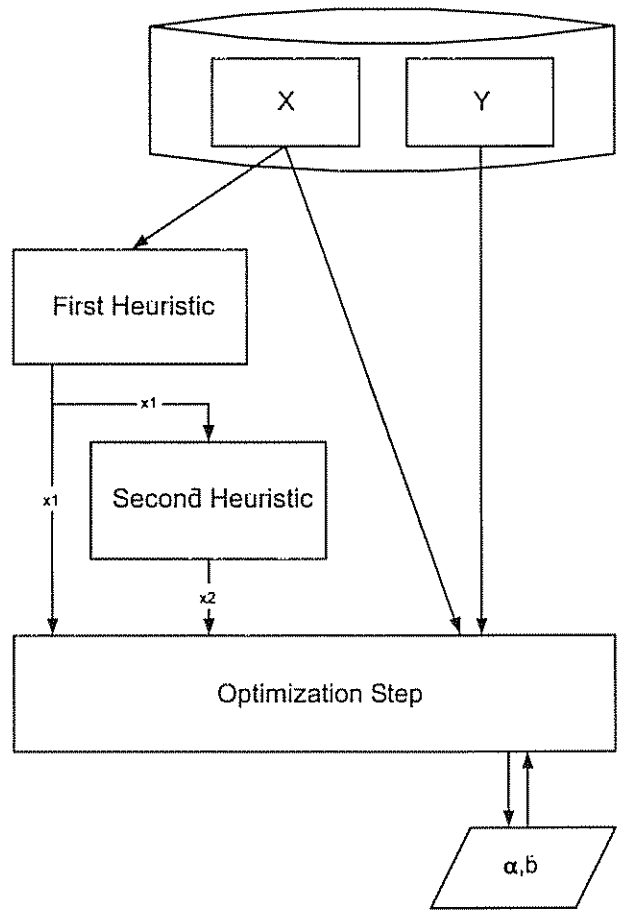


Figure 4.2: Schematic drawing of the internal workings of SMO.

QP problem determined by two Lagrange multipliers corresponding to the two instances just selected from the training set. The final output of the algorithm is given after it converges to the optimal solution, when all optimization steps have updated the values of the vector of Lagrange multipliers α and the bias term b.

4.3 Proposed Algorithms

In this section we describe our proposed hybrid algorithms that combine SMO with AdaBoost.M1 using different degrees of coupling. All algorithms described next, namely **SMO-B$_\beta$**, **SMO-B$_\alpha$**, **SMO-B$_\gamma$**, **SMO-B$_\delta$**, will be addressed as such in Chapter 5, where we describe their experiments and results. We shall then refer to the standard form of the SMO algorithm from Figures 3.5.3 and 4.2, as originally described

by John Platt [Pla98a, Pla98b], simply as **SMO**.

4.3.1 Naive Integration (SMO-B_α)

The most straight forward way of integrating AdaBoost and SMO is using both of their native forms, where AdaBoost uses SMO as its weak classifier. The schematic for this approach is presented in Figure 4.3. Since no modifications have yet been introduced to AdaBoost or SMO in this hybrid version, few remarks remain for how they interface. First, the selected data used to train SMO is drawn using selection with replacement, which may induce duplicate instances in the training set. Second, each weak hypothesis output by SMO must be stored by AdaBoost in order to be recreated later when computing the final strong hypothesis. We do that by storing, for each weak hypothesis, the vector of Lagrange multipliers α, the bias term b, and the mapping between each instance fed to SMO and its original tuple on AdaBoost's training set. Notice that we need not store which vectors are considered support vectors for each hypothesis, since they are all instances whose corresponding Lagrange multipliers are greater than zero, that is, those instances i such that $\alpha_i > 0$.

Notice that the behavior of this hybrid algorithm is governed by two input regularization parameters. The first, n, determines the size of the subset fed to SMO by AdaBoost's probabilistic selection mechanism at each iteration. We implemented this parameter as a proportion of the size of the original training set in AdaBoost's repository, ρ, where for this case of selection with replacement, n may be larger than the cardinality of the training set. The second, T, is the number of Boosting iterations that must be executed, consequently also corresponding to the number of weak hypotheses to be evaluated and later combined into the strong hypothesis.

4.3.2 Improved Subset Selection (SMO-B_β)

The first modification proposed to the SMO-B_α algorithm, described in Section 4.3.1, is a different probabilistic selection mechanism that enforces the selection of a proper subset of a training set, thus not allowing the duplication of instances. We refer to this method of probabilistic selection as selection *without replacement*, where we have that $\langle X', Y' \rangle \subseteq \langle X, Y \rangle$. In literature, this selection algorithm is known as probabilistic roulette selection [BLC00], which in this case is drawn with respect to distribution D. The schematic for this modified version of the hybrid algorithm is presented in Figure 4.4.

There are two motivations for replacing the original probabilistic selector. First, we have empirically observed that the distribution update mechanism in Ad-

aBoost often places too much emphasis on outliers. The unfortunate side effect of using selection with replacement is that the training set passed to SMO tends to contain repeated duplicates of these outliers, thus excessively biasing the generation of weak hypothesis. Second, SMO handles as an exception the case where the two Lagrange multipliers chosen for optimization correspond to equal input vectors. Recall from Equation (3.81) that, in this case, the second derivative of the objective function is zero, therefore implying in a division by zero in the Lagrange multiplier update expression from Equation (3.82).

Recall from Chapter 2 that, for binary classification problems, AdaBoost requires weak hypotheses with average error ϵ marginally better than random, that is, $\epsilon < 1/2$. Despite having enough motivation to replace the original probabilistic selector by a probabilistic roulette, this brings a most undesirable effect in AdaBoost that one of the convergence properties of the recursive weight distribution update function ceases to hold. The result is that, for some degenerated subsets of the training set, the requirement which demands that $\epsilon < 1/2$ also fails to hold. Notice that if we allow a hypothesis with $\epsilon < 1/2$, its corresponding weight ω assigned by AdaBoost would be non-positive, thus breaking the majority voting principle. Most Boosting implementations abort their iteration loop when confronted with such bad hypothesis, thus computing the strong hypothesis with whatever number of good hypothesis have been calculated up that point. Instead, we have introduced an extra hypothesis check step, where good-enough, $\epsilon < 1/2$, hypotheses from SMO get to be stored by AdaBoost, whereas hypothesis where $\epsilon \geq 1/2$ are simply ignored and the algorithm proceeds to the next iteration. Notice that if $\epsilon = 1/2$, we have also that $\omega = 0$, that is, such hypotheses would be automatically disregarded in the final weighted majority voting. Also, notice that since now some weak hypothesis may be ignored, after the T iterations we may have T' valid hypothesis to compute the final strong hypothesis, where $T' \leq T$.

4.3.3 First Heuristic Bypass (SMO-B$_\gamma$)

Starting with the two simple integration strategies used in SMO-B$_\alpha$ and SMO-B$_\beta$, we now proceed not only with unilateral modifications in Boosting or SMO, but also with the integration of some of their common components. The most obvious components that share the same functionality are AdaBoost's probabilistic selection mechanism, already considering its modified roulette form, and SMO's heuristics for selecting two instances that create small QP problems to be analytically solved. In our first attempt, we eliminate AdaBoost's probabilistic drawing of the training subset and SMO's first selection heuristics. At each iteration, the same probabilistic principle behind roulette selection is employed to draw the first instance of the QP

problem, originally selected using SMO's first heuristic. This instance is then fed to SMO's second heuristic, which examines the complete training set to choose the second instance that tries to maximize the contribution of the optimization step toward the solution.

Notice how we rely on Boosting's maintenance of the distribution D to select the first instance of the QP problem, which in turn directly influences the selection of the second heuristic. The principle of SMO's first heuristic, which is to select instances that violate KKT conditions, is hence replaced by the probabilistic assumption that these violating KKT conditions will be automatically selected based on their contribution to the overall training error of all hypotheses computed thus far. As with **SMO-B$_\beta$**, the same hypothesis check is necessary to avoid hypothesis with $\epsilon < 1/2$, and consequently $\omega < 0$. The schematic for this merger between AdaBoost and SMO components is presented in Figure 4.5.

4.3.4 First and Second Heuristics Bypass (SMO-B$_\delta$)

We now extend the approach used in **SMO-B$_\gamma$** to SMO's both first and second heuristics. These two heuristics are therefore replaced by a modified probabilistic selection mechanism, where the two instances corresponding to the two Lagrange multipliers in the QP problem are drawn from the training set with respect to D. The schematic for this algorithm is presented in Figure 4.6.

More so than with **SMO-B$_\gamma$**, we now rely on Boosting to select both points to be analytically optimized regardless of any heuristic based on KKT conditions and the estimation of contribution toward the solution. This selection corresponds to a 2-element probabilistic roulette selection on the training set, where it is expected that the distribution of probability values implicitly emphasizes those instances that are harder to learn, which in turn intuitively tend to most violate KKT conditions. Notice that as with **SMO-B$_\beta$** and **SMO-B$_\gamma$**, the same hypothesis check is again necessary to avoid negative hypothesis weights.

4.3.5 Failed Experiments

In order to get to **SMO-B$_\alpha$**, **SMO-B$_\beta$**, **SMO-B$_\gamma$**, and **SMO-B$_\delta$**, previously described in Section 4.3, we have attempted to create several other algorithms that systematically failed to converge in most of the databases in which they were executed. Although we do not address these algorithms in detail, nor did we run exhaustive tests with them, we now provide a brief description of most relevant approaches experimented.

Incremental generation of hypothesis. Boosting techniques, including AdaBoost, assume that the generation of a weak hypothesis in time t has no relationship whatsoever with the weak hypothesis generated in time $t-1$, except for the update of the distribution D with which examples are drawn to be presented to the weak learner. Using various approaches, we tried to extend this relationship by using the hypothesis generated in time $t-1$ as the starting point for the generation of the hypothesis in time t. We found that these attempts failed to converge since the current hypothesis training was excessively biased toward the direction of the previous hypothesis. Specially in cases of small n, that is, where the size of the subset selected by AdaBoost was much smaller than the original training set, the subsets drawn in different iterations were substantially different, thus resulting in great instability by SMO which started optimizations from initial solutions fit for rather different training subsets.

Hypothesis evaluation based on validation set. Boosting re-evaluation of the distribution D, with which the subset presented to the weak learner is drawn, is based on the training error of the weak hypothesis measured over this same training subset. In order to evaluate the generalization error of the final hypothesis, we divide the complete data set into two separate and non-overlapping sets, one for training and another for testing. We attempted to change this evaluation mechanism by dividing the complete data set into three non-overlapping subsets, namely training, testing, and validation sets. The training and testing sets were used to train weak hypotheses and evaluate the generalization error of the final hypothesis, respectively. Instead of evaluating the training error of each weak hypothesis with the same training set used for its training, we attempted to evaluate this measure based on the error of validation set. Unfortunately, Boosting failed to converge with this strategy.

SMO reweighting. Besides the probabilistic selection of samples based on a distribution maintained according to the evaluation of weak hypothesis, Freund and Schapire [FS96a, FS99b] describe the use of a reweighting strategy that requires weak learners able to accommodate multiplicative weight parameters that regulate the influence of each of the training instances in the training process. As we describe in more detail in Section 7.4, SMO does not originally have the ability to handle such weights. We attempted to modify SMO's analytical solver to do so, where the computationally expensive steps of probabilistic selection are skipped and each weak hypothesis is computed using different contributions for each of the vectors used, where weights correspond to AdaBoost's distribution values. Although the algorithm failed to converge,

further investigation of this problem may result in a more efficient algorithm that is equivalent or better than its probabilistic counterpart, as we suggest in Chapter 7.

4.4 Algorithmic Complexity

In this section we make a few considerations about the complexity of the four hybrid algorithms just introduced. It is clear that the complexity analyses of these algorithms depend on the individual analysis of both SMO and AdaBoost. We hence start by examining SMO, later proceeding to AdaBoost.

As we described in Section 3.5.3, when John Platt introduced SMO [Pla98a], he stated that it required an amount of memory linear in the training set size in order to execute, as opposed to the traditional quadratic requirements of other SVM training algorithms. In terms of execution time, Platt described SMO as requiring time somewhere between linear and quadratic in the training set size, which was considered a great improvement over other algorithms such as projected conjugate gradient chunking, which required time between linear and cubic in the training set size. More formally, using the big-O notation, given a training set with n samples, we say that SMO has space complexity $O(n)$, and time complexity between $O(n)$ and $O(n^2)$.

Recall from Chapter 2 that the analysis of AdaBoost, on the other hand, is bound to T, the number of weak hypotheses evaluated by the algorithm, instead of the training set size n. The analysis of AdaBoost in terms of n is rather straight forward, where it requires $O(n)$ space to store its data subsets while evaluating weak hypotheses, and $O(n)$ time to select this subset using its original selection-with-replacement mechanism. In terms of T, we have that AdaBoost requires $O(T)$ for both space and time, where the main loop of the algorithm that computes and stores weak hypotheses is repeated T times. This is consistent with the analysis performed by Kaynak and Alpaydin [KA00], where they also extend their investigation

Algorithm	Space Requirements		Time Requirements	
	in terms of n	in terms of T	in terms of n	in terms of T
SMO-B$_\alpha$	linear	linear	between linear and quadratic	linear
SMO-B$_\beta$	linear	linear	between log-linear and quadratic	linear
SMO-B$_\gamma$	linear	linear	between log-linear and quadratic	linear
SMO-B$_\delta$	linear	linear	between log-linear and quadratic	linear

Table 4.1: Space and time requirements for the four hybrid algorithms proposed.

to several other classification machines.

When we combined AdaBoost and SMO in **SMO-B**$_\alpha$, we kept AdaBoost's original probabilistic selection mechanism. For all three other algorithms, though, we have that the modified version of this mechanism no longer requires time $O(n)$, but $O(n \log n)$ instead since it performs selection without replacement. Table 4.4 summarizes these complexity analyses and depicts the specific requirements for the four hybrid algorithms proposed, in space and time, for both parameters n and T.

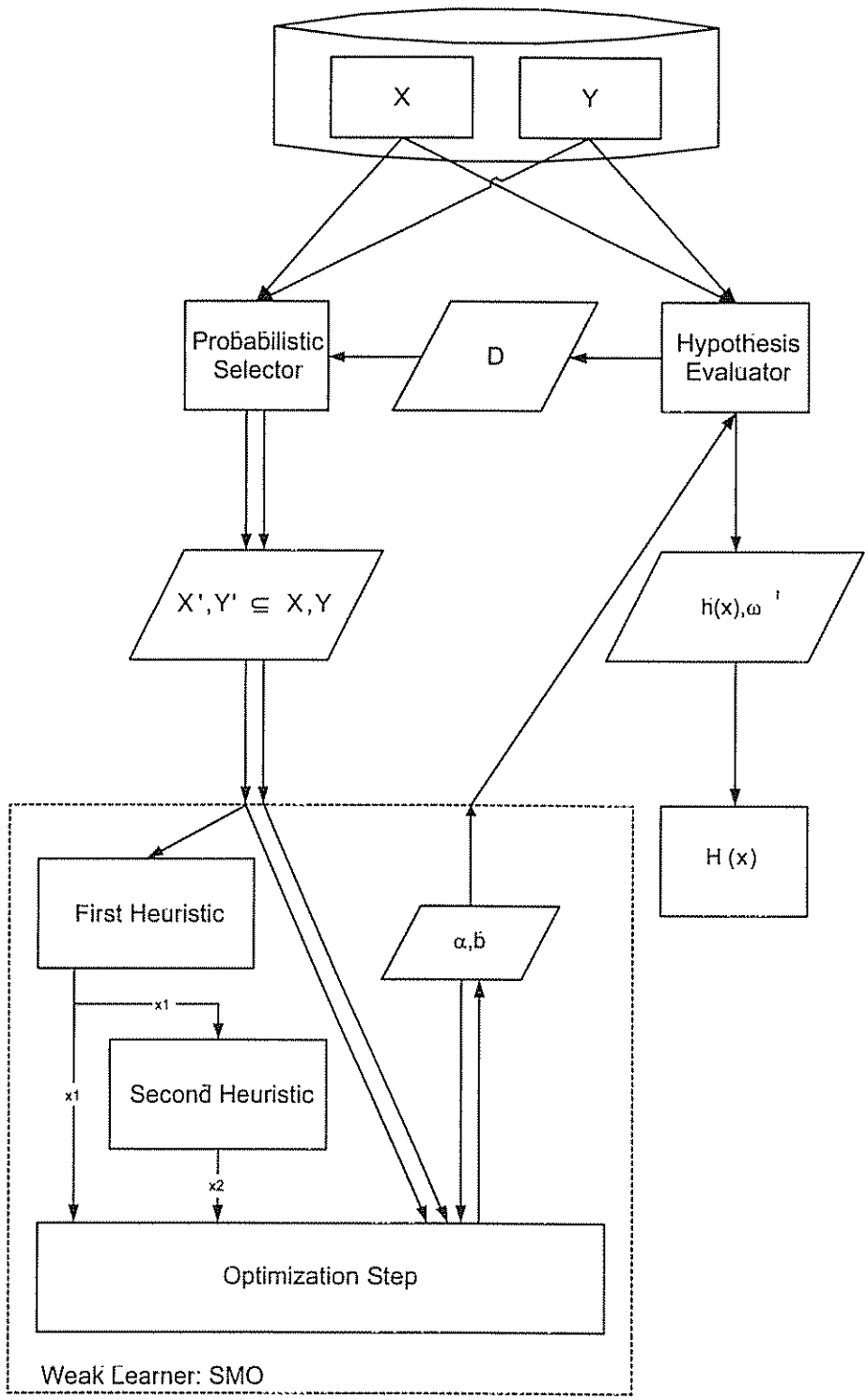


Figure 4.3: Schematic drawing for the proposed **SMO-B**$_\alpha$ algorithm.

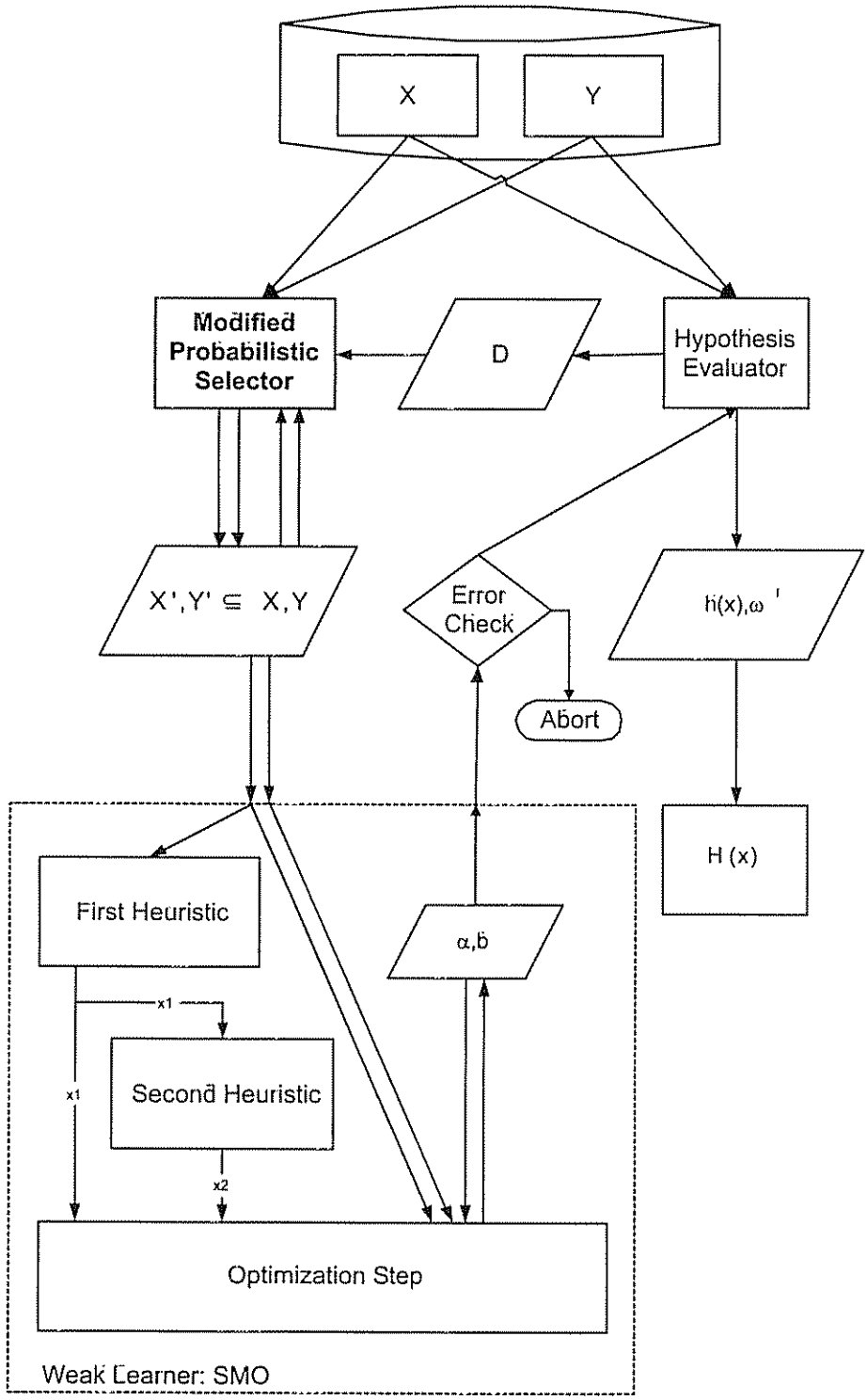


Figure 4.4: Schematic drawing for the proposed **SMO-B**$_\beta$ algorithm.

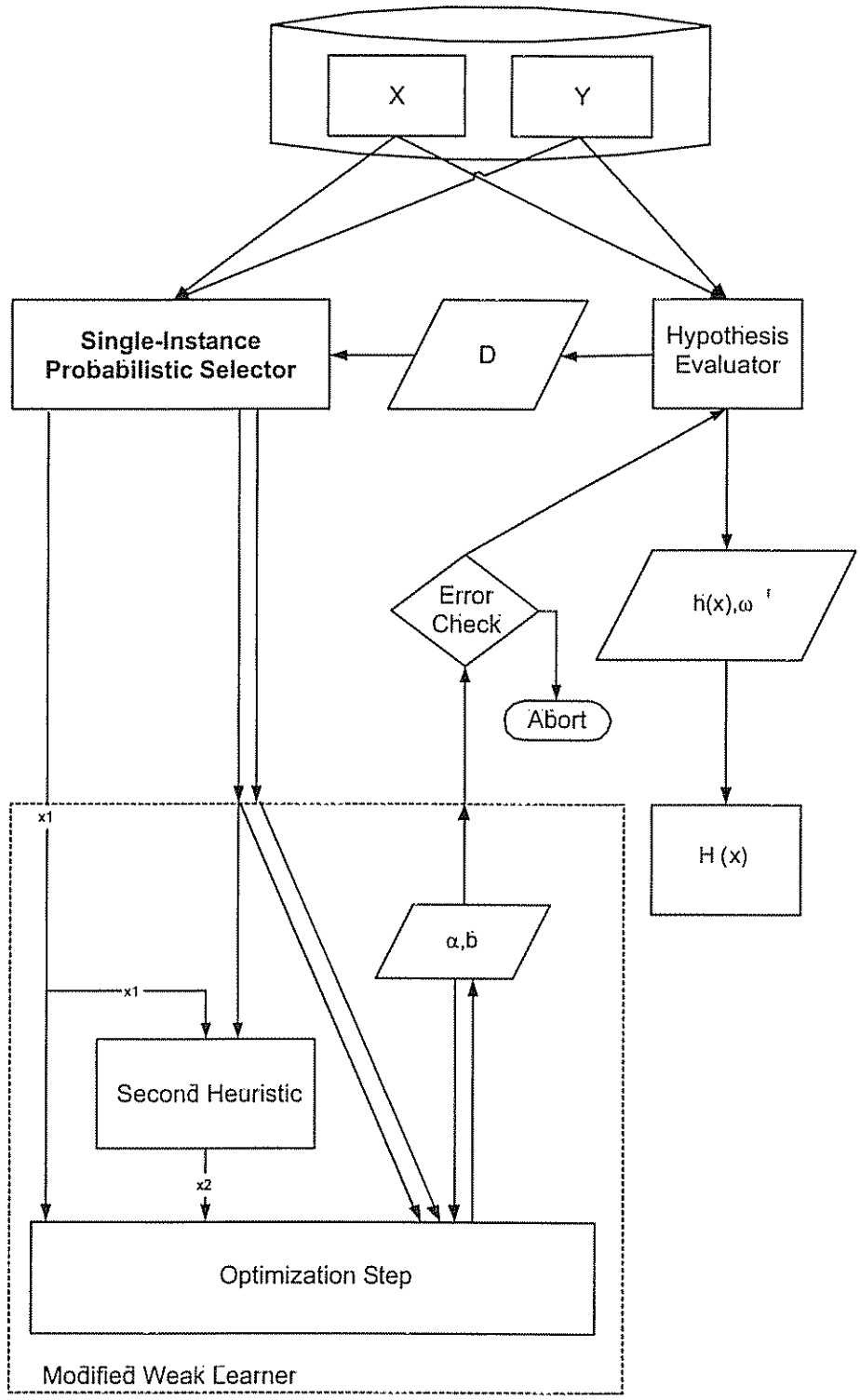


Figure 4.5: Schematic drawing for the proposed **SMO-B**$_\gamma$ algorithm.

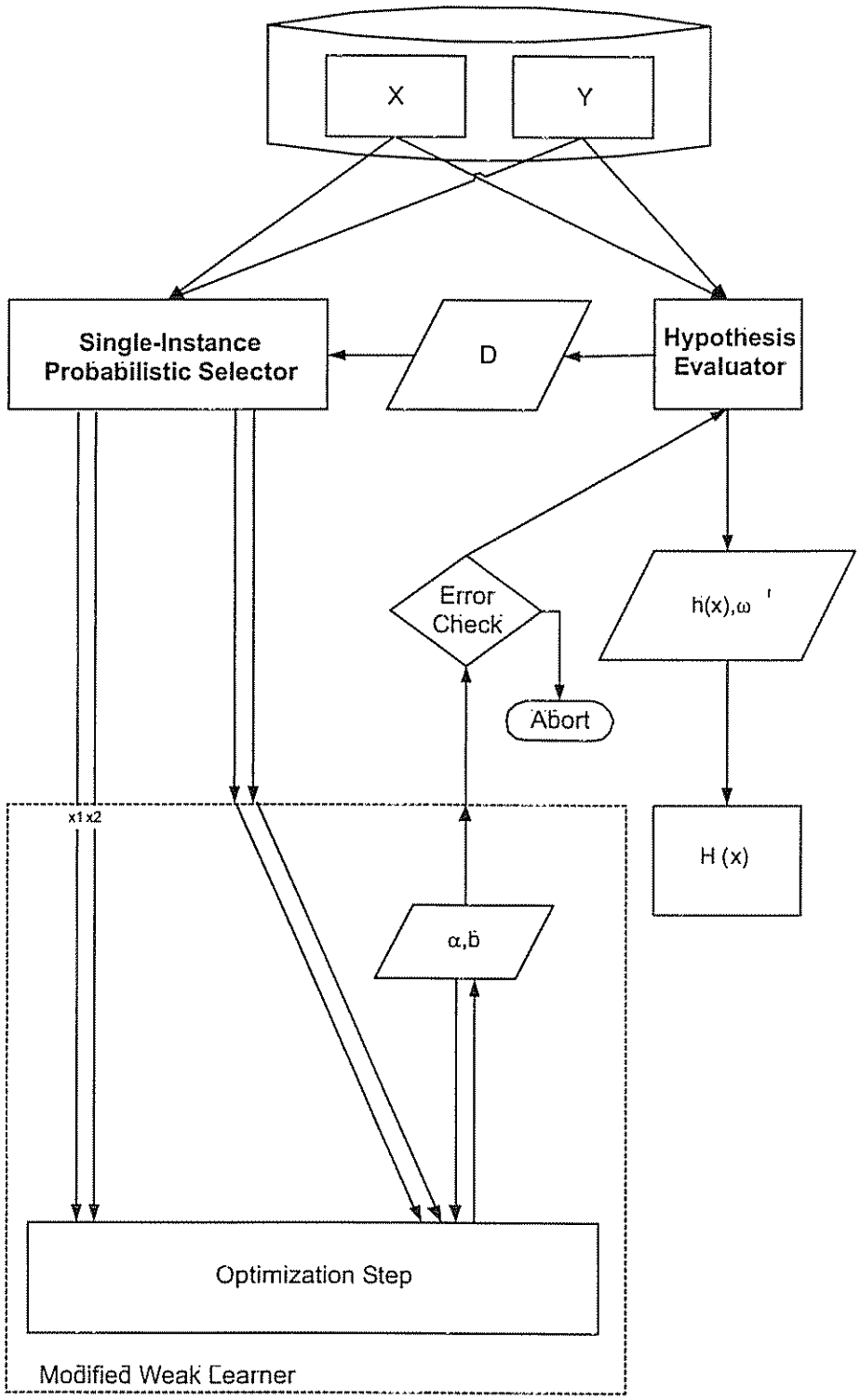


Figure 4.6: Schematic drawing for the proposed SMO-B$_\delta$ algorithm.

Chapter 5

Experiments and Results

This chapter describes the methodology of all experiments conducted in this work, as well as their results and analyses. Many decisions regarding kernel and parameter tuning have been made during each of the steps described in the next sections. These decisions are most relevant, since they determine the incremental tuning procedure the learning machines suffered, starting from a simple analysis with linear discriminants up to tweaking Boosting parameters in hybrid algorithms.

We start by describing each database in detail in Section 5.1, where we also display results and references of experiments with different algorithms by other authors. Secondly, in Section 5.2, we describe results of preliminary analyses performed over all databases using linear discriminants and the standard SMO algorithm. Thirdly, we describe results for the hybrid algorithms proposed in Chapter 4 in Section 5.3. Finally, we discuss the results obtained in all these experiments in Section 5.4.

5.1 Descriptions of Databases

In this section we give an outline of the datasets used to evaluate all learning algorithms described in this work. We have selected 22 different datasets from various sources, each one with distinct characteristics. All datasets have binary outputs in $\{-1, +1\}$ and inputs in $[-\infty, +\infty]$, thus meeting all requirements for the binary classification experiments conducted. Some databases have binary dimensions, which were all converted to the $\{-1, +1\}$ domain in order to remain orthogonal with the common output domain.

We may classify these 22 datasets into four groups based on their main features and their impact over learning problems:

Biology There are 8 biological databases, many of them describing problems of disease diagnosis. Out of these 8 databases, 2 of them have been made available by their authors with distinct training and testing sets, which are very useful when comparing generalization capabilities between different learning algorithms. Many of these datasets are classics well explored in the machine learning literature, mostly available at the UCI Repository of Machine Learning Databases [BM98]. Others have been collected in recent biology studies, combining advances in molecular biology with modern pattern recognition techniques.

Genomics There are 2 genomics databases, one about cancer diagnosis using microarray data of tissue samples, and another about determining promoter gene sequences. The first dataset was obtained from a recent study on multi-class classification of tissue types using Support Vector Machines, while the second is available at the UCI Repository of Machine Learning Databases [BM98].

Physics There is one single physics database describing an experiment with radar waves, also available at the UCI Repository of Machine Learning Databases [BM98].

Bidimensional Synthetic There are 9 synthetic databases with bidimensional input spaces. These databases were generated from three functions, each one parametrized in 3 different ways to generate sets with different degrees of overlapping between classes. Since these bidimensional databases may be easily visualized, they are displayed in Figures 5.1 to 5.9 in the following sections.

Multidimensional Synthetic There are 2 synthetic databases with 20-dimensional input spaces. These databases were generated by Leo Breiman in a study of bias and variance of learning algorithms [Bre96]. Each class was drawn using Gaussian distributions bound to specific constraints that provide a theoretical expectation of misclassification by binary classifiers.

Except for the 2 biology databases with independent training and testing sets available, all other databases had no such separation. All experiments with these databases drew independent training and testing complementary subsets from the original datasets, where for each new experiment conducted, a new selection procedure was executed.

The bias toward selecting biology and genomics databases in this work is intentional and justified. These problems often pose interesting challenges to learning

machines, since it is not unusual for them to contain massive quantities of data organized in vectors with large dimensionality. Therefore, the use of advanced learning machines capable to deal with these difficult large problems, such as SVMs, motivate the interest in testing new approaches with the potential to outperform previous ones. Due to this reason, SVMs have received a great deal of attention from the bioinformatics community, where several new applications have been attempted. For instance, taking direct advantage of SVMs' ability to handle large data sets, different problems related to microarray gene expression data have been tacked at different levels, such as classifying genes according to their specific functions [BGL+99, KK01], and classifying cancer tissue samples [CDH+00, RTR+01].

The following sections detail each database used, describing benchmark results of other learning algorithms and literature references when available.

5.1.1 Wisconsin Breast Cancer Database (bcw)

This breast cancer database was obtained from the UCI Repository of Machine Learning Databases [BM98]. It was originally created at the University of Wisconsin Hospitals, Madison, by Dr. William H. Wolberg. It describes a binary classification problem that labels tissue samples with cancer as *benign* or *malignant*. Each tissue sample underwent a series of cytological tests that described its cells' characteristics, creating feature-vectors with 9 dimensions of continuous values that form the input of the problem. There are 699 samples, 458 (65.5%) being benign, 241 (34.5%) being malignant.

A subset of this database has been studied before by Wolberg and Mangasarian [WM90], who used a multisurface method of separating hyperplanes pairs to divide the enneadimensional input space in two halves. The subset had only 369 instances then, which were all tissue samples collected so far. They report the result to be consistent with 50% of the dataset when using two pairs, and 93.5% accurate on the remaining 50%. When using three pairs, they report the result to be consistent with 67% of the dataset, and 95.9% accurate on the remaining 33%.

Zhang also reports experiments with this same database subset [Zha92], to which he applied a nearest-neighbors classifier. The average of ten executions of the 1-nearest-neighbor algorithm yielded 93.7% accuracy, where the training set was a random sample with 54% of the instances, and the testing set was the remaining 46%.

This database originally contained 16 missing values in one of its input dimensions. These missing values are due to a change in the way data were collected during the base construction. Since Support Vector Machines are not tolerant to missing data, each missing value has been replaced by zero for the purpose of con-

ducting the experiments described in this work.

5.1.2 Wisconsin Diagnostic Breast Cancer (wdbc)

There are many studies in both machine learning and medical literatures about breast cancer diagnosis. This database, also obtained from the UCI Repository of Machine Learning Databases [BM98], is based on the same studies from the University of Wisconsin Hospitals, Madison, by Dr. W. H. Wolberg, described in Section 5.1.1. Wolberg, Street and Mangasarian alone have published six papers with results of different learning techniques on the same problem, the latest being [MSW95]. This binary prediction problem has two output classes, labeled *benign* and *malignant*, to describe breast cancer tissues. Each of the 569 samples contains a real-valued vector of 30 features that describe properties of cancer cells nucleus. There are 357 instances of benign samples, and 212 instances of malignant samples.

Wolberg, Street and Mangasarian report results using separating hyperplanes in a subspace of the input space with only 3 dimensions, where they disregarded the remaining 27 features available. Using 10-fold cross-validation, they estimate an accuracy of 97.5%, which was later confirmed with 176 new patients all correctly diagnosed.

5.1.3 Wisconsin Prognostic Breast Cancer (wpbc)

A follow-up study of the breast cancer problem described in Sections 5.1.1 and 5.1.2 consists of determining cancer prognostics, that is, whether or not patients experienced recurrence of the disease after there had been no further symptoms for a predetermined amount of time (24 months). Wolberg, Street and Mangasarian used the same 30 features from their previous work [MSW95], together with 3 new features that described properties of the recurrent disease [SMW95]. This binary problem is therefore made of a 33-dimensional input space of real values, and a binary output that corresponds to labels *recurred* and *did not recur*. This database, as described in [SMW95], is available from the UCI Repository of Machine Learning Databases [BM98].

Wolberg, Street and Mangasarian report results using the same technique of separating hyperplanes in a restricted 4-dimensional subspace of the original 33-dimensional input space. They obtained 86.3% accuracy over the 198 input instances, of which 151 were labeled *did not recur* and 47 were labeled *recurred*.

5.1.4 Cancer Diagnosis Using Gene Expression Signatures (cdges)

Many of the recent advances in bioinformatics rely on gene expression signatures to diagnose different types of cancer. The use of microarray-based tumor gene expression profiles provides an objective technique to cancer diagnosis, outperforming standard subjective interpretations of clinical and histopathological information. One of these studies was published by Ramaswamy et al. [RTR+01], where they used Support Vector Machines (SVMs) along with a One Versus All (OVA) strategy to distinguish between 14 classes of cancer based on 16063 genes and expressed sequence tags per sample. In this paper, Ramaswamy et al. focus on the problem of differentiating between 14 types of cancer and normal tissue. They used 314 tumor and 98 normal tissue samples, from which 218 tumor and 90 normal tissue samples passed quality control criteria and were used for subsequent data analysis. Out of these 318 samples, 28 yielded low-confidence predictions, and were disregarded by the authors.

The database we built using the data used by Ramaswamy et al. [RTR+01] combined all 14 classes of cancer into a single class labeled *tumor*, and all healthy samples in a class labeled *normal*. This binary-output database contained 280 tissue samples, of which 190 were some type of tumor, and 90 were normal. Each sample was formed by 16063 dimensions that corresponded to continuous gene expression levels.

5.1.5 Hepatitis Domain (hepatitis)

This database was obtained from the UCI Repository of Machine Learning Databases [BM98]. It describes a binary classification problem that distinguishes patients with hepatitis between those who died, and those who managed to survive the disease. Each sample consists of 13 binary and 6 continuous dimensions that describe patients' characteristics, symptoms and clinical test results. There are 155 instances of patients, of whom 32 died and 123 lived.

This prediction problem has been studied before by Cestnik et al. [CKB87]. They proposed a new machine learning algorithm called Assistant-86, which performed with 83% accuracy over the database.

This database contained 167 missing values scattered across different input dimensions in a few samples. These missing values are due to lack of specific information about patients during the base construction. Since Support Vector Machines are not tolerant to missing data, each missing value has been replaced by zero for the purpose of conducting experiments. All binary dimension representations in the database have been converted to the $\{-1, +1\}$ set, which is particularly useful when

representing data blanks as zeros.

5.1.6 Musk Database (musk)

This dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 to be non-musks[1]. It therefore describes a binary classification problem that uses 166 real-valued features to describe these molecules according to their shape and conformation. There are 476 samples of molecules, of which 207 are found to be musks and 269 are found to be non-musks. This database was obtained from the UCI Repository of Machine Learning Databases [BM98].

Dietterich et al. have studied this problem with axis-parallel rectangles in [DLLP97], where they also report results on a second musk dataset. Using 10-fold cross-validation, they describe average, minimum and maximum generalization results for five of their proposed algorithms, 92.4% [87.0%–97.8%], 91.3% [85.5%–97.1%], 90.2% [84.2%–96.3%], 83.7% [76.2%–91.2%], 80.4% [72.3%–88.5%], respectively. These results are also compared to those of a back-propagation neural network with 127 hidden units, 75.0% [66.2%–83.8%], and to those of C4.5 (pruned), 68.5% [40.9%–61.3%].

5.1.7 Escherichia coli promoter gene sequences (DNA) (pgs)

The problem of determining promoter gene sequences is one of the classic problems of bioinformatics. Many studies about the bacterium Escherichia coli have been published, both in the light of biological sciences and in the light of machine learning. Harley and Reynolds compiled and analyzed 263 promoters with known transcriptional start points for Escherichia coli DNA genes in [HR87], which served as basis for many other developments. Based on their results, Towell et al. used the 106-instances compiled database to evaluate a hybrid learning algorithm that uses examples to inductively refine preexisting knowledge [TSN90]. Using a leave-one-out methodology, they obtained generalization errors as low as (4/106), which they compared against other machine learning techniques such as a back-propagation neural network (8/106), nearest-neighbor algorithm (13/106), Quinlan's decision

[1]According to the Webster's Revised Unabriged Dictionary [Por98], *musk* is a substance of a reddish brown color, and when fresh of the consistence of honey, obtained from a bag being behind the navel of the male musk deer. It has a slightly bitter taste, but is specially remarkable for its powerful and enduring odor. It is used in medicine as a stimulant antispasmodic. The term is also applied to secretions of various other animals, having a similar odor.

tree builder (19/106) [Qui92], and O'Neill (12/106), which is an ad hoc technique from the biological literature [O'N89, OC89].

The database contains 106 instances, of which 53 are promoters and 53 are not promoters. For each sample, the original database contained a sequence of 57 bases, each being either adenine (A), citosine (C), guanine (G), or timine (T). These bases were naively translated from the $\{A, G, T, C\}$ input space to the $\{-1, +1\}^2$ space, each base having a unique pair in $\{-1, +1\}^2$, hence loosing all sequence information in purpose. Each base originated two independent dimensions, thus forming a 114-dimensional binary input space. A much more sophisticated approach that does not ignore sequence information is suggested in Section 7.1, though it was not attempted due to research time constraints. Finally, the output space was also translated to the $\{-1, +1\}$ binary domain. The original sequence database derived by [HR87] and used by [TSN90] was obtained from the UCI Repository of Machine Learning Databases [BM98].

5.1.8 Pima Indians Diabetes Database (pid)

This database describes another case of disease diagnosis problem, this time of diabetes mellitus on a specific population of Pima American Indians. Diagnostics were binary, determining whether each patient showed signs of diabetes according to World Health Organization criteria, that is, if their 2-hour post-load plasma glucose was at least 200 mg/dl at any survey examination, or if found during routine medical care. There are 768 samples of patients, 500 of which tested negative for diabetes and the remaining 278 tested positive. Each sample contains 8 real-valued features describing patients' blood test results, body mass, and age. This database was obtained from the UCI Repository of Machine Learning Databases [BM98].

Smith et al. studied this problem using the ADAP algorithm [SED+88]. This algorithm makes continuous predictions in $[0, 1]$, which were discretized using a cutoff function. Using 75% of the data as a training set, they report 76% success in predicing the remaining testing set.

5.1.9 Johns Hopkins University Ionosphere Database (ionosphere)

Sigillito et al. investigated the problem of finding evidence of structure in the ionosphere using radar signals [SWHB89]. The data were collected by a system in Goose Bay, Labrador. This system consisted of a phased array of 16 high-frequency antennas with a total transmitted power of around 6.4 kilowatts. Targets were free electrons in the ionosphere. Good radar returns were those showing evidence of some type of structure in the ionosphere, and bad returns were those whose signals

passed through the ionosphere. The learning problem was consisted of determining whether a reading is *good* or *bad*, based on a 34-dimensional real-valued feature vector. The database contains 351 instances of readings, where 126 are labeled as bad and 225 are labeled as good.

Sigillito et al. report investigating the problem with back-propagation and Perceptron neural networks. Using the first 200 equally-balanced instances for training, they found 90.7% accuracy with a linear Perceptron, 92% with a non-linear Perceptron, and 96% with a back-propagation-trained multi-layer Perceptron network.

5.1.10 Bidimensional Normal Distributions With Overlapping (gauss^0, gauss^1, gauss^2)

One of the most classic binary problems is that of separating two classes of bidimensional points with a separating hyperplane. We have generated three instances of this problem, where we used different degrees of overlapping between the two classes, starting from non-overlapping. The three datasets, namely **gauss^0**, **gauss^1**, and **gauss^2**, can be seen in Figures 5.1, 5.2, and 5.3, respectively. Each of the three datasets, drawn with their corresponding distribution parameters, contain 1000 points evenly distributed between the two classes.

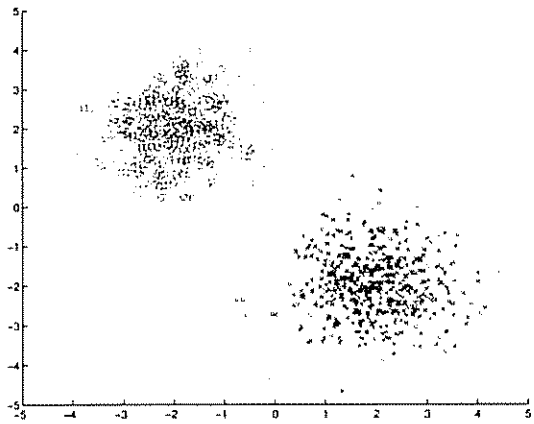


Figure 5.1: **gauss^0**: two normal-distributed classes with no overlapping.

A preliminary analysis of these databases, along with all other databases used, was performed prior to any other more advanced experiments. A detailed discussion of this analysis, specifically for databases **gauss^0** and **gauss^1**, are described ahead in Section 5.4.1.

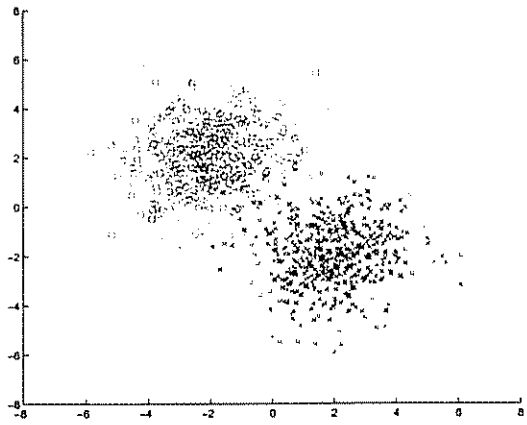


Figure 5.2: **gauss**^{1}: two normal-distributed classes with little overlapping.

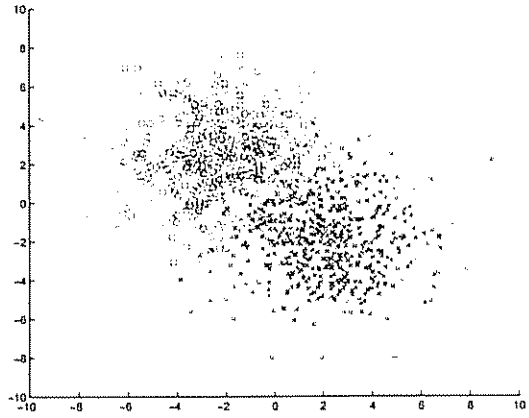


Figure 5.3: **gauss**^{2}: two normal-distributed classes with significant overlapping.

5.1.11 Bidimensional Uniform Distributions Over Chessboard With Noise (chess^0, chess^1, chess^2)

A very interesting binary bidimensional classification problem is that of distributing points in a chessboard-like fashion. Each class is drawn form a uniform distribution, and Gaussian noise is added to each point to obtain overlapping. We have generated three instances of this problem, where we used different degrees of overlapping between the two classes, starting from non-overlapping. The three datasets, namely **chess**^{0}, **chess**^{1}, and **chess**^{2}, can be seen in Figures 5.4, 5.5, and 5.6, respectively. Each of the three datasets, drawn with their corresponding distribution parameters, contain 1000 points evenly distributed between the two classes.

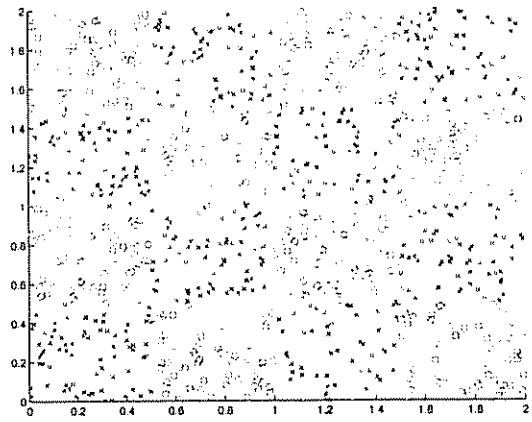


Figure 5.4: **chess**^{0}: two uniform-distributed classes over chessboard with no overlapping.

5.1.12 Bidimensional Spiral with Noise (spiral^0, spiral^1, spiral^2)

Another interesting binary bidimensional classification problem consists of using a spiral function to draw points along semi-parallel spiral lines. Each point is drawn from $(r\sin(\alpha), r\cos(\alpha))$, with α as a linear function of r, and r ranging between the minimum and maximum radii desired. Gaussian noise was added to both dimensions of the points to obtain overlapping. We have generated three instances of this problem, where we used different degrees of overlapping between the two classes, starting from non-overlapping. The three datasets, namely **spiral**^{0}, **spiral**^{1}, and **spiral**^{2}, can be seen in Figures 5.7, 5.8, and 5.9, respectively. Each of the three datasets, drawn with their corresponding distribution parameters, contain 1000 points evenly distributed between the two classes.

5.1.13 Multivariate Normal Distribution (ringnorm)

Breiman describes an algorithm for generating a synthetic dataset for binary learning problems in [Bre96]. The binary-output problem generated has an input space with 20 dimensions, drawn from a multivariate normal distribution. Class 1 has mean zero and covariance 4 times the identity, while class 2 has mean $(a, a, \ldots, a)$ and unit covariance, where $a = \frac{2}{\sqrt{20}}$. Breiman reports a theoretical expected misclassification rate of 1.3% for the problem. The dataset generated has 1000 instances, of which 503 are from class 1 and 497 from class 2.

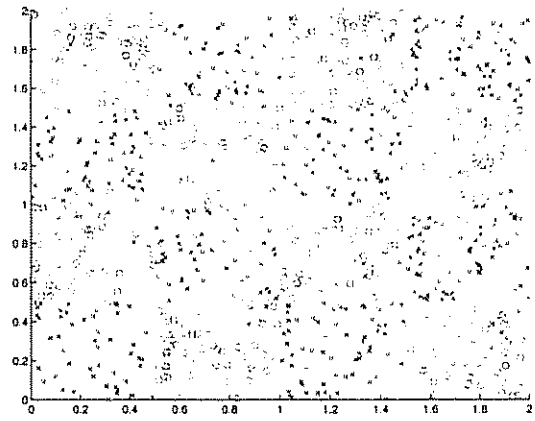


Figure 5.5: **chess**[1]: two uniform-distributed classes over chessboard with little overlapping caused by Gaussian noise.

5.1.14 Overlapping Multivariate Normal Distribution (twonorm)

Along with the dataset described in Section 5.1.13, Breiman also describes another algorithm for generating a dataset with two overlapping normal distributions [Bre96]. The binary-output problem generated has an input space with 20 dimensions, drawn from a multivariate normal distribution. Class 1 has mean $(a, a, \ldots, a)$, while class 2 has mean $(-a, -a, \ldots, -a)$, where $a = \frac{2}{\sqrt{20}}$. Breiman reports a theoretical expected misclassification rate of 2.3% for the problem. The dataset generated has 1000 instances, of which 505 are from class 1 and 495 from class 2.

5.1.15 Real SPECT (spect')

This dataset describes the diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. It has been studied by Kurgan et al. [KCT+01], where they used computer vision techniques and an integer programming algorithm. All of the 267 SPECT images, each corresponding to a single patient, were classified as either *normal* or *abnormal*. These images were processed with feature extraction algorithms to produce 44-dimensional real-valued feature vectors for each patient.

Kurgan et al. describe the results of their integer programming algorithm, trained with a training set of 80 instances, over a distinct testing set of 269 instances, where they obtained 77.0% accuracy using cardiologists' diagnoses as reference. This same database, as described in [KCT+01], may be obtained from the UCI Repository of Machine Learning Databases [BM98].

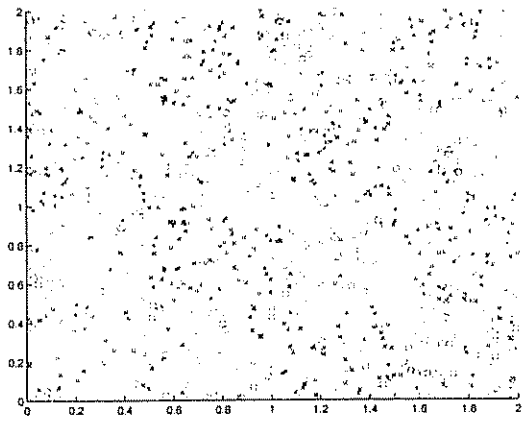


Figure 5.6: **chess**^{2}: two uniform-distributed classes over chessboard with significant overlapping caused by Gaussian noise.

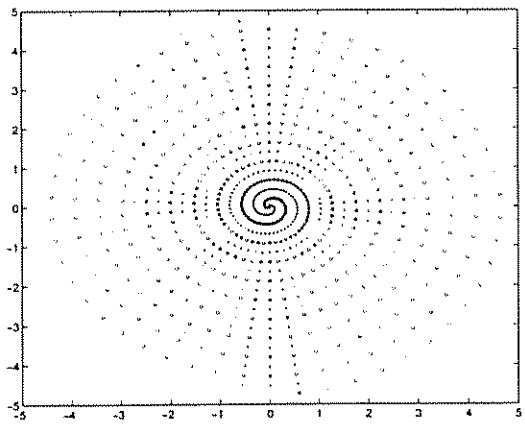


Figure 5.7: **spiral**^{0}: two classes drawn over spiral lines with no overlapping.

5.1.16 Binary SPECT (spect^b)

The same problem of cardiac SPECT images described in Section 5.1.15 was further refined by Kurgan et al. and translated into a binary problem [KCT+01]. Each of the 44-dimensional real-valued feature vectors they had previously obtained was transformed into 22-dimensional binary feature vectors, that is, an input space translation from $[-\infty, +\infty]^{44}$ to $\{-1, +1\}^{22}$.

Kurgan et al. then used the same integer programming algorithm on the binary database, trained with a training set of 80 instances, and this time tested over a testing set of 187 instances. Their results improved significantly, from 77.0% to 84.0% accuracy, again using cardiologists' diagnoses as reference. This same

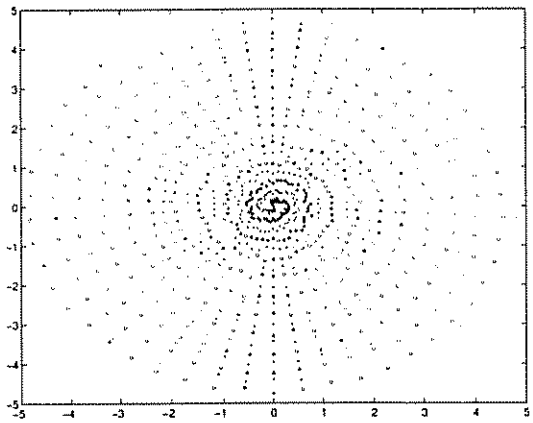


Figure 5.8: **spiral**[1]: two classes drawn over spiral lines with little overlapping caused by Gaussian noise.

post-processed binary database, as described in [KCT+01], may obtained from the UCI Repository of Machine Learning Databases [BM98].

5.2 Preliminary Analyses

Before undergoing experiments with hybrid algorithms combining Support Vector Machines and Boosting proposed in Chapter 4, we have preliminarily analyzed each dataset using other well-known learning machines. The first preliminary analysis attempted to determine the linear separability of the datasets, in which we used a simple linear learning machine.

The second analysis was performed with the standard version of SMO on a very restricted subset of each dataset. The goal of such study was to verify the applicability of the selected SVM kernels and kernel parameters to each problem, and occasionally coarse-tune them.

The third and last preliminary analysis conducted used the same standard SMO version, only this time with the complete datasets and using the kernels and kernel parameters from the previous analysis. The goal of this study was to explore the effects of different kernel parameters and SMO's C parameter, which regulates the trade-off between the training error and the margin, in order to select the best combinations to proceed to the next experiments.

Notice that for each of the three analyses presented next, in this section we restrict ourselves to presenting results for the **bcw** (Wisconsin Breast Cancer) database for illustration purposes only. Experiment methodologies and result dis-

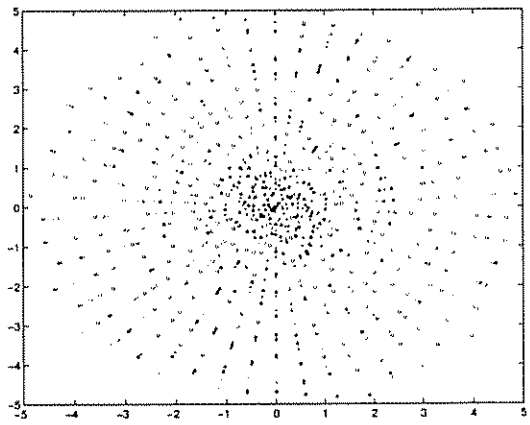


Figure 5.9: **spiral**^{2}: two classes drawn over spiral lines with significant overlapping caused by Gaussian noise.

cussions are still held in this section, where we omit other result tables and graphs for remaining databases due to size constraints. The complete results for all these databases are presented in Appendix A so as not to clutter the main text.

5.2.1 Linear Discriminant

The first analysis to which we submitted all databases aimed to verify their property of linear separability. This analysis consisted of attempting to solve each binary classification problem with one of the simplest learning machines available, a Perceptron neural network with a single neuron [Ros58]. We now revisit the classic linear classification machine previously described in Section 3.2.1.

According to the theory behind the McCulloch and Pitts neurons [MP43], thoroughly described in [Hay94] and [BLC00], a single neuron is only able to correctly classify all instances of a learning problem if the latter is linearly separable. Though a complete description and proof of these simple properties is beyond the scope of this text, which is focused on much more sophisticated learning models and algorithms, we may easily assert this linearity property by analyzing the output of a single McCulloch-Pitts neuron:

$$y = sign\left(\mathbf{w}\mathbf{x} - b\right) \tag{5.1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the weight vector associated with the neuron, and b is a threshold value. Since the output solely depends upon an inner product between two vectors exceeding or not the threshold value, this learning machine may only describe the solution for a linearly separable problem [DHS01, Hay94,

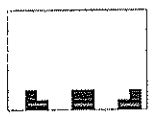
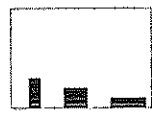
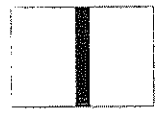
BLC00]. Such linear discriminant is not capable of describing what Duda, Hart, and Stork called the simplest non-linear problem of all, which is the result of a Boolean bi-dimensional XOR function [DHS01]. This was one of the main arguments that Minsky and Papert used in [MP69], where they outlined many of the deficiencies of the Perceptron model.

Each databased was repeatedly analyzed with the Perceptron node for 10 times, each time with different selections of non-overlapping complementary training and testing sets corresponding to 70% and 30% of the dataset, respectively. The Perceptron node used a learning rate $\eta = 0.01$, an error tolerance $tol = 0.1$, and a maximum number of epochs $epoch_{max} = 100000$.

Table A.1 displays a complete description of the results with the linear discriminant, whereas Table 5.1 brings example results for the **bcw** dataset only. For each of the databases, the average ratio of correct classifications (denoted *accuracy*) is displayed along with the medium squared error, the average number of iterations and the average execution time[2] required by the algorithm.

Together with the average values and their respective standard deviations for each of the quantities described in Tables 5.1 and A.1, we also display the histogram of such measures collected through the 10 rounds of experiments. This display is a visual aid that aims to provide a better notion of the dispersion of the experiments, thus complementing the information contained in those two statistical properties calculated.

Table 5.1: Average results for single-neuron Perceptron network after 10 rounds of experiments with distinct training and testing sets.

Database	Accuracy	MSE	Iterations	Execution time
				
bcw	96.24% ± 1.81%	0.0376 ± 0.0181	100000 ± 0	4.18 s ± 0.03 s

5.2.2 SVM Parameter Coarse Tuning

The aim of this analysis was to quickly evaluate whether the default SMO parameter set could be used to solve each of the problems selected. This default parameter

[2]By execution time we mean the time required to train the algorithm as well as the time required to evaluate the testing set.

set consisted of using an RBF kernel, with a variance parameter $p_{\sigma^2} = 1.000$ and scale parameter $p_s = 1.000$. Recall the definition of a radial-basis function (RBF), which may be expressed as $\exp\left(-\frac{\|x-x_i\|^2}{2\sigma^2 s}\right)$, where s is known as the linear scale parameter, and σ as the variance parameter.

Subsets of size 100 where drawn from each dataset, with complementary non-overlapping training and testing sets of size 70 and 30, respectively. Table 5.2 displays a description of the results with SMO and the default parameter set for **bcw** only, where results for remaining datasets are omitted due to space constraints. Three-dimensional graphs of the ratio of correct classifications (denoted *accuracy*) are displayed along with graphs for medium squared error, ratio of support vectors, norm of the separating hyperplane, number of iterations and execution time required by the algorithm. The z axis of each graph corresponds to each of the six measures just described, where the x and y axis contain variations of p_{σ^2} and p_s. Finally, for each of the six measure, there are three independent graphs for three different values for the limit of Lagrange multipliers in SMO, otherwise known to SVMs as C.

In summary, Table 5.2 contains graphs describing results of all possible permutations of three parameter that govern the generalization performance of SMO:

- The trade-off between training error and the margin (C), where C was chosen from the set $\{10^{-3}, 10^0, 10^3\}$.
- RBF kernel parameter for variance (p_{σ^2}), where p_{σ^2} was chosen from the set $\{10^{-3}, 10^0, 10^3\}$.
- RBF kernel parameter for scale (p_s), p_s chosen from the set $\{10^{-3}, 10^0, 10^3\}$.

Notice that the x and y axis of all graphs were plotted using logarithmic scales, and the z axis was plotted in a scale that maximizes the visualization of details in the graphs. The surface drawn between the neighboring points was produced with a low pass filter that converted scattered data to grid data using a norm-4 distance measure. The objective of the graphs in Tables 5.2 is to provide a visual comparison between different parameter sets for the same algorithm on the same problem.

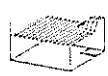
Table 5.2: Results for standard SMO algorithm over databases subsets of 100 instances at the most.

bcw (average data results)

C	p_{σ^2}	p_s	Accuracy (%)	Support Vectors (%)	Norm	MSE	Iterations	Execution Time (s)
10^0	10^0	10^0	73.33% ± 0.00%	82.86% ± 0.00%	30.8512 ± 0.0000	0.2667 ± 0.0000	27 ± 0	0.07 s ± 0.00 s
10^0	10^0	10^{-3}	46.67% ± 0.00%	95.71% ± 0.00%	54.0112 ± 0.0000	0.5333 ± 0.0000	46 ± 0	0.10 s ± 0.00 s
10^0	10^0	10^3	96.67% ± 0.00%	65.71% ± 0.00%	23.4067 ± 0.0000	0.0333 ± 0.0000	19 ± 0	0.02 s ± 0.00 s
10^0	10^{-3}	10^0	46.67% ± 0.00%	95.71% ± 0.00%	54.0112 ± 0.0000	0.5333 ± 0.0000	46 ± 0	0.10 s ± 0.00 s
10^0	10^{-3}	10^{-3}	46.67% ± 0.00%	95.71% ± 0.00%	54.0112 ± 0.0000	0.5333 ± 0.0000	46 ± 0	0.10 s ± 0.00 s
10^0	10^{-3}	10^3	46.67% ± 0.00%	95.71% ± 0.00%	54.0112 ± 0.0000	0.5333 ± 0.0000	46 ± 0	0.11 s ± 0.00 s
10^{-3}	10^0	10^0	66.67% ± 0.00%	98.57% ± 0.00%	0.0002 ± 0.0000	0.3333 ± 0.0000	16 ± 0	0.01 s ± 0.00 s
10^0	10^3	10^0	66.67% ± 0.00%	97.14% ± 0.00%	0.1249 ± 0.0000	0.3333 ± 0.0000	12 ± 0	0.00 s ± 0.00 s
10^{-3}	10^0	10^{-3}	66.67% ± 0.00%	98.57% ± 0.00%	0.0001 ± 0.0000	0.3333 ± 0.0000	26 ± 0	0.02 s ± 0.00 s
10^0	10^3	10^{-3}	96.67% ± 0.00%	64.29% ± 0.00%	23.4080 ± 0.0000	0.0333 ± 0.0000	26 ± 0	0.01 s ± 0.00 s
10^{-3}	10^0	10^3	66.67% ± 0.00%	97.14% ± 0.00%	0.0001 ± 0.0000	0.3333 ± 0.0000	15 ± 0	0.01 s ± 0.00 s
10^0	10^3	10^3	66.67% ± 0.00%	97.14% ± 0.00%	0.0001 ± 0.0000	0.3333 ± 0.0000	15 ± 0	0.01 s ± 0.00 s
10^{-3}	10^{-3}	10^0	66.67% ± 0.00%	98.57% ± 0.00%	0.0001 ± 0.0000	0.3333 ± 0.0000	26 ± 0	0.01 s ± 0.00 s
10^{-3}	10^{-3}	10^{-3}	66.67% ± 0.00%	98.57% ± 0.00%	0.0001 ± 0.0000	0.3333 ± 0.0000	26 ± 0	0.01 s ± 0.00 s
10^3	10^0	10^0	73.33% ± 0.00%	80.00% ± 0.00%	37.7399 ± 0.0000	0.2667 ± 0.0000	12 ± 0	0.04 s ± 0.00 s
10^{-3}	10^{-3}	10^3	66.67% ± 0.00%	98.57% ± 0.00%	0.0001 ± 0.0000	0.3333 ± 0.0000	26 ± 0	0.01 s ± 0.00 s
10^3	10^0	10^{-3}	46.67% ± 0.00%	95.71% ± 0.00%	61.4228 ± 0.0000	0.5333 ± 0.0000	7 ± 0	0.03 s ± 0.00 s
10^{-3}	10^3	10^0	66.67% ± 0.00%	97.14% ± 0.00%	0.0000 ± 0.0000	0.3333 ± 0.0000	11 ± 0	0.02 s ± 0.00 s
10^3	10^0	10^3	96.67% ± 0.00%	20.00% ± 0.00%	3265.6689 ± 0.0000	0.0333 ± 0.0000	88 ± 0	0.06 s ± 0.00 s
10^{-3}	10^3	10^{-3}	66.67% ± 0.00%	97.14% ± 0.00%	0.0001 ± 0.0000	0.3333 ± 0.0000	15 ± 0	0.01 s ± 0.00 s
10^3	10^{-3}	10^0	46.67% ± 0.00%	95.71% ± 0.00%	61.4228 ± 0.0000	0.5333 ± 0.0000	7 ± 0	0.03 s ± 0.00 s
10^{-3}	10^3	10^3	90.00% ± 0.00%	94.29% ± 0.00%	0.0000 ± 0.0000	0.1000 ± 0.0000	3 ± 0	0.03 s ± 0.00 s
10^3	10^{-3}	10^{-3}	46.67% ± 0.00%	95.71% ± 0.00%	61.4228 ± 0.0000	0.5333 ± 0.0000	7 ± 0	0.03 s ± 0.00 s
10^3	10^{-3}	10^3	46.67% ± 0.00%	95.71% ± 0.00%	61.4228 ± 0.0000	0.5333 ± 0.0000	7 ± 0	0.05 s ± 0.00 s
10^3	10^3	10^0	96.67% ± 0.00%	62.86% ± 0.00%	22168.2188 ± 0.0000	0.0333 ± 0.0000	14 ± 0	0.00 s ± 0.00 s
10^3	10^3	10^{-3}	96.67% ± 0.00%	20.00% ± 0.00%	3272.8018 ± 0.0000	0.0333 ± 0.0000	98 ± 0	0.07 s ± 0.00 s
10^3	10^3	10^3	66.67% ± 0.00%	97.14% ± 0.00%	103.1250 ± 0.0000	0.3333 ± 0.0000	3 ± 0	0.02 s ± 0.00 s

Table 5.2: (continued)

bcw (average result graphs for $p_{\sigma^2} \times p_s$)

$C = 10^{-3}$	$C = 10^{0}$	$C = 10^{3}$	$C = 10^{-3}$	$C = 10^{0}$	$C = 10^{3}$
			Accuracy (%)		
			Norm		
			Iterations		
				Support Vectors (%)	
				MSE	
				Execution Time (s)	

After these preliminary results, we found that SMO managed to partially classify the following databases, hence validating the RBF kernel and coarse parameter selection:

- **bcw**, **chess**^{0}, **chess**^{1}, **chess**^{2}, **hepatitis**, **ionosphere**, **musk**, **pgs**, **pid**, **ringnorm**, **spect**^{b}, **spect**^{r}, **twonorm**, **wdbc**, **wpbc**.

Meanwhile, the following databases required further tweaking of parameters and kernel types, since not at all were properly classified by SMO, which produced a 100% error rate:

- **cdges**, **gauss**^{0}, **gauss**^{1}, **gauss**^{2}, **musk**, **spiral**^{0}, **spiral**^{1}, **spiral**^{2}.

For these databases, we found out that the upper bound imposed on the database subset drawn to carry out the experiments was far too small, eliminating important features without which the problem became unlearnable to SMO with RBF kernel setups. In order to proceed, we first explored the results of the previous analysis using a linear discriminant. Since five of these eight databases, namely **cdges**, **gauss**^{0}, **gauss**^{1}, **gauss**^{2}, and **musk**, were linearly separable, we safely assumed that their learning would be possible via an RBF kernel with the trivial solution of having two radial-basis areas, one on each side of the linear separating hyperplane. Second, for databases **spiral**^{0}, **spiral**^{1}, and **spiral**^{2}, we relied on Lee

[Lee00], who reported successful experiments with the same binary spiral problems using SVMs with an RBF kernel. Reassuring these considerations, we executed the same experiments as above, this time eliminating the 100-instance subset upper bound and using the complete database to create training and testing sets, still keeping their 30%/70% cardinality proportion. For illustration purposes, Table 5.3 displays results for database **cdges** in the same format as Table 5.2, where results for all remaining datasets are omitted due to space constraints.

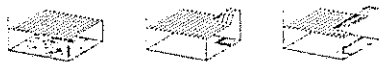
Table 5.3: Results for standard SMO algorithm over unbounded databases that previously failed with at most 100 instances.

cdges (average data results)

C	p_{σ^2}	p_s	Accuracy (%)	Support Vectors (%)	Norm	MSE	Iterations	Execution Time (s)
10^0	10^0	10^0	67.86% ± 0.00%	100.00% ± 0.00%	92.8422 ± 0.0000	0.3214 ± 0.0000	10 ± 0	26.59 s ± 0.00 s
10^0	10^0	10^{-3}	67.86% ± 0.00%	100.00% ± 0.00%	92.8421 ± 0.0000	0.3214 ± 0.0000	11 ± 0	26.21 s ± 0.00 s
10^0	10^0	10^3	67.86% ± 0.00%	100.00% ± 0.00%	94.9912 ± 0.0000	0.3214 ± 0.0000	13 ± 0	26.28 s ± 0.00 s
10^0	10^{-3}	10^0	67.86% ± 0.00%	100.00% ± 0.00%	92.8421 ± 0.0000	0.3214 ± 0.0000	10 ± 0	23.49 s ± 0.00 s
10^0	10^{-3}	10^{-3}	67.86% ± 0.00%	100.00% ± 0.00%	92.8421 ± 0.0000	0.3214 ± 0.0000	11 ± 0	24.79 s ± 0.00 s
10^0	10^{-3}	10^3	67.86% ± 0.00%	100.00% ± 0.00%	92.8421 ± 0.0000	0.3214 ± 0.0000	10 ± 0	23.76 s ± 0.00 s
10^{-3}	10^0	10^0	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	9 ± 0	12.48 s ± 0.00 s
10^0	10^3	10^0	67.86% ± 0.00%	92.86% ± 0.00%	43.3778 ± 0.0000	0.3214 ± 0.0000	185 ± 0	141.52 s ± 0.00 s
10^{-3}	10^0	10^{-3}	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	9 ± 0	12.53 s ± 0.00 s
10^0	10^3	10^{-3}	67.86% ± 0.00%	100.00% ± 0.00%	94.9911 ± 0.0000	0.3214 ± 0.0000	13 ± 0	25.80 s ± 0.00 s
10^{-3}	10^0	10^3	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	10 ± 0	12.59 s ± 0.00 s
10^0	10^3	10^3	77.38% ± 0.00%	83.16% ± 0.00%	49.0887 ± 0.0000	0.2262 ± 0.0000	52 ± 0	56.32 s ± 0.00 s
10^{-3}	10^{-3}	10^0	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	7 ± 0	12.17 s ± 0.00 s
10^{-3}	10^{-3}	10^{-3}	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	9 ± 0	12.42 s ± 0.00 s
10^3	10^0	10^0	67.86% ± 0.00%	100.00% ± 0.00%	170.9628 ± 0.0000	0.3214 ± 0.0000	7 ± 0	30.26 s ± 0.00 s
10^{-3}	10^{-3}	10^3	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	9 ± 0	12.63 s ± 0.00 s
10^3	10^0	10^{-3}	67.86% ± 0.00%	100.00% ± 0.00%	170.9711 ± 0.0000	0.3214 ± 0.0000	7 ± 0	27.92 s ± 0.00 s
10^{-3}	10^3	10^0	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	7 ± 0	12.55 s ± 0.00 s
10^3	10^0	10^3	67.86% ± 0.00%	100.00% ± 0.00%	167.4624 ± 0.0000	0.3214 ± 0.0000	7 ± 0	26.97 s ± 0.00 s
10^{-3}	10^3	10^{-3}	67.86% ± 0.00%	64.80% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	11 ± 0	12.60 s ± 0.00 s
10^3	10^{-3}	10^0	67.86% ± 0.00%	100.00% ± 0.00%	170.9600 ± 0.0000	0.3214 ± 0.0000	7 ± 0	28.75 s ± 0.00 s
10^{-3}	10^3	10^3	67.86% ± 0.00%	65.31% ± 0.00%	0.0001 ± 0.0000	0.3214 ± 0.0000	15 ± 0	13.61 s ± 0.00 s
10^3	10^{-3}	10^{-3}	67.86% ± 0.00%	100.00% ± 0.00%	170.9942 ± 0.0000	0.3214 ± 0.0000	7 ± 0	26.61 s ± 0.00 s
10^3	10^{-3}	10^3	67.86% ± 0.00%	100.00% ± 0.00%	170.9966 ± 0.0000	0.3214 ± 0.0000	7 ± 0	30.14 s ± 0.00 s
10^3	10^3	10^0	75.00% ± 0.00%	95.92% ± 0.00%	7367.5454 ± 0.0000	0.2500 ± 0.0000	55 ± 0	165.60 s ± 0.00 s
10^3	10^3	10^{-3}	67.86% ± 0.00%	100.00% ± 0.00%	167.4231 ± 0.0000	0.3214 ± 0.0000	7 ± 0	27.62 s ± 0.00 s
10^3	10^3	10^3	78.57% ± 0.00%	88.27% ± 0.00%	1639.0208 ± 0.0000	0.2143 ± 0.0000	186 ± 0	269.44 s ± 0.00 s

Table 5.3: (continued)

cdges (average result graphs for $p_{\sigma^2} \times p_s$)

$C = 10^{-3}$	$C = 10^{0}$	$C = 10^{3}$	$C = 10^{-3}$	$C = 10^{0}$	$C = 10^{3}$
					
Accuracy (%)			Support Vectors (%)		
					
Norm			MSE		
					
Iterations			Execution Time (s)		

5.2.3 SVM Parameter Fine Tuning

Now that we qualitatively evaluated the feasibility of using SMO with an RBF kernel for all databases, we must use this setup to run a more thorough experiment followed by a quantitatively analysis. There are two very important reasons to carry out this analysis:

Reliable Parameter Tuning. Although we already used SMO on all databases during the experiments described in Section 5.2.2, databases were cropped to 100 instances and there was no concern with statistical relevance. In this experiment, not only entire databases were used to generate different training and testing sets at each run, each parameter configuration was repeated for 10 runs to avoid statistical flukes. The reason for such rigid criteria was to determine reasonably good pseudo-optimal values for C, p_{σ^2}, and p_s for all databases. These values for the standard version of SMO, for consistency of results, were the same used in experiments with the proposed hybrid algorithms.

Performance Benchmark. As we describe results for the proposed hybrid algorithms in Section 5.3, it is interesting to compare these results with the standard version of SMO proposed by Platt [Pla98a]. Furthermore, parameters

C, p_{σ^2}, and p_s are kept the same for SMO and all four hybrids, thus avoiding wrongful performance comparisons due to different SVM parameters.

As before, all three parameters were varied in ranges where all possible configurations were experimented with. For the trade-off between training error and the margin, C, the possible values were chosen from the set $\{10^{-3}, 10^{0}, 10^{3}\}$. For the RBF kernel parameter for variance, p_{σ^2}, values were chosen from $\{10^{-3}, 10^{0}, 10^{3}\}$. For the RBF kernel parameter for scale, p_s, from $\{10^{-3}, 10^{0}, 10^{3}\}$. Before each execution, non-overlapping complementary selections for training and testing sets, corresponding to 70% and 30% of the dataset, respectively, were drawn with respect to a uniform distribution.

Results for this experiment are presented in Table 5.4 for the **bcw** database, where results for all remaining databases are omitted due to space constraints. Results are formatted the same way as those from Tables 5.2 and 5.3, with three-dimensional graphs that vary the two kernel parameters in x and y axis, which are plotted in logarithmic scale, and each quantitative performance measure is given in the z axis using a range that maximizes the visualization of details in the graphs. Surfaces drawn between neighboring points were produced with a low pass filter that converted scattered data to grid data using a norm-4 distance measure. Besides these three-dimensional plots, Table 5.4 also brings complete average data description for each parameter setup, as well as histograms for each measure using the best parameter configuration, determined through the 10 rounds of experiments. These best parameter configurations, one for each of the databases used, are shown highlighted in the tables. The two criteria with which to sort and pick a best parameter configuration were, in order:

- Best accuracy, where better configurations yield smaller generalization errors;
- Faster execution times, where if there is a tie with the accuracy criterion, the fastest configuration is selected.

If a tie still results from these two criteria, a random choice for best configuration is made. The three parameters that describe this best configuration are the basis for the new setups built for all further experiments with hybrid algorithms described in Section 5.3.

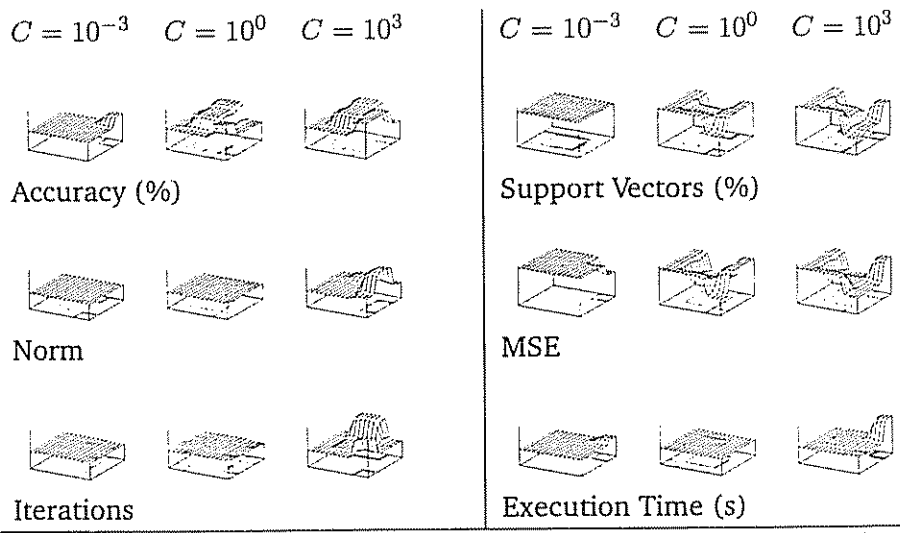
Table 5.4: Average and best results for standard SMO algorithm after 10 rounds of experiments with distinct training and testing sets.

bcw (average data results)

C	$p_{\sigma}2$	p_{ε}	Accuracy (%)	Support Vectors (%)	Norm	MSE	Iterations	Execution Time (s)
10^{-3}	10^{0}	10^{-3}	65 29% ± 2 43%	69 08% ± 2 17%	0.0003 ± 0 0000	0.3471 ± 0.0243	32 ± 10	0 45 s ± 0 03 s
10^{-3}	10^{0}	10^{3}	65 29% ± 2 43%	68.79% ± 2 11%	0 0040 ± 0.0003	0.3471 ± 0 0243	27 ± 8	0.25 s ± 0 03 s
10^{0}	10^{3}	10^{0}	65 29% ± 2 43%	68 77% ± 2 11%	4 3661 ± 0 3347	0.3471 ± 0 0243	27 ± 6	0 25 s ± 0 02 s
10^{-3}	10^{3}	10^{-3}	65 29% ± 2 43%	68 79% ± 2 08%	0.0040 ± 0.0003	0.3471 ± 0 0243	23 ± 6	0.24 s ± 0.02 s
10^{-3}	10^{-3}	10^{0}	65 29% ± 2 43%	69 20% ± 2 21%	0.0003 ± 0 0000	0.3471 ± 0 0243	33 ± 11	0 41 s ± 0 04 s
10^{0}	10^{-3}	10^{3}	67 90% ± 3 99%	75 13% ± 2 80%	325 5565 ± 13 4327	0.3210 ± 0.0399	36 ± 11	2 08 s ± 0 81 s
10^{0}	10^{3}	10^{3}	65 29% ± 2 43%	68.75% ± 2.09%	0 0021 ± 0 0015	0.3471 ± 0.0243	10 ± 2	0.23 s ± 0.02 s
10^{0}	10^{3}	10^{-3}	96.00% ± 1 09%	18.88% ± 0 61%	43 2219 ± 0 7215	0.0400 ± 0 0109	53 ± 15	0 28 s ± 0 07 s
10^{0}	10^{0}	10^{-3}	68 43% ± 3 94%	75 13% ± 2 72%	325 5555 ± 13 4260	0.3157 ± 0.0394	48 ± 30	2 26 s ± 0.99 s
10^{3}	10^{3}	10^{3}	65 29% ± 2 43%	68.83% ± 2 11%	735.2062 ± 314 7489	0.3471 ± 0 0243	30 ± 27	92.08 s ± 87 88 s
10^{-3}	10^{-3}	10^{3}	65.29% ± 2.43%	68.96% ± 2.09%	0.0003 ± 0.0000	0.3471 ± 0.0243	29 ± 10	0.44 s ± 0.02 s
10^{3}	10^{0}	10^{3}	96.10% ± 0.97%	9.06% ± 1.43%	6075.9512 ± 584.4494	0.0390 ± 0.0097	531 ± 209	5.46 s ± 2.40 s
10^{3}	10^{3}	10^{0}	95 76% ± 0.96%	18 16% ± 0 74%	41721 1188 ± 725 6512	0.0424 ± 0 0096	33 ± 5	0 16 s ± 0 01 s
10^{3}	10^{0}	10^{-3}	68 43% ± 3 94%	78 51% ± 0.98%	335.1146 ± 3 9536	0.3157 ± 0 0394	9 ± 2	1 72 s ± 0.19 s
10^{3}	10^{-3}	10^{-3}	68.43% ± 3 94%	78 69% ± 1 49%	335 1190 ± 3 9558	0.3157 ± 0 0394	10 ± 2	1.63 s ± 0 21 s
10^{-3}	10^{3}	10^{3}	83 05% ± 24 06%	68 75% ± 2 09%	0 0000 ± 0.0000	0.1695 ± 0.2406	3 ± 0	19 02 s ± 0 70 s
10^{3}	10^{0}	10^{0}	91 76% ± 1 76%	58 53% ± 1 29%	189 0334 ± 4 9804	0.0824 ± 0 0176	51 ± 12	5 88 s ± 1 27 s
10^{3}	10^{-3}	10^{3}	67 90% ± 3 99%	78 73% ± 1 40%	335 1064 ± 3 9696	0.3210 ± 0 0399	10 ± 2	1 77 s ± 0 24 s
10^{0}	10^{0}	10^{3}	96.00% ± 1 09%	18.96% ± 0.68%	43 2211 ± 0 7283	0.0400 ± 0.0109	49 ± 11	0 34 s ± 0 19 s
10^{-3}	10^{-3}	10^{-3}	65.29% ± 2 43%	69.06% ± 2 19%	0 0003 ± 0 0000	0.3471 ± 0 0243	32 ± 10	0.40 s ± 0 02 s
10^{0}	10^{-3}	10^{-3}	68 43% ± 3 94%	75 44% ± 4 33%	325 5636 ± 13 4304	0.3157 ± 0 0394	29 ± 11	1 67 s ± 0 57 s
10^{3}	10^{-3}	10^{0}	68 43% ± 3 94%	78.75% ± 1 24%	335 0858 ± 3 9545	0.3157 ± 0 0394	9 ± 2	1 56 s ± 0 18 s
10^{0}	10^{0}	10^{0}	91 24% ± 2 09%	58 63% ± 1 26%	149 5965 ± 5 3023	0.0876 ± 0 0209	50 ± 24	4 13 s ± 2 28 s
10^{-3}	10^{3}	10^{0}	65 29% ± 2 43%	68 75% ± 2 09%	0 0000 ± 0 0000	0.3471 ± 0 0243	8 ± 2	0 19 s ± 0 01 s
10^{0}	10^{-3}	10^{0}	67 90% ± 3 99%	74 76% ± 3 69%	325 5480 ± 13 4255	0.3210 ± 0 0399	48 ± 21	2 31 s ± 0 80 s
10^{3}	10^{3}	10^{-3}	96 10% ± 0 97%	9 10% ± 1 41%	6076.7823 ± 584 6272	0.0390 ± 0 0097	509 ± 169	5 25 s ± 2 25 s
10^{-3}	10^{0}	10^{0}	65 29% ± 2 43%	69 35% ± 1 98%	0.0011 ± 0.0001	0.3471 ± 0.0243	39 ± 12	0.46 s ± 0.04 s

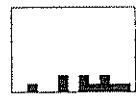
Table 5.4: (continued)

bcw (average result graphs for $p_{\sigma^2} \times p_s$)

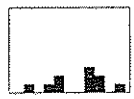


bcw (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 96.10% ± 0.97%

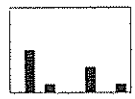


Support Vectors (%): 9.06% ± 1.43%



Norm: 6075.9512 ± 584.4494

MSE: 0.0390 ± 0.0097



Iterations: 531 ± 209

Execution Time (s): 5.46 s ± 2.40 s

5.3 Experimental Results

In Section 5.2, besides exploring interesting properties of each of the 22 learning problems tacked, we were able to tune SVM parameters, namely the trade-off parameter C between training error and the margin, and the two kernel parameters for

the RBF kernel, for each of the corresponding datasets. In this section we describe the results for the four proposed hybrid algorithms in Chapter 4, which are tested for each dataset using the SVM parameters tuned in Section 5.2 as configuration bases.

Each of these experiments were performed using different parameter configurations attempting to induce different behavior patterns from the Boosting algorithm and from the hybrid frontier between the booster and the weak learner. As we did in Section 5.2, in these experiments we observed the results for the algorithms based on the variation of two regularization parameters:

- T, the target number of weak hypotheses that the Boosting algorithm must build using the weak learner and later combine into the strong hypothesis using majority voting, chosen from the set $\{1, 100, 1000\}$;
- ρ, the fraction of the training set cardinality which determines the cardinality of the training subset selected using the probability distribution maintained by the Boosting algorithm, chosen from the set $\{0.1, 0.4, 0.7, 1.0\}$.

The 12 possible combinations of these two sets of parameters were tested over all 22 datasets with each one of the four hybrid algorithms. In most cases, experiments were repeated 10 times for statistical relevancy, except in specific instances where computational resource limitations forced the experiments to be carried out lesser times. These experiments and their results are described in the following sections.

5.3.1 Results for SMO-B_α

The first hybrid algorithm proposed, **SMO-B_α**, is the most straight forward and naive way of integrating Boosting and Support Vector Machines. Complete results for experiments with **SMO-B_α** are presented in Table A.2, where partial results for the **bcw** database are also presented in Table 5.5. Before each execution of the algorithm, non-overlapping complementary selections for training and testing sets, corresponding to 70% and 30% of the dataset, respectively, were drawn with respect to a uniform distribution.

For each dataset, these tables first present average results for the final strong hypotheses computed by the Boosting algorithm. Besides the textual representation of the results, which display average results and corresponding standard deviations, three-dimensional graphs are used to describe each quantitative performance measure over the variation of the two regularization parameters, T and ρ, where ρ is given in the y axis, T is given in logarithmic scale in the x axis, and all measures are

given in the z axis using a range that maximizes the visualization of details in the graphs. Surfaces drawn between neighboring points were produced with a low pass filter that converts scattered data to grid data using a norm-4 distance measure.

Secondly, Tables 5.5 and A.2 bring average results collected from each individual weak hypothesis created by the weak learner. As with the results for the strong hypotheses, data are presented in a descriptive table as well as in graphs. For both strong and weak hypotheses data displays, the best parameter configuration obtained is highlighted. The three criteria with which to sort and pick the best parameter configurations were, in order:

- Best accuracy, where better configurations yield smaller generalization errors;
- Faster execution times, where if there is a tie with the accuracy criterion, the fastest configuration is selected;
- Greater percentage of valid weak hypotheses, where if there is a tie with the two criteria above, the configuration with lesser occurrences of invalid weak hypotheses is selected.

If a tie still results from these three criteria, a random choice for best configuration is made. Notice that for **SMO-B$_\alpha$**, which uses the standard heuristics from AdaBoost.M1, the algorithm halts processing whenever an invalid weak hypothesis is found. This issue, which is changed in the upcoming algorithms, is most relevant when using the percentage of valid weak hypotheses as a selection criterion.

Finally, Tables 5.5 and A.2 bring the histograms for the repeated executions of the best configuration for the algorithm and each database, where the dispersion of the three quantitative measures for strong hypotheses are displayed.

Notice that due to limitations in computational resources, not all datasets were able to be experimented with **SMO-B$_\alpha$**. Therefore, Table A.2 brings results for only those problems which were able to be solved in reasonable time. Moreover, due to the excessive training time required by the algorithm, as depicted in Tables 5.5 and A.2, not all experiments were repeated 10 times, some being executed only once. For this reason, corresponding standard deviations for measures taken over the average of different executions are shown as zero.

Table 5.5: Average and best results for hybrid algorithm **SMO-B$_\alpha$** after 10 rounds of experiments with distinct training and testing sets.

bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	51.00% ± 16.81%	3.74% ± 1.84%	24.36 s ± 15.15 s
1	0.4	53.67% ± 14.99%	100.00% ± 0.00%	0.99 s ± 0.87 s
1	**0.1**	**87.71% ± 12.38%**	**100.00% ± 0.00%**	**0.03 s ± 0.03 s**
1000	0.1	50.14% ± 15.43%	4.90% ± 1.84%	1.13 s ± 0.12 s
1000	0.7	49.19% ± 15.41%	3.39% ± 1.70%	102.87 s ± 63.15 s
100	1	47.19% ± 15.17%	16.40% ± 10.86%	186.25 s ± 112.08 s
1	0.7	56.52% ± 13.98%	100.00% ± 0.00%	18.03 s ± 24.23 s
1000	1	44.71% ± 14.50%	1.60% ± 0.56%	178.40 s ± 98.95 s
100	0.7	50.05% ± 15.43%	30.20% ± 8.96%	127.54 s ± 118.44 s
100	0.4	53.86% ± 14.94%	35.30% ± 19.62%	21.25 s ± 12.50 s
1	1	47.29% ± 15.19%	100.00% ± 0.00%	228.99 s ± 271.93 s
100	0.1	49.19% ± 19.95%	55.10% ± 33.19%	0.59 s ± 0.34 s

bcw (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

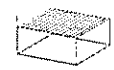
Table 5.5: (continued)

bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.22% ± 0.11%	251.2781 ± 161.4524	6 ± 3
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	6.81% ± 2.54%	17973.7100 ± 17899.1636	301 ± 243
1	**0.1**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**3.38% ± 0.84%**	**969.5497 ± 2283.8921**	**38 ± 50**
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.15% ± 0.06%	27.4609 ± 11.5640	2 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	0.25% ± 0.11%	395.3559 ± 203.2713	9 ± 5
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	1.37% ± 0.75%	4123.0772 ± 1374.9117	86 ± 37
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	10.81% ± 2.51%	76494.4166 ± 57033.0937	1797 ± 2095
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	0.14% ± 0.04%	403.1479 ± 178.5044	8 ± 3
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	2.27% ± 0.70%	4831.3097 ± 3590.3131	102 ± 68
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	2.06% ± 1.12%	2562.8477 ± 1283.3616	57 ± 30
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	16.67% ± 5.35%	322822.2248 ± 248327.9780	8017 ± 7173
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	1.75% ± 1.04%	334.7604 ± 184.4323	16 ± 9

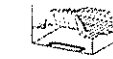
bcw (weak hypotheses' average result graphs for $T \times \rho$)


Alpha


Error Rate (%)

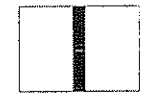

Support Vectors (%)

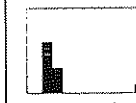

Norm


SMO Iterations

bcw (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 87.71% ± 12.38%


Good Weak Hyp. (%): 100.00% ± 0.00%


Execution Time (s): 0.03 s ± 0.03 s

5.3.2 Results for SMO-B$_\beta$

The second hybrid algorithm proposed, **SMO-B$_\beta$**, is a natural evolution of **SMO-B$_\alpha$** aimed to solve the problem of ignoring far too many weak hypotheses, as previously shown in the data from Tables 5.5 and A.2. The results for **SMO-B$_\beta$** are shown in

Tables 5.6 and A.3, where results are presented using the very same format as in Tables 5.5 and A.2, respectively. These tables first show results for the final strong hypotheses, then average results for all generated weak hypotheses, and finally the histogram for the best configuration achieved.

The criteria for selecting the best configuration were the same as described in Section 5.3.1 for **SMO-B$_\alpha$**, the only implicit difference being the new mechanism of ignoring invalid weak hypotheses without halting processing.

It is worthwhile noticing that, as in experiments with **SMO-B$_\alpha$**, not all datasets were able to be tested due to limitations in computational resources. Also, due to excessive training time required by the algorithm, no experiment with **SMO-B$_\alpha$** was repeated more than once. For this reason, standard deviations for these measures are shown as zero.

Table 5.6: Average and best results for hybrid algorithm **SMO-B$_\beta$** after 10 rounds of experiments with distinct training and testing sets.

bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	97.62% ± 0.00%	3.70% ± 0.00%	905.42 s ± 0.00 s
1	**0.4**	**97.62% ± 0.00%**	**100.00% ± 0.00%**	**0.04 s ± 0.00 s**
1	0.1	96.19% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	96.67% ± 0.00%	99.90% ± 0.00%	14.31 s ± 0.00 s
1000	0.7	96.19% ± 0.00%	1.10% ± 0.00%	2805.17 s ± 0.00 s
100	1	96.67% ± 0.00%	100.00% ± 0.00%	444.31 s ± 0.00 s
1	0.7	97.14% ± 0.00%	100.00% ± 0.00%	2.12 s ± 0.00 s
1000	1	96.67% ± 0.00%	100.00% ± 0.00%	4491.92 s ± 0.00 s
100	0.7	97.14% ± 0.00%	10.00% ± 0.00%	285.85 s ± 0.00 s
100	0.4	95.24% ± 0.00%	14.00% ± 0.00%	100.29 s ± 0.00 s
1	1	96.67% ± 0.00%	100.00% ± 0.00%	5.04 s ± 0.00 s
100	0.1	97.14% ± 0.00%	100.00% ± 0.00%	1.20 s ± 0.00 s

bcw (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

Table 5.6: (continued)

bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	-0.1294 ± 0.0000	0.02% ± 0.00%	13.76% ± 0.00%	5602.5713 ± 0.0000	325 ± 0
1	**0.4**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**3.33% ± 0.00%**	**1666.2279 ± 0.0000**	**21 ± 0**
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	2.86% ± 0.00%	187.4276 ± 0.0000	11 ± 0
1000	0.1	9.3123 ± 0.0000	0.00% ± 0.00%	3.59% ± 0.00%	833.1179 ± 0.0000	39 ± 0
1000	0.7	-0.3749 ± 0.0000	0.04% ± 0.00%	19.46% ± 0.00%	8896.0666 ± 0.0000	454 ± 0
100	1	0.0193 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10056.9649 ± 0.0000	547 ± 0
1	0.7	2.0482 ± 0.0000	0.04% ± 0.00%	15.24% ± 0.00%	5772.3096 ± 0.0000	368 ± 0
1000	1	0.0019 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10054.5734 ± 0.0000	549 ± 0
100	0.7	-0.1583 ± 0.0000	0.04% ± 0.00%	19.17% ± 0.00%	7206.9525 ± 0.0000	459 ± 0
100	0.4	0.0247 ± 0.0000	0.02% ± 0.00%	14.10% ± 0.00%	6115.5771 ± 0.0000	335 ± 0
1	1	1.9346 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10022.3125 ± 0.0000	663 ± 0
100	0.1	9.8656 ± 0.0000	0.00% ± 0.00%	3.39% ± 0.00%	703.8516 ± 0.0000	36 ± 0

bcw (weak hypotheses' average result graphs for $T \times \rho$)


Alpha


Error Rate (%)

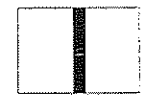

Support Vectors (%)

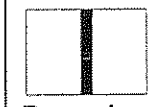

Norm


SMO Iterations

bcw (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 97.62% ± 0.00%


Good Weak Hyp. (%): 100.00% ± 0.00%


Execution Time (s): 0.04 s ± 0.00 s

5.3.3 Results for SMO-B$_\gamma$

SMO-B$_\gamma$ is the first hybrid algorithm that attempts to merge components from SMO together with AdaBoost.M1, instead of simply integrating them as in **SMO-B$_\alpha$** and **SMO-B$_\beta$**. Results for **SMO-B$_\gamma$** are shown in Tables 5.7 and A.4. As in Section 5.3.2,

the format of these two tables is the very same as the one described in Section 5.3.1, also used in Tables 5.5 and A.2. Notice that Table 5.7 brings complete results for the **bcw** dataset only, where Table A.4 brings complete results for all datasets.

The criteria for selecting the best configuration were the same as described in Section 5.3.2 for **SMO-B**$_\beta$, where the same mechanism of ignoring invalid weak hypotheses without halting the algorithm was used and considered by the last selection criterion.

Finally, although we managed to experiment **SMO-B**$_\gamma$ with all 22 databases, executions for 8 of them were not repeated more than once due to computational resource limitations. For these databases, namely **cdges**, **gauss**^{0}, **gauss**^{1}, **gauss**^{2}, **musk**, **spiral**^{0}, **spiral**^{1}, and **spiral**^{2}, standard deviations show as zero.

Table 5.7: Average and best results for hybrid algorithm **SMO-B**$_\gamma$ after 10 rounds of experiments with distinct training and testing sets.

bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	96.29% ± 1.14%	18.15% ± 2.54%	35.43 s ± 4.12 s
1	0.4	90.24% ± 9.43%	100.00% ± 0.00%	0.05 s ± 0.01 s
1	0.1	95.52% ± 2.38%	100.00% ± 0.00%	0.03 s ± 0.01 s
1000	0.1	96.14% ± 1.12%	32.91% ± 4.01%	18.06 s ± 0.98 s
1000	**0.7**	**96.52% ± 1.17%**	**14.37% ± 3.00%**	**45.96 s ± 4.66 s**
100	1	96.19% ± 1.11%	21.40% ± 4.92%	5.31 s ± 0.60 s
1	0.7	91.19% ± 7.89%	100.00% ± 0.00%	0.06 s ± 0.01 s
1000	1	96.29% ± 0.85%	12.07% ± 4.28%	51.64 s ± 5.96 s
100	0.7	96.33% ± 1.04%	23.00% ± 3.69%	4.64 s ± 0.56 s
100	0.4	96.19% ± 1.02%	27.90% ± 5.56%	3.67 s ± 0.43 s
1	1	91.33% ± 8.41%	100.00% ± 0.00%	0.08 s ± 0.01 s
100	0.1	96.38% ± 1.19%	45.30% ± 7.84%	1.83 s ± 0.10 s

bcw (strong hypotheses' average result graphs for $T \times \rho$)


Accuracy (%)


Good Weak Hyp. (%)


Execution Time (s)

Table 5.7: (continued)

bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.3065 ± 0.0576	0.16% ± 0.03%	25.54% ± 3.33%	639.6772 ± 71.2488
1	0.4	1.3624 ± 0.4054	0.19% ± 0.19%	15.76% ± 3.36%	399.5661 ± 122.6807
1	0.1	1.5457 ± 0.2571	0.12% ± 0.07%	5.43% ± 1.57%	194.0201 ± 111.1667
1000	0.1	-0.0853 ± 0.0188	0.33% ± 0.05%	13.26% ± 0.83%	400.7593 ± 26.2831
1000	**0.7**	**-0.3853 ± 0.0892**	**0.13% ± 0.02%**	**32.47% ± 3.74%**	**725.3301 ± 97.6372**
100	1	-0.1961 ± 0.0672	0.12% ± 0.02%	35.89% ± 4.82%	756.3575 ± 114.6263
1	0.7	1.3731 ± 0.4324	0.19% ± 0.17%	26.38% ± 5.13%	526.2639 ± 156.5876
1000	1	-0.4039 ± 0.0785	0.13% ± 0.02%	35.92% ± 4.73%	789.2475 ± 96.5533
100	0.7	-0.2106 ± 0.0500	0.14% ± 0.03%	31.87% ± 4.38%	714.4174 ± 104.9769
100	0.4	-0.1293 ± 0.0438	0.15% ± 0.03%	25.65% ± 3.48%	626.1287 ± 79.9194
1	1	1.3330 ± 0.4916	0.22% ± 0.22%	37.71% ± 5.19%	656.4435 ± 164.6508
100	0.1	-0.0144 ± 0.0393	0.35% ± 0.08%	12.67% ± 0.67%	398.4358 ± 21.4161

bcw (weak hypotheses' average result graphs for $T \times \rho$)


Alpha


Error Rate (%)

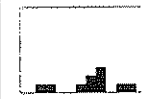

Support Vectors (%)


Norm

bcw (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 96.52% ± 1.17%


Good Weak Hyp. (%): 14.37% ± 3.00%


Execution Time (s): 45.96 s ± 4.66 s

5.3.4 Results for SMO-B_δ

As SMO-B_β succeeded SMO-B_α, SMO-B_δ is the natural evolution of SMO-B_γ, tightening even more the merger between SMO and AdaBoost.M1. Results for SMO-B_δ are shown in Tables 5.8 and A.5. Again, the format of these two tables is the

very same as the one described in Section 5.3.1, previously used in many other tables. While Table 5.8 is restricted to the **bcw** dataset only, complete results for all databases are presented in Table A.5.

Again, the criteria for selecting the best configuration were the same as described in Section 5.3.2 for SMO-B_β, and later in Section 5.3.3 for SMO-B_γ. Finally, although we managed to experiment SMO-B_δ with all 22 databases, executions for 8 of them were not repeated more than once due to computational resource limitations. For these databases, namely **cdges**, **gauss**^{0}, **gauss**^{1}, **gauss**^{2}, **musk**, **spiral**^{0}, **spiral**^{1}, and **spiral**^{2}, standard deviations show as zero.

Table 5.8: Average and best results for hybrid algorithm SMO-B_δ after 10 rounds of experiments with distinct training and testing sets.

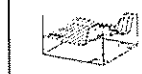
bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	96.14% ± 1.45%	70.54% ± 1.46%	68.12 s ± 0.91 s
1	0.4	92.67% ± 4.41%	100.00% ± 0.00%	0.03 s ± 0.01 s
1	0.1	91.86% ± 6.38%	100.00% ± 0.00%	0.02 s ± 0.01 s
1000	**0.1**	**96.29% ± 1.63%**	**69.17% ± 1.54%**	**17.62 s ± 0.16 s**
1000	0.7	96.00% ± 1.21%	71.67% ± 2.46%	118.28 s ± 2.35 s
100	1	96.14% ± 1.56%	80.40% ± 5.02%	16.08 s ± 0.38 s
1	0.7	92.14% ± 5.98%	100.00% ± 0.00%	0.03 s ± 0.01 s
1000	1	95.81% ± 1.24%	73.61% ± 1.94%	168.21 s ± 3.33 s
100	0.7	96.00% ± 1.45%	80.10% ± 3.21%	11.29 s ± 0.22 s
100	0.4	96.19% ± 1.52%	80.20% ± 2.60%	6.51 s ± 0.10 s
1	1	94.76% ± 3.47%	100.00% ± 0.00%	0.03 s ± 0.00 s
100	0.1	96.14% ± 1.35%	74.50% ± 3.50%	1.73 s ± 0.05 s

bcw (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)



Execution Time (s)

Table 5.8: (continued)

bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0646 ± 0.0079	0.96% ± 0.05%	23.46% ± 2.81%	825.2713 ± 116.4942
1	0.4	1.3973 ± 0.3459	0.16% ± 0.11%	7.71% ± 2.86%	211.5841 ± 34.3790
1	0.1	1.3876 ± 0.3919	0.18% ± 0.15%	5.05% ± 1.82%	130.3803 ± 49.4667
1000	**0.1**	**0.0599 ± 0.0086**	**0.97% ± 0.03%**	**11.77% ± 0.58%**	**489.3496 ± 42.2209**
1000	0.7	0.0732 ± 0.0151	0.94% ± 0.06%	27.19% ± 3.75%	1253.2555 ± 238.3426
100	1	0.1546 ± 0.0249	0.83% ± 0.11%	39.98% ± 5.36%	2002.2801 ± 406.5583
1	0.7	1.3892 ± 0.3426	0.17% ± 0.12%	10.00% ± 3.99%	276.8681 ± 123.6461
1000	1	0.0808 ± 0.0158	0.91% ± 0.08%	29.69% ± 4.23%	1721.3789 ± 362.4685
100	0.7	0.1459 ± 0.0226	0.86% ± 0.08%	34.93% ± 4.44%	1395.4959 ± 207.6325
100	0.4	0.1355 ± 0.0149	0.86% ± 0.08%	27.08% ± 2.33%	900.5703 ± 129.9236
1	1	1.5918 ± 0.2485	0.10% ± 0.06%	13.24% ± 4.63%	340.5600 ± 95.3159
100	0.1	0.1214 ± 0.0182	0.92% ± 0.05%	11.65% ± 0.44%	447.1555 ± 56.9464

bcw (weak hypotheses' average result graphs for $T \times \rho$)

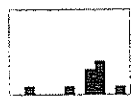

Alpha


Error Rate (%)

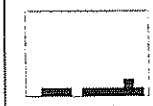

Support Vectors (%)


Norm

bcw (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 96.29% ± 1.63%


Good Weak Hyp. (%): 69.17% ± 1.54%


Execution Time (s): 17.62 s ± 0.16 s

5.4 Discussion of Results

In this section we discuss the results partially presented in Sections 5.2 and 5.3, and further completed for all databases in Appendix A. We first highlight the main findings of the three steps of preliminary analysis for each of the databases, namely

the step that explores linear separability of data and the other two that attempt to properly tune SVM parameters. Secondly, we comment on the results of the four hybrid algorithms proposed in Chapter 4, which are the main focus of this work.

5.4.1 Linear Separability

The aim of the experiment described in Section 5.2.1 was to determine the linear separability of databases. Hence, we may divide all databases into two distinct groups, one for the linearly separable and another for those non linearly separable. All databases found to be linearly separable were **cdges**, **gauss**^{0}, **ionosphere**, **musk**, **pgs**, **spect**^{r}, and **twonorm**, where all remaining databases were not linearly separable.

It is important to describe the criterion with which these databases were just classified. Notice that since we use a single-neuron Perceptron network, the Perceptron training algorithm will only stop when either all training instances are correctly classified, or the maximum number of iterations is reached. Furthermore, we repeated the experiment with each database 10 times, each time drawing new training and testing sets from the original dataset. Even though stating with 100% accuracy that these database were linearly separable was only possible if each complete dataset was used the training set for these experiments[3], we consider low the probability of repeatedly drawing degenerated training and testing sets that might have led to false conclusions. Therefore, those datasets where the Perceptron algorithm managed to halt before reaching the maximum iteration stop criterion are considered as linearly separable.

For those linearly separable databases such as, for instance, **gauss**^{0}, it may seem rather trivial employing more advanced learning machines to solve a trivial binary classification problem. Nonetheless, there are two reasons for doing so. First, there are groups of similar databases where only one member of these groups is linearly separable. In these cases, it is interesting to analyze the change in behavior of learning algorithms from one group member to another, regardless of the triviality of some databases. One such example is database **gauss**^{0}, which is similar to the non linear separable databases **gauss**^{1} and **gauss**^{2}. The trivial linear result of this experiment for **gauss**^{0} may be observed in Figure 5.10, in a similar dataset whose points where drawn with the same Gaussian probability distributions.

Databases **gauss**^{1} and **gauss**^{2}, on the other hand, impose an unbeatable challenge on linear learning algorithms. Since these two problems contain overlapping

[3]We chose to draw distinct non-overlapping training and testing sets for these experiments in order to be able to measure the generalization abilities of the Perceptron neuron and training algorithm for the databases used.

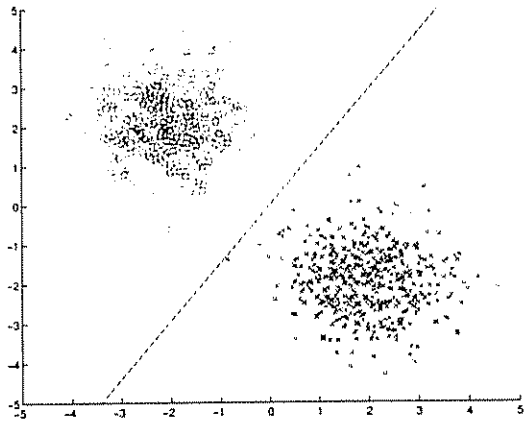


Figure 5.10: Experiment with linear discriminant (single-neuron Perceptron network) over a linearly separable problem similar to **gauss**^{0}.

classes, linear discriminants will never reach a solution with zero training error. Specifically in the case of Perceptrons, the criterion of maximum training epochs must be used to avoid the training algorithm never ending, where the separating line floats around the overlapping area between the two classes and no 100% accurate solution is ever found. The result of this experiment may be observed in Figure 5.11 in a problem similar to **gauss**^{1}.

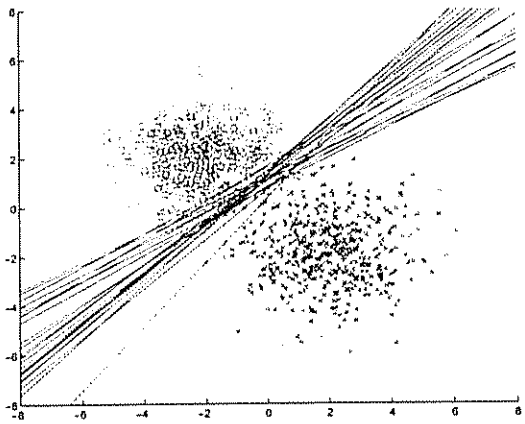


Figure 5.11: Experiment with linear discriminant (single-neuron Perceptron network) over a problem similar to **gauss**^{1}.

The second reason for conducting experiments on linearly separable problems is to examine the behavior of the algorithms proposed for databases with particular characteristics. Take, for instance, the **cdges** database, which is composed

of vectors with 16063 dimensions. Despite its apparent triviality, the study of this database in this work is still very relevant to evaluate the effects of such high dimensionality on the proposed algorithms, which is known to be one of the key factors that limit the expression power of many learning machines, as well as significantly increase their training times.

5.4.2 SVM Tuning

In order to execute the experiments with hybrid algorithms, we first investigated which SVM parameters would best suit each of the databases used. The goal of this analysis was to determine good values for each of the three parameters specific to the standard version of SMO, namely the parameter for trade-off between training error and margin (C), and the two RBF kernel parameters. Each of these 3-tuples, for each dataset, was used as the basis of the configuration of all following hybrid algorithm experiments, where we varied other parameters with respect to Boosting and the interface between Boosting and the SVM.

This tuning process was divided into two steps, one for coarse-grained tuning and another for fine-grained tuning. The coarse-grained tuning step, described in Section 5.2.2, aimed to quickly evaluate whether the default SMO parameter set could be used to solve each of the problems selected. This default parameter set consisted of using an RBF kernel with a variance parameter $p_{\sigma^2} = 1.000$ and scale parameter $p_s = 1.000$. In order to speed up experiments, all databases were cropped to at most 100 instances, where the standard 30%/70% ratio between testing and training sets, respectively, was maintained. Although speeding up this step was rather beneficial, it caused a degradation of results for 8 databases, namely **cdges**, **gauss**^{0}, **gauss**^{1}, **gauss**^{2}, **musk**, **spiral**^{0}, **spiral**^{1}, **spiral**^{2}. These results are partially presented in Table 5.2. Since **SMO** failed to perform well for these cropped datasets, producing error rates of 100%, we attempted the same experiments with these entire datasets, regardless of how many instances they had. Results for this experimental follow-up are partially presented in Table 5.3. Notice that for these two coarse-grained experiments, each experiment was executed only once. Thus, standard deviations for all results collected show zero.

Coarse-grained tuning yielded that all 22 databases selected were learned by a Support Vector Machine using an RBF kernel. This result was most welcome and necessary to both simplify the testing of algorithms without worrying about switching between machines with different kernels, and also assure that each and every dataset used could be learned with reasonable generalization by an SVM.

Now that at least the kernel type for all problems was set to be RBF, we proceeded with the fine-grained tuning of SVMs for each dataset. Again, we used **SMO**

and varied the possible configurations using different values for the three parameters described above. Values for C were chosen from $\{10^{-3}, 10^{0}, 10^{3}\}$, and values for both RBF kernel parameters were also chosen from $\{10^{-3}, 10^{0}, 10^{3}\}$. In order to increase the statistical relevance of the experiment, each configuration was run over all datasets for 10 times. For illustration purposes, partial results for this experiment are shown in Table 5.4.

After the execution of all tests, we determined which configurations were best suited to handle each learning problem using an RBF-based SVM. The criteria for ranking each configuration, previously described in Section 5.2.3, were based on each one's average outcome for accuracy and execution time. One example of such best configuration is highlighted in Table 5.4. Interestingly, although a few configurations were repeatedly chosen as best in more than one dataset, no obvious pattern was obtained from all best configurations, which widely explored the range of possible parameter combinations.

Another objective of enduring this SVM tuning procedure, aside the determination of parameter configurations itself, was to set benchmarks for the standard **SMO** algorithm, both in terms of accuracy and execution time. Except in datasets with excessive added noise, such as **chess**[2], **SMO** managed to yield reasonably good accuracy results, all above 75%, most within the $85\% - 100\%$ range. The remarkable outlier for this good performance was the spiral-function family of databases, **spiral**[0], **spiral**[1], and **spiral**[2]. Although the accuracy rates obtained in these experiments was rather poor, roughly around 50%, similar databases were reportedly well learned by SVMs with RBF kernels in literature. Motivated by these studies, we attempted to fine-tune the SVM with even finer-grained parameters in order to improve accuracy. We found that this problem is particularly sensible to all SVM parameters, where small variations of 10^{-3} often yielded significantly different results. Since in our automated tuning procedure we varied our parameters by 10^{3}, 6 orders of magnitude more than these variations, it is clear that a more sophisticated heuristic for parameter tuning would certainly enhance the performance of **SMO**. Since these parameter-determination heuristics are not the focus of this work, we solely relied on our own not-so-fine-grained tuning procedure, where the performance of hybrid algorithms was to be verified even in cases with badly-tuned parameters.

5.4.3 SMO-B$_\alpha$ and SMO-B$_\beta$: A Simple Hybrid Algorithm and its Evolution

The first couple of hybrid algorithms we experimented with were **SMO-B$_\alpha$** and **SMO-B$_\beta$**, proposed in Section 4.3.1 and 4.3.2, respectively. The **SMO-B$_\alpha$** algorithm is the simplest and most naive form of integrating Boosting and SVMs, and **SMO-B$_\beta$**

may be considered as an evolution of the original **SMO-B$_\alpha$** toward efficiency. **SMO-B$_\alpha$** is the classical implementation of AdaBoost.M1 with SMO as a weak learner, where **SMO-B$_\beta$** proposes its enhancement by extending the original probabilistic selection mechanism.

Using the best configurations found for each dataset during the tuning procedure previously described, hybrid algorithms were set up to execute with different configurations for two regularization parameters, one for the number of weak hypothesis to be evaluated, T, and another for the fraction of the training set to be selected by the booster and presented to the weak learner, T. T was chosen from set $\{1, 100, 1000\}$, while ρ was chosen from set $\{0.1, 0.4, 0.7, 1.0\}$. All 12 parameter combinations were merged with the previous best configurations and executed over each dataset.

Notice that due to limitations in computational resources, we could not afford, time wise, to repeat all experiments 10 times as originally planned. Instead, most experiments for **SMO-B$_\alpha$** and **SMO-B$_\beta$** were executed only once, as one may easily infer from some zero standard deviation values in Tables A.2 and A.3. Additionally, due to the same reason, we have also omitted specific experiments with **SMO-B$_\alpha$** and **SMO-B$_\beta$** over a few databases. The complete results for all databases used to evaluated these two hybrid algorithms are presented in Table A.2 for **SMO-B$_\alpha$**, and Table A.3 for **SMO-B$_\beta$**.

Although the construction of **SMO-B$_\alpha$** was well supported by concepts from both AdaBoost.M1 and SMO, its results were far from ideal, almost always performing worse than the standard **SMO**. Schapire [Sch02] advocates that Boosting may fail to perform well when overly complex weak learners are used, so one may wonder if that may be the cause for **SMO-B$_\alpha$**'s failure since SVMs are rather complex learning machines. A deeper investigation of the results in Table A.2 explained how such phenomenon may take place, where most weak hypotheses generated by SMO with the subsets drawn by AdaBoost.M1 where either invalid or saturated. Recall from Chapter 2 that weak hypotheses for binary classification problems must produce error rates of at most 50%. Since many hypotheses did not make such requirement, the original implementation of AdaBoost.M1 in **SMO-B$_\alpha$** halted the algorithm once an invalid weak hypothesis was built, thus yielding very poor results. Furthermore, another phenomenon, this time numerical, made that many null hypotheses with norm zero, obtained from training with degenerate cases of training subsets, were saturated with the largest possible weight values for ω_t. These degenerate cases, in turn, were obtained by **SMO-B$_\alpha$**'s selection-with-replacement mechanism. Recall from Chapter 3, Equation (3.81) and the SMO algorithm in Figure 3.5.3, that SMO may reach a division by zero if two input instances refer to equal input vectors.

Therefore, when using selection with replacement to select the training subset, repeated instances were often presented to SMO, which produced ill weak hypotheses. These ill weak hypotheses were wrongfully saturated with very high weights by the AdaBoost grading mechanism, hence totally biasing the vital principle of majority voting.

In an attempt to solve this problem, **SMO-B$_\beta$** was proposed with a selection-without-replacement selection mechanism. This new mechanism makes it impossible to insert repetitions in SMO's training subset unless the original training set contains duplicates. An extra step was also added to the modified hybrid algorithm, allowing for invalid weak hypotheses to be ignored without halting the algorithm. The results for **SMO-B$_\beta$**, presented in Table A.3, show that it often outperforms **SMO** both in terms of accuracy and execution time, thus validating the idea that the appropriate combination of Boosting and SVMs may lead to better learning algorithms that not only have better generalization abilities than the standard SMO, but also execute in less time. Despite these promising results, specially for the enhanced accuracy compared to that of **SMO**, we noticed that most training times were not faster to **SMO**'s, although there were a few exceptions. On the contrary, for cases with large values for T, **SMO-B$_\beta$** required a substantial amount of time to complete, some runs lasting for hours and even days [4].

A complete comparison of results between all hybrid algorithms and **SMO**, both in terms of accuracy and execution time, is later presented in Section 5.4.5.

5.4.4 SMO-B$_\gamma$ and SMO-B$_\delta$: Hybrid Algorithms with Merged Components

Finally, after discussing the promising results of the first two hybrid algorithms in Section 5.4.3, we now discuss the results for **SMO-B$_\gamma$** and **SMO-B$_\delta$**, two hybrid algorithms built with much tighter integration between AdaBoost.M1 and SMO components. Instead of simply integrating AdaBoost.M1 and SMO using a weak learner interface with minor modifications, the structural changes introduced by **SMO-B$_\gamma$** and **SMO-B$_\delta$** take apart and eliminate some of the main building blocks from its two parent algorithms. **SMO-B$_\gamma$** changes the paradigm of selecting a subset using the distribution maintained by the booster, where this same distribution is used to draw individual instances from the whole training set. Moreover, **SMO-B$_\gamma$** eliminates the first SMO selection heuristic, which implemented one of SMO's checking mechanisms to ensure that all KKT-violating instances were selected for analytical optimization. Extending this concept even further, **SMO-B$_\delta$** disposes of both first

[4]Please refer to Appendix B for notes on program instrumentation and execution environment.

and second SMO selection heuristics, relying the very convergence of the machine, originally achieved by selecting two training instances that most contribute toward the optimal solution, to the probabilistic selection mechanism fed by the booster.

When experimenting with SMO-B$_\gamma$ and SMO-B$_\delta$, as with SMO-B$_\alpha$ and SMO-B$_\beta$, we used the best configurations found for each dataset during the SVM tuning procedure to execute the algorithms with different configurations for two regularization parameters, one for the number of weak hypothesis to be evaluated, T, and another for the fraction of the training set to be selected by the booster and presented to the weak learner, T. T was chosen from set $\{1, 100, 1000\}$, while ρ was chosen from set $\{0.1, 0.4, 0.7, 1.0\}$. All 12 parameter combinations were merged with the previous best configurations and executed over each dataset.

Also analogously to the first two hybrid algorithms, due to limitations in computational resources we could not afford, time wise, to repeat all experiments with all datasets 10 times. Experiments with 8 databases, namely **cdges**, **gauss**^{0}, **gauss**^{1}, **gauss**^{2}, **musk**, **spiral**^{0}, **spiral**^{1}, **spiral**^{2}, were repeated only once, where those with the remaining 14 datasets were carried out 10 times as planned. The complete results for all databases used to evaluated these two hybrid algorithms are presented in Table A.4 for SMO-B$_\gamma$, and Table A.5 for SMO-B$_\delta$.

Despite the rather drastic approach of eliminating the first SMO heuristic, we found that SMO-B$_\gamma$ performed remarkably well, outperforming SMO's accuracy in 11 datasets, exactly 50% of the databases. Those in which SMO performed better, we found that the difference between accuracy results obtained with SMO and SMO-B$_\gamma$ were marginal, validating the robustness of the novel hybrid learning machine. In terms of execution time, SMO-B$_\gamma$ was faster than SMO in only 4 of the 22 databases.

Results for SMO-B$_\delta$ were equally good, specially considering its even more drastic approach of eliminating SMO strategies and relying solely on Boosting's probabilistic selection mechanism. Like SMO-B$_\gamma$, SMO-B$_\delta$ outperformed SMO for 11 datasets, the same number of datasets in which it outperformed SMO-B$_\gamma$ itself. Although slightly faster than SMO-B$_\gamma$, SMO-B$_\delta$ was still slower than SMO in average, where it manage to execute in less time only over 6 databases.

Besides comparing the overall performance of all algorithms with respect to the number of datasets with which they obtained good results, we may also identify recurring patterns in these datasets that highlight strengths and weaknesses of each algorithm. For instance, SMO-B$_\gamma$ and SMO-B$_\delta$ were consistently more accurate than SMO in some synthetic problems with artificial noise added, such as the chess family and the spiral family, SMO-B$_\delta$ even more so than SMO-B$_\gamma$. Interestingly, this trend was reversed for the synthetic databases with multivariate

normal distributions, **ringnorm** and **twonorm**, which were also one order of magnitude slower than **SMO** on both **SMO-B**$_\gamma$ and **SMO-B**$_\delta$. Dimensionality of datasets also seems to greatly impact the time performance of algorithms, where for the 16063-dimensional **cdges** database, **SMO-B**$_\gamma$ executed more than 10 times faster than **SMO**, and 20 times faster than **SMO-B**$_\delta$.

It is worthwhile pointing out that, despite accuracy differences between **SMO** and the four hybrid algorithms proposed, these 5 algorithms most often outperformed other studies published with these datasets, described in Section 5.1. One such exception is the **cdges** database, previously analyzed in Section 5.2.1, that despite being linearly separable, did not produce good results with any the SVM setups used.

A complete comparison of results between all hybrid algorithms and **SMO**, both in terms of accuracy and execution time, is presented next in Section 5.4.5.

5.4.5 Performance Summary

This section brings performance summary tables that describe the best results obtained with each one of the algorithms experimented with over all 22 datasets, namely **SMO**, **SMO-B**$_\alpha$, **SMO-B**$_\beta$, **SMO-B**$_\gamma$, and **SMO-B**$_\delta$. These best results were selected from the complete set of experimental results described in Appendix A using the sorting criteria previously described in Sections 5.2.3 and 5.3.

Table 5.9 compares the accuracy of the best results for all algorithms, and Table 5.10 compares their execution times. Notice that the entries in Table 5.10 are not the fastest execution times obtained with each algorithm. Instead, these entries represent fastest execution times obtained with the most accurate configurations, as implicitly pointed out by the sorting criteria.

Notice that some cells in Tables 5.9 and 5.10 are left blank, where not all datasets were able to be executed by all algorithms due to computational resource limitations. For this same reason, some experiments which were originally planned to be repeated 10 times were executed only once. These experiments are identified by the standard deviation of some of their measures, which therefore are equal to zero.

Table 5.9: Accuracy summary for best configuration results obtained with SMO, SMO-B_α, SMO-B_β, SMO-B_γ, and SMO-B_δ.

Dataset	SMO	SMO-B_α	SMO-B_β	SMO-B_γ	SMO-B_δ
bcw	96.10% ± 0.97%	87.71% ± 12.38%	97.62% ± 0.00%	96.52% ± 1.17%	96.29% ± 1.63%
cdges	80.48% ± 3.33%	70.24% ± 0.00%	77.38% ± 0.00%	77.38% ± 0.00%	83.33% ± 0.00%
chess^0	93.77% ± 1.38%	85.63% ± 8.01%	-	93.90% ± 1.49%	93.20% ± 1.68%
chess^1	85.37% ± 1.41%	78.60% ± 4.76%	-	84.57% ± 1.65%	82.40% ± 1.54%
chess^2	68.37% ± 1.26%	64.33% ± 3.91%	-	66.53% ± 2.17%	66.63% ± 2.40%
gauss^0	100.00% ± 0.00%	100.00% ± 0.00%	100.00% ± 0.00%	100.00% ± 0.00%	100.00% ± 0.00%
gauss^1	98.57% ± 0.68%	-	-	98.67% ± 0.00%	99.33% ± 0.00%
gauss^2	92.07% ± 1.05%	-	-	93.00% ± 0.00%	93.00% ± 0.00%
hepatitis	80.64% ± 4.61%	-	-	78.30% ± 6.51%	75.96% ± 6.32%
ionosphere	95.09% ± 1.83%	-	-	95.00% ± 0.95%	94.72% ± 2.74%
musk	93.01% ± 2.32%	-	-	93.01% ± 0.00%	93.71% ± 0.00%
pgs	81.25% ± 8.95%	-	87.50% ± 0.00%	88.12% ± 4.80%	82.50% ± 5.96%
pid	76.80% ± 2.51%	-	75.32% ± 0.00%	75.71% ± 2.02%	76.19% ± 2.28%
ringnorm	98.47% ± 0.70%	-	-	98.13% ± 0.92%	97.90% ± 0.56%
spect^b	76.36% ± 0.21%	-	-	77.81% ± 0.36%	78.50% ± 3.31%
spect^r	82.53% ± 0.00%	-	-	84.31% ± 2.88%	82.86% ± 1.65%
spiral^0	57.07% ± 2.63%	-	-	61.00% ± 0.00%	62.67% ± 0.00%
spiral^1	52.63% ± 1.39%	-	-	56.67% ± 0.00%	54.67% ± 0.00%
spiral^2	53.17% ± 1.21%	-	-	52.67% ± 0.00%	51.00% ± 0.00%
twonorm	97.97% ± 0.67%	-	-	97.50% ± 0.70%	96.67% ± 0.80%
wdbc	95.38% ± 1.37%	-	-	93.80% ± 2.01%	94.04% ± 1.47%
wpbc	79.83% ± 5.65%	-	-	75.67% ± 7.54%	77.67% ± 5.39%

Table 5.10: Execution time summary for best configuration results obtained with SMO, SMO-B_α, SMO-B_β, SMO-B_γ, and SMO-B_δ.

Dataset	SMO	SMO-B_α	SMO-B_β	SMO-B_γ	SMO-B_δ
bcw	5.46 s ± 2.40 s	0.03 s ± 0.03 s	0.04 s ± 0.00 s	45.96 s ± 4.66 s	17.62 s ± 0.16 s
cdges	158.45 s ± 21.21 s	28.82 s ± 0.00 s	97.81 s ± 0.00 s	13.37 s ± 0.00 s	282.34 s ± 0.00 s
chess^0	79.22 s ± 3.54 s	25.25 s ± 10.94 s	-	79.69 s ± 0.46 s	355.78 s ± 3.48 s
chess^1	79.74 s ± 17.01 s	9.84 s ± 9.01 s	-	751.56 s ± 17.27 s	57.81 s ± 1.61 s
chess^2	62.79 s ± 3.19 s	14.96 s ± 9.49 s	-	77.71 s ± 0.80 s	35.91 s ± 0.16 s
gauss^0	0.04 s ± 0.01 s	0.06 s ± 0.00 s	0.02 s ± 0.00 s	0.02 s ± 0.00 s	0.02 s ± 0.00 s
gauss^1	0.43 s ± 0.13 s	-	-	44.60 s ± 0.00 s	10.69 s ± 0.00 s
gauss^2	0.62 s ± 0.11 s	-	-	42.11 s ± 0.00 s	11.22 s ± 0.00 s
hepatitis	0.01 s ± 0.00 s	-	-	1.03 s ± 0.04 s	3.31 s ± 0.03 s
ionosphere	1.00 s ± 0.43 s	-	-	2.71 s ± 0.05 s	2.21 s ± 0.02 s
musk	8.87 s ± 3.86 s	-	-	17.95 s ± 0.00 s	165.83 s ± 0.00 s
pgs	0.19 s ± 0.08 s	-	5.11 s ± 0.00 s	10.81 s ± 0.13 s	3.61 s ± 0.03 s
pid	8.28 s ± 1.56 s	-	88.99 s ± 0.00 s	9.73 s ± 0.13 s	117.75 s ± 0.77 s
ringnorm	21.23 s ± 5.62 s	-	-	328.96 s ± 6.99 s	283.86 s ± 3.70 s
spect^b	0.18 s ± 0.03 s	-	-	3.49 s ± 0.04 s	0.07 s ± 0.04 s
spect^r	0.27 s ± 0.02 s	-	-	0.05 s ± 0.00 s	0.17 s ± 0.03 s
spiral^0	38.72 s ± 27.51 s	-	-	127.82 s ± 0.00 s	353.34 s ± 0.00 s
spiral^1	5.96 s ± 0.85 s	-	-	13.11 s ± 0.00 s	64.38 s ± 0.00 s
spiral^2	171.59 s ± 2.66 s	-	-	49156.51 s ± 0.00 s	1.24 s ± 0.00 s
twonorm	0.72 s ± 0.03 s	-	-	9.20 s ± 0.36 s	46.22 s ± 0.16 s
wdbc	2.98 s ± 0.58 s	-	-	34.99 s ± 1.41 s	80.04 s ± 0.97 s
wpbc	4.84 s ± 3.30 s	-	-	0.21 s ± 0.01 s	5.01 s ± 0.04 s

Chapter 6

Conclusion

In the light of the results described and discussed in Chapter 5, the most fundamental conclusion of this work is that it is possible to assemble Boosting and Support Vector Machines together in hybrid algorithms that will often outperform other sophisticated learning methods such as Support Vector Machines themselves. This conclusion contradicts an important remark from the Boosting literature. Many authors have argued, including Schapire [Sch02], that overly complex weak hypothesis may fail to produce good strong hypothesis using a Boosting algorithm. This issue, previously discussed in Chapter 2, imposed a serious threat to this work, since SVMs are among the most sophisticated learning machines available [CST00]. Recall from Chapter 5 that the naive hybrid algorithm **SMO-B**$_\alpha$ failed to beat **SMO** practically in all databases. Nonetheless, we were able to locate the source of the problem and build a hybrid extension, **SMO-B**$_\beta$, with consistently better accuracy than that of **SMO**, thus contracting the warning by Schapire and others.

Another interesting point comes from the theory of Support Vector Machines, described in Chapter 3, which formulates the training of learning machines as solving convex constraint optimization problems, where an optimal solution is reached when the training algorithm converges. Based on the principle of structural risk minimization, SVMs are supposed to be the most accurate learning machines, i.e. with the greatest generalization abilities, given that SVM parameters and kernel are properly tuned for the problem in hand. Since tuning SVMs is often a problem as hard as solving the learning problem itself, it is expected that SVMs perform suboptimally. The results obtained in this work show that by using of principles such as those from Boosting theory, one is able to effectively boost the accuracy of SVMs to compensate for this lack of proper parameters.

This issue leads to another interesting conclusion. Regardless if coupled or not with Boosting in hybrid algorithms, one of the main practical problems of em-

bracing SVMs, as well as other kernel methods, as generally-preferred learning machines is their necessity to have machine and kernel parameters properly setup in order to perform well. In our experiments, we examined such problem for the spiral family of datasets, **spiral**^{0}, **spiral**^{1}, and **spiral**^{2}. These problems were fined-tuned by hand in a much tighter parameter range, roughly 6 orders of magnitude more precise. As a result, **SMO** was able to nearly achieve 100% accuracy in **spiral**^{0}. Since the tuning procedures undertaken in this work were based on a much coarser grain, **SMO** and all four hybrid algorithms proposed failed to perform well for this dataset, with best accuracy around 63%. The conclusion is hence that although these hybrid approaches tend to compensate for a slight misconfiguration of the machine and its kernel, big discrepancies are still the Achilles' heel of SVMs, together with Boosting or not.

Specific deficiencies known to Boosting and Support Vector Machines also appear to have benefited from the hybrid approach. According to Dietterich [Die00], Boosting is specially susceptible to noise, yet hybrid algorithms outperformed **SMO** in synthetic databases with artificial noise added. Furthermore, Support Vector Machine training algorithms, such as SMO, often suffer in terms of training time when datasets have great dimensionality, great cardinality, or both. As expected, the probabilistic selection strategy from Boosting greatly reduced the expensive evaluation of internal kernels, allowing hybrid algorithms to be faster than the standard **SMO** for high-dimensional datasets such as **cdges**.

Except for **SMO-B**$_\alpha$, which served as basis for the development of its customized counterpart, one of the most interesting properties observed for **SMO-B**$_\beta$, **SMO-B**$_\gamma$, and **SMO-B**$_\delta$ is the robustness associated with the high quality of the solutions obtained. For all datasets experimented, these algorithms either outperformed or were marginally worse than each other and than the standard **SMO**. This robustness in accuracy is most desirable, meaning the algorithms tend to converge to narrow near-optimal ranges. Considering that one of Boosting's properties is to enhance accuracy as more weak hypotheses are evaluated, provided there is enough training data, these proposed algorithms have the potential to greatly outperform other standard learning machines, including SVMs. Although some algorithms did require a great amount of time to execute, it is important to point out that they were also consistently faster that **SMO** in different databases with specific properties, such as the **cdges** dataset just described.

The final and perhaps most important conclusion of this work has to do with the promising results already obtained and the development approach used so far. The development process carried out during the undertaking of this work was rather horizontal, meaning it spanned over many different possibilities, options, and even

datasets, in search for promising results that would lead the way to even more groundbreaking algorithms. Now that these promising results have been found, it is clear that many of the issues briefly tacked in this work deserve to be further explored in a more vertical approach. Therefore, the final conclusion is that the further exploration of strategies for combining Boosting and Support Vector Machines has a great potential to result in even better learning machines, both in terms of accuracy and execution time. Several suggestions for different types of follow-up works which may lead to yet more interesting conclusions and contributions are presented next in Chapter 7.

Chapter 7

Future Work

The results and conclusions of this work gave rise to a series of issues that suggest the further deepening of the research conducted so far. Due to time limitations, the investigation of these issues have been currently postponed, though it does not mean they are any less relevant or promising than the topics so far explored.

Some of these ideas are based on previous developments in either Support Vector Machine or Boosting literatures, where it is yet to be seen what their effect would be on algorithms that combine both together. Each one has the potential to not only increase the applicability of the algorithms developed up to now, but also to lead to new algorithms with better generalization and faster execution times. The following sections describe these ideas, and sow the seed to future developments and new research work.

7.1 Non-Euclidean Input Spaces

In real-world applications, there are many classification problems that cannot be encoded using Euclidean input spaces. Despite most often having a standard finite set of labels as output, the fact that their input space is non-Euclidean substantially narrows the availability of learning machines to solve them.

There are many papers in the Support Vector Machine literature about customized kernels that have been used to classify different kinds of input data. By using the principles of kernel induced feature spaces described in Section 3.3.2, SVMs are able to map non-Euclidean input spaces into linearly-separable feature spaces, thus making learning possible.

One of the applications with greatest appeal for such technique is text classification, where one must classify chunks of text according to their content. Note how difficult it is to encode textual information in Euclidean spaces, where the sense

of sentences represented by the selection and disposal of words in the text must be preserved. There are several examples of SVM uses in text classification problems, for instance the ones by Joachims [Joa98b] and Dumais, Platt et al. [DPHS98]. Furthermore, there have also been examples of the application of Boosting techniques to text classification, such as BoosTexter from Schapire and Singer [SS00], RankBoost from Iyer et al. [ILS+00], and a text filtering scheme from Schapire et at [SSS98]. The combination of these Boosting techniques with the appropriate SVM setup for text classification is yet to be experimented.

Another application that has earned the interest of the machine learning community is the classification of biological sequences, powered by the recent advances in bioinformatics and biological data acquisition techniques. Take, for instance, the problem of determining promoter gene sequences described in Section 5.1.7. Its encoding must not only consider which bases form the sequences, as we in purpose did, but also consider the order in which they appear. A much better approach than the one we adopted would have been the use of a customized kernel able to incorporate sequence information. Examples of such advanced kernel techniques are plenty in the bioinformatics literature, namely the work by Jaakkola and Haussler [JH99] using hidden Markov models in protein homology detection, the work by Watkins [Wat99a, Wat99b, Wat99c] using probabilistic context free grammars to build kernels, and the work by Haussler [Hau99] on kernels for discrete and sequential data such as biological sequences. Other examples are also discussed by Cristianini et al. [CST00].

The effect of using such customized kernels with the algorithms described in Chapter 4 is yet to be determined. Since using problem-specific kernels have the potential to increase the quality of each weak hypothesis evaluated by those algorithms, interesting observations may arise with respect to hypotheses weight distributions and the impact of the number of weak hypotheses on the quality of the strong hypothesis.

7.2 Sparse Vector Operations

A very common optimization of the standard SMO algorithm was proposed by Platt himself in [Pla98a]. It consists of implementing customized operations for dealing with sparse vectors more efficiently. According to Platt, much of the computation time of SMO is spent on kernel evaluations, which may be significantly improved for sparse data problems if customized operators are used. If the sparse data are binary, the optimization is even more efficient, where floating-point operations are replaced by simple increment and decrement operations.

Since one of the drawbacks of the algorithms proposed in Chapter 4 is their relatively large consumption of computer resources when repeatedly evaluating different weak hypotheses, the use of such small optimization techniques may lead to a great reduction in the algorithms' execution times, therefore potentially enhancing their overall applicability for large problems with sparse or binary sparse inputs.

7.3 Fixed-threshold SVMs

Platt describes the origins of SMO in [Pla98a], where he compares SMO to previous SVM and other optimization algorithms. According to him, SMO can be considered a special case of the Osuna algorithm, where the working set is of size two and both Lagrange multipliers are replaced with new multipliers at every step.

It is interesting, though, to consider a modification of SMO that uses a working set of size one, hence only replacing one Lagrange multiplier at each step. The immediate consequence of such change is that the SVM would not have the linear restrictions imposed by Equation (7.1) on the dual representation of the optimization problem (Chapter 3), therefore fixating the threshold variable.

$$\sum_{i=1}^{l} y_i \alpha_i = 0 \text{ and } 0 \leq \alpha_i \leq C \; \forall \; i = 1, 2, \ldots, l \tag{7.1}$$

The standard Lagrange multiplier update rule used by SMO, described in Equation (7.2), would therefore be replaced by the new rule described in Equation (7.3), which only considers one multiplier update at a time.

$$\alpha_2^{new} = \alpha_2^{old} - \frac{y_2 \left(E_1 - E_2\right)}{2k\left(\vec{x_1}, \vec{x_2}\right) - k\left(\vec{x_1}, \vec{x_1}\right) - k\left(\vec{x_2}, \vec{x_2}\right)} \tag{7.2}$$
$$\text{where } E_1 = f^{old}\left(\vec{x_1}\right) - y_i$$

$$\alpha^{new} = \alpha^{old} + \frac{y_1 E_1}{k\left(\vec{x_1}, \vec{x_1}\right)} \tag{7.3}$$
$$\text{where } E_1 = f^{old}\left(\vec{x_1}\right) - y_i$$

Platt states that the fixed-threshold version of SMO for a linear SVM is similar in concept to the Perceptron relaxation rule [Pla98a, Hay94], where the output of a node is adjusted every time there is an error, so its output lies over the separating margin. The drawback of this approach is exactly the loss of the maximum margin property of SVMs, which is the core of the structural risk minimization principle

described in Chapter 3. Its benefit, though, is that different selection heuristics for SMO may be used, only this time for selecting a single Lagrange multiplier instead of two. The impact of this major change in SMO's capacity to derive good hypothesis on its own must be evaluated separately before attempting to use it together with Boosting. Nevertheless, the potential for different coupling strategies of this single-multiplier selection strategy with Boosting selection mechanisms is great, specially considering that the probable loss of generalization caused by the structural risk minimization violation may be absorbed when combining weak hypotheses to derive a strong hypothesis.

Yet another potential benefit of using a fixed-threshold SMO, also a direct consequence of the single multiplier update rule, is described further ahead in Section 7.10.2.

7.4 Reweighting with SVMs

The AdaBoost algorithm described in Chapter 2 contains an implementation detail that consists of selecting the mechanism for exchanging weight information with the weak hypothesis generator. For learning algorithms that easily accept weighting parameters, this is done by maintaining weight values together with each instance of the training set, which is the booster's distribution vector $\mathbf{D}_t$. This weight vector is then passed to the weak hypothesis generator, which uses this information to bias its learning toward the instances with greater weight. This technique is referred to as *reweighting* [FS96a, FS99b].

Consider a Multi-Layer Perceptron neural network, for instance. The output of neurons on the first hidden layer may be evaluated as:

$$y = sign\,(\mathbf{wx} - b) \tag{7.4}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the weight vector associated with the neuron, and b is a threshold value. Applying reweighting for such simple algorithm would multiply $\mathbf{D}_t$ by each input vector $\mathbf{x}$, hence producing:

$$y = sign\,(\mathbf{wxD}_t - b) \tag{7.5}$$

For learning algorithms which do not accept weighting parameters, another means for passing this information along is necessary. *Resampling* is then used, which consists of selecting a subset of the training set using $\mathbf{D}_t$ as a probability distribution. This technique was used for interfacing SMO with AdaBoost in this work, since there is no obvious way for biasing SVM training on specific input vectors.

Further research into SMO mechanics may suggest an efficient way for using vector biasing information during SVM training. This modification of SMO for reweighting would be integrated with AdaBoost, as we already did using resampling, and its results would provide interesting data for comparison between reweighting and resampling when using SMO as a weak hypothesis generator. Furthermore, the combination of both techniques could be explored, evaluating the effects of overemphasizing the principle of weighted majority voting behind the Boosting theory.

7.5 Multiclass Classification

A natural extension of the standard binary classification problems we explored in this work would be the replacement of the output domain $\{-1, +1\}$ by a finite set of n labels $\{1, \ldots, n\}$. In order to do that, we would first need a Boosting algorithm and an SVM-based weak hypothesis generator able to solve such problems.

There are several methods of extending AdaBoost to work with multiclass problems [SS99, Sch02]. Some of them are simple generalizations of AdaBoost.M1 [FS95], while others divide a multiclass problem into multiple binary problems, such as Schapire and Singer's AdaBoost.MH [SS98] and Freund and Schapire's AdaBoost.M2 [FS95].

Just like Boosting algorithms, there are many Support Vector Machines techniques for transforming a multiclass problem into a set of binary problems. One such example is the *one versus all* (OVA) pairwise comparison technique, for instance used in [RTR+01] for classifying tissue samples among 15 different types of cancer.

The combination of such multiclass Boosting and Support Vector Machine algorithms may lead to an new class of algorithms able to solve a much wider range of problems than the binary version explored in this work.

7.6 Multilabel Classification

Another extension that follows the multiclass classification described in Section 7.5 is multilabel classification. Given a problem with a finite output set with n labels $\{1, \ldots, n\}$, multilabel classification would assign confidence levels to each of the n labels when presented a new data sample.

The most well-known multilabel extension of AdaBoost is AdaBoost.M2, due to Freund and Schapire [FS95, FS96a, FS99b]. AdaBoost.M2 extends the communication with the weak learner by considering a set of plausible labels from each

weak hypothesis instead of a single output label. Thus, each hypothesis outputs a vector $[0,1]^n$, where components close to 0 or 1 correspond to lower or higher degrees of plausibility, respectively. Also, AdaBoost.M2 replaces the usual prediction error of each hypothesis by a much more sophisticated error measure that Freund and Schapire refer to as *pseudo-loss*. The internal workings of AdaBoost.M2 are thoroughly described in [FS96a, SS99], where they are also compared to those of AdaBoost.M1.

As with Boosting, Support Vector Machines also must provide extensions to cope with confidence measures. One of the most straight forward confidence measures for SVMs is the Euclidean distance of a sample point to the separating hyperplane. A similar approach has been explored in [RTR+01] for choosing between results of machines trained in *one versus all* pairwise comparisons.

Similarly with multiclass classification, multilabel extensions to both Boosting and Support Vector Machines may be combined to deal of more sophisticated classification problems, increasing even more the applicability of the algorithms.

7.7 Regression Problems

Aside from exploring different input spaces, we may also tackle problems with different output spaces other than the discrete ones we have so far described. One of the most interesting applications of Support Vector Machines is on regression problems, that is, problems with output space in $[-\infty, +\infty]$ [CST00]. Many recent advances have also widened the applicability range of SVMs in regression problems, such as the work of Flake and Lawrence [FL01] to build a new version of SMO [Pla98a] for real-value output problems. Due to its structural risk minimization principle, SVMs are often found to outperform other regression machines.

Using Boosting on regression problems has also drawn much attention from the Machine Learning community. New papers and theses have been published [RMR99, Rät01], where previous methods of Boosting in classification problems have been adapted to work on regression as well. The combination of Boosting regression techniques with SVM regression may yield learning algorithms with a great potential for solving non-discrete problems.

7.8 Automatic Kernel and Parameter Selection

One of the decisions made in this work was that the selection of SVM kernels and kernel parameters would be fixed for each problem solved, thus sparing the Boosting algorithm from this decision. One of the newest research topics in the Support

Vector Machine community is the discovery of methods for automatically determining SVM kernels and kernel parameters.

Examples of such techniques are described by Jaakkola [JH98], Cristianini et al. [CSTC98], and Amari and Wu [AW89], which use decisions derived from the problem data. Other works explore empirical findings based on specific problems, such as Ali et al. [AA02]. Other examples of automatic kernel selection are discussed by Cristianini et al. in [CST00].

Another interesting area is determining the regularization parameter C of SVMs with soft margins. Pontil and Verri [PV98] of study the effects of C over the solution, where they find interesting results by identifying more relevant support vectors that play a key role in determining the optimal separating hyperplane of the problem.

It may be possible to derive techniques for automatic kernel and parameters selection and tuning based on Boosting principles, where instead of training weak hypotheses based on the reweighting or resampling of input vectors, a initial training step would take place for producing a strong hypothesis containing a refined kernel for the given problem. Among the multiple ways of translating such idea into a Boosting algorithm, one possibility would be to keep a set of possible kernel parameters from which the algorithm would use distribution values to select from and build new weak hypothesis at each step.

7.9 Study of Theoretical Bounds

One of the most promising post-studies of the algorithms proposed in this work is the derivation of their theoretical bounds on both training error and testing error (generalization). These derivations may take place in various ways, for instance exploring the theories of Support Vector Machines and Boosting, or even verifying the behavior of algorithms bound to well-known curves.

7.9.1 Bounds Imposed by Combining SVM and Boosting Theories

According to the concepts in Boosting theory based on the Probably Approximately Correct (PAC) model [Fre95, Vap95, CST00], AdaBoost.M1, which has been used in all algorithms described in Chapter 4 in its generalized form, outputs a strong hypothesis with the following training error upper bound:

$$\epsilon_H \leq \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp\left(-2 \sum_{t=1}^{T} \gamma_t^2\right) \tag{7.6}$$

where T is the number of weak hypotheses generated, $\gamma_t = 1/2 - \epsilon_t$, and ϵ_t is the error measure associated with weak hypothesis t [FS96a]. Schapire shows in [Sch02] that the customized selection of Z_t leads to narrower error bounds, hence improving the algorithms' convergence by producing better weak hypotheses. Notice, however, that the selection of Z_t depends on the output domain of each weak hypothesis $h_t(x)$. In order to determine and restrict the output of $h_t(x)$, Support Vector Machine theory may be used to derive consistent ways of ensuring restricted output bounds, thus improving the use of SVMs as better weak hypothesis generators.

Similar approaches may be used to restrict error bounds on Support Vector Machines, which in turn rely on the Vapnik Chervonenkis (VC) theory [Vap82, Vap95]. Several aspects of the VC theory provide tools for determining margin-based bounds on SVM generalization [CST00], which may be explored with the intent of combining weak hypotheses using weighted majority voting. This voting strategy is too powered by the PAC model, which coincidently is founded on the same principles as is the VC theory.

7.9.2 Investigating Behavior Bounded to L or Pareto Curves

A new stream of research work on learning machines has attempted to study generalization bounds using well-known curves. Examples of such work have been developed for Multi-Layer Perceptron neural networks using Pareto curves by Teixeira, Braga et al. [Tei01, TBTS00, TBTS01]. Their proposed Multi-objective algorithm minimizes the norm of MLP weight vectors along with the sum of squared errors to obtain Pareto-optimal solutions.

A similar approach may be derived for SVM algorithms as well as the combination of SVMs and Boosting described in this work. These bounds may follow a Pareto-curve as discovered for MLPs by Teixeira, Braga et al., or even other more sophisticated shapes such as L-curves [Han99]. Either way, such discovery would enhance both the knowledge and the applicability of the algorithms here proposed.

7.10 Parallel Architectures

In order to overcome the substantial consumption of computer resources demanded by the algorithms described in Chapter 4 when repeatedly evaluating different weak hypotheses, parallel implementation techniques may be used to take advantages of advanced architectures such as multi-processed machines and networks of clustered machines. Whether distributing processing among several microprocessors or sev-

eral nodes in a cluster, the principles and problems behind parallelizing Boosting and SMO still remain the same. Either way, such implementations may transform these algorithms into viable solutions for real-world applications with strong data and execution time constraints.

7.10.1 Parallelizing Boosting

Implementing parallel versions of Boosting algorithms, such as those from the the AdaBoost family, has no theoretical limitations whatsoever. Since the evaluation of each weak hypothesis is done independently from one another, threads or cluster nodes may keep copies of the entire set of input samples and evaluate separate batches of hypotheses in parallel. After a predetermined amount of hypotheses or time, these threads or cluster nodes would synchronize in order to produce a strong hypothesis.

This simple approach would reduce the weak hypothesis generation time of Boosting algorithms by a factor of $1/n$, where n is the number of processors or cluster nodes used. Notice, however, that neither the complexity of the algorithm nor the time required to combine all hypotheses would be reduced with such approach.

7.10.2 Parallelizing SMO

Implementing a parallel version of SMO, which is quite controversial since SMO stands for Sequential Minimal Optimization, has theoretical limitations that have not yet been overcome. As described in Chapter 3, SMO trains a Support Vector Machine by solving the corresponding QP problem with the decomposition method taken to its extreme, that is, by selecting and optimizing two Lagrange multipliers at each step [CST00, Bur98]. This way, each pair of multipliers is solved analytically and therefore ceases to violate Karush-Kuhn-Tucker (KKT) conditions.

The problem with parallel implementations of SMO lie exactly on the pairwise replacement of multipliers. Since every two multipliers in the original set may be selected by the SMO selection heuristics, it is highly likely that inconsistencies between two parallel instances are created. These inconsistencies, in turn, would result in the eventual optimization of multipliers selected with an inconsistent mate, allowing for former to violate KKT conditions as other multipliers change, even after being previously updated. Considering this difficulty, two approaches may be used to implement naive parallel versions:

- Consistently synchronizing the updated values of multipliers, thus avoiding all inconsistencies but at the same time eliminating almost all parallelism possible.

- Neglect these inconsistencies and bet that the benefits of evaluating Lagrange multipliers in parallel will overcome the hassle of re-evaluating all multipliers with violating KKT conditions during synchronization cycles.

Though it seams rather obvious that the former approach is doomed to fail, the latter has been attempted by Etin and Elias [EE00], which report marginal performance benefits over standard SMO with twice as much computer resources (a network with two identical computer nodes).

A novel approach to building parallel versions of SMO relies on the principles of using fixed-threshold SVMs, briefly described in Section 7.3. Since training fixed-threshold SVMs with a modified version of SMO implies that single Lagrange multipliers will be updated at each step, we now may distribute this processing across parallel units without the problem of incorrectly updating Lagrange multipliers due to inconsistencies in their pairs.

Notice that this approach also comes with the generalization drawbacks also described in Section 7.3. Nevertheless, considering the capabilities of Boosting algorithms to combine weak hypotheses into a strong hypothesis, evaluating poorer SVMs with parallel implementations may be not bring serious generalization decreases to the algorithm and yet improve its execution times.

Appendix A

Complete Results For All Datasets

This appendix brings the complete results for experiments with all datasets described in Chapter 5. These results have been omitted from the main text body due the great number of databases used and consequently their relatively large size. Section A.1 brings partial results for preliminary analyses previously described in Section 5.2, where only results for the linear discriminant analysis are shown. Complete results for the SVM tuning procedure were omitted due to their relatively large size, where we restrain ourselves to showing examples for datasets **bcw** and **cdges** in Sections 5.2.2 and 5.2.3. Section A.2 brings complete results with all datasets for the algorithms proposed in Chapter 4, whose experiments were first commented in Section 5.3. Besides the description of experiments distributed among Chapter 5 and Appendix A, algorithms' analyses and results are discussed in more detail in Section 5.4.

A.1 Preliminary Analysis

Table A.1 brings the complete results of the experiment using a Perceptron single-neuron network as a linear discriminant. Each databased was repeatedly analyzed with one Perceptron node for 10 times, each time with different selections of non-overlapping complementary training and testing sets corresponding to 70% and 30% of the dataset, respectively. The Perceptron node used a learning rate $\eta = 0.01$, an error tolerance $tol = 0.1$, and a maximum number of epochs $epoch_{max} = 100000$.

Table A.1: Average results for single-neuron Perceptron network after 10 rounds of experiments with distinct training and testing sets.

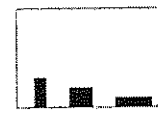
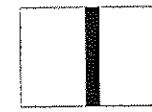
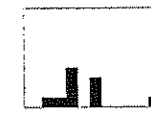
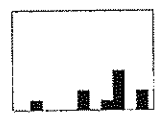
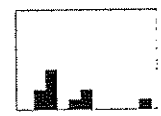
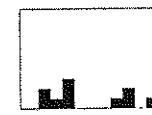
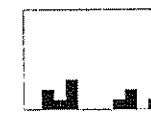
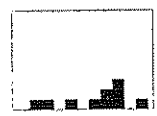
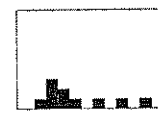
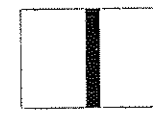
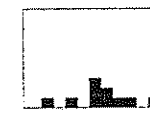
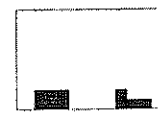
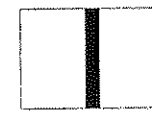
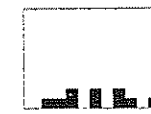
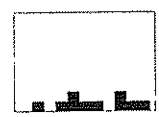
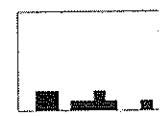
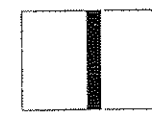
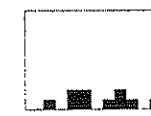
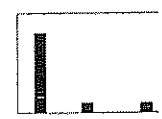
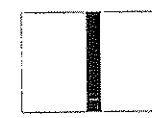
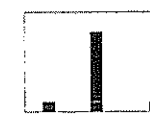
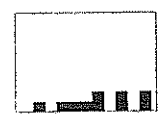
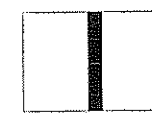
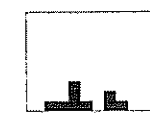
Database	Accuracy	MSE	Iterations	Execution time
bcw	 96.24% ± 1.81%	 0.0376 ± 0.0181	 100000 ± 0	 4.18 s ± 0.03 s
cdges	 89.52% ± 2.05%	 0.1048 ± 0.0205	 279 ± 101	 15.74 s ± 2.23 s
chess[0]	 48.67% ± 2.54%	 0.5133 ± 0.0254	 100000 ± 0	 3.45 s ± 0.04 s
chess[1]	 51.00% ± 1.90%	 0.4900 ± 0.0190	 100000 ± 0	 3.48 s ± 0.03 s
chess[2]	 50.83% ± 2.77%	 0.4917 ± 0.0277	 100000 ± 0	 3.48 s ± 0.02 s
gauss[0]	 99.90% ± 0.21%	 0.0010 ± 0.0021	 2 ± 0	 0.01 s ± 0.00 s
gauss[1]	 97.97% ± 0.85%	 0.0203 ± 0.0085	 100000 ± 0	 3.47 s ± 0.03 s

Table A.1: (continued)

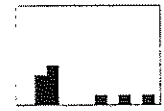
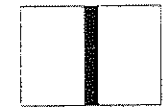
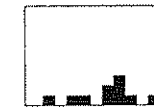
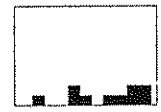
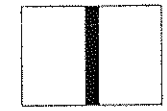
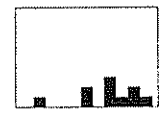
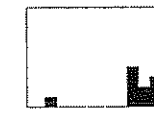
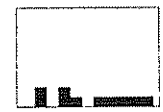
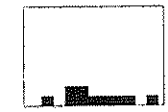
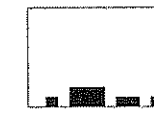
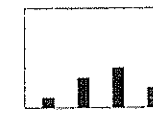
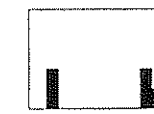
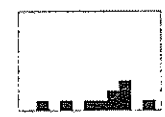
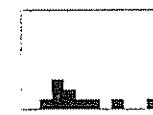
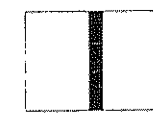
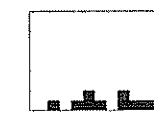
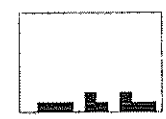
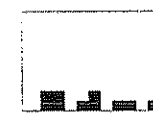
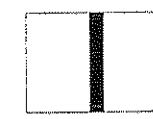
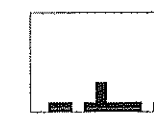
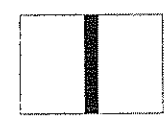
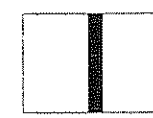
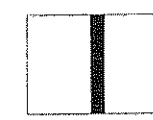
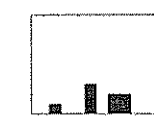
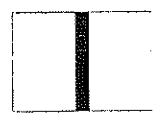
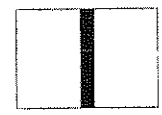
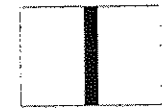
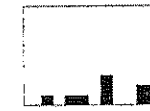
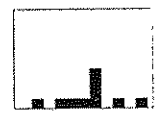
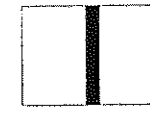
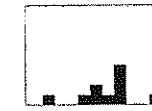
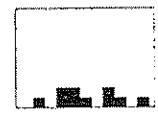
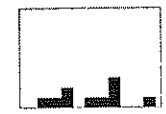
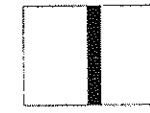
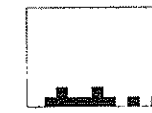
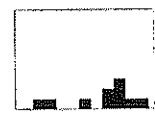
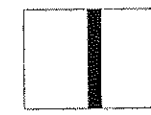
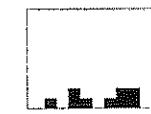
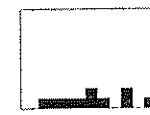
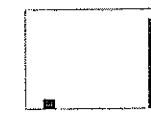
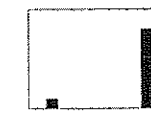
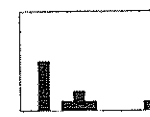
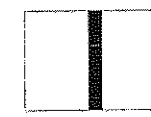
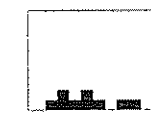
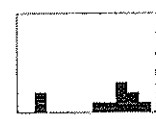
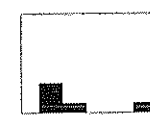
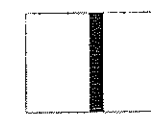
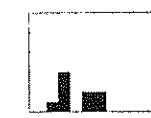
Database	Accuracy	MSE	Iterations	Execution time
gauss[2]	 87.73% ± 5.54%	 0.1227 ± 0.0554	 100000 ± 0	 3.45 s ± 0.03 s
hepatitis	 76.17% ± 8.34%	 0.2383 ± 0.0834	 100000 ± 0	 1.75 s ± 0.02 s
ionosphere	 82.83% ± 2.34%	 0.1717 ± 0.0234	 99580 ± 1258	 6.21 s ± 0.08 s
musk	 82.87% ± 3.13%	 0.1713 ± 0.0313	 1064 ± 284	 0.56 s ± 0.10 s
pgs	 76.56% ± 6.74%	 0.2344 ± 0.0674	 5 ± 1	 0.03 s ± 0.00 s
pid	 68.31% ± 8.31%	 0.3169 ± 0.0831	 100000 ± 0	 4.27 s ± 0.04 s
ringnorm	 66.00% ± 4.70%	 0.3400 ± 0.0470	 100000 ± 0	 12.12 s ± 0.02 s
spect[b]	 75.94% ± 0.00%	 0.2406 ± 0.0000	 100000 ± 0	 1.43 s ± 0.01 s

Table A.1: (continued)

Database	Accuracy	MSE	Iterations	Execution time
spect^{r}	 70.63% ± 0.00%	 0.2937 ± 0.0000	 32038 ± 0	 0.82 s ± 0.01 s
spiral^{0}	 48.93% ± 3.46%	 0.5107 ± 0.0346	 100000 ± 0	 3.44 s ± 0.03 s
spiral^{1}	 49.20% ± 2.11%	 0.5080 ± 0.0211	 100000 ± 0	 3.45 s ± 0.02 s
spiral^{2}	 50.50% ± 2.25%	 0.4950 ± 0.0225	 100000 ± 0	 3.45 s ± 0.03 s
twonorm	 96.63% ± 0.85%	 0.0337 ± 0.0085	 90303 ± 29092	 10.97 s ± 3.51 s
wdbc	 92.63% ± 3.44%	 0.0737 ± 0.0344	 100000 ± 0	 9.26 s ± 0.05 s
wpbc	 69.67% ± 10.40%	 0.3033 ± 0.1040	 100000 ± 0	 3.38 s ± 0.02 s

A.2 Experimental Results

In this section we present the results previously described in Section 5.3 and later discussed in Section 5.4. Results for each of the four hybrid algorithms proposed in Chapter 4 are displayed in the following sections.

A.2.1 Complete Results for SMO-B_α

Table A.2 brings the complete results for the hybrid algorithm SMO-B_α, as previously described in Section 5.3.1. The algorithm was experimented with variations of T and ρ, its two regularization parameters that influence the behavior of the Boosting module and its interface with the weak learner. The results presented in this section are further analyzed in Section 5.4.3, where they are thoroughly discussed.

Table A.2: Average and best results for hybrid algorithm SMO-B_α after 10 rounds of experiments with distinct training and testing sets.

bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	51.00% ± 16.81%	3.74% ± 1.84%	24.36 s ± 15.15 s
1	0.4	53.67% ± 14.99%	100.00% ± 0.00%	0.99 s ± 0.87 s
1	**0.1**	**87.71% ± 12.38%**	**100.00% ± 0.00%**	**0.03 s ± 0.03 s**
1000	0.1	50.14% ± 15.43%	4.90% ± 1.84%	1.13 s ± 0.12 s
1000	0.7	49.19% ± 15.41%	3.39% ± 1.70%	102.87 s ± 63.15 s
100	1	47.19% ± 15.17%	16.40% ± 10.86%	186.25 s ± 112.08 s
1	0.7	56.52% ± 13.98%	100.00% ± 0.00%	18.03 s ± 24.23 s
1000	1	44.71% ± 14.50%	1.60% ± 0.56%	178.40 s ± 98.95 s
100	0.7	50.05% ± 15.43%	30.20% ± 8.96%	127.54 s ± 118.44 s
100	0.4	53.86% ± 14.94%	35.30% ± 19.62%	21.25 s ± 12.50 s
1	1	47.29% ± 15.19%	100.00% ± 0.00%	228.99 s ± 271.93 s
100	0.1	49.19% ± 19.95%	55.10% ± 33.19%	0.59 s ± 0.34 s

Table A.2: (continued)

bcw (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.22% ± 0.11%	251.2781 ± 161.4524	6 ± 3
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	6.81% ± 2.54%	17973.7100 ± 17899.1636	301 ± 243
1	**0.1**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**3.38% ± 0.84%**	**969.5497 ± 2283.8921**	**38 ± 50**
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.15% ± 0.06%	27.4609 ± 11.5640	2 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	0.25% ± 0.11%	395.3559 ± 203.2713	9 ± 5
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	1.37% ± 0.75%	4123.0772 ± 1374.9117	86 ± 37
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	10.81% ± 2.51%	76494.4166 ± 57033.0937	1797 ± 2095
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	0.14% ± 0.04%	403.1479 ± 178.5044	8 ± 3
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	2.27% ± 0.70%	4831.3097 ± 3590.3131	102 ± 68
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	2.06% ± 1.12%	2562.8477 ± 1283.3616	57 ± 30
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	16.67% ± 5.35%	322822.2248 ± 248327.9780	8017 ± 7173
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	1.75% ± 1.04%	334.7604 ± 184.4323	16 ± 9

bcw (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

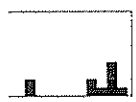
Error Rate (%)

Support Vectors (%)

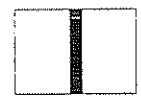
Norm

SMO Iterations

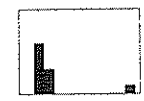
bcw (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 87.71% ± 12.38%



Good Weak Hyp. (%): 100.00% ± 0.00%



Execution Time (s): 0.03 s ± 0.03 s

Table A.2: (continued)

cdges (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	51.19% ± 0.00%	0.70% ± 0.00%	52.35 s ± 0.00 s
1	0.4	51.19% ± 0.00%	100.00% ± 0.00%	24.20 s ± 0.00 s
1	0.1	48.81% ± 0.00%	100.00% ± 0.00%	9.85 s ± 0.00 s
1000	0.1	51.19% ± 0.00%	1.20% ± 0.00%	14.27 s ± 0.00 s
1000	0.7	70.24% ± 0.00%	2.00% ± 0.00%	214.78 s ± 0.00 s
100	1	51.19% ± 0.00%	12.00% ± 0.00%	214.85 s ± 0.00 s
1	**0.7**	**70.24% ± 0.00%**	**100.00% ± 0.00%**	**28.82 s ± 0.00 s**
1000	1	51.19% ± 0.00%	0.80% ± 0.00%	150.64 s ± 0.00 s
100	0.7	70.24% ± 0.00%	12.00% ± 0.00%	123.07 s ± 0.00 s
100	0.4	51.19% ± 0.00%	21.00% ± 0.00%	131.07 s ± 0.00 s
1	1	70.24% ± 0.00%	100.00% ± 0.00%	57.87 s ± 0.00 s
100	0.1	51.19% ± 0.00%	64.00% ± 0.00%	36.85 s ± 0.00 s

cdges (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

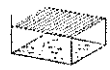
Execution Time (s)

cdges (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.41% ± 0.00%	15.4946 ± 0.0000	1 ± 0
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	75.00% ± 0.00%	2645.6855 ± 0.0000	38 ± 0
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	22.62% ± 0.00%	731.3799 ± 0.0000	16 ± 0
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.24% ± 0.00%	9.2934 ± 0.0000	1 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	1.27% ± 0.00%	38.0956 ± 0.0000	1 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	8.90% ± 0.00%	266.3392 ± 0.0000	2 ± 0
1	**0.7**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**122.62% ± 0.00%**	**4147.3276 ± 0.0000**	**21 ± 0**
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	0.64% ± 0.00%	19.4240 ± 0.0000	1 ± 0
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	8.92% ± 0.00%	276.1070 ± 0.0000	2 ± 0
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	11.54% ± 0.00%	422.3412 ± 0.0000	4 ± 0
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	153.57% ± 0.00%	5781.1938 ± 0.0000	22 ± 0
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	11.81% ± 0.00%	458.2767 ± 0.0000	9 ± 0

Table A.2: (continued)

cdges (weak hypotheses' average result graphs for $T \times \rho$)



Alpha

Error Rate (%)

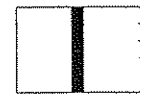
Support Vectors (%)

Norm

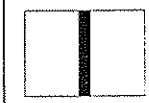
SMO Iterations

cdges (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 70.24% ± 0.00%



Good Weak Hyp. (%): 100.00% ± 0.00%



Execution Time (s): 28.82 s ± 0.00 s

chess^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	83.97% ± 8.02%	7.89% ± 2.43%	39.15 s ± 5.99 s
1	0.4	76.57% ± 5.78%	100.00% ± 0.00%	1.35 s ± 0.35 s
1	0.1	56.97% ± 3.58%	100.00% ± 0.00%	0.08 s ± 0.02 s
1000	0.1	82.17% ± 8.13%	18.05% ± 12.33%	10.27 s ± 6.00 s
1000	0.7	69.90% ± 11.58%	6.04% ± 1.33%	57.53 s ± 5.64 s
100	1	64.67% ± 7.38%	37.00% ± 13.53%	42.86 s ± 7.67 s
1	0.7	81.13% ± 4.08%	100.00% ± 0.00%	4.17 s ± 2.06 s
1000	1	63.33% ± 10.40%	3.71% ± 1.36%	73.07 s ± 6.80 s
100	0.7	68.50% ± 16.09%	57.90% ± 14.92%	42.75 s ± 6.30 s
100	**0.4**	**85.63% ± 8.01%**	**60.10% ± 32.18%**	**25.25 s ± 10.94 s**
1	1	82.27% ± 2.49%	100.00% ± 0.00%	9.65 s ± 5.09 s
100	0.1	85.60% ± 7.28%	74.00% ± 30.78%	3.91 s ± 1.55 s

Table A.2: (continued)

chess^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

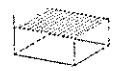
Execution Time (s)

chess^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	3.84% ± 1.06%	7.5144 ± 1.7952	1 ± 0
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	82.67% ± 1.64%	196.3264 ± 9.4921	22 ± 5
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	22.67% ± 0.33%	61.9287 ± 1.5567	10 ± 4
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	3.58% ± 2.42%	9.2388 ± 6.1715	2 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	3.24% ± 0.50%	5.0552 ± 0.4924	1 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	23.00% ± 6.84%	32.7467 ± 6.9988	4 ± 0
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	127.33% ± 5.30%	355.8742 ± 93.3104	28 ± 12
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	2.33% ± 0.74%	3.2827 ± 0.9158	1 ± 0
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	31.50% ± 6.03%	48.7881 ± 7.5404	6 ± 0
100	**0.4**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**30.17% ± 15.32%**	**59.6627 ± 28.0318**	**7 ± 2**
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	162.67% ± 8.35%	371.1604 ± 78.1281	48 ± 29
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	14.76% ± 6.07%	38.7785 ± 15.9310	7 ± 2

chess^0 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha



Error Rate (%)

Support Vectors (%)

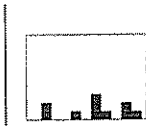
Norm

SMO Iterations

chess^0 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 85.63% ± 8.01%

Good Weak Hyp. (%): 60.10% ± 32.18%



Execution Time (s): 25.25 s ± 10.94 s

Table A.2: (continued)

chess[1] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	74.33% ± 5.72%	9.33% ± 4.28%	45.59 s ± 12.93 s
1	0.4	66.97% ± 4.03%	100.00% ± 0.00%	1.36 s ± 0.53 s
1	0.1	57.77% ± 3.57%	100.00% ± 0.00%	0.08 s ± 0.02 s
1000	**0.1**	**78.60% ± 4.76%**	**16.68% ± 18.14%**	**9.84 s ± 9.01 s**
1000	0.7	63.97% ± 4.54%	6.36% ± 1.48%	62.17 s ± 6.92 s
100	1	63.10% ± 8.62%	34.50% ± 13.92%	47.26 s ± 9.65 s
1	0.7	75.23% ± 4.55%	100.00% ± 0.00%	5.42 s ± 5.32 s
1000	1	62.67% ± 9.98%	3.76% ± 1.25%	76.65 s ± 6.80 s
100	0.7	64.97% ± 6.68%	52.90% ± 13.17%	45.45 s ± 5.45 s
100	0.4	76.10% ± 5.41%	67.60% ± 25.36%	31.52 s ± 10.40 s
1	1	72.87% ± 3.32%	100.00% ± 0.00%	9.28 s ± 4.36 s
100	0.1	76.07% ± 4.87%	74.00% ± 32.62%	3.98 s ± 1.62 s

chess[1] (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

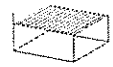
chess[1] (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	4.39% ± 1.77%	13.3867 ± 7.3156	1 ± 0
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	80.30% ± 2.57%	438.5983 ± 442.6472	25 ± 11
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	22.67% ± 0.49%	99.6662 ± 107.5999	11 ± 6
1000	**0.1**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**3.30% ± 3.55%**	**11.8840 ± 12.6099**	**2 ± 1**
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	3.31% ± 0.55%	10.8595 ± 9.7191	1 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	21.25% ± 6.69%	65.0907 ± 32.1331	4 ± 0
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	123.90% ± 7.39%	1510.1827 ± 2386.2431	35 ± 28
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	2.34% ± 0.64%	8.0160 ± 6.4058	1 ± 0
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	29.75% ± 5.73%	116.1931 ± 115.0793	6 ± 0
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	33.06% ± 11.90%	96.2734 ± 27.0194	8 ± 2
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	159.43% ± 10.10%	2594.7407 ± 3563.6063	37 ± 18
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	14.90% ± 6.47%	57.4934 ± 51.2699	7 ± 2

Table A.2: (continued)

chess[1] (weak hypotheses' average result graphs for $T \times \rho$)

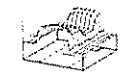
Alpha



Error Rate (%)

Support Vectors (%)

Norm



SMO Iterations

chess[1] (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 78.60% ± 4.76%

Good Weak Hyp. (%): 16.68% ± 18.14%

Execution Time (s): 9.84 s ± 9.01 s

chess[2] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	63.77% ± 3.88%	9.21% ± 4.91%	46.33 s ± 17.81 s
1	0.4	58.93% ± 2.91%	100.00% ± 0.00%	1.84 s ± 1.78 s
1	0.1	51.67% ± 2.06%	100.00% ± 0.00%	0.07 s ± 0.01 s
1000	**0.1**	**64.33% ± 3.91%**	**26.72% ± 18.69%**	**14.96 s ± 9.49 s**
1000	0.7	60.47% ± 5.90%	5.97% ± 1.59%	65.03 s ± 11.39 s
100	1	56.53% ± 4.44%	38.50% ± 12.53%	55.12 s ± 13.95 s
1	0.7	62.20% ± 2.32%	100.00% ± 0.00%	6.51 s ± 3.64 s
1000	1	55.03% ± 4.30%	3.39% ± 1.45%	78.81 s ± 11.41 s
100	0.7	61.07% ± 4.71%	47.50% ± 13.63%	40.62 s ± 5.94 s
100	0.4	62.37% ± 3.03%	93.60% ± 10.90%	43.82 s ± 5.62 s
1	1	62.80% ± 2.81%	100.00% ± 0.00%	10.52 s ± 2.26 s
100	0.1	64.10% ± 3.50%	61.40% ± 30.00%	3.37 s ± 1.53 s

Table A.2: (continued)

chess^2 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

chess^2 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	4.25% ± 2.06%	17.2177 ± 9.6422	1 ± 0
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	81.10% ± 3.06%	431.6170 ± 412.4177	27 ± 24
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	22.90% ± 0.30%	68.3830 ± 4.0520	8 ± 1
1000	**0.1**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**5.24% ± 3.64%**	**19.2020 ± 13.9891**	**3 ± 1**
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	3.13% ± 0.61%	12.6131 ± 3.7084	1 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	23.55% ± 6.07%	81.2367 ± 20.5459	4 ± 0
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	121.53% ± 3.60%	1143.2826 ± 693.6101	44 ± 19
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	2.10% ± 0.77%	8.2996 ± 2.8439	1 ± 0
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	26.70% ± 5.40%	109.5397 ± 44.6596	5 ± 0
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	44.35% ± 5.29%	199.9251 ± 66.6741	11 ± 1
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	152.37% ± 6.17%	1523.2606 ± 671.0082	41 ± 10
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	12.33% ± 5.89%	41.8047 ± 22.0975	6 ± 2

chess^2 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

Support Vectors (%)

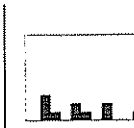
Norm

SMO Iterations

chess^2 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 64.33% ± 3.91%

Good Weak Hyp. (%): 26.72% ± 18.69%



Execution Time (s): 14.96 s ± 9.49 s

Table A.2: (continued)

gauss^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	52.67% ± 0.00%	0.40% ± 0.00%	6.93 s ± 0.00 s
1	0.4	47.33% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1	0.1	47.33% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	52.67% ± 0.00%	7.70% ± 0.00%	1.51 s ± 0.00 s
1000	0.7	47.33% ± 0.00%	5.30% ± 0.00%	17.25 s ± 0.00 s
100	1	52.67% ± 0.00%	49.00% ± 0.00%	3.05 s ± 0.00 s
1	0.7	100.00% ± 0.00%	100.00% ± 0.00%	0.13 s ± 0.00 s
1000	1	47.33% ± 0.00%	4.40% ± 0.00%	31.77 s ± 0.00 s
100	0.7	47.33% ± 0.00%	32.00% ± 0.00%	1.72 s ± 0.00 s
100	0.4	52.67% ± 0.00%	52.00% ± 0.00%	0.98 s ± 0.00 s
1	**1**	**100.00% ± 0.00%**	**100.00% ± 0.00%**	**0.06 s ± 0.00 s**
100	0.1	100.00% ± 0.00%	12.00% ± 0.00%	0.15 s ± 0.00 s

gauss^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

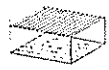
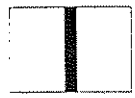
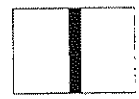
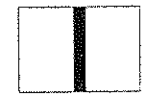
Good Weak Hyp. (%)

Execution Time (s)

gauss^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.00% ± 0.00%	3.9032 ± 0.0000	1 ± 0
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	1.00% ± 0.00%	1444.5056 ± 0.0000	11 ± 0
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	1.33% ± 0.00%	695.9318 ± 0.0000	19 ± 0
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.07% ± 0.00%	45.8893 ± 0.0000	1 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	0.08% ± 0.00%	35.6975 ± 0.0000	1 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	0.74% ± 0.00%	322.1009 ± 0.0000	4 ± 0
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	1.33% ± 0.00%	979.9987 ± 0.0000	48 ± 0
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	0.08% ± 0.00%	27.2773 ± 0.0000	1 ± 0
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	0.45% ± 0.00%	222.5380 ± 0.0000	3 ± 0
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.64% ± 0.00%	367.1458 ± 0.0000	6 ± 0
1	**1**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**1.00% ± 0.00%**	**1281.9319 ± 0.0000**	**8 ± 0**
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.11% ± 0.00%	76.1266 ± 0.0000	1 ± 0

Table A.2: (continued)

gauss^0 (weak hypotheses' average result graphs for $T \times \rho$)				
 Alpha	 Error Rate (%)	 Support Vectors (%)	 Norm	 SMO Iterations
gauss^0 (experiment histogram for parameter setup that yielded best results)				
 Accuracy (%): 100.00% ± 0.00%	 Good Weak Hyp. (%): 100.00% ± 0.00%	 Execution Time (s): 0.06 s ± 0.00 s		

A.2.2 Complete Results for SMO-B_β

Table A.3 brings the complete results for the hybrid algorithm SMO-B_β, as previously described in Section 5.3.2. The algorithm was experimented with variations of T and ρ, its two regularization parameters that influence the behavior of the Boosting module and its interface with the weak learner. The results presented in this section are further analyzed in Section 5.4.3, where they are thoroughly discussed.

Table A.3: Average and best results for hybrid algorithm **SMO-B$_\beta$** after 10 rounds of experiments with distinct training and testing sets.

bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	97.62% ± 0.00%	3.70% ± 0.00%	905.42 s ± 0.00 s
1	**0.4**	**97.62% ± 0.00%**	**100.00% ± 0.00%**	**0.04 s ± 0.00 s**
1	0.1	96.19% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	96.67% ± 0.00%	99.90% ± 0.00%	14.31 s ± 0.00 s
1000	0.7	96.19% ± 0.00%	1.10% ± 0.00%	2805.17 s ± 0.00 s
100	1	96.67% ± 0.00%	100.00% ± 0.00%	444.31 s ± 0.00 s
1	0.7	97.14% ± 0.00%	100.00% ± 0.00%	2.12 s ± 0.00 s
1000	1	96.67% ± 0.00%	100.00% ± 0.00%	4491.92 s ± 0.00 s
100	0.7	97.14% ± 0.00%	10.00% ± 0.00%	285.85 s ± 0.00 s
100	0.4	95.24% ± 0.00%	14.00% ± 0.00%	100.29 s ± 0.00 s
1	1	96.67% ± 0.00%	100.00% ± 0.00%	5.04 s ± 0.00 s
100	0.1	97.14% ± 0.00%	100.00% ± 0.00%	1.20 s ± 0.00 s

bcw (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

Table A.3: (continued)

bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	-0.1294 ± 0.0000	0.02% ± 0.00%	13.76% ± 0.00%	5602.5713 ± 0.0000	325 ± 0
1	**0.4**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**3.33% ± 0.00%**	**1666.2279 ± 0.0000**	**21 ± 0**
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	2.86% ± 0.00%	187.4276 ± 0.0000	11 ± 0
1000	0.1	9.3123 ± 0.0000	0.00% ± 0.00%	3.59% ± 0.00%	833.1179 ± 0.0000	39 ± 0
1000	0.7	-0.3749 ± 0.0000	0.04% ± 0.00%	19.46% ± 0.00%	8896.0666 ± 0.0000	454 ± 0
100	1	0.0193 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10056.9649 ± 0.0000	547 ± 0
1	0.7	2.0482 ± 0.0000	0.04% ± 0.00%	15.24% ± 0.00%	5772.3096 ± 0.0000	368 ± 0
1000	1	0.0019 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10054.5734 ± 0.0000	549 ± 0
100	0.7	-0.1583 ± 0.0000	0.04% ± 0.00%	19.17% ± 0.00%	7206.9525 ± 0.0000	459 ± 0
100	0.4	0.0247 ± 0.0000	0.02% ± 0.00%	14.10% ± 0.00%	6115.5771 ± 0.0000	335 ± 0
1	1	1.9346 ± 0.0000	0.05% ± 0.00%	22.38% ± 0.00%	10022.3125 ± 0.0000	663 ± 0
100	0.1	9.8656 ± 0.0000	0.00% ± 0.00%	3.39% ± 0.00%	703.8516 ± 0.0000	36 ± 0

bcw (weak hypotheses' average result graphs for $T \times \rho$)


Alpha

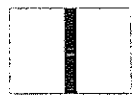

Error Rate (%)

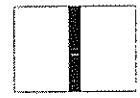

Support Vectors (%)


Norm


SMO Iterations

bcw (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 97.62% ± 0.00%


Good Weak Hyp. (%): 100.00% ± 0.00%

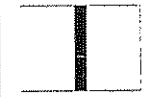

Execution Time (s): 0.04 s ± 0.00 s

Table A.3: (continued)

cdges (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	76.19% ± 0.00%	100.00% ± 0.00%	17109.49 s ± 0.00 s
1	0.4	76.19% ± 0.00%	100.00% ± 0.00%	30.46 s ± 0.00 s
1	0.1	67.86% ± 0.00%	100.00% ± 0.00%	9.78 s ± 0.00 s
1000	0.1	76.19% ± 0.00%	99.90% ± 0.00%	579.43 s ± 0.00 s
1000	0.7	77.38% ± 0.00%	100.00% ± 0.00%	72729.07 s ± 0.00 s
100	1	77.38% ± 0.00%	100.00% ± 0.00%	17091.03 s ± 0.00 s
1	**0.7**	**77.38% ± 0.00%**	**100.00% ± 0.00%**	**97.81 s ± 0.00 s**
1000	1	77.38% ± 0.00%	100.00% ± 0.00%	71991.00 s ± 0.00 s
100	0.7	77.38% ± 0.00%	100.00% ± 0.00%	7211.17 s ± 0.00 s
100	0.4	76.19% ± 0.00%	100.00% ± 0.00%	1670.10 s ± 0.00 s
1	1	77.38% ± 0.00%	100.00% ± 0.00%	143.89 s ± 0.00 s
100	0.1	76.19% ± 0.00%	100.00% ± 0.00%	66.27 s ± 0.00 s

cdges (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

cdges (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	90.91% ± 0.00%	4369.0028 ± 0.0000	34 ± 0
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	92.86% ± 0.00%	3819.5847 ± 0.0000	48 ± 0
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	22.62% ± 0.00%	483.1850 ± 0.0000	16 ± 0
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	22.13% ± 0.00%	1100.6379 ± 0.0000	15 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	158.12% ± 0.00%	7134.6612 ± 0.0000	49 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	224.98% ± 0.00%	9630.2028 ± 0.0000	56 ± 0
1	**0.7**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**160.71% ± 0.00%**	**7992.8071 ± 0.0000**	**58 ± 0**
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	224.95% ± 0.00%	9630.2196 ± 0.0000	55 ± 0
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	157.99% ± 0.00%	7111.6097 ± 0.0000	48 ± 0
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	90.63% ± 0.00%	4331.4845 ± 0.0000	33 ± 0
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	225.00% ± 0.00%	9629.5186 ± 0.0000	46 ± 0
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	22.25% ± 0.00%	1110.1486 ± 0.0000	16 ± 0

Table A.3: (continued)

cdges (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

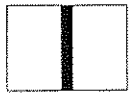
Error Rate (%)

Support Vectors (%)

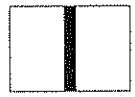
Norm

SMO Iterations

cdges (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 77.38% ± 0.00%



Good Weak Hyp. (%): 100.00% ± 0.00%

Execution Time (s): 97.81 s ± 0.00 s

gauss^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	100.00% ± 0.00%	100.00% ± 0.00%	13.78 s ± 0.00 s
1	**0.4**	**100.00% ± 0.00%**	**100.00% ± 0.00%**	**0.02 s ± 0.00 s**
1	0.1	100.00% ± 0.00%	100.00% ± 0.00%	0.03 s ± 0.00 s
1000	0.1	100.00% ± 0.00%	100.00% ± 0.00%	3.65 s ± 0.00 s
1000	0.7	100.00% ± 0.00%	100.00% ± 0.00%	28.72 s ± 0.00 s
100	1	100.00% ± 0.00%	100.00% ± 0.00%	3.97 s ± 0.00 s
1	0.7	100.00% ± 0.00%	100.00% ± 0.00%	0.03 s ± 0.00 s
1000	1	100.00% ± 0.00%	100.00% ± 0.00%	39.56 s ± 0.00 s
100	0.7	100.00% ± 0.00%	100.00% ± 0.00%	2.62 s ± 0.00 s
100	0.4	100.00% ± 0.00%	100.00% ± 0.00%	1.45 s ± 0.00 s
1	1	100.00% ± 0.00%	100.00% ± 0.00%	0.04 s ± 0.00 s
100	0.1	100.00% ± 0.00%	100.00% ± 0.00%	0.37 s ± 0.00 s

Table A.3: (continued)

gauss^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

gauss^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.96% ± 0.00%	1172.8804 ± 0.0000	12 ± 0
1	**0.4**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**1.00% ± 0.00%**	**848.4375 ± 0.0000**	**8 ± 0**
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	1.00% ± 0.00%	381.5545 ± 0.0000	12 ± 0
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.92% ± 0.00%	647.3489 ± 0.0000	10 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	1.00% ± 0.00%	1418.5664 ± 0.0000	16 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	1.01% ± 0.00%	1540.5376 ± 0.0000	19 ± 0
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	1.00% ± 0.00%	1445.3323 ± 0.0000	12 ± 0
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	1.01% ± 0.00%	1540.5912 ± 0.0000	20 ± 0
100	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	1.00% ± 0.00%	1416.4849 ± 0.0000	14 ± 0
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	0.97% ± 0.00%	1159.3766 ± 0.0000	14 ± 0
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	1.00% ± 0.00%	1536.5115 ± 0.0000	10 ± 0
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	0.92% ± 0.00%	651.4590 ± 0.0000	10 ± 0

gauss^0 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

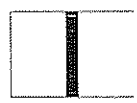
Error Rate (%)

Support Vectors (%)

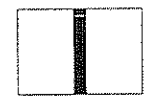
Norm

SMO Iterations

gauss^0 (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 100.00% ± 0.00%



Good Weak Hyp. (%): 100.00% ± 0.00%

Execution Time (s): 0.02 s ± 0.00 s

Table A.3: (continued)

pgs (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	81.25% ± 0.00%	100.00% ± 0.00%	10.11 s ± 0.00 s
1	0.4	71.88% ± 0.00%	100.00% ± 0.00%	0.04 s ± 0.00 s
1	0.1	40.62% ± 0.00%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	0.1	71.88% ± 0.00%	98.70% ± 0.00%	0.62 s ± 0.00 s
1000	0.7	87.50% ± 0.00%	100.00% ± 0.00%	50.89 s ± 0.00 s
100	1	84.38% ± 0.00%	100.00% ± 0.00%	20.11 s ± 0.00 s
1	0.7	78.12% ± 0.00%	100.00% ± 0.00%	0.06 s ± 0.00 s
1000	1	84.38% ± 0.00%	100.00% ± 0.00%	188.02 s ± 0.00 s
100	**0.7**	**87.50% ± 0.00%**	**100.00% ± 0.00%**	**5.11 s ± 0.00 s**
100	0.4	81.25% ± 0.00%	100.00% ± 0.00%	0.99 s ± 0.00 s
1	1	84.38% ± 0.00%	100.00% ± 0.00%	0.16 s ± 0.00 s
100	0.1	75.00% ± 0.00%	98.00% ± 0.00%	0.07 s ± 0.00 s

pgs (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

pgs (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	85.14% ± 0.00%	248.6430 ± 0.0000	9 ± 0
1	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	84.38% ± 0.00%	237.0542 ± 0.0000	9 ± 0
1	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	21.88% ± 0.00%	43.1786 ± 0.0000	6 ± 0
1000	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	21.51% ± 0.00%	54.9214 ± 0.0000	6 ± 0
1000	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	131.41% ± 0.00%	442.4716 ± 0.0000	18 ± 0
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	176.78% ± 0.00%	611.1197 ± 0.0000	38 ± 0
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	134.38% ± 0.00%	434.8931 ± 0.0000	13 ± 0
1000	1	10.0000 ± 0.0000	0.00% ± 0.00%	176.45% ± 0.00%	611.1284 ± 0.0000	36 ± 0
100	**0.7**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**131.44% ± 0.00%**	**446.4440 ± 0.0000**	**18 ± 0**
100	0.4	10.0000 ± 0.0000	0.00% ± 0.00%	85.72% ± 0.00%	246.3571 ± 0.0000	9 ± 0
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	175.00% ± 0.00%	611.2856 ± 0.0000	27 ± 0
100	0.1	10.0000 ± 0.0000	0.00% ± 0.00%	21.38% ± 0.00%	53.9706 ± 0.0000	6 ± 0

Table A.3: (continued)

pgs (weak hypotheses' average result graphs for $T \times \rho$)


Alpha

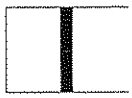

Error Rate (%)


Support Vectors (%)


Norm


SMO Iterations

pgs (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 87.50% ± 0.00%


Good Weak Hyp. (%): 100.00% ± 0.00%


Execution Time (s): 5.11 s ± 0.00 s

pid (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	74.03% ± 0.00%	1.90% ± 0.00%	1455.94 s ± 0.00 s
1	0.4	74.03% ± 0.00%	100.00% ± 0.00%	2.68 s ± 0.00 s
1	0.1	72.29% ± 0.00%	100.00% ± 0.00%	0.06 s ± 0.00 s
1000	**0.1**	**75.32% ± 0.00%**	**4.90% ± 0.00%**	**88.99 s ± 0.00 s**
1000	0.7	74.03% ± 0.00%	1.80% ± 0.00%	4276.86 s ± 0.00 s
100	1	74.89% ± 0.00%	100.00% ± 0.00%	749.56 s ± 0.00 s
1	0.7	73.16% ± 0.00%	100.00% ± 0.00%	5.38 s ± 0.00 s
1000	1	74.89% ± 0.00%	100.00% ± 0.00%	7628.79 s ± 0.00 s
100	0.7	74.03% ± 0.00%	6.00% ± 0.00%	415.90 s ± 0.00 s
100	0.4	73.59% ± 0.00%	17.00% ± 0.00%	150.09 s ± 0.00 s
1	1	74.89% ± 0.00%	100.00% ± 0.00%	6.32 s ± 0.00 s
100	0.1	72.29% ± 0.00%	36.00% ± 0.00%	7.16 s ± 0.00 s

Table A.3: (continued)

pid (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

pid (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm	SMO Iterations
1000	0.4	-0.1425 ± 0.0000	0.29% ± 0.00%	65.12% ± 0.00%	6365.9361 ± 0.0000	1275 ± 0
1	0.4	1.2082 ± 0.0000	0.19% ± 0.00%	49.35% ± 0.00%	5041.8750 ± 0.0000	2050 ± 0
1	0.1	2.3336 ± 0.0000	0.02% ± 0.00%	8.23% ± 0.00%	3624.7500 ± 0.0000	153 ± 0
1000	**0.1**	**-0.3175 ± 0.0000**	**0.07% ± 0.00%**	**18.27% ± 0.00%**	**3470.8524 ± 0.0000**	**395 ± 0**
1000	0.7	-0.0810 ± 0.0000	0.43% ± 0.00%	99.39% ± 0.00%	8425.5555 ± 0.0000	1948 ± 0
100	1	0.0064 ± 0.0000	0.51% ± 0.00%	121.71% ± 0.00%	11711.6100 ± 0.0000	2101 ± 0
1	0.7	0.9013 ± 0.0000	0.33% ± 0.00%	83.55% ± 0.00%	10614.5000 ± 0.0000	2957 ± 0
1000	1	0.0006 ± 0.0000	0.51% ± 0.00%	121.72% ± 0.00%	11717.6790 ± 0.0000	2165 ± 0
100	0.7	-0.0451 ± 0.0000	0.43% ± 0.00%	99.28% ± 0.00%	8744.7141 ± 0.0000	1861 ± 0
100	0.4	-0.0633 ± 0.0000	0.30% ± 0.00%	66.73% ± 0.00%	6398.8391 ± 0.0000	1325 ± 0
1	1	0.6390 ± 0.0000	0.51% ± 0.00%	121.21% ± 0.00%	11787.8750 ± 0.0000	1750 ± 0
100	0.1	-0.0160 ± 0.0000	0.07% ± 0.00%	17.67% ± 0.00%	3549.5643 ± 0.0000	320 ± 0

pid (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

Support Vectors (%)

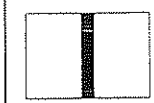
Norm

SMO Iterations

pid (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 75.32% ± 0.00%

Good Weak Hyp. (%): 4.90% ± 0.00%



Execution Time (s): 88.99 s ± 0.00 s

A.2.3 Complete Results for SMO-B$_\gamma$

Table A.4 brings the complete results for the hybrid algorithm SMO-B$_\gamma$, as previously described in Section 5.3.3. The algorithm was experimented with variations of T and ρ, its two regularization parameters that influence the behavior of the Boosting module and its interface with the weak learner. The results presented in this section are further analyzed in Section 5.4.4, where they are thoroughly discussed.

Table A.4: Average and best results for hybrid algorithm SMO-B$_\gamma$ after 10 rounds of experiments with distinct training and testing sets.

bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	96.29% ± 1.14%	18.15% ± 2.54%	35.43 s ± 4.12 s
1	0.4	90.24% ± 9.43%	100.00% ± 0.00%	0.05 s ± 0.01 s
1	0.1	95.52% ± 2.38%	100.00% ± 0.00%	0.03 s ± 0.01 s
1000	0.1	96.14% ± 1.12%	32.91% ± 4.01%	18.06 s ± 0.98 s
1000	**0.7**	**96.52% ± 1.17%**	**14.37% ± 3.00%**	**45.96 s ± 4.66 s**
100	1	96.19% ± 1.11%	21.40% ± 4.92%	5.31 s ± 0.60 s
1	0.7	91.19% ± 7.89%	100.00% ± 0.00%	0.06 s ± 0.01 s
1000	1	96.29% ± 0.85%	12.07% ± 4.28%	51.64 s ± 5.96 s
100	0.7	96.33% ± 1.04%	23.00% ± 3.69%	4.64 s ± 0.56 s
100	0.4	96.19% ± 1.02%	27.90% ± 5.56%	3.67 s ± 0.43 s
1	1	91.33% ± 8.41%	100.00% ± 0.00%	0.08 s ± 0.01 s
100	0.1	96.38% ± 1.19%	45.30% ± 7.84%	1.83 s ± 0.10 s

bcw (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

Table A.4: (continued)

bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.3065 ± 0.0576	0.16% ± 0.03%	25.54% ± 3.33%	639.6772 ± 71.2488
1	0.4	1.3624 ± 0.4054	0.19% ± 0.19%	15.76% ± 3.36%	399.5661 ± 122.6807
1	0.1	1.5457 ± 0.2571	0.12% ± 0.07%	5.43% ± 1.57%	194.0201 ± 111.1667
1000	0.1	-0.0853 ± 0.0188	0.33% ± 0.05%	13.26% ± 0.83%	400.7593 ± 26.2831
1000	**0.7**	**-0.3853 ± 0.0892**	**0.13% ± 0.02%**	**32.47% ± 3.74%**	**725.3301 ± 97.6372**
100	1	-0.1961 ± 0.0672	0.12% ± 0.02%	35.89% ± 4.82%	756.3575 ± 114.6263
1	0.7	1.3731 ± 0.4324	0.19% ± 0.17%	26.38% ± 5.13%	526.2639 ± 156.5876
1000	1	-0.4039 ± 0.0785	0.13% ± 0.02%	35.92% ± 4.73%	789.2475 ± 96.5533
100	0.7	-0.2106 ± 0.0500	0.14% ± 0.03%	31.87% ± 4.38%	714.4174 ± 104.9769
100	0.4	-0.1293 ± 0.0438	0.15% ± 0.03%	25.65% ± 3.48%	626.1287 ± 79.9194
1	1	1.3330 ± 0.4916	0.22% ± 0.22%	37.71% ± 5.19%	656.4435 ± 164.6508
100	0.1	-0.0144 ± 0.0393	0.35% ± 0.08%	12.67% ± 0.67%	398.4358 ± 21.4161

bcw (weak hypotheses' average result graphs for $T \times \rho$)


Alpha


Error Rate (%)


Support Vectors (%)


Norm

bcw (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 96.52% ± 1.17%


Good Weak Hyp. (%): 14.37% ± 3.00%

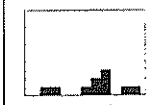

Execution Time (s): 45.96 s ± 4.66 s

Table A.4: (continued)

cdges (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	77.38% ± 0.00%	94.30% ± 0.00%	2310.22 s ± 0.00 s
1	0.4	51.19% ± 0.00%	100.00% ± 0.00%	11.60 s ± 0.00 s
1	0.1	0.00% ± 0.00%	0.00% ± 0.00%	9.81 s ± 0.00 s
1000	0.1	77.38% ± 0.00%	88.90% ± 0.00%	600.70 s ± 0.00 s
1000	0.7	77.38% ± 0.00%	95.80% ± 0.00%	4002.05 s ± 0.00 s
100	1	77.38% ± 0.00%	91.00% ± 0.00%	560.16 s ± 0.00 s
1	**0.7**	**77.38% ± 0.00%**	**100.00% ± 0.00%**	**13.37 s ± 0.00 s**
1000	1	77.38% ± 0.00%	94.90% ± 0.00%	5665.90 s ± 0.00 s
100	0.7	77.38% ± 0.00%	94.00% ± 0.00%	407.01 s ± 0.00 s
100	0.4	77.38% ± 0.00%	99.00% ± 0.00%	240.42 s ± 0.00 s
1	1	75.00% ± 0.00%	100.00% ± 0.00%	14.93 s ± 0.00 s
100	0.1	61.90% ± 0.00%	89.00% ± 0.00%	69.11 s ± 0.00 s

cdges (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

cdges (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.7625 ± 0.0000	0.63% ± 0.00%	83.07% ± 0.00%	3923.7487 ± 0.0000
1	0.4	0.2828 ± 0.0000	0.85% ± 0.00%	82.14% ± 0.00%	2398.3770 ± 0.0000
1	0.1	-0.0306 ± 0.0000	1.20% ± 0.00%	19.05% ± 0.00%	1758.7495 ± 0.0000
1000	0.1	0.1475 ± 0.0000	1.07% ± 0.00%	21.73% ± 0.00%	959.9760 ± 0.0000
1000	0.7	1.5132 ± 0.0000	0.29% ± 0.00%	143.76% ± 0.00%	6133.3806 ± 0.0000
100	1	4.6086 ± 0.0000	0.11% ± 0.00%	199.23% ± 0.00%	7870.3905 ± 0.0000
1	**0.7**	**1.4616 ± 0.0000**	**0.12% ± 0.00%**	**139.29% ± 0.00%**	**7380.6626 ± 0.0000**
1000	1	5.1173 ± 0.0000	0.09% ± 0.00%	200.17% ± 0.00%	8037.4230 ± 0.0000
100	0.7	1.3251 ± 0.0000	0.31% ± 0.00%	143.70% ± 0.00%	6008.3653 ± 0.0000
100	0.4	0.7255 ± 0.0000	0.67% ± 0.00%	82.67% ± 0.00%	3717.7524 ± 0.0000
1	1	2.6365 ± 0.0000	0.01% ± 0.00%	204.76% ± 0.00%	8333.9365 ± 0.0000
100	0.1	0.1414 ± 0.0000	1.07% ± 0.00%	21.96% ± 0.00%	903.1495 ± 0.0000

Table A.4: (continued)

cdges (weak hypotheses' average result graphs for $T \times \rho$)

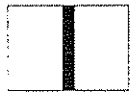
Alpha

Error Rate (%)

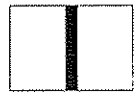
Support Vectors (%)

Norm

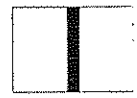
cdges (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 77.38% ± 0.00%



Good Weak Hyp. (%): 100.00% ± 0.00%



Execution Time (s): 13.37 s ± 0.00 s

chess^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	93.77% ± 1.52%	99.76% ± 0.21%	473.78 s ± 2.12 s
1	0.4	70.23% ± 4.13%	100.00% ± 0.00%	0.49 s ± 0.01 s
1	0.1	56.37% ± 1.94%	100.00% ± 0.00%	0.13 s ± 0.01 s
1000	0.1	93.17% ± 1.54%	97.12% ± 1.19%	124.37 s ± 0.31 s
1000	0.7	93.67% ± 1.35%	99.61% ± 0.43%	796.00 s ± 4.54 s
100	1	93.27% ± 2.07%	99.50% ± 0.67%	110.18 s ± 0.89 s
1	0.7	82.00% ± 5.62%	100.00% ± 0.00%	0.82 s ± 0.01 s
1000	1	93.00% ± 2.01%	98.07% ± 2.83%	1100.37 s ± 10.19 s
100	**0.7**	**93.90% ± 1.49%**	**99.60% ± 0.92%**	**79.69 s ± 0.46 s**
100	0.4	93.27% ± 1.90%	99.80% ± 0.40%	47.45 s ± 0.22 s
1	1	85.53% ± 6.70%	100.00% ± 0.00%	1.12 s ± 0.01 s
100	0.1	91.77% ± 2.15%	97.70% ± 2.05%	12.44 s ± 0.03 s

Table A.4: (continued)

chess^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

chess^0 (average results for weak hypotheses)

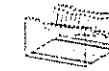
T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	1.5102 ± 0.0262	0.38% ± 0.02%	90.72% ± 0.50%	291.1688 ± 27.1275
1	0.4	0.7588 ± 0.1113	0.42% ± 0.08%	91.00% ± 1.12%	211.4808 ± 14.8662
1	0.1	0.2167 ± 0.0321	0.92% ± 0.04%	23.60% ± 0.13%	60.6072 ± 3.3533
1000	0.1	0.2223 ± 0.0126	0.97% ± 0.01%	23.51% ± 0.04%	81.6400 ± 9.2504
1000	0.7	2.7891 ± 0.1016	0.11% ± 0.00%	152.45% ± 1.27%	434.0435 ± 37.9946
100	1	8.1435 ± 1.2775	0.00% ± 0.01%	209.73% ± 2.17%	552.4172 ± 53.3453
1	0.7	1.4665 ± 0.1490	0.12% ± 0.03%	153.77% ± 2.45%	384.5562 ± 74.8701
1000	1	8.1030 ± 2.4036	0.01% ± 0.01%	209.86% ± 2.10%	545.7416 ± 49.6236
100	**0.7**	**2.6472 ± 0.0946**	**0.11% ± 0.01%**	**152.60% ± 1.21%**	**431.5629 ± 37.1213**
100	0.4	1.4096 ± 0.0482	0.38% ± 0.02%	90.74% ± 0.48%	283.9617 ± 24.1754
1	1	6.6027 ± 3.3987	0.00% ± 0.00%	210.97% ± 2.44%	504.7328 ± 35.3104
100	0.1	0.2225 ± 0.0092	0.97% ± 0.01%	23.51% ± 0.02%	64.9219 ± 1.9526

chess^0 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

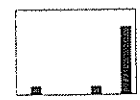
Support Vectors (%)



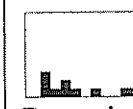
Norm

chess^0 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 93.90% ± 1.49%



Good Weak Hyp. (%): 99.60% ± 0.92%



Execution Time (s): 79.69 s ± 0.46 s

Table A.4: (continued)

chess[1] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	83.83% ± 1.83%	89.01% ± 7.07%	457.52 s ± 9.14 s
1	0.4	64.53% ± 6.66%	100.00% ± 0.00%	0.49 s ± 0.01 s
1	0.1	54.87% ± 2.96%	100.00% ± 0.00%	0.13 s ± 0.00 s
1000	0.1	82.10% ± 2.46%	94.78% ± 1.72%	123.12 s ± 0.56 s
1000	**0.7**	**84.57% ± 1.65%**	**81.91% ± 7.68%**	**751.56 s ± 17.27 s**
100	1	83.40% ± 1.55%	65.60% ± 19.95%	100.84 s ± 4.07 s
1	0.7	75.97% ± 5.87%	100.00% ± 0.00%	0.81 s ± 0.01 s
1000	1	82.97% ± 3.14%	49.88% ± 21.59%	990.40 s ± 38.57 s
100	0.7	84.07% ± 2.20%	84.30% ± 7.36%	75.49 s ± 1.49 s
100	0.4	83.00% ± 1.32%	92.60% ± 4.84%	46.17 s ± 0.70 s
1	1	74.03% ± 7.98%	100.00% ± 0.00%	1.09 s ± 0.02 s
100	0.1	83.47% ± 1.53%	98.60% ± 1.69%	12.44 s ± 0.03 s

chess[1] (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

chess[1] (average results for weak hypotheses)

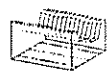
T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.5617 ± 0.2703	0.49% ± 0.02%	89.05% ± 0.83%	1134.4328 ± 589.7147
1	0.4	0.6637 ± 0.1086	0.49% ± 0.08%	89.80% ± 0.90%	251.9552 ± 16.1336
1	0.1	0.1571 ± 0.0287	0.98% ± 0.03%	23.40% ± 0.20%	58.8958 ± 3.7823
1000	0.1	0.1352 ± 0.0196	1.03% ± 0.01%	23.37% ± 0.06%	309.9936 ± 169.3858
1000	**0.7**	**0.6571 ± 0.4133**	**0.22% ± 0.02%**	**147.40% ± 2.06%**	**1458.2767 ± 682.7649**
100	1	0.6373 ± 0.7010	0.07% ± 0.03%	201.08% ± 3.75%	1502.7503 ± 640.2227
1	0.7	1.2484 ± 0.1546	0.18% ± 0.05%	151.90% ± 1.93%	474.6025 ± 29.1068
1000	1	0.3014 ± 0.7024	0.13% ± 0.15%	201.54% ± 3.76%	1307.2925 ± 540.4607
100	0.7	0.7217 ± 0.3761	0.21% ± 0.02%	147.68% ± 1.91%	1377.3964 ± 671.5939
100	0.4	0.5894 ± 0.2510	0.49% ± 0.02%	89.38% ± 0.73%	951.5431 ± 440.2647
1	1	1.9507 ± 0.7375	0.13% ± 0.21%	206.07% ± 4.25%	686.7002 ± 38.7519
100	0.1	0.1730 ± 0.0103	1.00% ± 0.01%	23.50% ± 0.03%	94.0435 ± 49.9580

Table A.4: (continued)

chess[1] (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

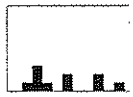
Error Rate (%)



Support Vectors (%)

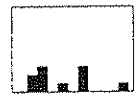
Norm

chess[1] (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 84.57% ± 1.65%

Good Weak Hyp. (%): 81.91% ± 7.68%



Execution Time (s): 751.56 s ± 17.27 s

chess[2] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	65.70% ± 1.91%	93.21% ± 3.57%	468.78 s ± 3.51 s
1	0.4	56.30% ± 1.77%	100.00% ± 0.00%	0.49 s ± 0.01 s
1	0.1	53.13% ± 2.11%	100.00% ± 0.00%	0.13 s ± 0.01 s
1000	0.1	65.33% ± 1.87%	93.40% ± 1.39%	123.44 s ± 0.32 s
1000	0.7	65.97% ± 1.93%	84.61% ± 5.25%	771.83 s ± 8.80 s
100	1	65.37% ± 1.91%	50.80% ± 11.96%	101.68 s ± 1.75 s
1	0.7	60.30% ± 4.27%	100.00% ± 0.00%	0.82 s ± 0.01 s
1000	1	66.43% ± 2.33%	32.02% ± 14.61%	993.66 s ± 16.82 s
100	**0.7**	**66.53% ± 2.17%**	**88.00% ± 4.60%**	**77.71 s ± 0.80 s**
100	0.4	65.67% ± 1.97%	95.90% ± 3.78%	47.24 s ± 0.31 s
1	1	62.23% ± 3.79%	100.00% ± 0.00%	1.11 s ± 0.01 s
100	0.1	63.70% ± 2.32%	95.80% ± 1.25%	12.41 s ± 0.03 s

Table A.4: (continued)

chess^2 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

chess^2 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.3923 ± 0.1125	0.63% ± 0.02%	90.75% ± 0.30%	513.6626 ± 85.0442
1	0.4	0.5157 ± 0.0649	0.61% ± 0.06%	90.97% ± 1.08%	305.5724 ± 24.3011
1	0.1	0.1338 ± 0.0367	1.01% ± 0.04%	23.53% ± 0.16%	66.5284 ± 4.7710
1000	0.1	0.0749 ± 0.0064	1.08% ± 0.00%	23.47% ± 0.02%	104.7901 ± 13.1210
1000	0.7	0.5728 ± 0.2494	0.31% ± 0.02%	151.51% ± 0.84%	809.5122 ± 104.7689
100	1	0.1633 ± 0.2394	0.12% ± 0.04%	207.93% ± 1.82%	984.1589 ± 74.0892
1	0.7	0.9487 ± 0.0821	0.31% ± 0.05%	153.57% ± 1.73%	605.5052 ± 33.3293
1000	1	-0.2731 ± 0.3161	0.14% ± 0.06%	208.18% ± 1.79%	974.9410 ± 92.1293
100	**0.7**	**0.6210 ± 0.2387**	**0.31% ± 0.02%**	**151.81% ± 0.82%**	**789.6537 ± 96.0523**
100	0.4	0.4419 ± 0.0928	0.64% ± 0.02%	91.00% ± 0.26%	455.8521 ± 54.5770
1	1	1.7811 ± 0.4038	0.09% ± 0.09%	211.57% ± 1.97%	859.4905 ± 33.3921
100	0.1	0.1183 ± 0.0030	1.04% ± 0.01%	23.53% ± 0.03%	74.2223 ± 5.3344

chess^2 (weak hypotheses' average result graphs for $T \times \rho$)

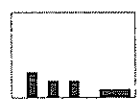
Alpha

Error Rate (%)

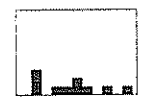
Support Vectors (%)

Norm

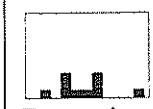
chess^2 (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 66.53% ± 2.17%



Good Weak Hyp. (%): 88.00% ± 4.60%



Execution Time (s): 77.71 s ± 0.80 s

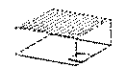
Table A.4: (continued)

gauss^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	100.00% ± 0.00%	99.80% ± 0.00%	10.17 s ± 0.00 s
1	**0.4**	**100.00% ± 0.00%**	**100.00% ± 0.00%**	**0.02 s ± 0.00 s**
1	0.1	100.00% ± 0.00%	100.00% ± 0.00%	0.03 s ± 0.00 s
1000	0.1	100.00% ± 0.00%	100.00% ± 0.00%	8.45 s ± 0.00 s
1000	0.7	100.00% ± 0.00%	100.00% ± 0.00%	12.03 s ± 0.00 s
100	1	100.00% ± 0.00%	100.00% ± 0.00%	1.90 s ± 0.00 s
1	0.7	100.00% ± 0.00%	100.00% ± 0.00%	0.03 s ± 0.00 s
1000	1	100.00% ± 0.00%	100.00% ± 0.00%	12.07 s ± 0.00 s
100	0.7	100.00% ± 0.00%	100.00% ± 0.00%	1.75 s ± 0.00 s
100	0.4	100.00% ± 0.00%	100.00% ± 0.00%	1.33 s ± 0.00 s
1	1	100.00% ± 0.00%	100.00% ± 0.00%	0.05 s ± 0.00 s
100	0.1	100.00% ± 0.00%	100.00% ± 0.00%	0.87 s ± 0.00 s

gauss^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)



Good Weak Hyp. (%)

Execution Time (s)

gauss^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	9.9027 ± 0.0000	0.00% ± 0.00%	2.45% ± 0.00%	1350.5541 ± 0.0000
1	**0.4**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**4.33% ± 0.00%**	**1323.1578 ± 0.0000**
1	0.1	3.2748 ± 0.0000	0.00% ± 0.00%	2.67% ± 0.00%	302.4907 ± 0.0000
1000	0.1	9.4905 ± 0.0000	0.00% ± 0.00%	2.09% ± 0.00%	1244.0704 ± 0.0000
1000	0.7	9.9436 ± 0.0000	0.00% ± 0.00%	2.72% ± 0.00%	1328.5686 ± 0.0000
100	1	9.8690 ± 0.0000	0.00% ± 0.00%	4.15% ± 0.00%	1323.3400 ± 0.0000
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	4.33% ± 0.00%	1323.1578 ± 0.0000
1000	1	9.9663 ± 0.0000	0.00% ± 0.00%	2.63% ± 0.00%	1450.7894 ± 0.0000
100	0.7	9.9327 ± 0.0000	0.00% ± 0.00%	3.82% ± 0.00%	1133.4881 ± 0.0000
100	0.4	9.7744 ± 0.0000	0.00% ± 0.00%	3.02% ± 0.00%	1227.0154 ± 0.0000
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	4.33% ± 0.00%	1273.7454 ± 0.0000
100	0.1	8.9067 ± 0.0000	0.00% ± 0.00%	2.14% ± 0.00%	1197.0902 ± 0.0000

Table A.4: (continued)

gauss^0 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

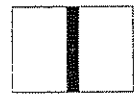
Error Rate (%)

Support Vectors (%)

Norm

gauss^0 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 100.00% ± 0.00%



Good Weak Hyp. (%): 100.00% ± 0.00%

Execution Time (s): 0.02 s ± 0.00 s

gauss^1 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	97.67% ± 0.00%	11.50% ± 0.00%	36.26 s ± 0.00 s
1	0.4	98.00% ± 0.00%	100.00% ± 0.00%	0.03 s ± 0.00 s
1	0.1	96.67% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	98.33% ± 0.00%	32.40% ± 0.00%	23.71 s ± 0.00 s
1000	**0.7**	**98.67% ± 0.00%**	**10.90% ± 0.00%**	**44.60 s ± 0.00 s**
100	1	98.33% ± 0.00%	11.00% ± 0.00%	4.64 s ± 0.00 s
1	0.7	96.00% ± 0.00%	100.00% ± 0.00%	0.05 s ± 0.00 s
1000	1	98.67% ± 0.00%	4.20% ± 0.00%	50.73 s ± 0.00 s
100	0.7	97.67% ± 0.00%	22.00% ± 0.00%	4.63 s ± 0.00 s
100	0.4	98.33% ± 0.00%	21.00% ± 0.00%	3.81 s ± 0.00 s
1	1	96.33% ± 0.00%	100.00% ± 0.00%	0.08 s ± 0.00 s
100	0.1	98.33% ± 0.00%	29.00% ± 0.00%	2.24 s ± 0.00 s

Table A.4: (continued)

gauss[1] (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

gauss[1] (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.4500 ± 0.0000	0.08% ± 0.00%	12.23% ± 0.00%	47794.2388 ± 0.0000
1	0.4	1.9837 ± 0.0000	0.04% ± 0.00%	3.33% ± 0.00%	15034.0742 ± 0.0000
1	0.1	1.6911 ± 0.0000	0.08% ± 0.00%	2.33% ± 0.00%	1540.8752 ± 0.0000
1000	0.1	-0.1653 ± 0.0000	0.18% ± 0.00%	8.31% ± 0.00%	28039.7499 ± 0.0000
1000	**0.7**	**-0.4228 ± 0.0000**	**0.06% ± 0.00%**	**14.93% ± 0.00%**	**40125.6203 ± 0.0000**
100	1	-0.4023 ± 0.0000	0.06% ± 0.00%	14.66% ± 0.00%	69097.9893 ± 0.0000
1	0.7	1.7632 ± 0.0000	0.07% ± 0.00%	8.67% ± 0.00%	4142.4697 ± 0.0000
1000	1	-0.6123 ± 0.0000	0.05% ± 0.00%	16.49% ± 0.00%	41228.2788 ± 0.0000
100	0.7	-0.1360 ± 0.0000	0.07% ± 0.00%	15.14% ± 0.00%	81610.2185 ± 0.0000
100	0.4	-0.4109 ± 0.0000	0.08% ± 0.00%	12.59% ± 0.00%	35343.1107 ± 0.0000
1	1	1.9459 ± 0.0000	0.05% ± 0.00%	17.00% ± 0.00%	3386.1904 ± 0.0000
100	0.1	-0.1224 ± 0.0000	0.18% ± 0.00%	7.63% ± 0.00%	7958.7905 ± 0.0000

gauss[1] (weak hypotheses' average result graphs for $T \times \rho$)

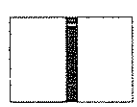
Alpha

Error Rate (%)

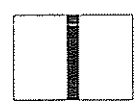
Support Vectors (%)

Norm

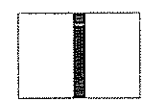
gauss[1] (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 98.67% ± 0.00%



Good Weak Hyp. (%): 10.90% ± 0.00%



Execution Time (s): 44.60 s ± 0.00 s

Table A.4: (continued)

gauss^2 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	90.00% ± 0.00%	22.30% ± 0.00%	122.60 s ± 0.00 s
1	0.4	78.67% ± 0.00%	100.00% ± 0.00%	0.09 s ± 0.00 s
1	0.1	90.33% ± 0.00%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	**0.1**	**93.00% ± 0.00%**	**46.50% ± 0.00%**	**42.11 s ± 0.00 s**
1000	0.7	89.00% ± 0.00%	12.60% ± 0.00%	169.42 s ± 0.00 s
100	1	92.33% ± 0.00%	19.00% ± 0.00%	22.33 s ± 0.00 s
1	0.7	68.00% ± 0.00%	100.00% ± 0.00%	0.18 s ± 0.00 s
1000	1	90.33% ± 0.00%	8.20% ± 0.00%	206.29 s ± 0.00 s
100	0.7	92.00% ± 0.00%	22.00% ± 0.00%	17.77 s ± 0.00 s
100	0.4	88.67% ± 0.00%	31.00% ± 0.00%	12.36 s ± 0.00 s
1	1	47.33% ± 0.00%	100.00% ± 0.00%	0.25 s ± 0.00 s
100	0.1	91.33% ± 0.00%	47.00% ± 0.00%	4.17 s ± 0.00 s

gauss^2 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

gauss^2 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.0595 ± 0.0000	0.50% ± 0.00%	47.96% ± 0.00%	4133.3771 ± 0.0000
1	0.4	0.5267 ± 0.0000	0.60% ± 0.00%	26.00% ± 0.00%	2568.6262 ± 0.0000
1	0.1	1.0454 ± 0.0000	0.26% ± 0.00%	9.00% ± 0.00%	1073.9448 ± 0.0000
1000	**0.1**	**-0.0045 ± 0.0000**	**0.78% ± 0.00%**	**15.82% ± 0.00%**	**2790.2747 ± 0.0000**
1000	0.7	-0.1616 ± 0.0000	0.39% ± 0.00%	66.36% ± 0.00%	5952.3016 ± 0.0000
100	1	-0.0581 ± 0.0000	0.42% ± 0.00%	84.35% ± 0.00%	11699.4812 ± 0.0000
1	0.7	0.6123 ± 0.0000	0.53% ± 0.00%	58.00% ± 0.00%	1350.0762 ± 0.0000
1000	1	-0.2271 ± 0.0000	0.34% ± 0.00%	79.91% ± 0.00%	9332.7204 ± 0.0000
100	0.7	-0.1176 ± 0.0000	0.42% ± 0.00%	67.33% ± 0.00%	2009.8065 ± 0.0000
100	0.4	-0.0658 ± 0.0000	0.49% ± 0.00%	47.02% ± 0.00%	5309.7124 ± 0.0000
1	1	0.1177 ± 0.0000	1.03% ± 0.00%	90.33% ± 0.00%	603.8593 ± 0.0000
100	0.1	0.0095 ± 0.0000	0.68% ± 0.00%	15.51% ± 0.00%	9153.1181 ± 0.0000

Table A.4: (continued)

gauss^2 (weak hypotheses' average result graphs for $T \times \rho$)

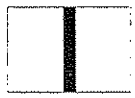
Alpha

Error Rate (%)

Support Vectors (%)

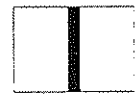
Norm

gauss^2 (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 93.00% ± 0.00%

Good Weak Hyp. (%): 46.50% ± 0.00%



Execution Time (s): 42.11 s ± 0.00 s

hepatitis (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	77.23% ± 5.79%	49.87% ± 1.90%	4.58 s ± 0.08 s
1	0.4	45.96% ± 32.42%	70.00% ± 45.83%	0.02 s ± 0.00 s
1	0.1	56.38% ± 29.25%	80.00% ± 40.00%	0.01 s ± 0.00 s
1000	0.1	74.68% ± 4.09%	59.36% ± 1.41%	1.24 s ± 0.01 s
1000	0.7	76.60% ± 7.25%	37.32% ± 3.09%	7.48 s ± 0.20 s
100	**1**	**78.30% ± 6.51%**	**41.50% ± 7.59%**	**1.03 s ± 0.04 s**
1	0.7	53.62% ± 28.14%	80.00% ± 40.00%	0.02 s ± 0.00 s
1000	1	75.32% ± 6.25%	25.79% ± 3.81%	9.91 s ± 0.33 s
100	0.7	76.60% ± 9.03%	50.70% ± 7.50%	0.77 s ± 0.03 s
100	0.4	74.89% ± 6.98%	56.60% ± 3.72%	0.47 s ± 0.01 s
1	1	61.06% ± 21.57%	90.00% ± 30.00%	0.02 s ± 0.00 s
100	0.1	75.11% ± 4.04%	63.70% ± 4.03%	0.14 s ± 0.00 s

Table A.4: (continued)

hepatitis (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

hepatitis (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.0008 ± 0.0044	0.96% ± 0.04%	63.57% ± 1.10%	1942240.8494 ± 426623.6388
1	0.4	0.1953 ± 0.3788	0.95% ± 0.39%	54.26% ± 6.18%	518308.1625 ± 1395607.0577
1	0.1	0.2934 ± 0.2678	0.84% ± 0.28%	13.40% ± 3.93%	1681441.3516 ± 1750213.9897
1000	0.1	0.0185 ± 0.0027	1.11% ± 0.01%	16.70% ± 0.14%	2663605.2557 ± 446452.2596
1000	0.7	-0.0400 ± 0.0109	0.87% ± 0.05%	105.66% ± 2.63%	1900553.4893 ± 403250.8987
100	**1**	**-0.0214 ± 0.0267**	**0.84% ± 0.07%**	**142.24% ± 5.30%**	**2522477.4449 ± 973428.0580**
1	0.7	0.2516 ± 0.3495	0.89% ± 0.37%	101.70% ± 10.46%	2329877.7984 ± 6026747.3535
1000	1	-0.1027 ± 0.0227	0.85% ± 0.08%	141.84% ± 4.47%	2765383.0264 ± 403173.9623
100	0.7	0.0101 ± 0.0190	0.87% ± 0.08%	104.94% ± 3.54%	2484364.5240 ± 957892.3037
100	0.4	0.0250 ± 0.0102	0.95% ± 0.05%	62.92% ± 1.16%	1900874.3873 ± 626200.2910
1	1	0.3112 ± 0.2055	0.81% ± 0.21%	147.87% ± 10.65%	799494.0656 ± 3983293.0464
100	0.1	0.0375 ± 0.0128	1.07% ± 0.04%	16.65% ± 0.29%	2282995.9531 ± 623600.0470

hepatitis (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

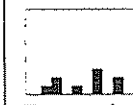
Support Vectors (%)

Norm

hepatitis (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 78.30% ± 6.51%

Good Weak Hyp. (%): 41.50% ± 7.59%



Execution Time (s): 1.03 s ± 0.04 s

Table A.4: (continued)

ionosphere (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	94.81% ± 1.05%	98.93% ± 0.46%	26.69 s ± 0.44 s
1	0.4	86.51% ± 12.85%	100.00% ± 0.00%	0.07 s ± 0.00 s
1	0.1	71.32% ± 8.77%	100.00% ± 0.00%	0.04 s ± 0.01 s
1000	0.1	94.72% ± 0.96%	98.76% ± 0.40%	8.35 s ± 0.08 s
1000	0.7	93.87% ± 0.63%	98.64% ± 0.72%	41.78 s ± 0.79 s
100	1	93.68% ± 0.85%	90.50% ± 16.51%	5.50 s ± 0.12 s
1	0.7	89.25% ± 8.16%	100.00% ± 0.00%	0.08 s ± 0.00 s
1000	1	93.77% ± 0.63%	93.33% ± 11.91%	54.59 s ± 1.46 s
100	0.7	94.06% ± 0.60%	98.40% ± 1.36%	4.22 s ± 0.09 s
100	**0.4**	**95.00% ± 0.95%**	**99.30% ± 0.64%**	**2.71 s ± 0.05 s**
1	1	92.08% ± 1.75%	100.00% ± 0.00%	0.09 s ± 0.00 s
100	0.1	94.34% ± 0.84%	99.30% ± 0.64%	0.87 s ± 0.01 s

ionosphere (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

ionosphere (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	2.1495 ± 0.1519	0.13% ± 0.01%	70.90% ± 1.31%	72.0871 ± 5.0645
1	0.4	1.4730 ± 0.4438	0.16% ± 0.16%	68.21% ± 4.98%	52.8561 ± 8.5701
1	0.1	0.4890 ± 0.2341	0.65% ± 0.19%	20.94% ± 1.83%	15.6440 ± 3.0738
1000	0.1	0.5505 ± 0.0441	0.64% ± 0.03%	22.60% ± 0.23%	27.2588 ± 1.4261
1000	0.7	4.0387 ± 0.5026	0.05% ± 0.01%	108.87% ± 2.20%	95.5864 ± 7.2172
100	1	7.4226 ± 1.9393	0.04% ± 0.03%	141.68% ± 3.91%	114.8033 ± 11.2721
1	0.7	1.8385 ± 0.5165	0.10% ± 0.15%	105.75% ± 5.17%	84.7279 ± 10.8406
1000	1	8.0601 ± 1.4797	0.03% ± 0.03%	141.12% ± 3.83%	114.6120 ± 9.5313
100	0.7	3.7323 ± 0.5087	0.06% ± 0.01%	108.82% ± 2.46%	95.1675 ± 6.9367
100	**0.4**	**2.1153 ± 0.1755**	**0.13% ± 0.02%**	**70.98% ± 1.42%**	**72.0412 ± 5.2055**
1	1	8.4595 ± 3.0813	0.00% ± 0.01%	140.57% ± 5.50%	112.8380 ± 9.6049
100	0.1	0.5580 ± 0.0483	0.64% ± 0.03%	22.58% ± 0.19%	25.7481 ± 1.5086

Table A.4: (continued)

ionosphere (weak hypotheses' average result graphs for $T \times \rho$)

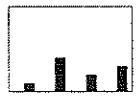

Alpha

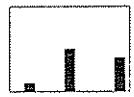

Error Rate (%)

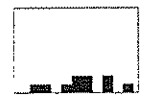

Support Vectors (%)


Norm

ionosphere (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 95.00% ± 0.95%


Good Weak Hyp. (%): 99.30% ± 0.64%


Execution Time (s): 2.71 s ± 0.05 s

musk (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	89.51% ± 0.00%	92.30% ± 0.00%	90.22 s ± 0.00 s
1	0.4	84.62% ± 0.00%	100.00% ± 0.00%	0.28 s ± 0.00 s
1	0.1	62.24% ± 0.00%	100.00% ± 0.00%	0.21 s ± 0.00 s
1000	0.1	90.21% ± 0.00%	94.60% ± 0.00%	26.98 s ± 0.00 s
1000	0.7	90.91% ± 0.00%	89.60% ± 0.00%	138.58 s ± 0.00 s
100	**1**	**93.01% ± 0.00%**	**73.00% ± 0.00%**	**17.95 s ± 0.00 s**
1	0.7	88.81% ± 0.00%	100.00% ± 0.00%	0.32 s ± 0.00 s
1000	1	93.01% ± 0.00%	73.60% ± 0.00%	179.10 s ± 0.00 s
100	0.7	91.61% ± 0.00%	92.00% ± 0.00%	14.08 s ± 0.00 s
100	0.4	90.21% ± 0.00%	90.00% ± 0.00%	9.13 s ± 0.00 s
1	1	83.22% ± 0.00%	100.00% ± 0.00%	0.36 s ± 0.00 s
100	0.1	90.21% ± 0.00%	90.00% ± 0.00%	2.80 s ± 0.00 s

Table A.4: (continued)

musk (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

musk (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.7131 ± 0.0000	0.38% ± 0.00%	72.02% ± 0.00%	271.5244 ± 0.0000
1	0.4	0.9154 ± 0.0000	0.32% ± 0.00%	66.43% ± 0.00%	223.5538 ± 0.0000
1	0.1	0.2739 ± 0.0000	0.85% ± 0.00%	17.48% ± 0.00%	62.3143 ± 0.0000
1000	0.1	0.1954 ± 0.0000	0.94% ± 0.00%	21.52% ± 0.00%	81.2018 ± 0.0000
1000	0.7	1.0166 ± 0.0000	0.21% ± 0.00%	108.14% ± 0.00%	383.2084 ± 0.0000
100	**1**	**0.7966 ± 0.0000**	**0.15% ± 0.00%**	**140.62% ± 0.00%**	**478.0366 ± 0.0000**
1	0.7	1.3005 ± 0.0000	0.16% ± 0.00%	106.99% ± 0.00%	391.3265 ± 0.0000
1000	1	1.1870 ± 0.0000	0.15% ± 0.00%	140.76% ± 0.00%	478.3758 ± 0.0000
100	0.7	0.9043 ± 0.0000	0.20% ± 0.00%	109.54% ± 0.00%	392.3124 ± 0.0000
100	0.4	0.5968 ± 0.0000	0.38% ± 0.00%	72.01% ± 0.00%	271.7855 ± 0.0000
1	1	1.2139 ± 0.0000	0.19% ± 0.00%	142.66% ± 0.00%	394.3179 ± 0.0000
100	0.1	0.2071 ± 0.0000	0.94% ± 0.00%	21.13% ± 0.00%	74.2074 ± 0.0000

musk (weak hypotheses' average result graphs for $T \times \rho$)

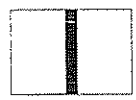
Alpha

Error Rate (%)

Support Vectors (%)

Norm

musk (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 93.01% ± 0.00%

Good Weak Hyp. (%): 73.00% ± 0.00%

Execution Time (s): 17.95 s ± 0.00 s

Table A.4: (continued)

pgs (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	87.19% ± 6.47%	99.61% ± 0.23%	4.90 s ± 0.02 s
1	0.4	74.69% ± 11.81%	100.00% ± 0.00%	0.03 s ± 0.00 s
1	0.1	55.00% ± 10.48%	100.00% ± 0.00%	0.03 s ± 0.00 s
1000	0.1	85.31% ± 6.26%	87.28% ± 1.60%	1.32 s ± 0.01 s
1000	0.7	86.88% ± 5.38%	98.86% ± 0.43%	7.96 s ± 0.07 s
100	1	85.62% ± 6.43%	99.30% ± 0.78%	1.10 s ± 0.01 s
1	0.7	78.75% ± 8.12%	100.00% ± 0.00%	0.03 s ± 0.01 s
1000	**1**	**88.12% ± 4.80%**	**98.70% ± 0.75%**	**10.81 s ± 0.13 s**
100	0.7	84.38% ± 5.41%	99.20% ± 0.98%	0.82 s ± 0.01 s
100	0.4	86.88% ± 5.56%	99.90% ± 0.30%	0.52 s ± 0.00 s
1	1	83.75% ± 8.71%	100.00% ± 0.00%	0.04 s ± 0.00 s
100	0.1	84.06% ± 6.77%	91.40% ± 4.22%	0.16 s ± 0.00 s

pgs (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

pgs (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	1.1787 ± 0.0773	0.42% ± 0.04%	88.33% ± 0.47%	269.9470 ± 9.5668
1	0.4	0.7067 ± 0.1739	0.47% ± 0.13%	86.25% ± 4.00%	214.3950 ± 31.2374
1	0.1	0.2393 ± 0.1035	0.89% ± 0.11%	24.69% ± 0.94%	62.6179 ± 16.2788
1000	0.1	0.1657 ± 0.0177	0.98% ± 0.02%	24.64% ± 0.04%	65.9010 ± 0.7574
1000	0.7	2.8079 ± 0.2713	0.16% ± 0.02%	140.44% ± 1.39%	397.5320 ± 17.4978
100	1	8.8283 ± 0.4086	0.02% ± 0.01%	185.97% ± 3.01%	506.7662 ± 27.5523
1	0.7	1.1809 ± 0.2388	0.22% ± 0.08%	137.50% ± 7.40%	363.4417 ± 37.1731
1000	**1**	**9.1858 ± 0.2847**	**0.02% ± 0.01%**	**186.79% ± 2.47%**	**506.2360 ± 29.1591**
100	0.7	2.5972 ± 0.3177	0.16% ± 0.02%	140.58% ± 1.25%	395.6311 ± 17.9595
100	0.4	1.1199 ± 0.0905	0.42% ± 0.04%	88.29% ± 0.46%	267.2556 ± 9.6664
1	1	8.4290 ± 3.1419	0.01% ± 0.01%	187.19% ± 6.77%	498.5016 ± 39.6821
100	0.1	0.1936 ± 0.0281	0.96% ± 0.02%	24.58% ± 0.09%	64.3529 ± 2.1929

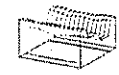
Table A.4: (continued)

pgs (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

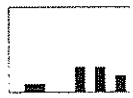
Error Rate (%)

Support Vectors (%)



Norm

pgs (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 88.12% ± 4.80%

Good Weak Hyp. (%): 98.70% ± 0.75%

Execution Time (s): 10.81 s ± 0.13 s

pid (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	72.60% ± 4.02%	43.32% ± 2.54%	96.24 s ± 1.07 s
1	0.4	40.22% ± 32.89%	60.00% ± 48.99%	0.11 s ± 0.01 s
1	0.1	60.52% ± 20.56%	90.00% ± 30.00%	0.04 s ± 0.01 s
1000	0.1	74.46% ± 2.94%	54.15% ± 1.54%	25.48 s ± 0.11 s
1000	0.7	73.03% ± 3.37%	30.31% ± 3.09%	159.33 s ± 2.85 s
100	1	74.42% ± 2.30%	36.10% ± 4.53%	21.76 s ± 0.46 s
1	0.7	45.84% ± 30.67%	70.00% ± 45.83%	0.18 s ± 0.01 s
1000	1	72.99% ± 2.58%	20.69% ± 2.65%	214.25 s ± 3.98 s
100	0.7	73.51% ± 2.73%	48.80% ± 4.92%	16.25 s ± 0.34 s
100	**0.4**	**75.71% ± 2.02%**	**58.00% ± 3.82%**	**9.73 s ± 0.13 s**
1	1	51.86% ± 26.35%	80.00% ± 40.00%	0.25 s ± 0.02 s
100	0.1	74.89% ± 1.96%	62.60% ± 3.41%	2.57 s ± 0.04 s

Table A.4: (continued)

pid (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

pid (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.0062 ± 0.0027	1.00% ± 0.02%	62.36% ± 0.52%	3755.7432 ± 263.8403
1	0.4	0.1535 ± 0.2597	0.99% ± 0.29%	54.11% ± 4.87%	5359.7477 ± 3064.8897
1	0.1	0.2997 ± 0.2238	0.83% ± 0.25%	12.99% ± 1.81%	4098.4712 ± 1025.5685
1000	0.1	0.0033 ± 0.0010	1.11% ± 0.01%	15.82% ± 0.06%	4089.3071 ± 270.3493
1000	0.7	-0.0236 ± 0.0055	0.92% ± 0.02%	105.12% ± 1.59%	5210.8183 ± 462.6854
100	1	-0.0162 ± 0.0075	0.84% ± 0.03%	141.90% ± 2.94%	7510.0867 ± 988.8801
1	0.7	0.1616 ± 0.2690	0.98% ± 0.30%	97.40% ± 6.04%	6022.7163 ± 5393.6630
1000	1	-0.0444 ± 0.0097	0.85% ± 0.04%	142.91% ± 2.68%	7537.0769 ± 933.8704
100	0.7	0.0043 ± 0.0096	0.93% ± 0.05%	104.41% ± 1.89%	5410.4378 ± 732.6878
100	**0.4**	**0.0177 ± 0.0044**	**0.98% ± 0.03%**	**61.72% ± 0.69%**	**3950.6172 ± 431.6465**
1	1	0.2428 ± 0.2396	0.90% ± 0.27%	143.55% ± 6.19%	8371.4107 ± 4273.5946
100	0.1	0.0229 ± 0.0047	1.09% ± 0.03%	15.63% ± 0.26%	4484.3355 ± 602.0800

pid (weak hypotheses' average result graphs for $T \times \rho$)

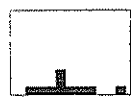
Alpha

Error Rate (%)

Support Vectors (%)

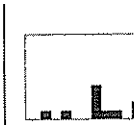
Norm

pid (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 75.71% ± 2.02%

Good Weak Hyp. (%): 58.00% ± 3.82%



Execution Time (s): 9.73 s ± 0.13 s

Table A.4: (continued)

ringnorm (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	97.67% ± 0.60%	65.23% ± 3.29%	167.75 s ± 2.64 s
1	0.4	86.30% ± 7.12%	100.00% ± 0.00%	0.24 s ± 0.01 s
1	0.1	70.13% ± 5.74%	100.00% ± 0.00%	0.12 s ± 0.01 s
1000	0.1	97.80% ± 0.65%	54.73% ± 2.47%	55.09 s ± 0.30 s
1000	0.7	97.83% ± 0.67%	66.41% ± 3.96%	257.58 s ± 3.96 s
100	1	97.83% ± 0.86%	64.40% ± 10.98%	33.13 s ± 0.42 s
1	0.7	94.63% ± 1.49%	100.00% ± 0.00%	0.34 s ± 0.01 s
1000	**1**	**98.13% ± 0.92%**	**56.81% ± 12.83%**	**328.96 s ± 6.99 s**
100	0.7	97.63% ± 0.95%	71.80% ± 3.68%	25.85 s ± 0.44 s
100	0.4	97.80% ± 0.78%	72.90% ± 1.97%	17.05 s ± 0.28 s
1	1	95.10% ± 3.16%	100.00% ± 0.00%	0.42 s ± 0.01 s
100	0.1	96.30% ± 0.80%	70.60% ± 4.43%	5.60 s ± 0.06 s

ringnorm (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

ringnorm (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.1350 ± 0.0339	0.25% ± 0.01%	53.44% ± 0.88%	4507.1654 ± 150.8922
1	0.4	1.0899 ± 0.3093	0.27% ± 0.14%	51.10% ± 2.06%	3887.5249 ± 318.1230
1	0.1	0.4975 ± 0.1509	0.64% ± 0.14%	14.50% ± 1.09%	843.1483 ± 162.4613
1000	0.1	0.0251 ± 0.0113	0.74% ± 0.01%	18.20% ± 0.07%	1118.7691 ± 31.4782
1000	0.7	0.1886 ± 0.0721	0.14% ± 0.01%	81.02% ± 1.47%	7156.6166 ± 292.6553
100	1	0.1927 ± 0.1413	0.10% ± 0.01%	103.77% ± 1.88%	9164.2359 ± 388.0517
1	0.7	1.5739 ± 0.1888	0.10% ± 0.03%	79.27% ± 2.27%	6539.8313 ± 409.3843
1000	**1**	**0.0202 ± 0.3190**	**0.10% ± 0.01%**	**103.98% ± 1.91%**	**9091.6499 ± 381.8765**
100	0.7	0.2344 ± 0.0647	0.14% ± 0.02%	80.96% ± 1.45%	7246.6255 ± 245.6685
100	0.4	0.2390 ± 0.0182	0.25% ± 0.02%	53.80% ± 0.78%	4544.1175 ± 94.9750
1	1	1.6818 ± 0.2742	0.09% ± 0.06%	104.43% ± 2.72%	8877.2621 ± 656.1083
100	0.1	0.1300 ± 0.0186	0.74% ± 0.02%	17.69% ± 0.17%	1114.9491 ± 33.2755

Table A.4: (continued)

ringnorm (weak hypotheses' average result graphs for $T \times \rho$)

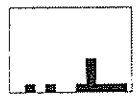
Alpha

Error Rate (%)

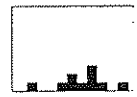
Support Vectors (%)

Norm

ringnorm (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 98.13% ± 0.92%



Good Weak Hyp. (%): 56.81% ± 12.83%

Execution Time (s): 328.96 s ± 6.99 s

spect^{b} (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	77.06% ± 0.61%	46.83% ± 1.70%	2.28 s ± 0.03 s
1	0.4	67.81% ± 23.79%	90.00% ± 30.00%	0.02 s ± 0.00 s
1	0.1	72.35% ± 9.22%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	77.27% ± 0.64%	46.85% ± 1.45%	1.06 s ± 0.01 s
1000	**0.7**	**77.81% ± 0.36%**	**52.77% ± 1.74%**	**3.49 s ± 0.04 s**
100	1	77.11% ± 0.89%	54.80% ± 12.34%	0.48 s ± 0.03 s
1	0.7	73.48% ± 10.18%	100.00% ± 0.00%	0.02 s ± 0.01 s
1000	1	77.54% ± 0.83%	63.73% ± 2.66%	4.91 s ± 0.07 s
100	0.7	77.38% ± 0.86%	66.10% ± 3.83%	0.41 s ± 0.02 s
100	0.4	76.58% ± 0.79%	66.90% ± 4.97%	0.31 s ± 0.01 s
1	1	72.25% ± 7.09%	100.00% ± 0.00%	0.02 s ± 0.00 s
100	0.1	77.65% ± 1.90%	76.10% ± 2.84%	0.15 s ± 0.01 s

Table A.4: (continued)

spect^b (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

spect^b (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0089 ± 0.0014	0.06% ± 0.00%	7.66% ± 0.06%	1080.5755 ± 10.9349
1	0.4	0.8629 ± 0.3372	0.07% ± 0.05%	8.45% ± 1.14%	847.3306 ± 150.9267
1	0.1	0.7756 ± 0.1173	0.08% ± 0.01%	3.21% ± 0.53%	299.5742 ± 130.3430
1000	0.1	0.0078 ± 0.0017	0.09% ± 0.00%	3.64% ± 0.03%	528.4209 ± 10.0255
1000	**0.7**	**0.0109 ± 0.0020**	**0.05% ± 0.00%**	**11.36% ± 0.08%**	**1394.5102 ± 21.6647**
100	1	0.0558 ± 0.0684	0.04% ± 0.00%	15.34% ± 0.24%	1890.9036 ± 57.5998
1	0.7	1.0247 ± 0.2942	0.05% ± 0.03%	11.93% ± 1.79%	1219.8344 ± 180.0926
1000	1	0.0080 ± 0.0061	0.05% ± 0.00%	15.15% ± 0.10%	1766.6958 ± 30.5167
100	0.7	0.1081 ± 0.0134	0.05% ± 0.00%	12.23% ± 0.31%	1598.3270 ± 57.9457
100	0.4	0.1205 ± 0.0104	0.06% ± 0.00%	9.01% ± 0.20%	1276.9132 ± 42.2702
1	1	1.2159 ± 0.2780	0.04% ± 0.02%	16.04% ± 1.04%	1886.5029 ± 296.7165
100	0.1	0.1412 ± 0.0086	0.12% ± 0.01%	4.09% ± 0.08%	601.8959 ± 25.1444

spect^b (weak hypotheses' average result graphs for $T \times \rho$)

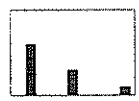
Alpha

Error Rate (%)

Support Vectors (%)

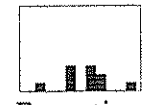
Norm

spect^b (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 77.81% ± 0.36%

Good Weak Hyp. (%): 52.77% ± 1.74%



Execution Time (s): 3.49 s ± 0.04 s

Table A.4: (continued)

spect^r (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	83.16% ± 0.24%	99.00% ± 0.34%	6.75 s ± 0.03 s
1	0.4	73.87% ± 11.93%	100.00% ± 0.00%	0.04 s ± 0.00 s
1	0.1	50.59% ± 23.02%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	0.1	82.12% ± 0.86%	87.56% ± 0.98%	1.92 s ± 0.02 s
1000	0.7	83.20% ± 0.32%	90.30% ± 1.07%	9.69 s ± 0.11 s
100	1	82.38% ± 0.83%	85.70% ± 9.28%	1.25 s ± 0.10 s
1	0.7	79.03% ± 11.75%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	1	82.83% ± 0.32%	88.58% ± 2.26%	12.45 s ± 0.27 s
100	0.7	83.12% ± 0.58%	93.60% ± 2.76%	1.03 s ± 0.02 s
100	0.4	82.94% ± 0.73%	99.00% ± 1.18%	0.70 s ± 0.01 s
1	**1**	**84.31% ± 2.88%**	**100.00% ± 0.00%**	**0.05 s ± 0.00 s**
100	0.1	83.09% ± 1.20%	90.10% ± 2.30%	0.22 s ± 0.00 s

spect' (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

spect^r (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	1.1390 ± 0.0128	0.06% ± 0.00%	11.20% ± 0.03%	42.6182 ± 0.1957
1	0.4	0.7832 ± 0.1375	0.05% ± 0.01%	11.08% ± 0.59%	34.8510 ± 3.8296
1	0.1	0.2637 ± 0.2377	0.11% ± 0.03%	3.20% ± 0.25%	7.3649 ± 2.4294
1000	0.1	0.1679 ± 0.0044	0.13% ± 0.00%	3.27% ± 0.00%	11.4738 ± 0.1083
1000	0.7	1.7987 ± 0.0296	0.03% ± 0.00%	16.67% ± 0.12%	56.5504 ± 0.4720
100	1	3.8963 ± 1.2713	0.02% ± 0.01%	21.40% ± 1.08%	71.9562 ± 3.3906
1	0.7	1.2610 ± 0.2357	0.02% ± 0.01%	17.47% ± 1.18%	56.0524 ± 6.0733
1000	1	4.7622 ± 0.2828	0.02% ± 0.00%	21.51% ± 0.31%	72.3587 ± 0.7647
100	0.7	1.7894 ± 0.1295	0.03% ± 0.00%	16.94% ± 0.24%	57.7153 ± 1.0709
100	0.4	1.0887 ± 0.0278	0.06% ± 0.00%	11.19% ± 0.10%	42.2094 ± 0.5682
1	**1**	**3.0218 ± 3.5500**	**0.03% ± 0.03%**	**23.35% ± 1.10%**	**76.3227 ± 7.1969**
100	0.1	0.1841 ± 0.0186	0.12% ± 0.00%	3.26% ± 0.02%	10.9349 ± 0.2804

Table A.4: (continued)

spect^r (weak hypotheses' average result graphs for $T \times \rho$)

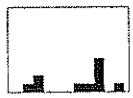
Alpha

Error Rate (%)

Support Vectors (%)

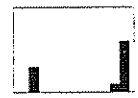
Norm

spect^r (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 84.31% ± 2.88%

Good Weak Hyp. (%): 100.00% ± 0.00%



Execution Time (s): 0.05 s ± 0.00 s

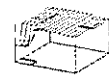
spiral^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	54.00% ± 0.00%	97.70% ± 0.00%	495.51 s ± 0.00 s
1	0.4	50.00% ± 0.00%	100.00% ± 0.00%	0.52 s ± 0.00 s
1	0.1	52.00% ± 0.00%	100.00% ± 0.00%	0.15 s ± 0.00 s
1000	**0.1**	**61.00% ± 0.00%**	**92.10% ± 0.00%**	**127.82 s ± 0.00 s**
1000	0.7	54.00% ± 0.00%	94.90% ± 0.00%	856.48 s ± 0.00 s
100	1	52.33% ± 0.00%	46.00% ± 0.00%	112.80 s ± 0.00 s
1	0.7	50.67% ± 0.00%	100.00% ± 0.00%	0.87 s ± 0.00 s
1000	1	53.33% ± 0.00%	97.10% ± 0.00%	1217.22 s ± 0.00 s
100	0.7	55.33% ± 0.00%	99.00% ± 0.00%	85.46 s ± 0.00 s
100	0.4	53.00% ± 0.00%	97.00% ± 0.00%	49.83 s ± 0.00 s
1	1	55.33% ± 0.00%	100.00% ± 0.00%	1.23 s ± 0.00 s
100	0.1	59.00% ± 0.00%	96.00% ± 0.00%	12.88 s ± 0.00 s

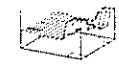
Table A.4: (continued)

spiral^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)



Good Weak Hyp. (%)



Execution Time (s)

spiral^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.6619 ± 0.0000	0.67% ± 0.00%	91.35% ± 0.00%	420.4900 ± 0.0000
1	0.4	0.4865 ± 0.0000	0.64% ± 0.00%	91.67% ± 0.00%	949.6679 ± 0.0000
1	0.1	0.0946 ± 0.0000	1.06% ± 0.00%	23.67% ± 0.00%	58.2032 ± 0.0000
1000	**0.1**	**0.1260 ± 0.0000**	**1.05% ± 0.00%**	**23.50% ± 0.00%**	**114.3259 ± 0.0000**
1000	0.7	1.5414 ± 0.0000	0.33% ± 0.00%	158.00% ± 0.00%	669.4644 ± 0.0000
100	1	2.3157 ± 0.0000	0.01% ± 0.00%	224.44% ± 0.00%	911.8566 ± 0.0000
1	0.7	0.8729 ± 0.0000	0.35% ± 0.00%	157.67% ± 0.00%	1080.9086 ± 0.0000
1000	1	1.3148 ± 0.0000	0.01% ± 0.00%	224.49% ± 0.00%	821.5846 ± 0.0000
100	0.7	1.4128 ± 0.0000	0.34% ± 0.00%	158.30% ± 0.00%	658.2381 ± 0.0000
100	0.4	0.6150 ± 0.0000	0.66% ± 0.00%	91.78% ± 0.00%	404.0470 ± 0.0000
1	1	2.9275 ± 0.0000	0.01% ± 0.00%	225.67% ± 0.00%	805.5492 ± 0.0000
100	0.1	0.1285 ± 0.0000	1.04% ± 0.00%	23.57% ± 0.00%	65.3634 ± 0.0000

spiral^0 (weak hypotheses' average result graphs for $T \times \rho$)

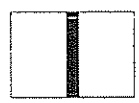
Alpha

Error Rate (%)

Support Vectors (%)

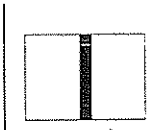
Norm

spiral^0 (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 61.00% ± 0.00%

Good Weak Hyp. (%): 92.10% ± 0.00%



Execution Time (s): 127.82 s ± 0.00 s

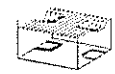
Table A.4: (continued)

spiral[1] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	54.67% ± 0.00%	100.00% ± 0.00%	523.16 s ± 0.00 s
1	0.4	52.67% ± 0.00%	100.00% ± 0.00%	0.54 s ± 0.00 s
1	0.1	52.67% ± 0.00%	100.00% ± 0.00%	0.15 s ± 0.00 s
1000	0.1	55.00% ± 0.00%	95.00% ± 0.00%	132.03 s ± 0.00 s
1000	0.7	55.00% ± 0.00%	100.00% ± 0.00%	898.54 s ± 0.00 s
100	1	54.33% ± 0.00%	100.00% ± 0.00%	128.35 s ± 0.00 s
1	0.7	49.33% ± 0.00%	100.00% ± 0.00%	0.92 s ± 0.00 s
1000	1	56.33% ± 0.00%	100.00% ± 0.00%	1288.83 s ± 0.00 s
100	0.7	51.33% ± 0.00%	100.00% ± 0.00%	90.26 s ± 0.00 s
100	0.4	53.67% ± 0.00%	100.00% ± 0.00%	52.30 s ± 0.00 s
1	1	50.00% ± 0.00%	100.00% ± 0.00%	1.29 s ± 0.00 s
100	**0.1**	**56.67% ± 0.00%**	**92.00% ± 0.00%**	**13.11 s ± 0.00 s**

spiral[1] (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)



Good Weak Hyp. (%)

Execution Time (s)

spiral[1] (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.8139 ± 0.0000	0.70% ± 0.00%	93.52% ± 0.00%	241.4674 ± 0.0000
1	0.4	0.4305 ± 0.0000	0.69% ± 0.00%	93.33% ± 0.00%	243.9210 ± 0.0000
1	0.1	0.0946 ± 0.0000	1.06% ± 0.00%	23.67% ± 0.00%	60.3085 ± 0.0000
1000	0.1	0.1333 ± 0.0000	1.05% ± 0.00%	23.64% ± 0.00%	61.9354 ± 0.0000
1000	0.7	2.2221 ± 0.0000	0.34% ± 0.00%	163.19% ± 0.00%	424.7696 ± 0.0000
100	1	10.0000 ± 0.0000	0.00% ± 0.00%	232.65% ± 0.00%	609.3001 ± 0.0000
1	0.7	0.8507 ± 0.0000	0.36% ± 0.00%	163.33% ± 0.00%	395.3812 ± 0.0000
1000	1	9.9927 ± 0.0000	0.00% ± 0.00%	232.69% ± 0.00%	612.3262 ± 0.0000
100	0.7	2.0013 ± 0.0000	0.33% ± 0.00%	163.16% ± 0.00%	425.5225 ± 0.0000
100	0.4	0.7816 ± 0.0000	0.69% ± 0.00%	93.52% ± 0.00%	240.8535 ± 0.0000
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	232.33% ± 0.00%	596.4028 ± 0.0000
100	**0.1**	**0.1140 ± 0.0000**	**1.05% ± 0.00%**	**23.64% ± 0.00%**	**61.5991 ± 0.0000**

Table A.4: (continued)

spiral[1] (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

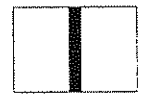
Error Rate (%)

Support Vectors (%)

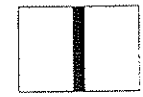
Norm

spiral[1] (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 56.67% ± 0.00%



Good Weak Hyp. (%): 92.00% ± 0.00%



Execution Time (s): 13.11 s ± 0.00 s

spiral[2] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	52.33% ± 0.00%	46.70% ± 0.00%	19156.59 s ± 0.00 s
1	0.4	0.00% ± 0.00%	0.00% ± 0.00%	18.07 s ± 0.00 s
1	0.1	47.67% ± 0.00%	100.00% ± 0.00%	6.10 s ± 0.00 s
1000	0.1	48.67% ± 0.00%	42.10% ± 0.00%	5089.58 s ± 0.00 s
1000	0.7	46.67% ± 0.00%	54.40% ± 0.00%	33968.40 s ± 0.00 s
100	1	51.00% ± 0.00%	60.00% ± 0.00%	4949.10 s ± 0.00 s
1	0.7	0.00% ± 0.00%	0.00% ± 0.00%	34.63 s ± 0.00 s
1000	**1**	**52.67% ± 0.00%**	**57.60% ± 0.00%**	**49156.51 s ± 0.00 s**
100	0.7	50.33% ± 0.00%	55.00% ± 0.00%	3459.17 s ± 0.00 s
100	0.4	49.00% ± 0.00%	51.00% ± 0.00%	1946.93 s ± 0.00 s
1	1	0.00% ± 0.00%	0.00% ± 0.00%	52.35 s ± 0.00 s
100	0.1	47.00% ± 0.00%	49.00% ± 0.00%	549.23 s ± 0.00 s

Table A.4: (continued)

spiral^2 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

spiral^2 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.0021 ± 0.0000	1.17% ± 0.00%	20.76% ± 0.00%	-2.6232e+14 ± 0.0000e+00
1	0.4	-0.0229 ± 0.0000	1.19% ± 0.00%	22.33% ± 0.00%	-1.5517e+12 ± 0.0000e+00
1	0.1	0.0543 ± 0.0000	1.10% ± 0.00%	9.00% ± 0.00%	-4.0265e+08 ± 0.0000e+00
1000	0.1	-0.0063 ± 0.0000	1.16% ± 0.00%	8.26% ± 0.00%	-3.2023e+09 ± 0.0000e+00
1000	0.7	-0.0014 ± 0.0000	1.17% ± 0.00%	31.42% ± 0.00%	-2.9315e+18 ± 0.0000e+00
100	1	-0.0014 ± 0.0000	1.17% ± 0.00%	41.11% ± 0.00%	-1.5436e+21 ± 0.0000e+00
1	0.7	-0.0229 ± 0.0000	1.19% ± 0.00%	32.00% ± 0.00%	-2.0786e+15 ± 0.0000e+00
1000	**1**	**-0.0013 ± 0.0000**	**1.17% ± 0.00%**	**41.32% ± 0.00%**	**-1.7932e+22 ± 0.0000e+00**
100	0.7	-0.0013 ± 0.0000	1.17% ± 0.00%	31.31% ± 0.00%	3.1475e+17 ± 0.0000e+00
100	0.4	-0.0020 ± 0.0000	1.16% ± 0.00%	20.79% ± 0.00%	-2.5035e+14 ± 0.0000e+00
1	1	-0.0314 ± 0.0000	1.20% ± 0.00%	44.33% ± 0.00%	-1.2794e+19 ± 0.0000e+00
100	0.1	-0.0001 ± 0.0000	1.17% ± 0.00%	8.02% ± 0.00%	1.6129e+10 ± 0.0000e+00

spiral^2 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

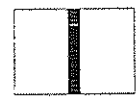
Error Rate (%)

Support Vectors (%)

Norm

spiral^2 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 52.67% ± 0.00%



Good Weak Hyp. (%): 57.60% ± 0.00%

Execution Time (s): 49156.51 s ± 0.00 s

Table A.4: (continued)

twonorm (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	97.27% ± 0.68%	27.90% ± 5.68%	70.39 s ± 3.08 s
1	0.4	94.57% ± 2.08%	100.00% ± 0.00%	0.14 s ± 0.01 s
1	0.1	94.27% ± 1.62%	100.00% ± 0.00%	0.10 s ± 0.01 s
1000	0.1	97.17% ± 0.64%	40.68% ± 6.49%	38.03 s ± 1.01 s
1000	0.7	97.43% ± 0.76%	25.03% ± 3.95%	91.10 s ± 3.80 s
100	1	97.33% ± 0.56%	34.40% ± 5.78%	10.47 s ± 0.42 s
1	0.7	94.73% ± 1.99%	100.00% ± 0.00%	0.17 s ± 0.01 s
1000	1	97.33% ± 0.61%	22.06% ± 7.06%	106.35 s ± 4.72 s
100	**0.7**	**97.50% ± 0.70%**	**39.60% ± 6.71%**	**9.20 s ± 0.36 s**
100	0.4	97.17% ± 0.70%	40.80% ± 3.92%	7.30 s ± 0.33 s
1	1	96.23% ± 0.56%	100.00% ± 0.00%	0.19 s ± 0.01 s
100	0.1	97.17% ± 0.67%	55.40% ± 5.04%	4.00 s ± 0.13 s

twonorm (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

twonorm (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	-0.1648 ± 0.0304	0.11% ± 0.01%	22.04% ± 1.25%	2.1942e+06 ± 1.2051e+05
1	0.4	1.5665 ± 0.1940	0.10% ± 0.04%	18.37% ± 1.41%	2.0825e+06 ± 3.2312e+05
1	0.1	1.5124 ± 0.1768	0.11% ± 0.05%	9.00% ± 0.91%	9.4658e+05 ± 1.7060e+05
1000	0.1	-0.0505 ± 0.0303	0.21% ± 0.02%	12.24% ± 0.50%	1.4967e+06 ± 4.6813e+04
1000	0.7	-0.2140 ± 0.0263	0.09% ± 0.01%	27.96% ± 1.58%	2.5974e+06 ± 1.0082e+05
100	1	-0.1044 ± 0.0396	0.08% ± 0.01%	31.30% ± 1.57%	2.9719e+06 ± 7.2526e+04
1	0.7	1.5570 ± 0.1985	0.11% ± 0.04%	25.33% ± 1.99%	2.5558e+06 ± 3.1535e+05
1000	1	-0.2382 ± 0.0356	0.08% ± 0.01%	32.24% ± 1.89%	2.8719e+06 ± 9.5798e+04
100	**0.7**	**-0.0872 ± 0.0718**	**0.09% ± 0.01%**	**27.58% ± 1.31%**	**2.7218e+06 ± 1.0497e+05**
100	0.4	-0.0560 ± 0.0385	0.11% ± 0.01%	22.04% ± 1.27%	2.2651e+06 ± 9.6902e+04
1	1	1.6757 ± 0.1687	0.08% ± 0.03%	32.13% ± 2.32%	3.0647e+06 ± 3.5066e+05
100	0.1	0.0585 ± 0.0246	0.21% ± 0.02%	12.41% ± 0.46%	1.5778e+06 ± 5.3867e+04

Table A.4: (continued)

twonorm (weak hypotheses' average result graphs for $T \times \rho$)

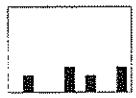
Alpha

Error Rate (%)

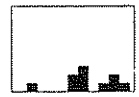
Support Vectors (%)

Norm

twonorm (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 97.50% ± 0.70%



Good Weak Hyp. (%): 39.60% ± 6.71%

Execution Time (s): 9.20 s ± 0.36 s

wdbc (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	**0.4**	**93.80% ± 2.01%**	**21.45% ± 2.81%**	**34.99 s ± 1.41 s**
1	0.4	81.52% ± 27.28%	90.00% ± 30.00%	0.09 s ± 0.01 s
1	0.1	85.96% ± 4.86%	100.00% ± 0.00%	0.07 s ± 0.01 s
1000	0.1	92.69% ± 2.62%	49.52% ± 2.21%	14.04 s ± 0.18 s
1000	0.7	92.87% ± 1.71%	14.99% ± 3.53%	46.31 s ± 2.18 s
100	1	92.11% ± 3.08%	24.20% ± 7.26%	5.62 s ± 0.32 s
1	0.7	77.49% ± 26.33%	90.00% ± 30.00%	0.11 s ± 0.01 s
1000	1	93.10% ± 2.34%	10.86% ± 4.25%	54.75 s ± 2.83 s
100	0.7	93.39% ± 1.53%	27.00% ± 4.10%	4.80 s ± 0.27 s
100	0.4	93.16% ± 1.41%	33.80% ± 6.85%	3.66 s ± 0.16 s
1	1	90.41% ± 2.88%	100.00% ± 0.00%	0.13 s ± 0.01 s
100	0.1	92.22% ± 2.27%	61.70% ± 5.85%	1.46 s ± 0.03 s

Table A.4: (continued)

wdbc (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

wdbc (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	**0.4**	**-0.1255 ± 0.0374**	**0.35% ± 0.03%**	**34.94% ± 1.51%**	**3.3704e+03 ± 5.1865e+02**
1	0.4	0.9677 ± 0.6312	0.39% ± 0.54%	21.99% ± 3.30%	1.5114e+03 ± 1.2233e+03
1	0.1	0.9433 ± 0.2627	0.33% ± 0.16%	6.32% ± 1.61%	3.5262e+02 ± 3.3601e+02
1000	0.1	-0.0006 ± 0.0076	0.70% ± 0.04%	13.79% ± 0.16%	1.8427e+03 ± 2.7510e+02
1000	0.7	-0.2027 ± 0.0538	0.28% ± 0.03%	45.42% ± 2.26%	4.5280e+03 ± 5.6315e+02
100	1	-0.1225 ± 0.0545	0.27% ± 0.05%	52.90% ± 3.47%	5.3082e+03 ± 6.7575e+02
1	0.7	0.8101 ± 0.4527	0.45% ± 0.40%	40.23% ± 4.60%	2.0909e+03 ± 1.1377e+03
1000	1	-0.3185 ± 0.0863	0.24% ± 0.03%	52.72% ± 3.00%	5.3309e+03 ± 4.2405e+02
100	0.7	-0.1504 ± 0.0346	0.30% ± 0.02%	45.48% ± 2.87%	4.4233e+03 ± 6.8457e+02
100	0.4	-0.0683 ± 0.0457	0.35% ± 0.05%	35.30% ± 1.57%	3.4271e+03 ± 6.0186e+02
1	1	1.1836 ± 0.1653	0.21% ± 0.06%	58.25% ± 4.88%	2.5770e+03 ± 9.2576e+02
100	0.1	0.0572 ± 0.0192	0.70% ± 0.05%	13.65% ± 0.20%	1.6725e+03 ± 2.2048e+02

wdbc (weak hypotheses' average result graphs for $T \times \rho$)

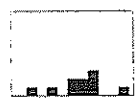
Alpha

Error Rate (%)

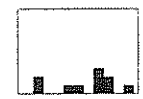
Support Vectors (%)

Norm

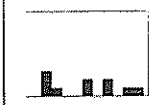
wdbc (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 93.80% ± 2.01%



Good Weak Hyp. (%): 21.45% ± 2.81%



Execution Time (s): 34.99 s ± 1.41 s

Table A.4: (continued)

wpbc (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	74.00% ± 5.88%	51.02% ± 2.37%	7.31 s ± 0.06 s
1	0.4	52.50% ± 27.40%	80.00% ± 40.00%	0.03 s ± 0.00 s
1	0.1	39.83% ± 33.73%	60.00% ± 48.99%	0.02 s ± 0.01 s
1000	0.1	75.17% ± 4.91%	58.40% ± 1.53%	1.93 s ± 0.01 s
1000	0.7	66.33% ± 8.52%	41.68% ± 2.33%	12.24 s ± 0.12 s
100	1	73.50% ± 3.69%	42.70% ± 8.60%	1.73 s ± 0.03 s
1	0.7	61.83% ± 11.46%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	1	71.33% ± 7.95%	28.55% ± 6.57%	16.73 s ± 0.30 s
100	0.7	71.33% ± 4.88%	54.10% ± 6.47%	1.26 s ± 0.03 s
100	0.4	73.17% ± 6.43%	60.40% ± 5.14%	0.76 s ± 0.01 s
1	1	54.00% ± 28.00%	80.00% ± 40.00%	0.04 s ± 0.01 s
100	**0.1**	**75.67% ± 7.54%**	**64.10% ± 4.68%**	**0.21 s ± 0.01 s**

wpbc (strong hypotheses' average result graphs for $T \times \rho$)


Accuracy (%)


Good Weak Hyp. (%)


Execution Time (s)

wpbc (average results for weak hypotheses)

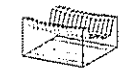
T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0008 ± 0.0028	1.04% ± 0.02%	61.45% ± 0.55%	3.8884e+02 ± 4.4272e+01
1	0.4	0.2189 ± 0.2811	0.92% ± 0.30%	55.17% ± 6.69%	5.5919e+02 ± 5.0777e+02
1	0.1	0.1526 ± 0.3305	0.99% ± 0.36%	12.67% ± 2.49%	3.9726e+02 ± 5.0596e+02
1000	0.1	0.0111 ± 0.0018	1.11% ± 0.02%	15.67% ± 0.07%	3.1021e+02 ± 3.8120e+01
1000	0.7	-0.0151 ± 0.0045	0.98% ± 0.02%	104.49% ± 1.13%	4.3775e+02 ± 4.1915e+01
100	1	-0.0124 ± 0.0264	0.91% ± 0.04%	144.19% ± 2.10%	5.3288e+02 ± 5.6168e+01
1	0.7	0.2513 ± 0.1322	0.87% ± 0.14%	95.00% ± 10.11%	5.7140e+02 ± 3.9682e+02
1000	1	-0.0489 ± 0.0193	0.93% ± 0.07%	145.00% ± 1.86%	5.3497e+02 ± 6.8301e+01
100	0.7	0.0127 ± 0.0123	0.97% ± 0.05%	103.72% ± 1.64%	4.5995e+02 ± 5.9258e+01
100	0.4	0.0233 ± 0.0079	1.02% ± 0.02%	61.04% ± 1.05%	3.5341e+02 ± 3.5637e+01
1	1	0.2202 ± 0.2868	0.92% ± 0.31%	134.33% ± 10.75%	5.0767e+02 ± 2.3048e+02
100	**0.1**	**0.0321 ± 0.0081**	**1.09% ± 0.03%**	**15.57% ± 0.29%**	**2.3255e+02 ± 3.5569e+01**

Table A.4: (continued)

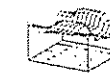
wpbc (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

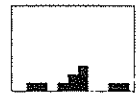


Support Vectors (%)



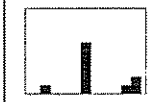
Norm

wpbc (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 75.67% ± 7.54%

Good Weak Hyp. (%): 64.10% ± 4.68%



Execution Time (s): 0.21 s ± 0.01 s

A.2.4 Complete Results for SMO-B$_\delta$

Table A.5 brings the complete results for the hybrid algorithm **SMO-B$_\delta$**, as previously described in Section 5.3.4. The algorithm was experimented with variations of T and ρ, its two regularization parameters that influence the behavior of the Boosting module and its interface with the weak learner. The results presented in this section are further analyzed in Section 5.4.4, where they are thoroughly discussed.

Table A.5: Average and best results for hybrid algorithm **SMO-B**$_\delta$ after 10 rounds of experiments with distinct training and testing sets.

bcw (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	96.14% ± 1.45%	70.54% ± 1.46%	68.12 s ± 0.91 s
1	0.4	92.67% ± 4.41%	100.00% ± 0.00%	0.03 s ± 0.01 s
1	0.1	91.86% ± 6.38%	100.00% ± 0.00%	0.02 s ± 0.01 s
1000	**0.1**	**96.29% ± 1.63%**	**69.17% ± 1.54%**	**17.62 s ± 0.16 s**
1000	0.7	96.00% ± 1.21%	71.67% ± 2.46%	118.28 s ± 2.35 s
100	1	96.14% ± 1.56%	80.40% ± 5.02%	16.08 s ± 0.38 s
1	0.7	92.14% ± 5.98%	100.00% ± 0.00%	0.03 s ± 0.01 s
1000	1	95.81% ± 1.24%	73.61% ± 1.94%	168.21 s ± 3.33 s
100	0.7	96.00% ± 1.45%	80.10% ± 3.21%	11.29 s ± 0.22 s
100	0.4	96.19% ± 1.52%	80.20% ± 2.60%	6.51 s ± 0.10 s
1	1	94.76% ± 3.47%	100.00% ± 0.00%	0.03 s ± 0.00 s
100	0.1	96.14% ± 1.35%	74.50% ± 3.50%	1.73 s ± 0.05 s

bcw (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

Table A.5: (continued)

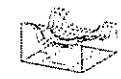
bcw (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0646 ± 0.0079	0.96% ± 0.05%	23.46% ± 2.81%	825.2713 ± 116.4942
1	0.4	1.3973 ± 0.3459	0.16% ± 0.11%	7.71% ± 2.86%	211.5841 ± 34.3790
1	0.1	1.3876 ± 0.3919	0.18% ± 0.15%	5.05% ± 1.82%	130.3803 ± 49.4667
1000	**0.1**	**0.0599 ± 0.0086**	**0.97% ± 0.03%**	**11.77% ± 0.58%**	**489.3496 ± 42.2209**
1000	0.7	0.0732 ± 0.0151	0.94% ± 0.06%	27.19% ± 3.75%	1253.2555 ± 238.3426
100	1	0.1546 ± 0.0249	0.83% ± 0.11%	39.98% ± 5.36%	2002.2801 ± 406.5583
1	0.7	1.3892 ± 0.3426	0.17% ± 0.12%	10.00% ± 3.99%	276.8681 ± 123.6461
1000	1	0.0808 ± 0.0158	0.91% ± 0.08%	29.69% ± 4.23%	1721.3789 ± 362.4685
100	0.7	0.1459 ± 0.0226	0.86% ± 0.08%	34.93% ± 4.44%	1395.4959 ± 207.6325
100	0.4	0.1355 ± 0.0149	0.86% ± 0.08%	27.08% ± 2.33%	900.5703 ± 129.9236
1	1	1.5918 ± 0.2485	0.10% ± 0.06%	13.24% ± 4.63%	340.5600 ± 95.3159
100	0.1	0.1214 ± 0.0182	0.92% ± 0.05%	11.65% ± 0.44%	447.1555 ± 56.9464

bcw (weak hypotheses' average result graphs for $T \times \rho$)

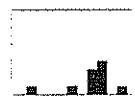

Alpha


Error Rate (%)


Support Vectors (%)


Norm

bcw (experiment histogram for parameter setup that yielded best results)


Accuracy (%): 96.29% ± 1.63%


Good Weak Hyp. (%): 69.17% ± 1.54%


Execution Time (s): 17.62 s ± 0.16 s

Table A.5: (continued)

cdges (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	83.33% ± 0.00%	95.50% ± 0.00%	1612.79 s ± 0.00 s
1	0.4	70.24% ± 0.00%	100.00% ± 0.00%	10.69 s ± 0.00 s
1	0.1	51.19% ± 0.00%	100.00% ± 0.00%	9.61 s ± 0.00 s
1000	0.1	77.38% ± 0.00%	82.20% ± 0.00%	412.14 s ± 0.00 s
1000	0.7	83.33% ± 0.00%	97.50% ± 0.00%	2768.94 s ± 0.00 s
100	1	83.33% ± 0.00%	99.00% ± 0.00%	388.96 s ± 0.00 s
1	0.7	48.81% ± 0.00%	100.00% ± 0.00%	12.58 s ± 0.00 s
1000	1	83.33% ± 0.00%	98.40% ± 0.00%	3831.25 s ± 0.00 s
100	**0.7**	**83.33% ± 0.00%**	**97.00% ± 0.00%**	**282.34 s ± 0.00 s**
100	0.4	82.14% ± 0.00%	97.00% ± 0.00%	172.01 s ± 0.00 s
1	1	58.33% ± 0.00%	100.00% ± 0.00%	12.84 s ± 0.00 s
100	0.1	79.76% ± 0.00%	84.00% ± 0.00%	47.17 s ± 0.00 s

cdges (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

cdges (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.3904 ± 0.0000	0.94% ± 0.00%	62.26% ± 0.00%	2282.3618 ± 0.0000
1	0.4	0.4963 ± 0.0000	0.63% ± 0.00%	59.52% ± 0.00%	1126.9927 ± 0.0000
1	0.1	0.0408 ± 0.0000	1.12% ± 0.00%	16.67% ± 0.00%	1131.0688 ± 0.0000
1000	0.1	0.1451 ± 0.0000	1.08% ± 0.00%	19.26% ± 0.00%	709.0286 ± 0.0000
1000	0.7	0.6571 ± 0.0000	0.80% ± 0.00%	87.72% ± 0.00%	3179.9887 ± 0.0000
100	1	0.9321 ± 0.0000	0.68% ± 0.00%	103.24% ± 0.00%	3562.6447 ± 0.0000
1	0.7	0.2939 ± 0.0000	0.83% ± 0.00%	129.76% ± 0.00%	2430.6873 ± 0.0000
1000	1	0.8689 ± 0.0000	0.71% ± 0.00%	102.50% ± 0.00%	3664.6736 ± 0.0000
100	**0.7**	**0.6800 ± 0.0000**	**0.78% ± 0.00%**	**87.57% ± 0.00%**	**2987.0381 ± 0.0000**
100	0.4	0.3867 ± 0.0000	0.92% ± 0.00%	63.90% ± 0.00%	2309.7233 ± 0.0000
1	1	0.5358 ± 0.0000	0.60% ± 0.00%	130.95% ± 0.00%	2932.7124 ± 0.0000
100	0.1	0.1758 ± 0.0000	1.05% ± 0.00%	18.20% ± 0.00%	574.7397 ± 0.0000

Table A.5: (continued)

cdges (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

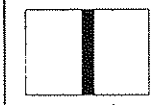
Support Vectors (%)

Norm

cdges (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 83.33% ± 0.00%

Good Weak Hyp. (%): 97.00% ± 0.00%



Execution Time (s): 282.34 s ± 0.00 s

chess^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	**0.4**	**93.20% ± 1.68%**	**98.62% ± 0.29%**	**355.78 s ± 3.48 s**
1	0.4	66.67% ± 7.23%	100.00% ± 0.00%	0.39 s ± 0.01 s
1	0.1	58.60% ± 7.84%	100.00% ± 0.00%	0.11 s ± 0.01 s
1000	0.1	91.17% ± 2.03%	93.78% ± 1.04%	92.32 s ± 0.92 s
1000	0.7	93.17% ± 1.65%	99.63% ± 0.25%	597.80 s ± 5.35 s
100	1	93.13% ± 1.53%	100.00% ± 0.00%	82.48 s ± 1.10 s
1	0.7	72.57% ± 7.55%	100.00% ± 0.00%	0.65 s ± 0.02 s
1000	1	93.03% ± 1.52%	99.91% ± 0.10%	823.25 s ± 8.62 s
100	0.7	92.80% ± 1.61%	99.80% ± 0.40%	59.98 s ± 0.76 s
100	0.4	92.53% ± 2.25%	98.80% ± 0.60%	35.62 s ± 0.36 s
1	1	82.77% ± 6.54%	100.00% ± 0.00%	0.90 s ± 0.03 s
100	0.1	87.67% ± 5.89%	93.60% ± 1.80%	9.16 s ± 0.11 s

Table A.5: (continued)

chess^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

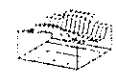
chess^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.6899 ± 0.0195	0.77% ± 0.01%	61.75% ± 0.86%	212.7448 ± 26.6444
1	0.4	0.6224 ± 0.1713	0.53% ± 0.14%	84.87% ± 2.25%	210.1454 ± 8.7988
1	0.1	0.2858 ± 0.1943	0.85% ± 0.20%	23.47% ± 2.69%	65.2067 ± 8.6006
1000	0.1	0.2328 ± 0.0091	1.00% ± 0.01%	20.22% ± 0.21%	79.6498 ± 10.0989
1000	0.7	1.0878 ± 0.0350	0.62% ± 0.01%	81.12% ± 0.72%	276.3994 ± 36.7599
100	1	1.3885 ± 0.0442	0.51% ± 0.01%	94.18% ± 2.47%	309.2881 ± 40.7065
1	0.7	0.9934 ± 0.1778	0.29% ± 0.11%	130.67% ± 2.80%	294.9164 ± 10.1433
1000	1	1.3979 ± 0.0345	0.52% ± 0.01%	89.62% ± 0.76%	310.7329 ± 42.6783
100	0.7	1.0663 ± 0.0391	0.61% ± 0.02%	84.66% ± 1.06%	273.7678 ± 34.2720
100	0.4	0.7072 ± 0.0365	0.75% ± 0.02%	63.48% ± 1.20%	207.0290 ± 23.4217
1	1	1.5029 ± 0.2723	0.12% ± 0.06%	161.70% ± 4.26%	341.4789 ± 12.4233
100	0.1	0.2617 ± 0.0147	0.96% ± 0.01%	21.09% ± 0.21%	67.5415 ± 5.1252

chess^0 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

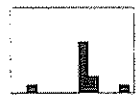
Error Rate (%)



Support Vectors (%)

Norm

chess^0 (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 93.20% ± 1.68%

Good Weak Hyp. (%): 98.62% ± 0.29%

Execution Time (s): 355.78 s ± 3.48 s

Table A.5: (continued)

chess[1] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	80.73% ± 2.39%	96.01% ± 1.30%	344.01 s ± 9.28 s
1	0.4	60.27% ± 7.46%	100.00% ± 0.00%	0.39 s ± 0.02 s
1	0.1	59.07% ± 7.87%	100.00% ± 0.00%	0.11 s ± 0.01 s
1000	0.1	78.60% ± 3.65%	91.69% ± 1.39%	92.72 s ± 0.90 s
1000	0.7	82.13% ± 1.54%	97.57% ± 0.79%	567.94 s ± 19.51 s
100	1	81.67% ± 1.57%	98.90% ± 1.37%	79.75 s ± 2.26 s
1	0.7	66.90% ± 7.15%	100.00% ± 0.00%	0.64 s ± 0.02 s
1000	1	82.00% ± 1.32%	98.77% ± 0.73%	787.46 s ± 27.98 s
100	**0.7**	**82.40% ± 1.54%**	**97.90% ± 0.94%**	**57.81 s ± 1.61 s**
100	0.4	79.47% ± 3.59%	97.40% ± 1.96%	34.95 s ± 0.88 s
1	1	71.63% ± 8.43%	100.00% ± 0.00%	0.89 s ± 0.01 s
100	0.1	76.73% ± 3.86%	93.90% ± 2.26%	9.29 s ± 0.13 s

chess[1] (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

chess[1] (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.4027 ± 0.0718	0.91% ± 0.04%	56.05% ± 3.27%	1690.8302 ± 1336.1698
1	0.4	0.5428 ± 0.1261	0.60% ± 0.11%	83.33% ± 3.84%	208.9996 ± 11.4563
1	0.1	0.2901 ± 0.1425	0.84% ± 0.15%	23.07% ± 2.01%	66.6651 ± 5.7268
1000	0.1	0.1207 ± 0.0179	1.07% ± 0.01%	20.38% ± 0.35%	401.0835 ± 292.6840
1000	0.7	0.6576 ± 0.1103	0.79% ± 0.05%	75.73% ± 4.59%	2253.6274 ± 1766.3481
100	1	0.8986 ± 0.1533	0.67% ± 0.06%	91.17% ± 3.65%	2349.9344 ± 1783.7175
1	0.7	0.8026 ± 0.2322	0.41% ± 0.17%	128.70% ± 3.93%	293.3474 ± 12.4519
1000	1	0.8836 ± 0.1445	0.69% ± 0.06%	87.26% ± 4.23%	2557.6547 ± 1985.0651
100	**0.7**	**0.6718 ± 0.1020**	**0.76% ± 0.04%**	**79.12% ± 3.99%**	**2017.1640 ± 1585.2221**
100	0.4	0.4258 ± 0.0619	0.88% ± 0.03%	60.31% ± 3.09%	1384.1524 ± 1136.3134
1	1	1.0680 ± 0.3051	0.28% ± 0.20%	160.50% ± 3.49%	349.9349 ± 7.3216
100	0.1	0.1765 ± 0.0082	1.01% ± 0.01%	21.90% ± 0.26%	89.8401 ± 32.0900

Table A.5: (continued)

chess[1] (weak hypotheses' average result graphs for $T \times \rho$)

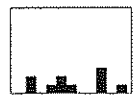
Alpha

Error Rate (%)

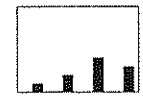
Support Vectors (%)

Norm

chess[1] (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 82.40% ± 1.54%



Good Weak Hyp. (%): 97.90% ± 0.94%

Execution Time (s): 57.81 s ± 1.61 s

chess[2] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	66.33% ± 2.47%	94.34% ± 0.72%	345.68 s ± 2.64 s
1	0.4	57.60% ± 4.24%	100.00% ± 0.00%	0.39 s ± 0.02 s
1	0.1	55.33% ± 3.22%	100.00% ± 0.00%	0.10 s ± 0.01 s
1000	0.1	66.17% ± 2.48%	90.70% ± 0.84%	93.38 s ± 0.32 s
1000	0.7	66.17% ± 2.53%	96.06% ± 0.61%	572.53 s ± 3.89 s
100	1	65.73% ± 2.65%	97.30% ± 1.49%	81.28 s ± 0.43 s
1	0.7	57.00% ± 6.32%	100.00% ± 0.00%	0.66 s ± 0.02 s
1000	1	66.60% ± 2.71%	96.80% ± 0.54%	793.49 s ± 4.06 s
100	0.7	65.53% ± 2.25%	96.30% ± 1.00%	59.26 s ± 0.28 s
100	**0.4**	**66.63% ± 2.40%**	**94.80% ± 1.72%**	**35.91 s ± 0.16 s**
1	1	58.37% ± 2.83%	100.00% ± 0.00%	0.91 s ± 0.02 s
100	0.1	61.27% ± 4.29%	93.80% ± 2.86%	9.26 s ± 0.11 s

Table A.5: (continued)

chess^2 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

chess^2 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.1798 ± 0.0235	1.03% ± 0.01%	56.05% ± 1.01%	397.7287 ± 140.2242
1	0.4	0.4615 ± 0.0552	0.66% ± 0.05%	84.47% ± 4.21%	216.1984 ± 18.7969
1	0.1	0.1952 ± 0.0886	0.94% ± 0.10%	22.40% ± 3.03%	64.2885 ± 10.6295
1000	0.1	0.0596 ± 0.0037	1.10% ± 0.00%	21.84% ± 0.11%	82.6262 ± 13.4838
1000	0.7	0.3039 ± 0.0372	0.96% ± 0.02%	74.29% ± 1.58%	616.8291 ± 225.1406
100	1	0.4479 ± 0.0565	0.86% ± 0.03%	93.55% ± 1.51%	751.9453 ± 262.3598
1	0.7	0.5126 ± 0.2577	0.64% ± 0.25%	131.57% ± 3.42%	310.1561 ± 12.3148
1000	1	0.4202 ± 0.0470	0.89% ± 0.02%	86.55% ± 2.01%	805.0883 ± 307.3652
100	0.7	0.3346 ± 0.0357	0.92% ± 0.02%	83.24% ± 1.32%	562.6228 ± 206.3076
100	**0.4**	**0.2340 ± 0.0147**	**0.97% ± 0.01%**	**65.66% ± 1.16%**	**284.4761 ± 68.7978**
1	1	0.7822 ± 0.1463	0.41% ± 0.10%	162.10% ± 3.98%	372.5934 ± 10.0216
100	0.1	0.1098 ± 0.0031	1.05% ± 0.00%	22.44% ± 0.21%	65.6937 ± 3.8760

chess^2 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

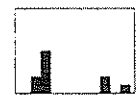
Error Rate (%)

Support Vectors (%)

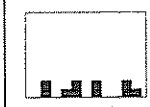
Norm

chess^2 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 66.63% ± 2.40%



Good Weak Hyp. (%): 94.80% ± 1.72%



Execution Time (s): 35.91 s ± 0.16 s

Table A.5: (continued)

gauss^0 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	100.00% ± 0.00%	100.00% ± 0.00%	19.95 s ± 0.00 s
1	**0.4**	**100.00% ± 0.00%**	**100.00% ± 0.00%**	**0.02 s ± 0.00 s**
1	0.1	99.33% ± 0.00%	100.00% ± 0.00%	0.01 s ± 0.00 s
1000	0.1	100.00% ± 0.00%	100.00% ± 0.00%	6.36 s ± 0.00 s
1000	0.7	100.00% ± 0.00%	100.00% ± 0.00%	11.40 s ± 0.00 s
100	1	100.00% ± 0.00%	100.00% ± 0.00%	1.86 s ± 0.00 s
1	0.7	100.00% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	1	100.00% ± 0.00%	100.00% ± 0.00%	15.13 s ± 0.00 s
100	0.7	100.00% ± 0.00%	100.00% ± 0.00%	1.63 s ± 0.00 s
100	0.4	100.00% ± 0.00%	100.00% ± 0.00%	1.13 s ± 0.00 s
1	1	100.00% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
100	0.1	100.00% ± 0.00%	100.00% ± 0.00%	0.89 s ± 0.00 s

gauss^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

gauss^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	9.9706 ± 0.0000	0.00% ± 0.00%	1.89% ± 0.00%	990.7921 ± 0.0000
1	**0.4**	**10.0000 ± 0.0000**	**0.00% ± 0.00%**	**2.67% ± 0.00%**	**403.0310 ± 0.0000**
1	0.1	2.7241 ± 0.0000	0.01% ± 0.00%	1.33% ± 0.00%	279.4646 ± 0.0000
1000	0.1	9.9865 ± 0.0000	0.00% ± 0.00%	1.20% ± 0.00%	1554.3041 ± 0.0000
1000	0.7	9.9766 ± 0.0000	0.00% ± 0.00%	1.02% ± 0.00%	1543.9565 ± 0.0000
100	1	9.6637 ± 0.0000	0.01% ± 0.00%	2.41% ± 0.00%	1214.9337 ± 0.0000
1	0.7	10.0000 ± 0.0000	0.00% ± 0.00%	2.67% ± 0.00%	403.0310 ± 0.0000
1000	1	9.9576 ± 0.0000	0.00% ± 0.00%	1.29% ± 0.00%	1544.2108 ± 0.0000
100	0.7	9.6618 ± 0.0000	0.00% ± 0.00%	3.04% ± 0.00%	1166.0681 ± 0.0000
100	0.4	9.8547 ± 0.0000	0.00% ± 0.00%	1.85% ± 0.00%	1313.2408 ± 0.0000
1	1	10.0000 ± 0.0000	0.00% ± 0.00%	3.33% ± 0.00%	408.8045 ± 0.0000
100	0.1	9.8579 ± 0.0000	0.00% ± 0.00%	1.61% ± 0.00%	1389.9964 ± 0.0000

Table A.5: (continued)

gauss^0 (weak hypotheses' average result graphs for $T \times \rho$)

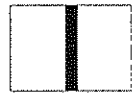
Alpha

Error Rate (%)

Support Vectors (%)

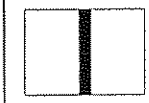
Norm

gauss^0 (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 100.00% ± 0.00%

Good Weak Hyp. (%): 100.00% ± 0.00%



Execution Time (s): 0.02 s ± 0.00 s

gauss^1 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	96.33% ± 0.00%	61.30% ± 0.00%	110.72 s ± 0.00 s
1	0.4	96.67% ± 0.00%	100.00% ± 0.00%	0.01 s ± 0.00 s
1	0.1	89.33% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	99.33% ± 0.00%	62.40% ± 0.00%	28.23 s ± 0.00 s
1000	0.7	75.33% ± 0.00%	58.10% ± 0.00%	194.14 s ± 0.00 s
100	1	99.33% ± 0.00%	59.00% ± 0.00%	25.67 s ± 0.00 s
1	0.7	87.33% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	1	93.33% ± 0.00%	61.40% ± 0.00%	278.04 s ± 0.00 s
100	0.7	98.67% ± 0.00%	57.00% ± 0.00%	18.83 s ± 0.00 s
100	**0.4**	**99.33% ± 0.00%**	**66.00% ± 0.00%**	**10.69 s ± 0.00 s**
1	1	90.67% ± 0.00%	100.00% ± 0.00%	0.03 s ± 0.00 s
100	0.1	97.67% ± 0.00%	72.00% ± 0.00%	2.79 s ± 0.00 s

Table A.5: (continued)

gauss[1] (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

gauss[1] (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0220 ± 0.0000	1.09% ± 0.00%	21.02% ± 0.00%	29106.5631 ± 0.0000
1	0.4	1.9837 ± 0.0000	0.04% ± 0.00%	2.67% ± 0.00%	769.4516 ± 0.0000
1	0.1	1.2212 ± 0.0000	0.19% ± 0.00%	2.00% ± 0.00%	1161.4712 ± 0.0000
1000	0.1	0.0252 ± 0.0000	1.07% ± 0.00%	9.28% ± 0.00%	33149.8684 ± 0.0000
1000	0.7	0.0186 ± 0.0000	1.08% ± 0.00%	27.54% ± 0.00%	-11716.5054 ± 0.0000
100	1	0.0804 ± 0.0000	1.13% ± 0.00%	50.96% ± 0.00%	-2236.6269 ± 0.0000
1	0.7	0.9505 ± 0.0000	0.30% ± 0.00%	3.33% ± 0.00%	1777.7134 ± 0.0000
1000	1	0.0262 ± 0.0000	1.07% ± 0.00%	34.09% ± 0.00%	-9995.6651 ± 0.0000
100	0.7	0.0728 ± 0.0000	1.05% ± 0.00%	36.13% ± 0.00%	20068.2938 ± 0.0000
100	**0.4**	**0.0960 ± 0.0000**	**0.99% ± 0.00%**	**25.18% ± 0.00%**	**10878.2323 ± 0.0000**
1	1	1.0168 ± 0.0000	0.27% ± 0.00%	4.00% ± 0.00%	1952.4985 ± 0.0000
100	0.1	0.0969 ± 0.0000	0.99% ± 0.00%	9.63% ± 0.00%	14247.8388 ± 0.0000

gauss[1] (weak hypotheses' average result graphs for $T \times \rho$)

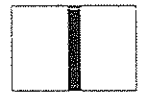
Alpha

Error Rate (%)

Support Vectors (%)

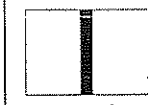
Norm

gauss[1] (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 99.33% ± 0.00%

Good Weak Hyp. (%): 66.00% ± 0.00%



Execution Time (s): 10.69 s ± 0.00 s

Table A.5: (continued)

gauss^2 (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	91.67% ± 0.00%	58.80% ± 0.00%	111.25 s ± 0.00 s
1	0.4	65.33% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1	0.1	88.67% ± 0.00%	100.00% ± 0.00%	0.02 s ± 0.00 s
1000	0.1	92.00% ± 0.00%	55.00% ± 0.00%	26.50 s ± 0.00 s
1000	0.7	91.67% ± 0.00%	56.00% ± 0.00%	202.12 s ± 0.00 s
100	1	93.00% ± 0.00%	64.00% ± 0.00%	28.90 s ± 0.00 s
1	0.7	80.67% ± 0.00%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	1	91.33% ± 0.00%	56.20% ± 0.00%	294.19 s ± 0.00 s
100	0.7	92.00% ± 0.00%	66.00% ± 0.00%	20.21 s ± 0.00 s
100	**0.4**	**93.00% ± 0.00%**	**68.00% ± 0.00%**	**11.22 s ± 0.00 s**
1	1	92.33% ± 0.00%	100.00% ± 0.00%	0.05 s ± 0.00 s
100	0.1	86.67% ± 0.00%	62.00% ± 0.00%	2.60 s ± 0.00 s

gauss^2 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

gauss^2 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0101 ± 0.0000	1.11% ± 0.00%	39.72% ± 0.00%	13408.8051 ± 0.0000
1	0.4	0.4135 ± 0.0000	0.71% ± 0.00%	4.67% ± 0.00%	1963.6981 ± 0.0000
1	0.1	1.0454 ± 0.0000	0.26% ± 0.00%	1.33% ± 0.00%	4930.0229 ± 0.0000
1000	0.1	0.0079 ± 0.0000	1.11% ± 0.00%	12.25% ± 0.00%	18235.3117 ± 0.0000
1000	0.7	0.0084 ± 0.0000	1.10% ± 0.00%	58.90% ± 0.00%	29430.1928 ± 0.0000
100	1	0.0457 ± 0.0000	1.02% ± 0.00%	73.26% ± 0.00%	61095.3076 ± 0.0000
1	0.7	0.6887 ± 0.0000	0.47% ± 0.00%	13.33% ± 0.00%	3188.9209 ± 0.0000
1000	1	0.0082 ± 0.0000	1.11% ± 0.00%	72.13% ± 0.00%	26480.5652 ± 0.0000
100	0.7	0.0462 ± 0.0000	1.09% ± 0.00%	59.47% ± 0.00%	-15635.0998 ± 0.0000
100	**0.4**	**0.0459 ± 0.0000**	**1.10% ± 0.00%**	**37.71% ± 0.00%**	**9439.8938 ± 0.0000**
1	1	1.2115 ± 0.0000	0.19% ± 0.00%	16.33% ± 0.00%	3666.0618 ± 0.0000
100	0.1	0.0526 ± 0.0000	1.05% ± 0.00%	11.78% ± 0.00%	8300.7815 ± 0.0000

Table A.5: (continued)

gauss^2 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

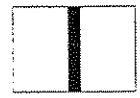
Error Rate (%)

Support Vectors (%)

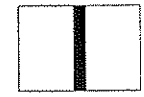
Norm

gauss^2 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 93.00% ± 0.00%



Good Weak Hyp. (%): 68.00% ± 0.00%



Execution Time (s): 11.22 s ± 0.00 s

hepatitis (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	**0.4**	**75.96% ± 6.32%**	**61.96% ± 2.25%**	**3.31 s ± 0.03 s**
1	0.4	58.51% ± 11.31%	100.00% ± 0.00%	0.01 s ± 0.01 s
1	0.1	20.64% ± 31.56%	30.00% ± 45.83%	0.01 s ± 0.00 s
1000	0.1	42.98% ± 25.15%	58.39% ± 6.82%	0.80 s ± 0.19 s
1000	0.7	75.32% ± 6.03%	60.75% ± 2.03%	5.68 s ± 0.05 s
100	1	75.11% ± 9.23%	65.60% ± 2.91%	0.83 s ± 0.02 s
1	0.7	53.40% ± 20.26%	90.00% ± 30.00%	0.01 s ± 0.00 s
1000	1	75.74% ± 7.13%	54.88% ± 5.48%	8.25 s ± 0.11 s
100	0.7	74.47% ± 5.94%	68.50% ± 4.18%	0.58 s ± 0.01 s
100	0.4	75.32% ± 7.50%	64.30% ± 6.56%	0.33 s ± 0.01 s
1	1	67.23% ± 11.35%	100.00% ± 0.00%	0.02 s ± 0.01 s
100	0.1	59.79% ± 23.00%	59.90% ± 8.93%	0.08 s ± 0.02 s

Table A.5: (continued)

hepatitis (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

hepatitis (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0269 ± 0.0067	1.16% ± 0.01%	48.90% ± 0.93%	5049477.6735 ± 786614.1044
1	0.4	0.2356 ± 0.1239	0.89% ± 0.13%	36.60% ± 10.37%	3932915.0312 ± 5070754.9105
1	0.1	-0.0919 ± 0.3330	1.25% ± 0.36%	10.21% ± 4.74%	1224065.2641 ± 1118489.6903
1000	0.1	0.6171 ± 1.0785	1.15% ± 0.11%	12.81% ± 4.03%	2094737.8838 ± 1094047.2819
1000	0.7	0.0237 ± 0.0061	1.14% ± 0.02%	72.12% ± 2.06%	4439633.5884 ± 1119465.4140
100	1	0.0501 ± 0.0076	1.10% ± 0.04%	96.88% ± 3.48%	4685277.1892 ± 1197583.2065
1	0.7	0.2689 ± 0.1946	0.86% ± 0.20%	61.70% ± 8.07%	4740229.7250 ± 2919735.2549
1000	1	0.0112 ± 0.0161	1.13% ± 0.01%	88.67% ± 3.12%	114421.3497 ± 4619108.1358
100	0.7	0.0526 ± 0.0094	1.12% ± 0.04%	75.20% ± 1.96%	4356726.9100 ± 1517071.8803
100	0.4	0.0433 ± 0.0147	1.16% ± 0.04%	49.15% ± 1.10%	3912832.5133 ± 714593.3025
1	1	0.4667 ± 0.1730	0.66% ± 0.17%	78.09% ± 6.39%	2659501.8930 ± 8242273.2483
100	0.1	0.7943 ± 1.2106	1.13% ± 0.12%	11.90% ± 4.36%	1445966.5308 ± 823693.8130

hepatitis (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

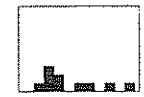
Error Rate (%)

Support Vectors (%)

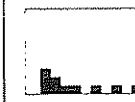
Norm

hepatitis (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 75.96% ± 6.32%



Good Weak Hyp. (%): 61.96% ± 2.25%



Execution Time (s): 3.31 s ± 0.03 s

Table A.5: (continued)

ionosphere (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	94.53% ± 2.59%	99.27% ± 0.25%	21.61 s ± 0.21 s
1	0.4	84.43% ± 8.79%	100.00% ± 0.00%	0.06 s ± 0.00 s
1	0.1	70.75% ± 8.20%	100.00% ± 0.00%	0.04 s ± 0.01 s
1000	0.1	94.53% ± 2.42%	93.52% ± 1.59%	6.06 s ± 0.09 s
1000	0.7	94.72% ± 2.54%	99.81% ± 0.10%	34.72 s ± 0.34 s
100	1	94.72% ± 2.87%	99.70% ± 0.64%	4.82 s ± 0.06 s
1	0.7	85.28% ± 10.43%	100.00% ± 0.00%	0.07 s ± 0.00 s
1000	1	94.62% ± 2.53%	99.89% ± 0.13%	46.90 s ± 0.81 s
100	0.7	94.25% ± 2.91%	99.90% ± 0.30%	3.58 s ± 0.06 s
100	**0.4**	**94.72% ± 2.74%**	**98.60% ± 1.69%**	**2.21 s ± 0.02 s**
1	1	88.87% ± 8.78%	100.00% ± 0.00%	0.08 s ± 0.00 s
100	0.1	92.64% ± 7.80%	93.20% ± 4.89%	0.61 s ± 0.12 s

ionosphere (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

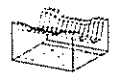
Execution Time (s)

ionosphere (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	1.0671 ± 0.1630	0.54% ± 0.05%	44.54% ± 1.15%	46.2627 ± 5.3174
1	0.4	1.1216 ± 0.3114	0.25% ± 0.14%	59.62% ± 4.97%	45.5418 ± 4.5944
1	0.1	0.4958 ± 0.1591	0.64% ± 0.14%	18.96% ± 4.14%	17.6444 ± 3.3787
1000	0.1	0.3264 ± 0.0796	0.89% ± 0.05%	18.45% ± 0.28%	21.3414 ± 1.5254
1000	0.7	1.5799 ± 0.1750	0.38% ± 0.04%	52.22% ± 2.08%	54.4694 ± 7.2754
100	1	1.8900 ± 0.1794	0.31% ± 0.04%	59.85% ± 2.32%	59.7365 ± 8.5230
1	0.7	1.1784 ± 0.3786	0.25% ± 0.18%	83.77% ± 4.11%	56.2518 ± 5.9302
1000	1	1.9310 ± 0.1766	0.31% ± 0.03%	55.00% ± 2.66%	58.6703 ± 8.4915
100	0.7	1.5795 ± 0.1797	0.38% ± 0.04%	56.15% ± 2.23%	55.7310 ± 7.3713
100	**0.4**	**1.0533 ± 0.1722**	**0.53% ± 0.05%**	**46.15% ± 1.72%**	**46.1658 ± 5.0677**
1	1	1.5223 ± 0.4575	0.16% ± 0.20%	103.96% ± 5.04%	65.7660 ± 7.7057
100	0.1	0.5059 ± 0.4605	0.89% ± 0.05%	17.66% ± 4.09%	19.3986 ± 5.7353

Table A.5: (continued)

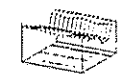
ionosphere (weak hypotheses' average result graphs for $T \times \rho$)



Alpha

Error Rate (%)

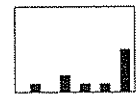
Support Vectors (%)



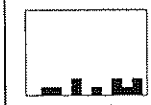
Norm

ionosphere (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 94.72% ± 2.74%



Good Weak Hyp. (%): 98.60% ± 1.69%



Execution Time (s): 2.21 s ± 0.02 s

musk (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	93.01% ± 0.00%	93.40% ± 0.00%	70.99 s ± 0.00 s
1	0.4	81.82% ± 0.00%	100.00% ± 0.00%	0.24 s ± 0.00 s
1	0.1	66.43% ± 0.00%	100.00% ± 0.00%	0.21 s ± 0.00 s
1000	0.1	90.21% ± 0.00%	85.40% ± 0.00%	18.58 s ± 0.00 s
1000	0.7	93.01% ± 0.00%	98.00% ± 0.00%	119.15 s ± 0.00 s
100	1	89.51% ± 0.00%	99.00% ± 0.00%	16.63 s ± 0.00 s
1	0.7	64.34% ± 0.00%	100.00% ± 0.00%	0.27 s ± 0.00 s
1000	**1**	**93.71% ± 0.00%**	**98.60% ± 0.00%**	**165.83 s ± 0.00 s**
100	0.7	91.61% ± 0.00%	97.00% ± 0.00%	11.86 s ± 0.00 s
100	0.4	91.61% ± 0.00%	95.00% ± 0.00%	7.10 s ± 0.00 s
1	1	72.73% ± 0.00%	100.00% ± 0.00%	0.30 s ± 0.00 s
100	0.1	83.92% ± 0.00%	84.00% ± 0.00%	1.96 s ± 0.00 s

Table A.5: (continued)

musk (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

musk (average results for weak hypotheses)

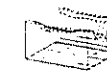
T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.2789 ± 0.0000	0.90% ± 0.00%	48.79% ± 0.00%	137.1992 ± 0.0000
1	0.4	0.7475 ± 0.0000	0.43% ± 0.00%	56.64% ± 0.00%	124.7623 ± 0.0000
1	0.1	0.2355 ± 0.0000	0.90% ± 0.00%	24.48% ± 0.00%	49.8657 ± 0.0000
1000	0.1	0.0758 ± 0.0000	1.08% ± 0.00%	19.10% ± 0.00%	49.3051 ± 0.0000
1000	0.7	0.5213 ± 0.0000	0.75% ± 0.00%	59.71% ± 0.00%	196.4148 ± 0.0000
100	1	0.7538 ± 0.0000	0.61% ± 0.00%	71.66% ± 0.00%	240.0581 ± 0.0000
1	0.7	0.4966 ± 0.0000	0.63% ± 0.00%	86.01% ± 0.00%	174.6977 ± 0.0000
1000	**1**	**0.7185 ± 0.0000**	**0.64% ± 0.00%**	**65.10% ± 0.00%**	**240.7219 ± 0.0000**
100	0.7	0.5107 ± 0.0000	0.73% ± 0.00%	65.64% ± 0.00%	190.0569 ± 0.0000
100	0.4	0.2726 ± 0.0000	0.89% ± 0.00%	53.38% ± 0.00%	133.5983 ± 0.0000
1	1	0.4890 ± 0.0000	0.64% ± 0.00%	108.39% ± 0.00%	200.8528 ± 0.0000
100	0.1	0.1215 ± 0.0000	1.01% ± 0.00%	19.04% ± 0.00%	48.2714 ± 0.0000

musk (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

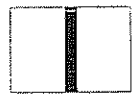
Error Rate (%)

Support Vectors (%)

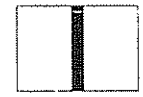


Norm

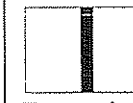
musk (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 93.71% ± 0.00%



Good Weak Hyp. (%): 98.60% ± 0.00%



Execution Time (s): 165.83 s ± 0.00 s

Table A.5: (continued)

pgs (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	82.50% ± 5.96%	97.38% ± 0.53%	3.61 s ± 0.03 s
1	0.4	68.44% ± 8.78%	100.00% ± 0.00%	0.03 s ± 0.00 s
1	0.1	58.75% ± 8.12%	100.00% ± 0.00%	0.03 s ± 0.00 s
1000	0.1	73.75% ± 11.11%	21.31% ± 9.69%	0.29 s ± 0.11 s
1000	0.7	82.19% ± 5.42%	98.97% ± 0.48%	5.87 s ± 0.07 s
100	1	81.25% ± 5.76%	99.20% ± 0.87%	0.83 s ± 0.01 s
1	0.7	67.19% ± 13.78%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	1	82.19% ± 5.24%	99.62% ± 0.21%	7.90 s ± 0.10 s
100	0.7	81.25% ± 4.84%	98.90% ± 1.14%	0.62 s ± 0.01 s
100	0.4	81.88% ± 5.90%	96.80% ± 0.75%	0.39 s ± 0.00 s
1	1	70.62% ± 9.50%	100.00% ± 0.00%	0.04 s ± 0.00 s
100	0.1	74.69% ± 12.14%	55.60% ± 27.22%	0.09 s ± 0.03 s

pgs (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

pgs (average results for weak hypotheses)

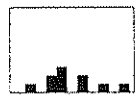
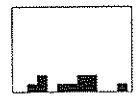
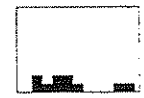
T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.5830 ± 0.0280	0.76% ± 0.02%	62.97% ± 1.13%	157.0038 ± 3.5178
1	0.4	0.7154 ± 0.2287	0.47% ± 0.16%	79.06% ± 9.06%	192.2077 ± 26.8090
1	0.1	0.3441 ± 0.1223	0.78% ± 0.13%	20.94% ± 7.53%	56.7272 ± 19.2485
1000	0.1	6.4323 ± 1.2721	0.36% ± 0.13%	4.96% ± 2.44%	13.2543 ± 6.7645
1000	0.7	0.9571 ± 0.0376	0.59% ± 0.02%	79.51% ± 2.62%	200.4554 ± 5.1125
100	1	1.2399 ± 0.0800	0.48% ± 0.03%	91.80% ± 2.80%	229.7050 ± 6.5868
1	0.7	0.8077 ± 0.2815	0.42% ± 0.18%	126.88% ± 10.57%	273.1393 ± 32.1228
1000	1	1.2817 ± 0.0533	0.48% ± 0.02%	87.43% ± 3.48%	227.6176 ± 6.8444
100	0.7	0.9604 ± 0.0545	0.58% ± 0.02%	82.97% ± 2.53%	202.8808 ± 4.6124
100	0.4	0.5856 ± 0.0368	0.75% ± 0.03%	65.43% ± 1.07%	159.7154 ± 3.6292
1	1	1.0588 ± 0.3315	0.29% ± 0.17%	155.62% ± 8.00%	318.4731 ± 26.9838
100	0.1	3.3580 ± 3.2776	0.65% ± 0.32%	13.31% ± 7.59%	37.7040 ± 21.7812

Table A.5: (continued)

pgs (weak hypotheses' average result graphs for $T \times \rho$)

 Alpha	 Error Rate (%)	 Support Vectors (%)	 Norm

pgs (experiment histogram for parameter setup that yielded best results)

 Accuracy (%): 82.50% ± 5.96%	 Good Weak Hyp. (%): 97.38% ± 0.53%	 Execution Time (s): 3.61 s ± 0.03 s

pid (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	75.54% ± 2.96%	56.96% ± 1.66%	64.94 s ± 0.84 s
1	0.4	68.31% ± 7.13%	100.00% ± 0.00%	0.06 s ± 0.01 s
1	0.1	49.91% ± 32.85%	70.00% ± 45.83%	0.03 s ± 0.01 s
1000	0.1	75.11% ± 3.91%	57.87% ± 1.71%	16.71 s ± 0.10 s
1000	**0.7**	**76.19% ± 2.28%**	**58.64% ± 1.33%**	**117.75 s ± 0.77 s**
100	1	76.02% ± 3.29%	63.70% ± 5.29%	17.05 s ± 0.13 s
1	0.7	54.37% ± 27.65%	80.00% ± 40.00%	0.09 s ± 0.02 s
1000	1	75.76% ± 2.50%	58.04% ± 1.66%	173.97 s ± 1.06 s
100	0.7	75.50% ± 3.40%	62.00% ± 5.48%	11.61 s ± 0.22 s
100	0.4	75.15% ± 2.79%	65.30% ± 6.25%	6.48 s ± 0.10 s
1	1	65.84% ± 6.47%	100.00% ± 0.00%	0.13 s ± 0.01 s
100	0.1	75.89% ± 2.63%	65.70% ± 6.89%	1.74 s ± 0.03 s

Table A.5: (continued)

pid (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

pid (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0063 ± 0.0014	1.15% ± 0.01%	47.30% ± 0.33%	15319.8160 ± 611.4799
1	0.4	0.4013 ± 0.1326	0.73% ± 0.14%	30.78% ± 5.31%	16836.7296 ± 11858.6247
1	0.1	0.2544 ± 0.2656	0.89% ± 0.29%	10.09% ± 2.40%	6158.9744 ± 2854.9066
1000	0.1	0.0066 ± 0.0012	1.16% ± 0.01%	13.66% ± 0.06%	10498.3356 ± 488.9582
1000	**0.7**	**0.0071 ± 0.0012**	**1.15% ± 0.01%**	**74.81% ± 0.71%**	**15668.5442 ± 901.9672**
100	1	0.0243 ± 0.0044	1.11% ± 0.03%	96.93% ± 1.20%	16413.2624 ± 1544.0768
1	0.7	0.2366 ± 0.2309	0.90% ± 0.26%	53.85% ± 8.14%	22051.8993 ± 7432.4476
1000	1	0.0073 ± 0.0013	1.15% ± 0.01%	96.86% ± 1.08%	15837.1379 ± 980.2540
100	0.7	0.0224 ± 0.0079	1.13% ± 0.03%	74.73% ± 1.00%	16160.8262 ± 1610.2331
100	0.4	0.0247 ± 0.0050	1.14% ± 0.02%	46.93% ± 0.55%	15355.8154 ± 1233.8818
1	1	0.3544 ± 0.1755	0.78% ± 0.18%	72.99% ± 8.25%	19226.9023 ± 7579.6699
100	0.1	0.0238 ± 0.0068	1.15% ± 0.03%	13.65% ± 0.16%	9899.9923 ± 1033.5808

pid (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

Support Vectors (%)

Norm

pid (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 76.19% ± 2.28%

Good Weak Hyp. (%): 58.64% ± 1.33%

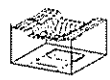
Execution Time (s): 117.75 s ± 0.77 s

Table A.5: (continued)

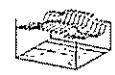
ringnorm (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	97.83% ± 0.48%	90.03% ± 0.68%	168.25 s ± 2.68 s
1	0.4	74.73% ± 6.95%	100.00% ± 0.00%	0.18 s ± 0.01 s
1	0.1	59.87% ± 21.62%	90.00% ± 30.00%	0.11 s ± 0.01 s
1000	0.1	96.57% ± 1.17%	83.11% ± 0.81%	44.15 s ± 0.39 s
1000	**0.7**	**97.90% ± 0.56%**	**93.58% ± 1.11%**	**283.86 s ± 3.70 s**
100	1	97.77% ± 0.54%	96.40% ± 1.20%	38.95 s ± 0.39 s
1	0.7	82.63% ± 8.13%	100.00% ± 0.00%	0.25 s ± 0.02 s
1000	1	97.77% ± 0.45%	95.49% ± 0.56%	395.03 s ± 3.86 s
100	0.7	97.50% ± 0.56%	93.40% ± 2.94%	27.81 s ± 0.22 s
100	0.4	97.67% ± 0.70%	92.90% ± 2.43%	16.03 s ± 0.15 s
1	1	85.70% ± 3.89%	100.00% ± 0.00%	0.31 s ± 0.02 s
100	0.1	94.53% ± 1.25%	86.30% ± 5.31%	4.07 s ± 0.12 s

ringnorm (strong hypotheses' average result graphs for $T \times \rho$)



Accuracy (%)



Good Weak Hyp. (%)

Execution Time (s)

ringnorm (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.1650 ± 0.0067	0.88% ± 0.01%	27.24% ± 0.89%	1636.5402 ± 23.8021
1	0.4	0.5746 ± 0.1722	0.57% ± 0.14%	42.30% ± 4.57%	1809.9296 ± 164.1390
1	0.1	0.3611 ± 0.2115	0.78% ± 0.22%	14.47% ± 2.79%	790.3080 ± 183.2613
1000	0.1	0.0808 ± 0.0021	1.03% ± 0.00%	16.15% ± 0.09%	801.6084 ± 13.7746
1000	**0.7**	**0.2636 ± 0.0176**	**0.74% ± 0.02%**	**26.68% ± 0.87%**	**2950.4370 ± 64.8932**
100	1	0.4293 ± 0.0401	0.57% ± 0.02%	38.13% ± 2.04%	4646.6654 ± 157.6675
1	0.7	0.8352 ± 0.2642	0.40% ± 0.19%	64.53% ± 6.31%	2724.1871 ± 252.0926
1000	1	0.3640 ± 0.0248	0.63% ± 0.02%	26.15% ± 1.14%	4442.0614 ± 61.0179
100	0.7	0.3370 ± 0.0320	0.67% ± 0.02%	38.95% ± 1.97%	3140.2346 ± 93.1060
100	0.4	0.2536 ± 0.0182	0.81% ± 0.02%	37.25% ± 0.91%	1882.7090 ± 26.2398
1	1	1.0079 ± 0.1640	0.28% ± 0.08%	78.03% ± 5.24%	3483.0452 ± 314.1620
100	0.1	0.1295 ± 0.0163	0.96% ± 0.01%	15.86% ± 0.34%	793.0031 ± 24.3473

Table A.5: (continued)

ringnorm (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

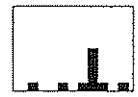
Error Rate (%)

Support Vectors (%)

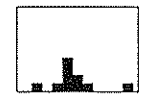
Norm

ringnorm (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 97.90% ± 0.56%



Good Weak Hyp. (%): 93.58% ± 1.11%



Execution Time (s): 283.86 s ± 3.70 s

spect[b] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	77.27% ± 1.10%	23.92% ± 4.47%	0.80 s ± 0.11 s
1	0.4	74.71% ± 8.72%	100.00% ± 0.00%	0.02 s ± 0.00 s
1	0.1	67.54% ± 23.28%	90.00% ± 30.00%	0.02 s ± 0.01 s
1000	0.1	76.90% ± 2.39%	9.03% ± 6.06%	0.23 s ± 0.07 s
1000	0.7	78.07% ± 1.29%	35.35% ± 7.74%	1.48 s ± 0.25 s
100	1	77.22% ± 1.38%	78.00% ± 3.44%	0.48 s ± 0.02 s
1	0.7	71.18% ± 25.66%	90.00% ± 30.00%	0.02 s ± 0.00 s
1000	1	77.11% ± 1.17%	48.80% ± 10.22%	2.47 s ± 0.42 s
100	0.7	77.54% ± 1.82%	79.40% ± 3.04%	0.39 s ± 0.02 s
100	0.4	77.59% ± 1.73%	80.70% ± 4.03%	0.29 s ± 0.00 s
1	1	77.70% ± 7.76%	100.00% ± 0.00%	0.02 s ± 0.00 s
100	**0.1**	**78.50% ± 3.31%**	**37.10% ± 30.70%**	**0.07 s ± 0.04 s**

Table A.5: (continued)

spect^b (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

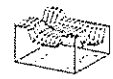
Execution Time (s)

spect^b (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	6.6327 ± 0.7211	0.06% ± 0.02%	1.60% ± 0.23%	248.9708 ± 31.1880
1	0.4	0.7284 ± 0.1971	0.08% ± 0.03%	6.79% ± 1.55%	491.3032 ± 160.9955
1	0.1	1.6313 ± 2.7978	0.08% ± 0.04%	2.67% ± 1.37%	236.7345 ± 134.2792
1000	0.1	8.6890 ± 0.7570	0.02% ± 0.01%	0.39% ± 0.24%	52.9425 ± 39.0435
1000	0.7	4.7381 ± 1.1878	0.10% ± 0.02%	2.45% ± 0.41%	397.3476 ± 64.8678
100	1	0.1371 ± 0.0060	0.13% ± 0.01%	8.31% ± 0.42%	1320.3503 ± 139.0509
1	0.7	0.6987 ± 0.3602	0.09% ± 0.06%	9.36% ± 1.71%	691.4671 ± 239.8080
1000	1	2.5923 ± 1.6515	0.14% ± 0.03%	3.55% ± 0.58%	590.8645 ± 107.3047
100	0.7	0.1463 ± 0.0075	0.14% ± 0.01%	7.71% ± 0.36%	1163.3804 ± 86.8304
100	0.4	0.1580 ± 0.0084	0.14% ± 0.01%	6.91% ± 0.30%	933.5958 ± 42.6099
1	1	0.9369 ± 0.1388	0.06% ± 0.01%	10.91% ± 1.99%	790.1033 ± 166.0420
100	**0.1**	**5.2834 ± 3.7111**	**0.08% ± 0.06%**	**1.48% ± 1.32%**	**188.5656 ± 183.4684**

spect^b (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

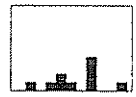


Error Rate (%)

Support Vectors (%)

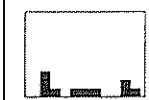
Norm

spect^b (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 78.50% ± 3.31%

Good Weak Hyp. (%): 37.10% ± 30.70%



Execution Time (s): 0.07 s ± 0.04 s

Table A.5: (continued)

spect^r (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	82.45% ± 0.59%	97.37% ± 0.50%	4.95 s ± 0.04 s
1	0.4	71.86% ± 10.26%	100.00% ± 0.00%	0.04 s ± 0.01 s
1	0.1	55.28% ± 14.19%	100.00% ± 0.00%	0.04 s ± 0.01 s
1000	0.1	81.41% ± 2.14%	14.52% ± 17.73%	0.39 s ± 0.29 s
1000	0.7	82.16% ± 0.47%	98.72% ± 0.40%	7.27 s ± 0.04 s
100	1	81.93% ± 0.48%	98.80% ± 1.17%	0.96 s ± 0.01 s
1	0.7	74.72% ± 8.03%	100.00% ± 0.00%	0.04 s ± 0.00 s
1000	1	81.93% ± 0.30%	99.32% ± 0.21%	9.07 s ± 0.02 s
100	0.7	82.68% ± 0.91%	98.90% ± 1.04%	0.78 s ± 0.01 s
100	0.4	82.45% ± 1.23%	98.10% ± 1.45%	0.54 s ± 0.01 s
1	1	80.30% ± 3.89%	100.00% ± 0.00%	0.05 s ± 0.00 s
100	**0.1**	**82.86% ± 1.65%**	**73.50% ± 20.48%**	**0.17 s ± 0.03 s**

spect^r (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

spect^r (average results for weak hypotheses)

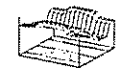
T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.5585 ± 0.0053	0.10% ± 0.00%	7.98% ± 0.04%	25.4390 ± 0.2526
1	0.4	0.6570 ± 0.1527	0.06% ± 0.02%	10.04% ± 1.39%	27.4857 ± 3.4553
1	0.1	0.3565 ± 0.1994	0.10% ± 0.03%	2.68% ± 1.25%	8.9356 ± 3.8840
1000	0.1	8.2835 ± 2.1615	0.02% ± 0.03%	0.46% ± 0.60%	1.6392 ± 2.1909
1000	0.7	0.9134 ± 0.0124	0.08% ± 0.00%	10.31% ± 0.07%	32.6020 ± 0.2625
100	1	1.2124 ± 0.0298	0.07% ± 0.00%	11.73% ± 0.20%	37.1265 ± 0.5761
1	0.7	0.8591 ± 0.1278	0.05% ± 0.01%	14.68% ± 1.01%	35.9601 ± 6.3936
1000	1	1.2181 ± 0.0137	0.07% ± 0.00%	11.34% ± 0.04%	37.0048 ± 0.2002
100	0.7	0.9026 ± 0.0395	0.08% ± 0.00%	10.60% ± 0.10%	32.7737 ± 0.6784
100	0.4	0.5738 ± 0.0258	0.10% ± 0.00%	8.20% ± 0.12%	25.4231 ± 0.6332
1	1	1.1201 ± 0.1350	0.03% ± 0.01%	17.81% ± 0.90%	42.9081 ± 4.9925
100	**0.1**	**1.4782 ± 2.4815**	**0.11% ± 0.03%**	**2.41% ± 0.70%**	**8.6834 ± 2.4882**

Table A.5: (continued)

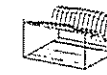
spect^{r} (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)



Support Vectors (%)

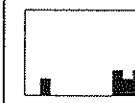


Norm

spect^{r} (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 82.86% ± 1.65%

Good Weak Hyp. (%): 73.50% ± 20.48%



Execution Time (s): 0.17 s ± 0.03 s

spiral^{0} (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	55.67% ± 0.00%	76.60% ± 0.00%	190.48 s ± 0.00 s
1	0.4	51.67% ± 0.00%	100.00% ± 0.00%	0.37 s ± 0.00 s
1	0.1	16.67% ± 0.00%	100.00% ± 0.00%	0.08 s ± 0.00 s
1000	0.1	51.67% ± 0.00%	12.80% ± 0.00%	4.91 s ± 0.00 s
1000	0.7	57.67% ± 0.00%	75.00% ± 0.00%	276.95 s ± 0.00 s
100	1	57.33% ± 0.00%	81.00% ± 0.00%	54.27 s ± 0.00 s
1	0.7	54.33% ± 0.00%	100.00% ± 0.00%	0.63 s ± 0.00 s
1000	**1**	**62.67% ± 0.00%**	**74.60% ± 0.00%**	**353.34 s ± 0.00 s**
100	0.7	56.33% ± 0.00%	80.00% ± 0.00%	42.76 s ± 0.00 s
100	0.4	55.00% ± 0.00%	86.00% ± 0.00%	30.39 s ± 0.00 s
1	1	54.00% ± 0.00%	100.00% ± 0.00%	0.92 s ± 0.00 s
100	0.1	51.33% ± 0.00%	28.00% ± 0.00%	1.43 s ± 0.00 s

Table A.5: (continued)

spiral^0 (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

spiral^0 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0397 ± 0.0000	1.08% ± 0.00%	23.77% ± 0.00%	136.2938 ± 0.0000
1	0.4	0.3934 ± 0.0000	0.73% ± 0.00%	79.33% ± 0.00%	215.3430 ± 0.0000
1	0.1	0.4512 ± 0.0000	0.67% ± 0.00%	19.00% ± 0.00%	57.3780 ± 0.0000
1000	0.1	8.9019 ± 0.0000	0.05% ± 0.00%	0.68% ± 0.00%	1.8481 ± 0.0000
1000	0.7	0.0302 ± 0.0000	1.08% ± 0.00%	26.33% ± 0.00%	162.8743 ± 0.0000
100	1	0.1961 ± 0.0000	0.93% ± 0.00%	65.82% ± 0.00%	314.4157 ± 0.0000
1	0.7	0.6370 ± 0.0000	0.51% ± 0.00%	133.33% ± 0.00%	331.0681 ± 0.0000
1000	**1**	**0.0269 ± 0.0000**	**1.08% ± 0.00%**	**28.25% ± 0.00%**	**184.7546 ± 0.0000**
100	0.7	0.2023 ± 0.0000	0.94% ± 0.00%	61.83% ± 0.00%	358.5158 ± 0.0000
100	0.4	0.2520 ± 0.0000	0.94% ± 0.00%	56.99% ± 0.00%	232.3269 ± 0.0000
1	1	0.8291 ± 0.0000	0.37% ± 0.00%	171.00% ± 0.00%	418.1260 ± 0.0000
100	0.1	7.4077 ± 0.0000	0.17% ± 0.00%	3.01% ± 0.00%	8.4547 ± 0.0000

spiral^0 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

Support Vectors (%)

Norm

spiral^0 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 62.67% ± 0.00%

Good Weak Hyp. (%): 74.60% ± 0.00%

Execution Time (s): 353.34 s ± 0.00 s

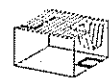
Table A.5: (continued)

spiral[1] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	53.33% ± 0.00%	98.70% ± 0.00%	361.75 s ± 0.00 s
1	0.4	49.67% ± 0.00%	100.00% ± 0.00%	0.37 s ± 0.00 s
1	0.1	53.00% ± 0.00%	100.00% ± 0.00%	0.08 s ± 0.00 s
1000	0.1	49.33% ± 0.00%	17.40% ± 0.00%	8.73 s ± 0.00 s
1000	0.7	54.00% ± 0.00%	99.40% ± 0.00%	643.46 s ± 0.00 s
100	1	54.00% ± 0.00%	100.00% ± 0.00%	91.06 s ± 0.00 s
1	0.7	49.00% ± 0.00%	100.00% ± 0.00%	0.69 s ± 0.00 s
1000	1	54.33% ± 0.00%	99.70% ± 0.00%	907.12 s ± 0.00 s
100	**0.7**	**54.67% ± 0.00%**	**100.00% ± 0.00%**	**64.38 s ± 0.00 s**
100	0.4	53.67% ± 0.00%	96.00% ± 0.00%	35.73 s ± 0.00 s
1	1	52.33% ± 0.00%	100.00% ± 0.00%	0.96 s ± 0.00 s
100	0.1	48.67% ± 0.00%	95.00% ± 0.00%	8.11 s ± 0.00 s

spiral[1] (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)



Good Weak Hyp. (%)

Execution Time (s)

spiral[1] (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.5318 ± 0.0000	0.83% ± 0.00%	75.69% ± 0.00%	184.4239 ± 0.0000
1	0.4	0.3316 ± 0.0000	0.79% ± 0.00%	82.67% ± 0.00%	217.3890 ± 0.0000
1	0.1	0.1119 ± 0.0000	1.04% ± 0.00%	19.33% ± 0.00%	56.9593 ± 0.0000
1000	0.1	8.2328 ± 0.0000	0.13% ± 0.00%	1.56% ± 0.00%	4.1945 ± 0.0000
1000	0.7	0.8370 ± 0.0000	0.71% ± 0.00%	111.65% ± 0.00%	248.7970 ± 0.0000
100	1	1.1471 ± 0.0000	0.59% ± 0.00%	132.73% ± 0.00%	288.9575 ± 0.0000
1	0.7	0.4034 ± 0.0000	0.72% ± 0.00%	137.33% ± 0.00%	356.7749 ± 0.0000
1000	1	1.1241 ± 0.0000	0.61% ± 0.00%	131.13% ± 0.00%	284.7723 ± 0.0000
100	**0.7**	**0.8555 ± 0.0000**	**0.69% ± 0.00%**	**112.66% ± 0.00%**	**252.2809 ± 0.0000**
100	0.4	0.5099 ± 0.0000	0.83% ± 0.00%	75.77% ± 0.00%	185.6105 ± 0.0000
1	1	0.6799 ± 0.0000	0.48% ± 0.00%	174.00% ± 0.00%	446.7588 ± 0.0000
100	0.1	0.8152 ± 0.0000	0.88% ± 0.00%	18.74% ± 0.00%	50.9288 ± 0.0000

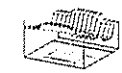
Table A.5: (continued)

spiral[1] (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

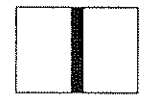
Support Vectors (%)



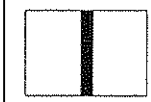
Norm

spiral[1] (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 54.67% ± 0.00%



Good Weak Hyp. (%): 100.00% ± 0.00%



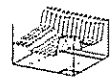
Execution Time (s): 64.38 s ± 0.00 s

spiral[2] (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	15.33% ± 0.00%	0.30% ± 0.00%	3.03 s ± 0.00 s
1	0.4	0.00% ± 0.00%	0.00% ± 0.00%	0.08 s ± 0.00 s
1	0.1	3.00% ± 0.00%	100.00% ± 0.00%	0.03 s ± 0.00 s
1000	0.1	9.67% ± 0.00%	0.20% ± 0.00%	2.26 s ± 0.00 s
1000	0.7	48.33% ± 0.00%	0.90% ± 0.00%	4.48 s ± 0.00 s
100	1	11.00% ± 0.00%	4.00% ± 0.00%	0.79 s ± 0.00 s
1	0.7	48.67% ± 0.00%	100.00% ± 0.00%	0.13 s ± 0.00 s
1000	1	50.67% ± 0.00%	0.60% ± 0.00%	4.71 s ± 0.00 s
100	**0.7**	**51.00% ± 0.00%**	**8.00% ± 0.00%**	**1.24 s ± 0.00 s**
100	0.4	13.67% ± 0.00%	2.00% ± 0.00%	0.43 s ± 0.00 s
1	1	0.00% ± 0.00%	100.00% ± 0.00%	0.17 s ± 0.00 s
100	0.1	16.67% ± 0.00%	2.00% ± 0.00%	0.28 s ± 0.00 s

Table A.5: (continued)

spiral^2 (strong hypotheses' average result graphs for $T \times \rho$)



Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

spiral^2 (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	9.9642 ± 0.0000	0.00% ± 0.00%	0.01% ± 0.00%	201326.5920 ± 0.0000
1	0.4	-0.0057 ± 0.0000	1.17% ± 0.00%	4.00% ± 0.00%	0.0000 ± 0.0000
1	0.1	1.5890 ± 0.0000	0.09% ± 0.00%	1.33% ± 0.00%	67108864.0000 ± 0.0000
1000	0.1	9.9828 ± 0.0000	0.00% ± 0.00%	0.00% ± 0.00%	67108.8640 ± 0.0000
1000	0.7	9.8952 ± 0.0000	0.01% ± 0.00%	0.02% ± 0.00%	868220.9280 ± 0.0000
100	1	9.5658 ± 0.0000	0.02% ± 0.00%	0.11% ± 0.00%	2013265.9200 ± 0.0000
1	0.7	0.0029 ± 0.0000	1.16% ± 0.00%	6.00% ± 0.00%	-536870912.0000 ± 0.0000
1000	1	9.9379 ± 0.0000	0.00% ± 0.00%	0.01% ± 0.00%	201326.5920 ± 0.0000
100	**0.7**	**8.9389 ± 0.0000**	**0.09% ± 0.00%**	**0.31% ± 0.00%**	**-3061841.9200 ± 0.0000**
100	0.4	9.7249 ± 0.0000	0.02% ± 0.00%	0.07% ± 0.00%	335544.3200 ± 0.0000
1	1	0.0000 ± 0.0000	1.17% ± 0.00%	6.67% ± 0.00%	-536870912.0000 ± 0.0000
100	0.1	9.8306 ± 0.0000	0.00% ± 0.00%	0.03% ± 0.00%	1677721.6000 ± 0.0000

spiral^2 (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

Support Vectors (%)

Norm

spiral^2 (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 51.00% ± 0.00%

Good Weak Hyp. (%): 8.00% ± 0.00%

Execution Time (s): 1.24 s ± 0.00 s

Table A.5: (continued)

twonorm (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	96.57% ± 0.88%	72.00% ± 3.90%	163.08 s ± 1.41 s
1	0.4	92.63% ± 3.70%	100.00% ± 0.00%	0.11 s ± 0.01 s
1	0.1	92.13% ± 4.73%	100.00% ± 0.00%	0.10 s ± 0.00 s
1000	**0.1**	**96.67% ± 0.80%**	**74.16% ± 2.89%**	**46.22 s ± 0.16 s**
1000	0.7	96.63% ± 0.67%	67.80% ± 5.38%	269.66 s ± 3.29 s
100	1	96.40% ± 0.71%	72.10% ± 4.61%	37.15 s ± 0.55 s
1	0.7	95.70% ± 1.71%	100.00% ± 0.00%	0.12 s ± 0.01 s
1000	1	96.40% ± 0.70%	63.15% ± 7.08%	378.33 s ± 5.33 s
100	0.7	96.57% ± 0.70%	74.70% ± 7.40%	26.65 s ± 0.20 s
100	0.4	96.53% ± 0.75%	77.90% ± 5.19%	15.83 s ± 0.14 s
1	1	94.03% ± 4.50%	100.00% ± 0.00%	0.13 s ± 0.01 s
100	0.1	96.53% ± 0.83%	81.80% ± 4.02%	4.52 s ± 0.09 s

twonorm (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

twonorm (average results for weak hypotheses)

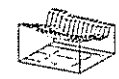
T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0892 ± 0.0270	0.62% ± 0.14%	15.80% ± 1.91%	1445673.0968 ± 305136.4598
1	0.4	1.3832 ± 0.3075	0.16% ± 0.11%	13.83% ± 1.92%	1122470.1938 ± 141998.1300
1	0.1	1.3248 ± 0.3210	0.18% ± 0.12%	7.67% ± 0.83%	708846.6875 ± 107073.9679
1000	**0.1**	**0.0899 ± 0.0139**	**0.77% ± 0.07%**	**11.42% ± 0.47%**	**1266186.5258 ± 51405.4585**
1000	0.7	0.0794 ± 0.0324	0.57% ± 0.15%	17.20% ± 2.25%	724257.8457 ± 801863.5204
100	1	0.1443 ± 0.0436	0.44% ± 0.12%	22.62% ± 3.30%	962667.9181 ± 1259685.7190
1	0.7	1.5974 ± 0.1257	0.09% ± 0.02%	17.87% ± 2.21%	1433305.0500 ± 248860.7809
1000	1	0.0609 ± 0.0368	0.52% ± 0.15%	17.65% ± 2.79%	-1023976.2484 ± 1544557.6291
100	0.7	0.1548 ± 0.0625	0.50% ± 0.14%	20.45% ± 2.84%	1926126.4553 ± 924236.0700
100	0.4	0.1651 ± 0.0399	0.49% ± 0.12%	17.93% ± 1.86%	1773930.9557 ± 348593.2286
1	1	1.6129 ± 0.3951	0.12% ± 0.12%	20.57% ± 2.60%	1509674.4250 ± 199655.5714
100	0.1	0.1676 ± 0.0176	0.64% ± 0.06%	11.59% ± 0.31%	1337570.7937 ± 75393.9190

Table A.5: (continued)

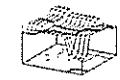
twonorm (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

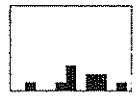


Support Vectors (%)



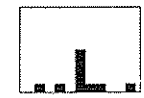
Norm

twonorm (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 96.67% ± 0.80%

Good Weak Hyp. (%): 74.16% ± 2.89%



Execution Time (s): 46.22 s ± 0.16 s

wdbc (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	92.75% ± 3.01%	63.32% ± 1.79%	44.88 s ± 0.51 s
1	0.4	84.27% ± 7.92%	100.00% ± 0.00%	0.08 s ± 0.01 s
1	0.1	77.43% ± 26.06%	90.00% ± 30.00%	0.07 s ± 0.01 s
1000	0.1	90.58% ± 4.76%	64.72% ± 1.52%	10.62 s ± 0.10 s
1000	**0.7**	**94.04% ± 1.47%**	**64.82% ± 2.51%**	**80.04 s ± 0.97 s**
100	1	93.45% ± 1.79%	74.90% ± 2.77%	10.88 s ± 0.16 s
1	0.7	84.15% ± 9.88%	100.00% ± 0.00%	0.07 s ± 0.00 s
1000	1	93.68% ± 1.61%	64.03% ± 2.14%	114.26 s ± 1.99 s
100	0.7	92.92% ± 2.02%	74.00% ± 4.98%	7.66 s ± 0.19 s
100	0.4	92.98% ± 1.46%	76.10% ± 5.07%	4.33 s ± 0.12 s
1	1	86.78% ± 6.78%	100.00% ± 0.00%	0.08 s ± 0.01 s
100	0.1	92.22% ± 2.04%	72.90% ± 4.99%	1.10 s ± 0.03 s

Table A.5: (continued)

wdbc (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

wdbc (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0313 ± 0.0058	1.05% ± 0.03%	29.78% ± 1.67%	7902.7766 ± 1434.3142
1	0.4	0.9276 ± 0.2510	0.34% ± 0.16%	11.17% ± 3.49%	807.7315 ± 450.7589
1	0.1	0.8442 ± 0.4636	0.43% ± 0.41%	5.32% ± 2.62%	331.8846 ± 457.1778
1000	0.1	0.0295 ± 0.0046	1.10% ± 0.02%	11.89% ± 0.20%	3873.2360 ± 726.3830
1000	**0.7**	**0.0349 ± 0.0078**	**1.03% ± 0.02%**	**37.95% ± 3.15%**	**9670.5489 ± 2251.5711**
100	1	0.1059 ± 0.0112	0.92% ± 0.05%	53.48% ± 3.01%	10031.3235 ± 1867.9671
1	0.7	0.9618 ± 0.2825	0.33% ± 0.19%	17.84% ± 7.11%	1632.4078 ± 1307.2061
1000	1	0.0350 ± 0.0060	1.02% ± 0.03%	43.56% ± 3.31%	11675.1315 ± 1895.1237
100	0.7	0.0996 ± 0.0151	0.93% ± 0.06%	44.04% ± 1.95%	7742.8310 ± 1146.5808
100	0.4	0.1058 ± 0.0154	0.95% ± 0.04%	32.67% ± 1.25%	5690.3505 ± 847.5184
1	1	0.9783 ± 0.2424	0.31% ± 0.13%	19.82% ± 8.01%	2322.5590 ± 1694.9120
100	0.1	0.0869 ± 0.0112	1.04% ± 0.05%	11.74% ± 0.26%	2426.9400 ± 256.2254

wdbc (weak hypotheses' average result graphs for $T \times \rho$)

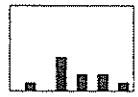
Alpha

Error Rate (%)

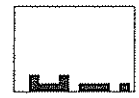
Support Vectors (%)

Norm

wdbc (experiment histogram for parameter setup that yielded best results)



Accuracy (%): 94.04% ± 1.47%



Good Weak Hyp. (%): 64.82% ± 2.51%

Execution Time (s): 80.04 s ± 0.97 s

Table A.5: (continued)

wpbc (average results for strong hypotheses)

T	ρ	Accuracy (%)	Good Weak Hypotheses (%)	Execution Time (s)
1000	0.4	77.67% ± 5.39%	61.84% ± 1.12%	5.01 s ± 0.04 s
1	0.4	48.67% ± 24.99%	80.00% ± 40.00%	0.03 s ± 0.00 s
1	0.1	35.00% ± 30.06%	60.00% ± 48.99%	0.02 s ± 0.01 s
1000	0.1	76.17% ± 5.58%	58.56% ± 1.83%	1.39 s ± 0.02 s
1000	0.7	76.00% ± 4.36%	62.94% ± 1.42%	8.85 s ± 0.07 s
100	1	73.00% ± 6.32%	70.70% ± 4.47%	1.31 s ± 0.02 s
1	0.7	37.33% ± 31.97%	60.00% ± 48.99%	0.03 s ± 0.00 s
1000	1	76.67% ± 5.63%	63.66% ± 1.15%	12.98 s ± 0.08 s
100	0.7	74.67% ± 6.05%	69.10% ± 4.68%	0.90 s ± 0.02 s
100	0.4	73.17% ± 5.24%	66.80% ± 5.10%	0.52 s ± 0.01 s
1	1	58.00% ± 29.25%	80.00% ± 40.00%	0.03 s ± 0.01 s
100	0.1	74.67% ± 6.57%	63.10% ± 4.57%	0.16 s ± 0.01 s

wpbc (strong hypotheses' average result graphs for $T \times \rho$)

Accuracy (%)

Good Weak Hyp. (%)

Execution Time (s)

wpbc (average results for weak hypotheses)

T	ρ	Alpha	Error Rate (%)	Support Vectors (%)	Norm
1000	0.4	0.0177 ± 0.0023	1.15% ± 0.02%	47.36% ± 0.38%	869.4542 ± 124.7982
1	0.4	0.2112 ± 0.2214	0.92% ± 0.24%	37.33% ± 6.72%	282.6220 ± 212.7706
1	0.1	0.0269 ± 0.2295	1.12% ± 0.25%	10.50% ± 3.95%	125.6832 ± 118.4744
1000	0.1	0.0113 ± 0.0029	1.16% ± 0.01%	14.34% ± 0.22%	301.9640 ± 24.5781
1000	0.7	0.0214 ± 0.0020	1.14% ± 0.01%	71.75% ± 1.36%	1151.3634 ± 157.1380
100	1	0.0466 ± 0.0104	1.12% ± 0.02%	94.39% ± 1.84%	1247.2487 ± 266.5689
1	0.7	0.1234 ± 0.2712	1.02% ± 0.29%	61.50% ± 10.07%	465.3461 ± 164.5376
1000	1	0.0229 ± 0.0020	1.13% ± 0.01%	89.75% ± 2.48%	1462.7027 ± 226.1739
100	0.7	0.0460 ± 0.0094	1.11% ± 0.02%	73.82% ± 1.97%	990.8765 ± 156.6613
100	0.4	0.0382 ± 0.0070	1.15% ± 0.02%	47.87% ± 0.80%	667.6229 ± 100.2174
1	1	0.3776 ± 0.3176	0.76% ± 0.33%	84.00% ± 8.83%	949.7512 ± 715.9812
100	0.1	0.0276 ± 0.0070	1.15% ± 0.04%	14.33% ± 0.59%	246.0793 ± 36.4175

Table A.5: (continued)

wpbc (weak hypotheses' average result graphs for $T \times \rho$)

Alpha

Error Rate (%)

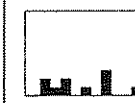
Support Vectors (%)

Norm

wpbc (experiment histogram for parameter setup that yielded best results)

Accuracy (%): 77.67% ± 5.39%

Good Weak Hyp. (%): 61.84% ± 1.12%



Execution Time (s): 5.01 s ± 0.04 s

Appendix B

Notes on Performance Measures

All experiments conducted in this work were executed in two identical server machines based on the Intel architecture. These machines were each powered by an Intel Pentium IV processor running at 2.0 GHz, 1.5 Gb of Double Data Rate (DDR) SDRAM memory on a bus running at 333 MHz, and a Seagate Barracuda ATA IV 80 Gb hard disk.

Each machine was installed with Linux distribution Red Hat version 7.3, where we used the Linux kernel version 2.4.18-3 originally shipped during all experiments. Time measurements were based on the kernel's instrumentation interface that returns the amount of processing time that user processes get from the machine's processor. Accesses to this interface was performed via the GNU bash – the Bourne Again Shell – utility `times`, where we used GNU bash version 2.05a.0(1)-release. Testing frameworks for all experiments were implemented using this version of the GNU bash and the GNU perl interpreter version 5.6.1. All algorithms were compiled with gcc – the GNU C Compiler – version 3.2, using dynamically-linked C library (libc) version 6.2.2.

Bibliography

[AA02] A. B. M. Shawkat Ali and Ajith Abraham. An empirical comparison of kernel selection for support vector machines. Technical report, Monash University, 2002.

[ABB00] M. B. Almeida, A. P. Braga, and J. P. Braga. SVM-KM: speeding SVMs learning with a priori cluster selection and k-means. In *Proceedings of the VIth Brazilian Symposium on Neural Networks*. IEEE Computer Society Press, November 2000.

[ABB01a] M. B. Almeida, A. P. Braga, and J. P. Braga. Training SVMs with EDR Algorithm. *International Journal of Neural Systems*, 11(3):257–263, 2001.

[ABB01b] M. B. Almeida, A. P. Braga, and J. P. Braga. Uma introdução a support vector machines. Technical report, Departamentos de Engenharia Eletrônica e Química da Universidade Federal de Minas Gerais, 2001.

[ASS99] S. Abney, R. E. Schapire, and Y. Singer. Boosting applied to tagging and pp attachment. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[ASS00] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[AW89] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Europhysics Letters*, 10:687–692, 1989.

[Bar98] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, March 1998.

[BGL+99] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, Jr. M. Ares, and D. Haussler. Support vector machine classification of microarray gene expression data. Technical report, University of California, Santa Cruz, 1999.

[BH89] E. B. Baum and D. Haussler. What size net gives valid generalization? *Advances in Neural Information Processing Systems*, I:81–90, 1989.

[BLC00] A. P. Braga, T. B. Ludermir, and A. F. Carvalho. *Redes Neurais Artificiais: Teoria e Aplicações*. LTC, 2000.

[BM98] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

[BMM99] P. S. Bradley, O. L. Mangasarian, and D. R. Musicant. Optimization methods in massive datasets. Technical report, University of Wisconsin in Madison, 1999.

[Bre94] Leo Breiman. Bagging predictors. Technical report, University of California at Berkeley, 1994.

[Bre96] Leo Breiman. Bias, variance and arcing classifiers. Technical report, University of California at Berkeley, April 1996.

[BSS92] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley, second edition, 1992.

[Bur98] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Cac94] Christian Cachin. Pedagogical pattern selection strategies. *Neural Networks*, 7(1):175–181, 1994.

[CBFH+93] N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, D. Haussler, and R. E. Schapire. How to use expert advice. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 382–391, May 1993.

[CDH+00] J. Cai, A. Dayanik, N. Hasan, T. Terauchi, and H. Yu. Supervised machine learning algorithms for classification of cancer tissue types using microarray gene expression data. Technical report, Columbia University, 2000.

[Chu94] T. H. Chung. Approximate methods for sequential decision making using expert advice. In *Proceedings of the Seventh Annual ACM Symposium on Computational Learning Theory*, pages 183–189, 1994.

[CKB87] G. Cestnik, I. Konenenko, and I. Bratko. Assistant-86: A knowledge-elicitation tool for sophisticated users. *Progress in Machine Learning*, pages 31–45, 1987.

[CSS00] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, AdaBoost and bregman distances. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, 2000.

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[CSTC98] N. Cristianini, J. Shawe-Taylor, and C. Campbell. Dynamically adapting kernels in support vector machines. *Advances in Neural Information Processing Systems*, 11, 1998.

[DB95] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., 2001.

[Die00] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–158, 2000.

[DLLP97] Thomas G. Dietterich, Richard H. Lathrop, and Tomas Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[DPHS98] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *7th International Conference on Information and Knowledge Management*, 1998.

[DSS93] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:705 – 719, 1993.

[EE00] Shmuel Etin and Uri Elias. Parallelizing SMO for solving SVMs. Technical report, Technion - Israel Institute of Technology, 2000.

[FISS98] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.

[FL01] Gary W. Flake and Steve Lawrence. Efficient SVM regression training with SMO. *Machine Learning*, 2001.

[Fle87] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Inc., second edition, 1987.

[FMS01] Y. Freund, Y. Mansour, and R. E. Schapire. Why averaging classifiers can protect against overfitting. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, 2001.

[Fre95] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 2(121):256–285, September 1995.

[FS95] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*. LNCS, March 1995.

[FS96a] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[FS96b] Y. Freund and R. E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the 9th Annual Conference on Computer Learning Theory*, pages 325–332. ACM Press, New York, NY, 1996.

[FS97] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[FS99a] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behaviour*, 29:79–103, 1999.

[FS99b] Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, September 1999. Appearing in Japanese, translation by Naoki Abe.

[FSSW97] Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth. Using and combining predictors that specialize. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 334–343, El Paso, Texas, 4–6 1997.

[Han99] Christian Hansen. The l-curve and its use in the numerical treatment of inverse problems. Technical report, Technical University of Denmark, 1999.

[Hau99] D. Haussler. Convolution kernels on discrete structures. Technical report, University of California in Santa Cruz, July 1999.

[Hay94] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company, Inc., 866 Third Avenue, New York, New York 10022, 1994.

[Heb49] D. O. Hebb. *The Organization of Behavior*. Wiley, 1949.

[HKLW91] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95(2):129–161, December 1991.

[HKW95] D. Haussler, J. Kivinen, and M. K. Warmuth. Tight worst-case loss bounds for predicting with expert advice. In *Proceedings of the Second European Conference on Computational Learning Theory*, 1995.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective properties. In *Proceedings of the National Academy of Sciences*, 79, pages 2554–2558, 1982.

[HR87] C. Harley and R. Reynolds. Analysis of e. coli promoter sequences. *Nucleic Acids Research*, 15:2343–2361, 1987.

[ILS+00] R. D. Iyer, D. D. Lewis, R. E. Schapire, Y. Singer, and A. Singhal. Boosting for document routing. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, 2000.

[JH98] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. *Advances in Neural Information Processing Systems*, 11, 1998.

[JH99] T. S. Jaakkola and D. Haussler. Probabilistic kernel regression models. In *Proceedings of the 1999 Conference on AI and Statistics*, 1999.

[Joa98a] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11. MIT Press, 1998.

[Joa98b] T. Joachims. Text categorization with support vector machines. In *Proceedings of European Conference on Machine Learning (ECML)*, 1998.

[KA00] C. Kaynak and E. Alpaydin. Multistage cascading of multiple classifiers: One man's noise is another man's data. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 455–462. Morgan Kaufmann, 2000.

[KCT+01] L. A. Kurgan, K. J. Cios, R. Tadeusiewicz, M. Ogiela, and L. S. Goodenday. Knowledge discovery approach to automated cardiac spect diagnosis. *Artificial Intelligence in Medicine*, 23(2):149–169, October 2001.

[KK01] Michihiro Kuramochi and George Karypis. Gene classification using expression profiles: A feasibility study. In *IEEE International Conference on Bioinformatics and Biomedical Egineering*, pages 191–200, 2001.

[KV94] M. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

[KW94] J. Kivinen and M. K. Warmuth. Using experts for predicting continuous outcomes. In *Computational Learning Theory: EuroCOLT '93*, pages 109–120, 1994.

[Lee00] Yuh-Jye Lee. Smooth support vector machines. Technical report, University of Wisconsin, 2000.

[LW94] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

[MM98] O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. Technical report, University of Wisconsin in Madison, 1998.

[MM99] O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10:1032–1037, 1999.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP69] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[MSW95] O. L. Mangasarian, W. N. Street, and W. H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, July-August 1995.

[Mun92] P. W. Munro. Repeat until bored: A pattern selection strategy. *Advances in neural information processing systems*, 4:1001–1008, 1992.

[OC89] M. C. O'Neill and F. Chiafari. Escherichia coli promoters. ii. a spacing-class dependent promoter search protocol. *Journal of Biological Chemistry*, 264:5531–5534, 1989.

[OFG97] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII – Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, 1997.

[O'N89] M. O'Neill. Escherichia coli promoters: Consensus as it relates to spacing class specificity, repeat substructure, and threedimensional organization. *Journal of Biological Chemistry*, 264:5522–5530, 1989.

[ORM00] Takashi Onoda, Gunnar Rätsch, and Klaus-Robert Muller. Applying support vector machines and boosting to a non-intrusive monitoring system for household electric appliances with inverters. In *Proceedings of Neural Computing 2000*, 2000.

[Pla98a] John C. Platt. *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. MIT Press, 1998.

[Pla98b] John C. Platt. Sequential minimal optimization: a fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

[Por98] Noah Porter, editor. *Webster's Revised Unabridged Dictionary*. C. & G. Merriam Co., 1998.

[PV98] Massimiliano Pontil and Alessandro Verri. Properties of support vector machines. *Neural Computation*, 10:955–974, 1998.

[Qui92] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. The Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann, October 1992.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[RMR99] Greg Ridgeway, David Madigan, and Thomas Richardson. Boosting methodology for regression problems. Technical report, University of Washington, 1999.

[RMSM02] Gunnar Rätsch, Sebastian Mika, Bernhard Schölkopf, and Klaus-Robert Müller. Constructing boosting algorithms from SVMs: An application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9), September 2002.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65:386–408, 1958.

[Rät01] Gunnar Rätsch. *Robust Boosting via Convex Optimization: Theory and Applications*. PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultät, 2001.

[RTR+01] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C. H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, , and T. R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *PNAS*, 98(26):15149–15154, 2001.

[RZH02] S. Rosset, J. Zu, and T. Hastie. Boosting as regularized path to a maximum margin classifier. Technical report, Stanford University, 2002.

[Sch90] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[Sch92] R. E. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. MIT Press, 1992.

[Sch97] R. E. Schapire. Using output codes to boost multiclass learning problems. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 313–321, 1997.

[Sch99a] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[Sch99b] R. E. Schapire. Theoretical views of boosting. In *Computational Learning Theory: Fourth European Conference, EuroCOLT'99*, 1999.

[Sch99c] R. E. Schapire. Theoretical views of boosting and applications. In *Tenth International Conference on Algorithmic Learning Theory*, 1999.

[Sch01] R. E. Schapire. Drifting games. *Machine Learning*, 2001. to appear.

[Sch02] R. E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[SED+88] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

[SFBL97] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proceedings of the 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997.

[SFBL98] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, October 1998.

[SMW95] W. N. Street, O. L. Mangasarian, and W. H. Wolberg. An inductive learning approach to prognostic prediction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 522–530. Morgan Kaufmann, 1995.

[SS98] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998.

[SS99] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, December 1999.

[SS00] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, May/June 2000.

[SSS98] R. E. Schapire, Y. Singer, and A. Singhal. Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval*, 1998.

[Str86] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[SWHB89] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[TBTS00] R. Teixeira, A. P. Braga, R. H. C. Takahashi, and R. R. Saldanha. Improving generalization of mlps with multi-objective optimization. *Neurocomputing*, 35(1-4), 2000.

[TBTS01] R. Teixeira, A. P. Braga, R. H. C. Takahashi, and R. R. Saldanha. Recent advances in the mobj algorithm for training artificial neural networks. *International Journal of Neural Systems*, 11(3), 2001.

[Tei01] R. Teixeira. *Treinamento de redes neurais artificiais através de otimização multi-objetivo: uma nova abordagem para o equilíbrio entre a polarização a variância*. PhD thesis, Programa de Pós-Graduação em Engenharia Elétrica, 2001.

[TSN90] G. Towell, J. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge-based artificial neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, 1990.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Vap82] V. N. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, New York, 1982.

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[VC71] V. N. Vapnik and A. J. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[Vov90] V. G. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383, 1990.

[Wat99a] C. Watkins. Dynamic alignment kernels. Technical report, University of London, January 1999.

[Wat99b] C. Watkins. Dynamic alignment kernels. In A. J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.

[Wat99c] C. Watkins. Kernels from matching operations. Technical report, University of London, July 1999.

[WD81] R. S. Wenocur and R. M. Dudley. Some special vapnik-chervonenkis classes. *Discrete Mathematics*, 33:313–318, 1981.

[WH60] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *Institute of Radio Engineers, Western Electronic Show and Convention*, 1960.

[WM90] W. H. Wolberg and O. L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In *Proceedings of the National Academy of Sciences*, pages 9193–9196, 1990.

[Zha92] J. Zhang. Selecting typical instances in instance-based learning. In *Proceedings of the Ninth International Machine Learning Conference*, pages 470–479. Morgan Kaufmann, 1992.