RAUL ACOSTA HERNANDEZ

***MP-SMO: UM ALGORITMO PARA A IMPLEMENTAÇÃO VLSI DO
TREINAMENTO DE MÁQUINAS DE VETORES DE SUPORTE***

São Paulo
2009

RAUL ACOSTA HERNANDEZ

***MP-SMO: UM ALGORITMO PARA A IMPLEMENTAÇÃO VLSI DO
TREINAMENTO DE MÁQUINAS DE VETORES DE SUPORTE***

**Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do título de Mestre
em Engenharia**

São Paulo
2009

RAUL ACOSTA HERNANDEZ

***MP-SMO: UM ALGORITMO PARA A IMPLEMENTAÇÃO VLSI DO
TREINAMENTO DE MÁQUINAS DE VETORES DE SUPORTE***

**Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do título de Mestre
em Engenharia**

Área de Concentração: Microeletrônica

Orientador: Prof. Dr. Marius Strum

São Paulo
2009

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 25 de setembro de 2009.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Hernandez, Raul Acosta

**MP-SMO: um algoritmo para a implementação VSLI do treinamento de máquinas de vetores de suporte / R.A. Hernandez. -- ed.rev. --São Paulo, 2009.
p. 112**

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

**1. Inteligência artificial 2. Algoritmos 3. Microeletrônica
4. Circuitos FPGA I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.**

Dedicada a minha avó Amália Reyes, com muito carinho

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Marius Strum, pelos conhecimentos recebidos de microeletrônica e de elaboração de trabalhos científicos, pelas recomendações que guiaram meu mestrado e este trabalho em particular. Agradeço também a ele a oportunidade de poder realizar meus estudos de pós-graduação em tão prestigiosa instituição. Também lhe agradeço pela amizade e preocupação por questões pessoais.

Também ao Prof. Dr. Wang Jiang Chau pela contribuição na elaboração dos trabalhos. Agradeço a ele e ao Prof. Dr. Marius Strum, líderes do Grupo de Projeto de Sistemas Eletrônicos Integrados e *Software* Aplicado (GSEIS), pela procura da excelência em todos os trabalhos realizados e condições de infraestrutura do grupo. Agradeço ao Prof. Dr. Artur Quilici Gonzalez, que junto ao Prof. Dr. Wang Jiang Chau propuseram o assunto de pesquisa e contribuíram na realização dele.

Um profundo agradecimento aos meus colegas do GSEIS por toda a ajuda recebida, do ponto de vista científico e em questões cotidianas. A todos eles agradeço a sua amizade.

Agradeço aos Professores Dr. Fernando Silveira e Eng. Conrado Rossi e aos meus colegas da Universidad de la República (UDELAR) do Uruguai.

Agradeço muito a minha mãe Obdulia Hernandez, a minha irmã Gabriela Acosta e seu esposo, Rafael Normey, por estarem sempre presentes e pelo afeto essencial.

Um especial agradecimento a minha namorada Márcia Teixeira por seu apoio e afeto, e a sua mãe Helena Bastos, por toda sua ajuda.

Às minhas avós e meus avôs, que são a gênese.

Ao resto da minha família pelo apoio incondicional, em especial Silvanina, Humberto, Paola, Homero, Flavia e Alberto.

Também aos meus amigos, pela preocupação e conselhos.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e à Universidade de São Paulo, por todos os recursos que recebi.

A todas essas pessoas e instituições com toda sinceridade, e com a esperança de não decepcioná-los.

“An artificial intelligent system must be capable of doing three things: 1) store knowledge, 2) apply the knowledge stored to solve problems, and 3) acquire new knowledge through experience.”

(Sage, 1990. In **(1)**)

RESUMO

Máquinas de aprendizagem, como Redes Neurais Artificiais (ANNs), Redes Bayesianas, Máquinas de Vetores de Suporte (SVMs) e outras, são aplicadas em problemas de classificação de padrões. Devido ao baixo erro de teste, a SVM possui uma grande quantidade de aplicações, como no reconhecimento de imagens, seleção de genes, classificação de textos, robótica, reconhecimento de escrita a mão e outras. Dos algoritmos desenvolvidos para o treinamento da SVM, o *Sequential Minimal Optimization* (SMO) é um dos mais rápidos e o mais fácil de implementar em *software*. Devido a sua importância, várias otimizações para diminuir ainda mais o seu tempo de execução têm sido reportadas.

A maioria das implementações do treinamento da SVM foram realizadas em *software*. Não obstante, a implementação em *hardware* é necessária em algumas aplicações com restrições: de área, e/ou de energia e/ou de tempo de treinamento, por exemplo, em algumas aplicações portáteis ou móveis. Nas implementações em *hardware* anteriores a este trabalho, o treinamento da SVM foi realizado com um conjunto de exemplos cuja quantidade é da ordem de somente dezenas, e unicamente uma delas usou o algoritmo SMO.

Neste trabalho é apresentada uma modificação do algoritmo SMO, que denominamos algoritmo SMO de Múltiplos Pares (MP-SMO), para a aceleração do treinamento da SVM. A diminuição do tempo de treinamento é obtida realizando a otimização de um ou mais pares de coeficientes, chamados Multiplicadores de Lagrange, em cada iteração. De modo diferente, o algoritmo SMO original otimiza somente um par. O algoritmo MP-SMO apresenta as seguintes características: 1) a otimização de cada par de coeficientes é mantida simples usando a solução analítica do algoritmo SMO original. 2) as heurísticas para a seleção dos múltiplos pares a otimizar são adaptações das soluções anteriores para a seleção de um par por iteração. Testou-se o algoritmo otimizando até dois, três e quatro pares de coeficientes por iteração, e melhores resultados foram obtidos quando comparados com os do algoritmo SMO. Nos testes realizados com sete *benchmarks*, o tempo de treinamento diminuiu entre 22,5% e 42,8%.

A diminuição do tempo de execução do algoritmo *SMO* em *hardware* é também abordada nesta dissertação. Os algoritmos *SMO* e *MP-SMO* foram completamente implementados em *hardware* dedicado para o *benchmark Tic-tac-toe endgame*. Este *benchmark* é composto por 958 exemplos, uma quantidade superior às usadas nas implementações anteriores. Com o algoritmo *MP-SMO* pretendeu-se reduzir o número de iterações, como na implementação em *software*, e poder incluir paralelismo na implementação em *hardware*. Para diminuir o tempo de execução de cada iteração, arquiteturas dos tipos *pipeline* e paralela foram usadas. Foram implementadas e testadas em um dispositivo do tipo *FPGA* (*Field Programmable Gate Array*) dezesseis diferentes arquiteturas no total, combinando ou não o algoritmo *SMO* ou o *MP-SMO* com *pipelining* e/ou paralelismo. O tempo de treinamento diminuiu no melhor caso para 1,8% do obtido com o algoritmo *SMO* implementado sem *pipelining* nem paralelismo, ou seja, diminuiu em mais de 50 vezes. Esta dissertação apresenta também a análise do custo em área e potência decorrente do aumento da velocidade de treinamento.

ABSTRACT

Learning Machines, like Artificial Neural Networks (ANNs), Bayesian Networks, Support Vector Machines (SVMs) and others are applied in pattern classification problems. As the test error in SVM is small, it has several applications, such as image recognition, gene selection, text classification, robotics, handwritten recognition and others. Among the developed algorithms for the SVM training, the Sequential Minimal Optimization (SMO) is one of the fastest and the simplest to implement in software. Due to its importance, many improvements have been proposed in order to obtain even faster solutions than the original algorithm.

Most of the SVM training implementations are in software. However, in some applications with restrictions of: area, and/or power and/or training time, a hardware implementation is necessary, for example, in some mobile or portable applications. In related previous works, the SVMs were trained in hardware using sets of only tens of examples, and in only one implementation the SMO algorithm was employed.

In this work, a modified version of the SMO algorithm, named here the Multiple Pairs SMO (MP-SMO) algorithm, for the SVM training acceleration is presented. The training time reduction is obtained optimizing per iteration one or more pairs of coefficients known as Lagrange Multipliers, instead of only one pair as in the original SMO algorithm. The MP-SMO algorithm has the following features: 1) the optimization of each pair is as simple as in the original SMO algorithm because of the use of the same analytical method. 2) the solution for the pairs of coefficients selection can be chosen between two adapted heuristics for the SMO algorithm. The algorithm was tested optimizing up to two, three and four pairs of coefficients per iteration, and the training time was improved, when compared against the SMO algorithm. The tests for seven benchmarks showed an improvement that ranged from 22.5% to 42.8%.

The reduction of the training time of the SMO algorithm executed in hardware is also treated in this dissertation. The algorithms SMO and MP-SMO were completely implemented in dedicated hardware for the Tic-tac-toe endgame benchmark. This benchmark is composed of 958 examples, a number greater than the used in the previous hardware implementations. The implementation of the MP-SMO algorithm is

intended to reduce the number of iterations, as in the software implementation, and to include parallelism in the hardware implementation. In order to reduce the iteration execution time, the pipeline and parallel architectures were realized. Sixteen different architectures were implemented and tested on a Field Programmable Gate Array (FPGA) device, combining or not the SMO or MP-SMO algorithm with pipelining and/or parallelism. The training time was reduced to 1.8% of that obtained with the SMO algorithm without neither pipelining nor parallelism, that is, more than 50 times. This dissertation also presents an analysis of the area and power cost of the training speed increase.

LISTA DE ABREVIATURAS E SIGLAS

ANN	Artificial Neural Network
ASIC	Application Specific Integrated Circuit
BSV	Bluespec System Verilog
CORDIC	Coordinate Rotation Digital Computer
F-heurística	Heurística de Fan; Chen e Lin
FPGA	Field Programmable Gate Array
JOP	Java Optimized Processor
K-heurística	Heurística de Keerthi et al.
KKT	Karush-Kuhn-Tucker
LUT	Look Up Table
MP-SMO	Multiple Pairs-Sequential Minimal Optimization
PDA	Personal Digital Assistant
QP	Quadratic Programming
RBF	Radial Basis Function
SMO	Sequential Minimal Optimization
SRAM	Static Random Access Memory
SVM	Support Vector Machine
UCI	University of California at Irvine

SUMÁRIO

1 INTRODUÇÃO	15
1.1 CONTEXTO	15
1.2 MOTIVAÇÕES.....	18
1.2.1 Motivações para aumentar a velocidade de execução do algoritmo <i>SMO</i> implementado em software	18
1.2.2 Motivações para a implementação e aceleração do algoritmo <i>SMO</i> em <i>hardware</i>	19
1.3 OBJETIVOS	21
1.4 ORGANIZAÇÃO DA DISSERTAÇÃO	22
2 TRABALHOS RELACIONADOS	24
2.1 ALGORITMOS DE TREINAMENTO ANTERIORES AO <i>SMO</i>	24
2.2 OTIMIZAÇÕES DO ALGORITMO <i>SMO</i>.....	25
2.3 CONVERGÊNCIA MONOTÔNICA DO ALGORITMO <i>SMO</i>	26
2.4 UM <i>KERNEL HARDWARE</i> AMIGÁVEL.....	27
2.5 IMPLEMENTAÇÕES EM <i>HARDWARE</i> DO TREINAMENTO DA <i>SVM</i>	27
2.6 RESUMO DO CAPÍTULO.....	30
3 FUNDAMENTOS TEÓRICOS	31
3.1 UM PROBLEMA DE CLASSIFICAÇÃO.....	31
3.2 MÁQUINA DE VETORES DE SUPORTE.....	32
3.3 ALGORITMO <i>SEQUENTIAL MINIMAL OPTIMIZATION</i>.....	36
3.3.1 Introdução.....	36
3.3.2 Tarefa 1: seleção do conjunto de trabalho	36
3.3.3 Tarefa 2: otimização do par de coeficientes	39
3.3.4 Tarefa 3: atualização de dados globais	41
3.4 RESUMO DO CAPÍTULO.....	41

4 ALGORITMO SEQUENTIAL MINIMAL OPTIMIZATION DE MÚLTIPLOS PARES.....	42
4.1 INTRODUÇÃO.....	42
4.2 TAREFA 1: SELEÇÃO DO CONJUNTO DE TRABALHO	42
4.3 TAREFA 2: OTIMIZAÇÃO DOS PARES DE COEFICIENTES	46
4.4 TAREFA 3: ATUALIZAÇÃO DE DADOS GLOBAIS	46
4.5 CONVERGÊNCIA MONOTÔNICA	46
4.6 CONFORMIDADE DAS RESTRIÇÕES DO PROBLEMA <i>SVM</i>	49
4.7 RESUMO DO CAPÍTULO.....	51
5 IMPLEMENTAÇÃO EM <i>HARDWARE</i> DO ALGORITMO <i>MP-SMO</i> ...	52
5.1 IMPLEMENTAÇÃO DO <i>KERNEL HARDWARE</i> AMIGÁVEL	52
5.1.1 Escolha do <i>kernel</i>	52
5.1.2 Implementação do <i>kernel</i> para o caso de atributos categóricos	53
5.2 IMPLEMENTAÇÃO DA OTIMIZAÇÃO	54
5.3 IMPLEMENTAÇÃO DA SELEÇÃO DO CONJUNTO DE TRABALHO E DA ATUALIZAÇÃO DE DADOS GLOBAIS	56
5.3.1 Implementação da seleção do conjunto de trabalho	56
5.3.2 Implementação da atualização de dados globais.....	58
5.4 REPRESENTAÇÃO DE DADOS EM PONTO FIXO.....	59
5.5 ARQUITETURA DO ALGORITMO <i>MP-SMO</i>	59
5.6 RESUMO DO CAPÍTULO.....	64
6 RESULTADOS EXPERIMENTAIS.....	66
6.1 RESULTADOS DA IMPLEMENTAÇÃO EM <i>SOFTWARE</i> DO ALGORITMO <i>MP-SMO</i>	66
6.2 RESULTADOS DA IMPLEMENTAÇÃO EM <i>HARDWARE</i> DO ALGORITMO <i>MP-SMO</i>	72
6.3 RESUMO DO CAPÍTULO.....	82
7 CONCLUSÕES E TRABALHOS FUTUROS	83
7.1 CONCLUSÕES SOBRE O ALGORITMO <i>MP-SMO</i>	83
7.2 TRABALHOS FUTUROS RELATIVOS AO ALGORITMO <i>MP-SMO</i>	83

7.3 CONCLUSÕES SOBRE A IMPLEMENTAÇÃO EM <i>HARDWARE</i> DO ALGORITMO <i>MP-SMO</i>	84
7.4 TRABALHOS FUTUROS RELATIVOS À IMPLEMENTAÇÃO EM <i>HARDWARE</i> DO ALGORITMO <i>MP-SMO</i>	87
REFERÊNCIAS	90
ANEXO – DESENVOLVIMENTO MATEMÁTICO DA <i>SVM</i>	95
A.1 INTRODUÇÃO	95
A.2 CASO LINEAR E SEPARÁVEL	95
A.2.1 Problema <i>SVM</i>	95
A.2.2 Multiplicadores de Lagrange	98
A.2.3 Problemas Primal e Dual.....	99
A.2.4 Aplicação dos Problemas Primal e Dual à <i>SVM</i>	101
A.3 CASO LINEAR E NÃO SEPARÁVEL	105
A.4 CASO NÃO LINEAR E NÃO SEPARÁVEL.....	109

1 INTRODUÇÃO

1.1 CONTEXTO

Máquinas de aprendizagem, como Redes Neurais Artificiais (*ANNs*), Redes Bayesianas (1), Máquinas de Vetores de Suporte (do inglês, ***Support Vector Machines - SVMs***)¹ (2) e outras, são aplicadas na classificação de padrões², por sua vez usada no reconhecimento de imagens, seleção de genes, classificação de textos e outras aplicações (3, 4). As mais conhecidas são as *ANNs* que emulam sistemas biológicos. Não obstante, *SVM* é uma máquina de aprendizagem não fundamentada biologicamente, mas que tem gerado muito interesse devido aos bons resultados obtidos em problemas reais (5).

Para alguns **classificadores de padrões**, é inapropriado o uso de modelos de sistemas obtidos com base no conhecimento de leis. Isto é devido à variabilidade dos padrões e a necessidade de adaptabilidade dos classificadores. Por exemplo, no reconhecimento de objetos em imagens, aquele a ser detectado pode apresentar grande variabilidade, como orientação, iluminação, oclusão e diferentes planos de fundo (6). Na detecção de objetos por sonar, o problema de variabilidade acontece devido a diferentes orientações, oclusão parcial e eco espúrio por sedimento (7). No caso da classificação de textos, é necessária a adaptabilidade do classificador à definição dinâmica de diferentes categorias (8). Outra característica dos problemas de classificação que dificulta a obtenção de um modelo é a importante dimensão do espaço das variáveis de entrada. Não obstante, bons resultados são obtidos com classificadores que utilizam máquinas de aprendizagem. Estas máquinas possuem duas fases. A primeira é chamada de treinamento ou de aprendizagem, na qual o modelo do sistema é obtido através de um conjunto de exemplos de treinamento. Para máquinas com aprendizagem supervisionada, este conjunto é formado por entradas e saídas conhecidas, por sua vez, em máquinas com aprendizagem não supervisionada somente por entradas (1). A outra fase é posterior à aprendizagem e denominada de teste.

¹ Usaremos a sigla *SVM* ao longo deste texto para designar a Máquina de Vetores de Suporte.

² *SVM* pode também ser usada em problemas de regressão não linear.

SVM (2) é uma máquina de aprendizagem do tipo supervisionada, desenvolvida por Vladimir Vapnik e seus colegas entre 1964 e 1995 **(9)**. Uma característica da SVM é que, ao ser ensaiada com padrões de teste, apresenta, na maioria dos casos índice de erro (*generalization performance*) menor ou igual ao de outros classificadores **(5)**. Do ponto de vista da teoria de aprendizado estatístico, SVM se aproxima da solução do método *structural risk minimization* **(1)**, o qual determina em uma classe de máquinas de aprendizagem, aquela que minimiza o limite máximo do índice de erro **(5)**. Também, em SVMs são obtidas soluções únicas, diferentemente de algumas ANNs, nas quais a solução pode depender de condições iniciais **(10)**. Essas propriedades subjazem na importante quantidade de aplicações que usam SVM para resolver problemas de classificação em diversas áreas **(10)**.

SVM foi originalmente desenvolvida para problemas de duas classes, posteriormente surgiram técnicas para ampliar sua aplicação a problemas de múltiplas classes³ **(11)** e para problemas de regressão não linear **(1)**.

O treinamento de uma SVM é executado usando um conjunto de T exemplos. Para problemas de duas classes, cada exemplo é formado por uma entrada representada por um vetor $\vec{x}_i$ de dimensão d e uma saída y_i pertencente ao conjunto $\{-1, +1\}$. O objetivo é determinar um hiperplano de dimensão $n-1$ que separe os exemplos positivos dos exemplos negativos. Consequentemente, a fase de treinamento de uma SVM é formulada como um problema matemático de otimização **(5)**. Ou seja, achar o mínimo ou máximo de uma função, chamada função objetivo, com os valores das variáveis limitados por restrições. Devido ao tipo de função objetivo, o problema de otimização no caso SVM é do tipo Programação Quadrática (QP) **(5)**. Este problema é transformado usando a teoria dos Multiplicadores de Lagrange e a solução é obtida quando achado o conjunto ótimo de coeficientes. Três casos podem ser identificados. O mais simples é conhecido como linear e separável, no qual n é igual a d . Outro caso é o linear ($n=d$) combinado com a não separabilidade dos exemplos. No caso geral, chamado não linear e não separável, as entradas dos exemplos são mapeadas em outro hiperespaço de dimensão $n>d$, também chamado **espaço de características (5)**.

³ Neste trabalho interessa o caso de classificação em duas classes.

Nos trabalhos de Mangasarian e Musicant (12) e de Joachims (13) foram implementados em *software* algoritmos específicos para o treinamento da SVM com alta velocidade de execução para o caso linear.

O algoritmo *Sequential Minimal Optimization* (SMO) (14) é reconhecido como um dos mais rápidos e o mais fácil para implementar em *software* para o caso geral de SVM, o não linear e não separável. O algoritmo é iterativo e adota uma **solução analítica** para a otimização de um par de Multiplicadores de Lagrange em cada iteração, evitando o armazenamento de matrizes de grandes dimensões em memória. SMO executa três tarefas por iteração: 1) seleção de um par de coeficientes (t_1), 2) otimização do par de coeficientes selecionados (t_2) e 3) atualização de dados globais (t_3). O algoritmo executa um total de I_t iterações até que todos os coeficientes satisfaçam um conjunto de condições denominadas de Karush-Kuhn-Tucker (KKT) (14, 15). O tempo total de execução (T_{SMO}) pode ser calculado segundo a igualdade (1.1), sendo $t_1+t_2+t_3$ o tempo médio de execução de cada iteração.

$$T_{SMO} = I_t(t_1 + t_2 + t_3) \quad (1.1)$$

Devido a sua importância, muitos autores têm realizado otimizações ao algoritmo SMO para reduzir o seu tempo de execução. Em implementações em *software*, a técnica de *shrinking* melhora t_1 e t_3 e *caching* melhora t_3 . *Shrinking* reduz o tamanho do problema QP e em *caching*, dados calculados são armazenados em memória para reuso. Uma maneira de reduzir o número de iterações I_t é melhorando a seleção dos pares a serem otimizados. Um exemplo é a heurística desenvolvida por Keerthi et al (16). Outras heurísticas reduzem ainda mais o número de iterações I_t à custa de maior complexidade computacional (17). O paralelismo em *software* também tem sido aplicado para acelerar a seleção do par de coeficientes a otimizar, diminuindo t_1 , e na atualização de dados globais, diminuindo t_3 (18).

As primeiras implementações dos algoritmos de treinamento de uma SVM foram realizadas em *software*, ao menos para verificar o seu funcionamento. Implementações em *hardware* existem de dois tipos, aquelas que realizam somente a fase de teste (6, 7, 19) e as que também realizam a fase de treinamento. Este trabalho trata somente do treinamento devido aos seguintes motivos: 1) o

treinamento é o principal foco nos problemas de máquinas com aprendizagem. 2) a complexidade computacional do treinamento é maior que a do teste. O teste consiste na avaliação de uma função. No treinamento é realizada a avaliação de uma função igual de complexa que a do teste e também é necessário otimizar coeficientes, e dependendo do algoritmo também selecioná-los. 3) existe um grande número de soluções eficientes para realizar o teste em *hardware*, enquanto que constata-se um número pequeno de soluções em *hardware* do treinamento.

Em ***hardware dedicado***, se destacam três implementações importantes da aprendizagem de uma SVM (10, 20, 21). Um desses trabalhos é uma implementação em ***software-hardware***, sendo a única conhecida que usa o algoritmo SMO. Nessa implementação, apresentada por Pedersen e Schoeberl, com a partição *hardware* foram realizadas operações aritméticas de multiplicação e acumulação, a parte do algoritmo com maior tempo de execução, usando uma arquitetura do tipo ***pipeline***. A velocidade de execução de treinamento melhorou em até 60%, quando comparada com uma implementação em puramente *software* (21). Não obstante, devido a que foi implementado o caso linear, foi obtida uma alta taxa de erro na fase de teste. Uma característica de todas essas implementações em *hardware* dedicado é que o número de exemplos de treinamento usado, dezenas ou centenas, é muito menor ao das implementações *software*, de milhares até centenas de milhares.

1.2 MOTIVAÇÕES

1.2.1 Motivações para aumentar a velocidade de execução do algoritmo SMO implementado em *software*

Mesmo sendo um dos algoritmos mais rápidos para o treinamento da SVM, devido a sua complexidade computacional, o tempo de execução do algoritmo SMO pode ser importante. Isso pode ser devido a um tempo de execução prolongado de cada iteração ($t_1+t_2+t_3$), e/ou a um grande número de iterações (I_t). Um exemplo é o treinamento realizado neste trabalho para o *benchmark Abalone* do repositório da

Universidade da Califórnia em Irvine (UCI) (22), o qual é composto por 4177 exemplos. Usando o algoritmo *SMO*, o treinamento desse *benchmark* foi realizado em 1122 s no ambiente de programação *OCTAVE*. Por esse motivo, foram realizadas otimizações ao algoritmo, como a introdução das técnicas *shrinking* e *caching* e das heurísticas para a seleção dos pares de coeficientes. Neste trabalho é apresentada uma modificação que complementa as anteriores, que consta da otimização de múltiplos pares de coeficientes por iteração. A simulação do algoritmo modificado em *software* serve também para prever resultados da implementação em *hardware*, ao menos de forma qualitativa.

1.2.2 Motivações para a implementação e aceleração do algoritmo *SMO* em *hardware*

Para certas aplicações que requerem um classificador, a **mobilidade, portabilidade** ou as **restrições nas dimensões físicas** do equipamento impõem que seja implementado em *hardware* embutido. Em alguns casos, o treinamento pode ser realizado *off line* em um computador de propósito geral e a fase de teste implementada no *hardware* embutido (23). Não obstante, para algumas aplicações, os requerimentos anteriores podem ser estendidos para a fase de treinamento devido à necessidade de adaptabilidade.

No campo da robótica, há uma quantidade importante de exemplos em que robôs se adaptam de forma autônoma a mudanças no ambiente de operação. Essa adaptação implica a necessidade de re-treinamento do robô, inclusive enquanto opera. Dois exemplos, entre outros, são: 1) Uma mão robótica que segura objetos (24), que pode ser aplicada na ajuda a pessoas com impedimentos físicos. A mão robótica deve aplicar uma força tal ao objeto, de forma que não provoque o deslizamento, a queda ou o dano do mesmo. No ambiente de operação, novos objetos podem surgir em relação àqueles para os quais foi treinada, exigindo adaptação. 2) Um robô móvel que evita obstáculos (25). Este tipo de veículo pode ser usado em aplicações industriais para transportar objetos. Este robô deve reconhecer novos tipos de obstáculos, com diferentes cores, formas e texturas que podem surgir no ambiente de operação. Para os dois casos, é mostrado que os

melhores resultados são obtidos com estratégias híbridas de aprendizagem (24, 25). Essas, por sua vez, são a mistura do aprendizado supervisionado, e o aprendizado por reforço. O primeiro é realizado através de exemplos, formados por entradas e saídas, dados por um usuário. Por sua vez, o aprendizado por reforço permite a adaptação, sendo baseado em uma ação e posterior avaliação do resultado que ela produz (24). No trabalho de Xue-ning Wang et al. (26), é apresentado um aprendizado híbrido de SVM e aprendizado por reforço. Nele, uma sequência de treinamento de SVM e a obtenção de um novo conjunto de treinamento através do aprendizado por reforço é realizada de forma iterativa.

O reconhecimento de escrita a mão é usado em dispositivos portáteis como as *personal digital assistants* (PDAs). Para este tipo de aplicação é usado um classificador que deve se adaptar ao estilo de escrita do usuário (27). No trabalho de McInerney e Reilly é mencionado que se ao menos uma parte do sistema de reconhecimento de escrita a mão é implementada em *hardware*, são liberados recursos da PDA para outras aplicações (28).

Aplicações móveis e portáteis em geral apresentam restrições de **energia** devido ao uso de bateria de alimentação. As portáteis principalmente apresentam também restrições de **área** e aquelas que devem executar tarefas em tempo real apresentam restrições de **tempo de execução**. Na comparação entre implementações do treinamento de uma SVM em *hardware* dedicado e microprocessadores de uso geral, o *hardware* dedicado é apontado como mais conveniente para sistemas com restrições de área, energia e tempo total de execução de treinamento⁴ (10). Na implementação de Pedersen e Schoeberl é demonstrada a conveniência em relação ao tempo de treinamento (21). Nela uma parte do algoritmo SMO é executada em *hardware* dedicado. Dessa maneira a velocidade de execução do treinamento foi aumentada em comparação com a execução completa em *software* em um microprocessador de uso geral, usando a mesma tecnologia (21), nesse caso, um dispositivo *Field Programmable Gate Array* (FPGA). No entanto, segundo nosso conhecimento, uma implementação completa do algoritmo SMO em *hardware* dedicado não foi realizada até este trabalho. Dados os argumentos anteriores, essa implementação melhoraria o desempenho quando comparada com a execução em *software* ou em *software-hardware*.

⁴ Será usada a denominação desempenho da implementação para fazer referência ao tempo de execução de treinamento, área e consumo de energia.

1.3 OBJETIVOS

Este trabalho apresenta dois objetivos gerais:

- 1) reduzir o tempo de treinamento da *SVM* em relação ao obtido com o algoritmo *SMO* e suas otimizações reportadas na literatura (para sua execução sequencial em *software*). Para isso, desenvolver uma modificação do algoritmo *SMO*, que chamamos **SMO de Múltiplos Pares (MP-SMO)**.
- 2) realizar uma implementação em *hardware* dedicado do algoritmo *MP-SMO* para a fase de treinamento da *SVM*, a fim de melhorar o tempo de treinamento da única implementação em *hardware-software* do algoritmo *SMO*. Realizar a implementação em dispositivo reconfigurável *FPGA* incluindo *pipelining* e paralelismo à arquitetura.

Para realizar os objetivos gerais, os seguintes objetivos específicos deverão ser atingidos:

- 1) realizar modificações do algoritmo *SMO* da seguinte maneira:
 - 1.1) selecionar N (dois ou mais) pares de coeficientes a cada iteração adaptando heurísticas existentes. Determinar para cada um desses pares, a partir do segundo, se pode ser otimizado a fim de garantir a convergência do algoritmo. Dado que a seleção de múltiplos pares implica maior complexidade computacional, o tempo da tarefa de seleção, t_1 , aumentará.
 - 1.2) otimizar M pares de coeficientes, tal que $1 \leq M \leq N$, mantendo a solução analítica do algoritmo *SMO*. Realizar a otimização de cada par independentemente dos outros. Na execução sequencial do algoritmo, o tempo da tarefa otimização t_2 aumentará. Não obstante, a otimização de múltiplos pares poderá reduzir o tempo de total do treinamento devido a uma maior diminuição do número de iterações I_t .

- 1.3) atualizar os dados globais, acrescentando a informação decorrente da otimização de múltiplos pares. Essa tarefa aumentará em complexidade, aumentando o tempo t_3 .
- 2) acrescentar na implementação em *hardware* as técnicas paralelismo e *pipelining* para diminuir o tempo de execução de cada iteração ($t_1 + t_2 + t_3$) da seguinte maneira:
 - 2.1) dado que as otimizações dos múltiplos pares são independentes entre si, podem ser executadas em paralelo em tempo t_2 aproximado ao da otimização de um só par de coeficientes.
 - 2.2) a seleção dos pares a otimizar e a atualização de dados gerais pode ser implementada usando *pipelining*, como no trabalho realizado por Pedersen e Schoeberl (21) para a atualização de dados, e adicionando paralelismo. Dessa maneira os tempos t_2 e t_3 são reduzidos.
- 3) para verificar o funcionamento da implementação em *software* do algoritmo *MP-SMO*, estudá-lo para sete casos selecionados do repositório da *UCI*. Esses casos deverão apresentar diferentes quantidades T de exemplos e diferentes quantidades e tipos de componentes das entradas $\bar{x}_i$. Realizar a implementação em *hardware* para um caso que representa um tipo de problema de classificação, o de entradas com componentes binárias. Este caso também possui uma quantidade de exemplos de aproximadamente 1000 exemplos, superior às das implementações anteriores em *hardware*. Analisar o custo em área e potência da diminuição do tempo de execução do algoritmo.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Neste capítulo foi apresentada a introdução, incluindo o contexto, as motivações e os objetivos do trabalho. No Capítulo 2 são apresentados os trabalhos anteriores a

este sobre otimizações do algoritmo *SMO* e implementações em *hardware* do treinamento da *SVM*. No Capítulo 3, primeiro é mostrado o problema de classificação que foi implementado em *hardware* neste trabalho e é dada uma breve revisão da *SVM*. Na segunda seção é apresentado o necessário para compreender os fundamentos básicos da *SVM*. Depois é descrito o algoritmo *SMO*, incluindo as heurísticas para a seleção do conjunto de trabalho que foram usadas neste trabalho (K-heurística e F-heurística). Na primeira seção do Capítulo 4 é introduzida nossa modificação do algoritmo *SMO*, que chamamos algoritmo *MP-SMO*. Da segunda até a quarta seção as tarefas executadas em cada iteração do novo algoritmo são descritas. Nas seções seguintes são demonstradas a convergência monotônica do algoritmo e a conformidade das restrições da *SVM*. O quinto capítulo apresenta as simplificações realizadas ao algoritmo *MP-SMO* para sua implementação em *hardware*. Também são descritas as arquiteturas para realizar as diferentes tarefas que compõem o algoritmo. Por último são mostradas as diferentes arquiteturas do algoritmo completo que foram implementadas. O Capítulo 6 que trata dos resultados experimentais é dividido em duas partes. Na primeira, são dados e analisados os resultados da implementação em *software* do algoritmo *MP-SMO*. Na segunda parte são mostrados os resultados dos testes em *FPGA*. É analisado o tempo de treinamento das diferentes arquiteturas e as implicações em área e potência consumida. No sétimo e último capítulo são dadas as conclusões do trabalho e as sugestões referentes a trabalhos futuros, tanto para o algoritmo *MP-SMO*, como para sua implementação em *hardware*. Esta dissertação possui também um ANEXO, no qual é apresentado o desenvolvimento matemático que resulta no problema *QP* da *SVM*. O ANEXO é também uma compilação de trabalhos anteriores sobre *SVM* e Otimização Matemática. Pretende-se dessa maneira, apresentar o desenvolvimento matemático da *SVM* com maior detalhe que em cada um desses trabalhos anteriores. Se o leitor não estiver interessado em aprofundar no conhecimento de *SVM* esse ANEXO pode ser obviado.

2 TRABALHOS RELACIONADOS

2.1 ALGORITMOS DE TREINAMENTO ANTERIORES AO *SMO*

Considerando que o treinamento da *SVM* consiste em resolver um problema *QP*, poderia se pensar em usar soluções em *software* de uso geral para problemas desse tipo. No entanto, esses programas para computador, tais como *MINOS* (29) e *LOQO* (30), são inadequados devido a: 1) armazenamento de grandes matrizes, de tamanho proporcional a T^2 . 2) tempo de treinamento muito grande devido à convergência lenta (14, 31). Para esse tipo de solução, a matriz a ser armazenada pode ocupar, por exemplo, 128 MB para um conjunto de treinamento de 4000 exemplos (14). Nos anos 90 esse tamanho de memória representava uma quantidade importante ou superior à disponível em computadores de uso geral. Hoje, para *hardware* dedicado pode representar uma quantidade acima da disponível. Em muitos casos a quantidade de exemplos é ainda superior, podendo ocupar uma quantidade de memória muito importante, inclusive para o treinamento executado em *software*. Por exemplo, o problema de classificação de texto *Reuters 21578 collection* possui 21578 exemplos (8), o problema *KDD04 Physics* possui 150000 exemplos, e o *Reuters C11*, 804414 exemplos (13).

O desenvolvimento de **algoritmos de decomposição** teve como objetivo o de resolver os problemas anteriores. Estes algoritmos são baseados na otimização de um subconjunto de exemplos de treinamento em cada iteração, chamado **conjunto de trabalho**. Consequentemente, menos recursos de *hardware* são necessários e o tempo de treinamento é melhorado. O primeiro algoritmo de decomposição foi introduzido por Vapnik e foi denominado *chunking* (9, 14). O conjunto de trabalho que este algoritmo otimiza é composto pelos Multiplicadores de Lagrange não nulos e um número W de piores coeficientes que não satisfazem as condições KKT. Osuna; Freund e Girosi demonstraram que se no conjunto de trabalho a otimizar há pelo menos um exemplo que não satisfaz as condições KKT, então o algoritmo converge (32). No mesmo trabalho, foi mostrado que se o algoritmo de

decomposição usa um conjunto de trabalho de tamanho fixo e arbitrário, o treinamento pode ser realizado para grandes conjuntos de treinamento.

Posteriormente, Joachims desenvolveu um algoritmo de decomposição com o qual diminuiu o tempo de treinamento incorporando técnicas de *caching* e *shrinking* (33). Com *caching*, a repetição de cálculos de sequências de *kernels*⁵ é evitada realizando armazenamento em memória. A técnica *shrinking* é usada durante o treinamento para reduzir o tamanho do problema *QP* removendo os Multiplicadores de Lagrange com alta probabilidade de não apresentar mudança no restante da execução do algoritmo. No mesmo trabalho Joachims apresentou o programa de computador *SVM Light*, no qual seu algoritmo é implementado. Este programa é um dos mais referenciados pela bibliografia para o treinamento da *SVM* (33).

Os algoritmos *chunking*, de Osuna e de Joachims usam métodos numéricos para a otimização dos Multiplicadores de Lagrange. O algoritmo *SMO* de Platt, por sua vez, realiza esta tarefa de forma analítica (14), otimizando um par de coeficientes por iteração.

2.2 OTIMIZAÇÕES DO ALGORITMO *SMO*

Keerthi et al. (16) perceberam que a principal ineficiência do algoritmo *SMO* era devida ao uso de um parâmetro b chamado limiar⁶ na seleção do conjunto de trabalho de cada iteração. Eles propuseram uma heurística, que aqui chamamos de **K-heurística**, diferente da do algoritmo *SMO* original que reduz o número total de iterações I_t do algoritmo. Essa heurística consiste em selecionar o **máximo par transgressor** das condições KKT. A velocidade de treinamento foi aumentada para a maioria dos *benchmarks* testados quando comparada com o algoritmo original.

Fan; Chen e Lin (17) propuseram outra forma de selecionar o conjunto de trabalho, que aqui denominamos **F-heurística**, com o objetivo de obter uma redução maior do número de iterações que com a K-heurística. Efeitos de segunda ordem são considerados durante a minimização da função objetivo não linear (quadrática). O

⁵ Funções *kernel* são usadas no caso não linear e representam o produto interno de dois vetores no espaço de características.

⁶ A definição do parâmetro limiar b será dada no Capítulo 3.

primeiro coeficiente do conjunto de trabalho é obtido da mesma maneira que na K-heurística e o segundo é selecionado produzindo a redução máxima da função objetivo. Pelo fato da seleção do máximo par transgressor na K-heurística ser relacionada à aproximação linear da função objetivo, a redução obtida é menor que com a F-heurística em cada iteração. Apesar do incremento computacional, o tempo de treinamento total com a F-heurística é reduzido para a maioria dos *benchmarks* testados. A F-heurística combinada com *shrinking* e *caching* foi incluída nas últimas versões de outro dos programas de computador para treinamento da SVM mais referenciados pela bibliografia especializada, o *LIBSVM* (34).

Glasmachers e Igel (35) também propuseram uma heurística que considera efeitos de segunda ordem da função objetivo. Com essa heurística, o primeiro coeficiente é selecionado entre aqueles pertencentes aos conjuntos de trabalho anteriores. Portanto, é garantido que a metade dos dados calculados por iteração está antecipadamente armazenada em memória. O número de iterações em comparação com *LIBSVM* é maior para todos os *benchmarks*. No entanto, devido à redução de cálculos, este método apresenta maior velocidade de treinamento quando comparado com *LIBSVM* para alguns casos.

Lian Zhou; Fen Xia e Yanwu Yang apresentaram um resumo das diferentes estratégias de *shrinking* e *caching* e propuseram um novo método de *caching* (36). Nele a probabilidade de uso futuro dos dados é considerada, decidindo se é útil ou não armazená-los em memória. Foram reportados melhores resultados de um parâmetro chamado “taxa de avaliação de *kernel*”, considerada proporcional ao tempo de treinamento, quando comparados com os dos métodos anteriores de *caching*. Não obstante, os cálculos realizados para a decisão de armazenar ou não os dados poderiam aumentar o tempo total de treinamento.

2.3 CONVERGÊNCIA MONOTÔNICA DO ALGORITMO SMO

Lin (37) demonstrou a convergência monotônica do algoritmo SMO usando a K-heurística. Pode ser demonstrado que este resultado é também válido para as outras heurísticas mencionadas anteriormente. Lin menciona demonstrações anteriores,

que são obtidas impondo certas condições no conjunto de treinamento. Sua demonstração é realizada sem a necessidade dessas condições.

2.4 UM *KERNEL* *HARDWARE* AMIGÁVEL

Anguita et al. (38) apresentaram uma função *kernel* e sua arquitetura de *hardware* que usa somente operações soma e deslocamento. Esse trabalho foi motivado pela disponibilidade limitada de recursos nas implementações em *hardware* da fase de teste das *SVMs* e a complexidade nos cálculos que apresentam as funções *kernels*. Essa arquitetura implementa um algoritmo do tipo *Coordinate Rotation Digital Computer (CORDIC)* em ponto fixo, evitando o uso de multiplicadores que requerem quantidades importantes de recursos. Dado que no treinamento da *SVM* são realizados cálculos da função *kernel*, esta arquitetura pode também ser adequada para sua implementação em *hardware*.

2.5 IMPLEMENTAÇÕES EM *HARDWARE* DO TREINAMENTO DA *SVM*

Para o treinamento da *SVM* em *hardware*, Anguita; Boni e Ridella implementaram uma arquitetura em *FPGA* (10). O problema de otimização é resolvido através de um algoritmo baseado em uma rede recorrente, considerado ***hardware amigável*** por sua simplicidade. Nele, um sistema dinâmico é gerado e descrito através de uma equação diferencial que é discretizada. O algoritmo é similar ao da primeira arquitetura *hardware* proposta para o treinamento de *SVM* por Anguita; Ridella e Rovetta (39). A implementação possui dois blocos principais de acordo com o algoritmo. Em cada iteração, primeiro é executado um bloco que resolve o problema *SVM* para um valor do parâmetro limiar b fixo. Depois, outro bloco recalcula o parâmetro b . Com o objetivo de obter alta velocidade de execução foi implementada uma arquitetura paralela. Nessa arquitetura são utilizados tantos multiplicadores incorporados no *FPGA* como a quantidade de exemplos do conjunto de treinamento.

Também foi realizada, por primeira vez, uma análise da influência do erro de quantificação. Foi realizado um estudo de pior caso e estabeleceu-se o número de bits necessários para que a quantificação não produza erros na fase de teste. O estudo teórico indicou um mínimo necessário de 10 bits para a representação dos dados em ponto fixo, porém, os resultados experimentais mostraram que 8 bits são suficientes. A implementação foi aplicada na equalização de um canal de comunicações. Dois circuitos foram realizados, um para o treinamento com 8 exemplos e outro com 32, sendo os vetores de entrada de dimensão 2. O dispositivo utilizado foi o XC2V8000 da família Virtex II da XILINX, e as porcentagens de *Look-Up Tables (LUTs)* utilizadas corresponderam a 1% e 6%. Os tempos de treinamento corresponderam a 0,4 ms e 6,6 ms. Como pode ser observado, o tempo de treinamento e a área utilizada registraram aumento maior que proporcional ao ser incrementado o número de exemplos. A importante velocidade de treinamento obtida com o paralelismo é a expensas de que o algoritmo usado requer o armazenamento de uma matriz de dimensão igual ao quadrado do número de exemplos de treinamento (T^2), demandando importante quantidade de área quando o número de exemplos aumenta. Além disso, a arquitetura descrita não realiza cálculos da função *kernel*, mas considera que são dados calculados anteriormente.

Sheng Yu Peng; Minch e Hasler (20) propuseram uma arquitetura VLSI analógica para treinar uma SVM não linear e não separável usando uma rede neuronal chamada *projection neural network*, ou rede neuronal de projeção. Chamada assim devido à solução do problema QP ser obtida por meio de um conjunto de equações diferenciais que contêm um operador de projeção. Uma função *kernel* não linear do tipo *Radial Basis Function (RBF)*⁷ é calculada na forma de uma corrente de saída de um circuito do tipo *floating gate bump*. A eficiência deste circuito para o cálculo da função *kernel* é estimada em duas ordens de magnitude superior à de um microprocessador. A computação do algoritmo é completada com um filtro passa baixas, um integrador e um circuito baseado em um espelho de corrente. A arquitetura completa do circuito que realiza o treinamento é paralela para obter alta velocidade de execução. Dado que o processamento é analógico, pode ser integrada a interfaces de sensores sem usar conversores analógico-digitais. O circuito foi simulado em nível de transistor para um conjunto de treinamento artificial

⁷ A função *kernel RBF* é uma função exponencial e será definida na seção 3.2.

formado por 4 exemplos de dimensão 2. O tempo de treinamento obtido foi de 0,4 ms. Não são apresentados dados de área, não obstante, nesse trabalho, o circuito é designado eficiente enquanto à área e à potência, dado que os blocos não usam resistores. Apesar disso, o número de blocos de *hardware* utilizados nessa arquitetura aumenta de forma quadrática com o número de exemplos (T^2).

Pedersen e Schoeberl (21) apresentaram a única implementação em *software* e *hardware* conhecida do algoritmo SMO, e mostraram que o tempo de treinamento é melhorado quando comparado com uma implementação em somente *software*. Na partição *hardware* implementou-se a função *kernel*, sendo uma parte da atualização de dados, a tarefa de maior tempo de execução. O caso considerado foi o linear. O *hardware* consiste em uma unidade do tipo multiplicador e acumulador (MAC), sendo usada uma arquitetura *pipeline* de 16 estágios. Os dados de entrada e saída da MAC foram representados em ponto fixo de 16.16 bits. Internamente usou-se uma representação de 32.32 bits, e o *kernel* é calculado em 244 ciclos de relógio. O resto do algoritmo foi executado em *software* em um Processador Java Otimizado (JOP) (21). Todo o algoritmo foi implementado em *FPGA* no dispositivo EP1C6 da família *Cyclone* da *Altera*. Como resultado da síntese da MAC, a frequência máxima de relógio é de 103 MHz, no entanto, a frequência usada foi de 100 MHz. O treinamento foi realizado usando um conjunto artificial formado por 60 exemplos de dimensão 2. A velocidade de execução foi melhorada em até 1,6 vezes em relação à implementação em puramente *software* no processador JOP em *FPGA*. O número de elementos lógicos totais usados foi de 7184. Diminuindo a memória cachê do processador, o número de elementos lógicos foi reduzido a 6438 a expensas de uma pequena redução da velocidade, sendo até 1,57 vezes maior em relação à implementação em puramente *software*. A leitura dos dados é apontada como dominante no tempo para o cálculo do *kernel*, e os autores propõem que para futuros trabalhos seja realizada diretamente pelo *hardware* específico. Foram realizados também testes para obter o número de bits necessários para que o algoritmo convirja e não aumente de forma importante o erro de teste. Para alguns testes, usando 12.12 bits o algoritmo não convergiu. Para três dos quatro conjuntos de teste, mesmo usando 4.4 bits o erro não aumentou de forma importante, mas em um deles, para 8.8 e 4.4 bits o erro foi mais de 5 vezes maior em um caso e 3 vezes maior no outro. Devido ao uso de *kernel* linear, foram reportados erros de até 58 % na fase de teste para conjuntos não separáveis. Finalmente, os resultados

experimentais do trabalho de Pedersen e Schoeberl (21), mostram que o tempo de treinamento de um conjunto composto por 60 exemplos é aproximadamente 2000 vezes maior que o de realizar um teste.

2.6 RESUMO DO CAPÍTULO

Neste capítulo foram apresentados os principais trabalhos anteriores que otimizaram o algoritmo *SMO* para o treinamento da *SVM*. Entre eles, os que propuseram novas heurísticas para a seleção do par de coeficientes a otimizar em cada iteração. Também os que incorporaram as técnicas *caching* e *shrinking*. Foi também apresentado o trabalho no qual é demonstrada a convergência monotônica do algoritmo *SMO*. Esta demonstração será estendida para o algoritmo *MP-SMO*. Foram também apresentados os trabalhos relacionados com a *SVM* em *hardware*. Em particular um *kernel* implementado em *hardware* que somente usa operações soma e deslocamento. Também foram mostrados os três trabalhos mais importantes sobre o treinamento de *SVM* em *hardware*. Dois deles usaram paralelismo para obter alta velocidade de execução e o terceiro é a única implementação em *software-hardware* do algoritmo *SMO* e usa uma arquitetura *pipeline*. Não obstante, nenhum deles implementa completamente o algoritmo *SMO* e o número de exemplos é no máximo de dezenas. Neste trabalho serão implementados os algoritmos *SMO* e *MP-SMO* completamente em *hardware*, para a execução sequencial e com combinações das técnicas *pipelining* e paralelismo. O número de exemplos do conjunto de treinamento será superior ao das implementações anteriores em pelo menos uma ordem de magnitude.

3 FUNDAMENTOS TEÓRICOS

3.1 UM PROBLEMA DE CLASSIFICAÇÃO

Uma *SVM* é treinada com um conjunto de exemplos formados por entradas e suas correspondentes saídas. As componentes dos vetores de entrada são também denominadas de **atributos**. Um exemplo de problema de classificação é o *benchmark Tic-tac-toe endgame* (jogo da velha) que é parte do repositório da *UCI* (22). Os atributos das entradas deste *benchmark* podem assumir um número finito de valores, e por esse motivo são denominados de **atributos categóricos**. No jogo, existem 9 posições que podem ter 3 valores possíveis cada uma, como mostrado na Figura 3.1 (40).

Figura 3.1 - Representação de um exemplo do *benchmark Tic-tac-toe endgame*

O objetivo do classificador é determinar se existem três círculos ou três cruzeiros em linha ou não. Quando usada a **codificação 1-of-N** para cada valor possível, cada posição é representada por 3 bits da seguinte maneira:

X → **100**
O → **010**
(vazia) → **001**.

Dessa maneira a entrada para o exemplo da Figura 1 é:

$$\vec{x} = (0,1,0,1,0,0,1,0,0,0,0,1,0,1,0,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0).$$

Portanto as entradas do classificador *Tic-tac-toe endgame* são representadas por vetores $\vec{x}_i$ de 27 componentes binárias ($\vec{x}_i \in B^{27}$). A saída y_i do *Tic-tac-toe endgame* indica se há 3 em linha ou não e pode ser representada da seguinte maneira:

$$\text{há 3 em linha} \rightarrow y_i = 1$$

$$\text{não há 3 em linha} \rightarrow y_i = -1.$$

3.2 MÁQUINA DE VETORES DE SUPORTE

No caso geral de problema de classificação de duas classes, os vetores de entradas possuem d atributos reais ($\vec{x}_i \in R^d$) e as saídas são representadas como no exemplo anterior ($y_i \in \{-1, 1\}$). O *benchmark Breast cancer* do repositório da UCI é um problema de classificação cujas entradas possuem atributos reais. As entradas desse problema são os dados obtidos da análise de células de um tumor. Por exemplo, a longitude do radio da célula, a concavidade do contorno, a textura dada pelo desvio padrão da escala de cinzas, etc (22). O caso de atributos binários pode ser incluso no caso geral de atributos reais, dado que nada impede que os mesmos possuam valores somente 0 ou 1.

O caso linear e separável de SVM consiste em achar um hiperplano de dimensão $d-1$, H_{d-1} , dado pela igualdade (3.1), que separe as entradas de exemplos positivos dos negativos.

$$\vec{w} \cdot \vec{x} + b = 0 \quad (3.1)$$

Sendo $\vec{w}$ um vetor ortogonal a H_{d-1} e b o parâmetro chamado limiar. Como existem infinitos hiperplanos de separação, o objetivo é achar aquele que maximiza o parâmetro **margem** e é equidistante das duas classes, sendo este hiperplano único

e chamado ótimo ou de decisão **(2, 5)**. A margem é definida como a soma das distâncias mínimas de cada classe ao hiperplano ótimo.

Em muitos casos depois de obter o hiperplano de decisão surgem erros, o que significa que alguns exemplos se alocam no lado errado em relação à separação. Para minimizar o número desses erros, o hiperplano é substituído por uma hipersuperfície não linear. Este caso é conhecido como não linear e não separável. Um hiperespaço de Hilbert é uma generalização dos hiperespaços Euclidianos, no qual não é necessário que o número de dimensões seja finito **(5)**. Uma hipersuperfície não linear de separação pode ser achada no hiperespaço de dimensão d mapeando os exemplos de entrada em outro hiperespaço de Hilbert de dimensão n ($n > d$). Nesse novo hiperespaço as entradas são separadas por um hiperplano H_{n-1} de dimensão $n-1$ **(5)**. Na Figura 3.2 é mostrada uma representação em duas dimensões do espaço das entradas transformadas, a margem, os erros e os vetores mais próximos ao hiperplano de separação, chamados **vetores de suporte**.

Figura 3.2 – Representação do espaço das entradas transformadas

A transformação do hiperespaço das entradas é realizada através de uma função de mapeamento $\vec{\phi}$ dada pela expressão (3.2).

$$\vec{\phi} : R^d \rightarrow R^n \quad (3.2)$$

Para obter o hiperplano ótimo, é preciso achar o máximo de uma função objetivo quadrática de múltiplas variáveis com restrições de desigualdade. Este é um problema do tipo *QP* **(5, 14)**. Introduzindo a Teoria de Multiplicadores de Lagrange, o problema *QP* original é substituído por outro problema *QP* equivalente e mais simples. Esse problema consiste em achar um conjunto de coeficientes de Lagrange do mesmo tamanho que o do conjunto de exemplos. Ele é formulado através da expressão (3.3) e sujeito às restrições dadas pelas expressões (3.4) e (3.5). No Anexo é mostrado o desenvolvimento matemático para obter esta formulação do problema *QP* da *SVM*.

$$\min_{\alpha_i} \{L = 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \vec{\phi}(\vec{x}_i) \vec{\phi}(\vec{x}_j) - \sum_{i=1..T} \alpha_i\} \quad (3.3)$$

$$\sum_{i=1..T} \alpha_i y_i = 0 \quad (3.4)$$

$$0 \leq \alpha_i \leq C; i = 1..T \quad (3.5)$$

Sendo L a função objetivo, α_i os Multiplicadores de Lagrange e C um parâmetro definido pelo usuário que representa o compromisso entre a margem e o número de erros no treinamento **(14)**. Esses dois últimos valores são importantes para ter um erro pequeno na fase de teste **(5)**. O problema pode ser simplificado se a função $\vec{\phi}$ é substituída na expressão (3.3) pela função *kernel* definida em (3.6) **(2, 5)**.

$$K(\vec{x}_i, \vec{x}_j) = \vec{\phi}(\vec{x}_i) \cdot \vec{\phi}(\vec{x}_j) \quad (3.6)$$

Algumas funções *kernels* que têm dado bons resultados de *generalization performance* (índice de erro na fase de teste), quando usadas em classificadores são: a função *kernel* polinomial (expressão (3.7)), a função de base radial gaussiana (*RBF*, expressão (3.8)) e a função tangente hiperbólica (expressão (3.9)) **(5)**.

$$K_1(\bar{x}_i, \bar{x}_j) = (\bar{x}_i \cdot \bar{x}_j + 1)^p \quad (3.7)$$

$$K_2(\bar{x}_i, \bar{x}_j) = e^{-\|\bar{x}_i - \bar{x}_j\|^2 / (2\sigma^2)} \quad (3.8)$$

$$K_3(\bar{x}_i, \bar{x}_j) = \tanh(k\bar{x}_i \cdot \bar{x}_j - \delta) \quad (3.9)$$

Os parâmetros p , σ , δ e k nas expressões das funções *kernel* anteriores são dados pelo usuário. O caso linear (hiperplano H_{d-1}) pode ser considerado um caso particular, no qual a função *kernel* é representada pela expressão (3.10).

$$K_4(\bar{x}_i, \bar{x}_j) = \bar{x}_i \cdot \bar{x}_j \quad (3.10)$$

Dada uma função $K(\bar{x}_i, \bar{x}_j)$ existe uma condição que determina se existe a função de mapeamento $\vec{\phi}$ e o espaço das entradas transformadas. Ou seja, se a função $K(\bar{x}_i, \bar{x}_j)$ é um *kernel* ou não. Essa condição é denominada **de Mercer (5)**, sendo necessária e suficiente. O enunciado da condição de Mercer é:

Dada uma função $K(\bar{x}_i, \bar{x}_j)$ aplicada a dois vetores $(\bar{x}_i, \bar{x}_j)$. Existe uma função $\vec{\phi}$ de mapeamento se e somente se, para toda função $g(\bar{x})$, tal que $\int g^2(\bar{x})d\bar{x}$ é finita, cumpre-se que:

$$\int K(\bar{x}_i, \bar{x}_j)g(\bar{x}_i)g(\bar{x}_j)d\bar{x}_i d\bar{x}_j \geq 0 \quad (3.11).$$

3.3 ALGORITMO SEQUENTIAL MINIMAL OPTIMIZATION

3.3.1 Introdução

O SMO é um algoritmo de decomposição que usa uma solução analítica para otimizar um par de Multiplicadores de Lagrange por iteração **(14)**. Em cada uma delas, três tarefas são executadas: 1) seleção de um par de coeficientes, 2) otimização do par de coeficientes selecionados e 3) atualização de dados globais. A parada do algoritmo ocorre quando todos os coeficientes satisfazem o conjunto de condições KKT **(14)**. O conjunto de trabalho é formado pelo par de coeficientes a otimizar.

3.3.2 Tarefa 1: seleção do conjunto de trabalho

As heurísticas mais frequentemente referenciadas para resolver esta tarefa são a K-heurística **(16, 17)** e a F-heurística **(17)**. No Algoritmo 3.1 é mostrado o pseudocódigo da K-heurística. Primeiro o conjunto de T exemplos é dividido em 2 grupos I_{up} e I_{low} de acordo com os valores dos coeficientes α_i e as saídas y_i (passo *i*). Depois são achados entre as funções $F_i = -y_i \partial L / \partial \alpha_i$ avaliadas em $(\alpha_1, \alpha_2, \dots, \alpha_T)$, o máximo em I_{up} e o mínimo em I_{low} para formar o conjunto de trabalho $(\alpha_{11}, \alpha_{12})$ (passo *ii*). Os limiares b_{up} e b_{low} são os valores das funções F_i correspondentes ao conjunto de trabalho (passo *iii*). A parada do algoritmo SMO ocorre quando b_{low} supera b_{up} menos um valor de tolerância ε dado pelo usuário, sendo essa condição equivalente a satisfazer as condições KKT. Essa tolerância ε deve ser considerada pelo fato da convergência do valor de b_{low} ao de b_{up} ser assintótica **(14)**.

Algoritmo 3.1 - Pseudocódigo da K-heurística

i. O conjunto de exemplos é dividido em dois grupos I_{up} e I_{low} .

$$1 \quad I_{up} \equiv \{i \mid \alpha_i < C, y_i = 1 \text{ or } \alpha_i > 0, y_i = -1\}$$

$$2 \quad I_{low} \equiv \{i \mid \alpha_i < C, y_i = -1 \text{ or } \alpha_i > 0, y_i = 1\}$$

ii. O conjunto de trabalho (i_{11}, i_{12}) é selecionado.

$$3 \quad i_{11} = \arg \max \{F_i = -y_i \partial L / \partial \alpha_i\}, i \in I_{up}$$

$$4 \quad i_{12} = \arg \min \{F_i = -y_i \partial L / \partial \alpha_i\}, i \in I_{low}$$

iii. Os coeficientes b_{up} e b_{low} (limiares) são determinados.

$$5 \quad b_{up} = F_{11}$$

$$6 \quad b_{low} = F_{12}$$

iv. A condição de parada do SMO é testada.

$$7 \quad \text{IF } b_{up} > b_{low} + \varepsilon$$

$$8 \quad \quad \quad \text{THEN} \quad \quad \quad \text{RETURN}(\alpha_{11}, \alpha_{12})$$

$$9 \quad \quad \quad \text{ELSE} \quad \quad \quad \text{END SMO}$$

$$10 \quad \text{END}$$

No Algoritmo 3.2 é mostrado o pseudocódigo da F-heurística. Essa heurística difere da anterior na seleção do segundo exemplo que é realizada achando o mínimo entre as funções $G_i = -(b_{up} - F_i)^2 / \eta_i$ em I_{low} (passo ii). Sendo o parâmetro η_i definido na linha 2 do Algoritmo 3.2.

Algoritmo 3.2 – Pseudocódigo da F-heurística

i. São obtidos os grupos I_{up} , I_{low} , o primeiro exemplo i_{11} e b_{up} como na K-heurística.

1 Linhas 1, 2, 3, e 5 do Algoritmo 3.1

ii. O segundo exemplo i_{12} é obtido minimizando a função G_i em I_{low} .

$$2 \quad \eta_i = K(\bar{x}_{11}, \bar{x}_{11}) + K(\bar{x}_i, \bar{x}_i) - 2K(\bar{x}_i, \bar{x}_{11})$$

$$3 \quad F_i = -y_i \partial L / \partial \alpha_i, i \in I_{low}$$

$$4 \quad i_{12} = \arg \min \{G_i = -(b_{up} - F_i)^2 / \eta_i\}, i \in I_{low}$$

iii. O limiar b_{low} é obtido minimizando F_i em I_{low} .

$$5 \quad i_{13} = \arg \min \{F_i = -y_i \partial L / \partial \alpha_i\}, i \in I_{low}$$

$$6 \quad b_{low} = F_{i_{13}}$$

iv. As últimas 3 linhas são as mesmas que a do Algoritmo 3.1.

7 Linhas 7, 8, 9 e 10 do Algoritmo 3.1

Posteriormente à seleção do conjunto de trabalho, a técnica *shrinking* pode ser aplicada para reduzir o tamanho do problema *QP*. Será considerada esta técnica como foi aplicada no *software LIBSVM (34)*. Os exemplos que satisfazem as condições (3.12) e (3.13) são removidos do conjunto de treinamento por apresentarem alta probabilidade de não serem modificados no restante da execução do algoritmo. Este método considera o fato experimental de que aqueles coeficientes que possuem os valores extremos (0 ou C) são os menos prováveis de serem modificados. Essa probabilidade diminui para os exemplos que satisfazem (3.12) e (3.13) considerando que em I_{up} é selecionado o máximo de F_i e o mínimo em I_{low} na K-heurística e um valor próximo a esse mínimo na F-heurística .

$$\{i \mid i \in I_{up}, F_i < b_{low}, \alpha_i = 0 \text{ or } \alpha_i = C\} \quad (3.12)$$

$$\{i \mid i \in I_{low}, F_i > b_{up}, \alpha_i = 0 \text{ or } \alpha_i = C\} \quad (3.13)$$

3.3.3 Tarefa 2: otimização do par de coeficientes

Para todos os algoritmos aqui estudados, considera-se que a função *kernel* **satisfaz a condição de Mercer e que o conjunto de exemplos não possui repetições**. O Algoritmo 3.3 mostra o pseudocódigo da otimização do par de coeficientes selecionados $(\alpha_{11}, \alpha_{12})$. Primeiro calcula-se os limites do valor de α_{12} , Lo e Hi . Estes limites são determinados pela aplicação das restrições dadas por (3.4) e (3.5) da SVM (passo *i*). Depois, o novo valor de α_{12} é achado, o qual depende da segunda derivada parcial η_1 da função objetivo L em relação a α_{12} (passos *ii* e *iii*). Logo, o novo valor de α_{12} é comparado com os limites Lo e Hi (passo *iv*). Por último, o novo valor do primeiro Multiplicador de Lagrange α_{11} é calculado (passo *v*). Para isso considera-se o erro numérico na computação do segundo coeficiente α_{12} . Se não for feita tal consideração o algoritmo pode não convergir **(31)**.

Algoritmo 3.3 – Pseudocódigo da otimização de um par de coeficientes

i. Cálculo dos limites de α_{12} , Lo e Hi.

```

1  If ( $y_{11} \neq y_{12}$ )
2    Lo =  $\max(0, \alpha_{12} - \alpha_{11})$ 
3    Hi =  $\min(C, C + \alpha_{12} - \alpha_{11})$ 
4  ELSE
5    Lo =  $\max(0, \alpha_{12} + \alpha_{11} - C)$ 
6    Hi =  $\min(C, \alpha_{12} + \alpha_{11})$ 
7  END

```

ii. A segunda derivada parcial da função objetivo L é calculada em relação a α_{12} .

```

8   $\eta_1 = K(\bar{x}_{11}, \bar{x}_{11}) + K(\bar{x}_{12}, \bar{x}_{12}) - 2K(\bar{x}_{12}, \bar{x}_{11})$ 

```

iii. O novo valor ótimo de α_{12} é calculado.

```

9   $\alpha_{12}^{new} = \alpha_{12}^{old} + y_{12} (b_{low} - b_{up}) / \eta_1$ 

```

iv. O valor de α_{12}^{new} é comparado com os limites Lo e Hi.

```

10 IF ( $\alpha_{12}^{new} > Hi$ )
11    $\alpha_{12}^{new} = Hi$ 
12 ELSEIF ( $\alpha_{12}^{new} < Lo$ )
13    $\alpha_{12}^{new} = Lo$ 
14 END

```

v. O novo valor do primeiro Multiplicador de Lagrange α_1 é obtido.

```

15 IF ( $\alpha_{12}^{new} = \alpha_{12}^{old} - \alpha_{11}^{old}$ ) & ( $y_{12} \neq y_{11}$ )
16    $\alpha_{11}^{new} = 0$ 
17 ELSEIF ( $\alpha_{12}^{new} = \alpha_{12}^{old} + \alpha_{11}^{old}$ ) & ( $y_{12} = y_{11}$ )
18    $\alpha_{11}^{new} = 0$ 
19 ELSEIF ( $\alpha_{12}^{new} = C - \alpha_{11}^{old} + \alpha_{12}^{old}$ ) & ( $y_{12} \neq y_{11}$ )
20    $\alpha_{11}^{new} = C$ 
21 ELSEIF ( $\alpha_{12}^{new} = \alpha_{12}^{old} - C + \alpha_{11}^{old}$ ) & ( $y_{12} = y_{11}$ )
22    $\alpha_{11}^{new} = C$ 
23 ELSE
24    $\alpha_{11}^{new} = \alpha_{11}^{old} + y_{11} y_{12} (\alpha_{12}^{old} - \alpha_{12}^{new})$ 
25 END

```

3.3.4 Tarefa 3: atualização de dados globais

Após realizada a tarefa de otimização, os valores das funções F_i são atualizados como na expressão (3.14).

$$\begin{aligned} F_i^{new} = -y_i \partial L / \partial \alpha_i = F_i^{old} + y_{11}(\alpha_{11}^{old} - \alpha_{11}^{new})K(\bar{x}_{11}, \bar{x}_i) \\ + y_{12}(\alpha_{12}^{old} - \alpha_{12}^{new})K(\bar{x}_{12}, \bar{x}_i); \quad i = 1..T \end{aligned} \quad (3.14)$$

Os *kernels* $K(\bar{x}_{11}, \bar{x}_i)$ e $K(\bar{x}_{12}, \bar{x}_i)$ devem ser calculados para a atualização de F_i . Após, a técnica *caching* pode ser aplicada. Os valores calculados dos *kernels*, são armazenados em memória para uso posterior durante a execução do algoritmo. Essa memória é de tamanho fixo e em quase todos os casos não é suficiente para armazenar todos os *kernels* calculados. Portanto, o primeiro valor armazenado é o primeiro a ser apagado quando a memória está completamente escrita.

Os valores dos Multiplicadores de Lagrange α_i e das funções F_i precisam ser iniciados. No caso dos Multiplicadores de Lagrange, esses valores iniciais devem respeitar as restrições dadas por (3.4) e (3.5). Iniciando os coeficientes α_i como no SMO original, seus valores serão iguais a zero **(14)**. Isso determina que as funções F_i sejam inicialmente iguais a y_i .

3.4 RESUMO DO CAPÍTULO

Neste capítulo, primeiro foi apresentado um exemplo de problema de classificação, o *Tic-tac-toe endgame*. Depois foi apresentada a SVM e na terceira seção do capítulo foi descrito o algoritmo SMO para seu treinamento. Nessa descrição do algoritmo SMO foram detalhadas as tarefas que são executadas em cada iteração. Em particular foram mostradas duas heurísticas (a K-heurística e a F-heurística) para a realização de uma dessas tarefas, a seleção do par de coeficientes a otimizar, as quais aceleram a execução do algoritmo.

4 ALGORITMO SEQUENTIAL MINIMAL OPTIMIZATION DE MÚLTIPLOS PARES

4.1 INTRODUÇÃO

O algoritmo *Sequential Minimal Optimization* de Múltiplos Pares (*MP-SMO*) (41) é resultado de uma modificação do algoritmo *SMO*, sendo proposto neste trabalho com o objetivo de acelerar o treinamento da *SVM*. A ideia central é a de poder otimizar dois ou mais pares de coeficientes em cada iteração, obtendo uma maior diminuição da função objetivo que com a otimização de um só par. Dessa maneira o número total de iterações também é menor.

Neste novo algoritmo as três tarefas executadas por iteração no *SMO* são modificadas da seguinte maneira:

- 1- seleção do conjunto de trabalho: seleciona N (≥ 2) pares de coeficientes.
- 2- otimização dos pares de coeficientes: otimiza M ($1 \leq M \leq N$) pares de coeficientes.
- 3- Atualização de dados globais: $2 \times M$ termos são adicionados aos valores de F_i .

4.2 TAREFA 1: SELEÇÃO DO CONJUNTO DE TRABALHO

Para selecionar o conjunto de trabalho, foram adaptadas as K- e F-heurísticas usadas no algoritmo *SMO*. No Algoritmo 4.1 é mostrado o pseudocódigo da seleção de dois pares de coeficientes usando uma adaptação da K-heurística. O primeiro par é escolhido como no algoritmo *SMO* (passo *i*). O segundo par de coeficientes é escolhido usando a K-heurística a partir do conjunto de exemplos sem o primeiro par selecionado (passos *ii* e *iii*). Depois o coeficiente *Coeff2* é calculado (passos *iv* e *v*)

para determinar se os dois pares de coeficientes serão otimizados ou somente o primeiro (passos vi e vii).

Algoritmo 4.1 – Pseudocódigo da seleção de dois pares de coeficientes com a K-heurística adaptada

-
- i. O primeiro par é selecionado usando a K-heurística (Algoritmo 3.1).
1 K – heurística $\rightarrow (i_{11}, i_{12}, \eta_1, b_{up1}, b_{low1})$
- ii. Os grupos I_{up2} e I_{low2} são gerados.
2 $I_{up2} \equiv \{i \mid \alpha_i < C, y_i = 1 \text{ or } \alpha_i > 0, y_i = -1, i \neq i_{11}, i \neq i_{12}\}$
3 $I_{low2} \equiv \{i \mid \alpha_i < C, y_i = -1 \text{ or } \alpha_i > 0, y_i = 1, i \neq i_{11}, i \neq i_{12}\}$
- iii. Seleção do segundo par de coeficientes $(\alpha_{21}, \alpha_{22})$.
4 $i_{21} = \arg \max \{F_i = -y_i \partial L / \partial \alpha_i\}, i \in I_{up2}$
5 $i_{22} = \arg \min \{F_i = -y_i \partial L / \partial \alpha_i\}, i \in I_{low2}$
- iv. Os limiares b_{up2} e b_{low2} são calculados.
6 $b_{up2} = F_{21}$
7 $b_{low2} = F_{22}$
- v. O coeficiente $coeff2$ é calculado.
8 $\eta_2 = K(\bar{x}_{21}, \bar{x}_{21}) + K(\bar{x}_{22}, \bar{x}_{22}) - 2K(\bar{x}_{22}, \bar{x}_{21})$
9 $coeff1 = (K(\bar{x}_{11}, \bar{x}_{21}) - K(\bar{x}_{11}, \bar{x}_{22}) - K(\bar{x}_{12}, \bar{x}_{21}) + K(\bar{x}_{12}, \bar{x}_{22}))$
10 $coeff2 = -(b_{up2} - b_{low2})^2 / (2\eta_2) + coeff1(b_{up1} - b_{low1})(b_{up2} - b_{low2}) / (\eta_1\eta_2)$
- vi. Se $coeff2$ é menor que zero o segundo par de coeficientes selecionados é otimizado.
11 IF ($coeff2 < 0$)
12 **THEN** **RETURN** $\{(\alpha_{11}, \alpha_{12}), (\alpha_{21}, \alpha_{22})\}$
- vii. Caso contrario somente o primeiro par de coeficientes é otimizado.
13 **ELSE** **RETURN** $(\alpha_{11}, \alpha_{12})$
14 END
-

No Algoritmo 4.2 é mostrado o pseudocódigo da seleção de dois pares de coeficientes usando uma adaptação da F-heurística. O primeiro par é escolhido como no algoritmo SMO. O primeiro coeficiente do segundo par é escolhido como na adaptação da K-heurística do Algoritmo 4.1 (passo i). Para a seleção do segundo coeficiente, as funções J_i são avaliadas, uma para cada exemplo de I_{low2} (passo ii). O

segundo coeficiente é o que corresponde ao valor J_i mínimo (passo *iii*). Finalmente, de acordo com o valor do J_i mínimo, o segundo par selecionado será otimizado ou não (passo *iv* e *v*).

Algoritmo 4.2 – Pseudocódigo da seleção de dois pares de coeficientes com a F-heurística adaptada

-
- i. Seleção do primeiro par e do primeiro coeficiente do segundo par.*
1 F – heuristic $\rightarrow (i_{11}, i_{12}, \eta_1, b_{up1}, b_{low1})$
2 Linhas 2, 3, 4 e 6 do Algoritmo 4.1
- ii. Os coeficientes J_i são calculados, um para cada exemplo de I_{low2} .*
3 FOR $i \in I_{low2}$
4 $b_{low2i} = F_i$
5 $\eta_{2i} = K(\bar{x}_{21}, \bar{x}_{21}) + K(\bar{x}_i, \bar{x}_i) - 2K(\bar{x}_i, \bar{x}_{21})$
6 $coeff1_i = (K(\bar{x}_{11}, \bar{x}_{21}) - K(\bar{x}_{11}, \bar{x}_i) - K(\bar{x}_{12}, \bar{x}_{21}) + K(\bar{x}_{12}, \bar{x}_i))$
7 $J_i = -(b_{up2} - b_{low2i})^2 / (2\eta_{2i}) + coeff1_i(b_{up1} - b_{low1})(b_{up2} - b_{low2i}) / (\eta_1\eta_{2i})$
8 END FOR
- iii. O segundo exemplo é selecionado minimizando J_i em I_{low2} .*
9 $i_{22} = \arg \min \{J_i\}, i \in I_{low2}$
- iv. Se J_{22} é menor que zero o segundo par de coeficientes é otimizado.*
10 IF $(J_{22} < 0)$
- v. As últimas linhas são as mesmas que as do Algoritmo 4.1 .*
11 Linhas 12, 13 e 14 do Algoritmo 4.1
-

Os Algoritmos 4.1 e 4.2 podem ser generalizados para a seleção de $N > 2$ pares de coeficientes.

Caso a K-heurística seja adaptada para a seleção de múltiplos pares, a seleção do k -ésimo par de coeficientes é realizada da seguinte maneira: os $k-1$ pares de Multiplicadores de Lagrange anteriores a otimizar são removidos do conjunto de exemplos. O resto do conjunto de exemplos é dividido em dois grupos I_{upk} e I_{lowk} . O primeiro exemplo α_{k1} é escolhido achando o máximo F_i em I_{upk} . O segundo exemplo α_{k2} é selecionado achando o mínimo F_i em I_{lowk} . A decisão para determinar se esse par será otimizado ou não é realizada avaliando a desigualdade (4.1).

$$\begin{aligned}
& -(b_{upk} - b_{lowk})^2 / (2\eta_k) + \\
& \sum_{\substack{j=1 \\ j\text{-par a} \\ \text{otimizar}}}^{j=k-1} \{ (K(\bar{x}_{j1}, \bar{x}_{k1}) - K(\bar{x}_{j1}, \bar{x}_{k2}) - K(\bar{x}_{j2}, \bar{x}_{k1}) + K(\bar{x}_{j2}, \bar{x}_{k2})) (b_{upj} - b_{lowj}) (b_{upk} - b_{lowk}) / (\eta_j \eta_k) \} < 0
\end{aligned} \tag{4.1}$$

Sendo b_{upk} e b_{lowk} os limiares definidos na seleção do k -ésimo par, η_k a segunda derivada parcial da função objetivo L em relação a α_{k2} , $(\bar{x}_{k1}, \bar{x}_{k2})$ as entradas do k -ésimo par selecionado, $(\bar{x}_{j1}, \bar{x}_{j2})$ as entradas do j -ésimo par a otimizar, sendo $1 \leq j \leq k-1$, b_{upj} e b_{lowj} os limiares do j -ésimo par e η_j a segunda derivada parcial da função objetivo L em relação a α_{j2} .

Caso a F-heurística seja adaptada para a seleção de múltiplos pares, a seleção do k -ésimo par de coeficientes é realizada da seguinte maneira: os $k-1$ pares de Multiplicadores de Lagrange anteriores a otimizar são removidos do conjunto de exemplos, e o resto é dividido em dois grupos I_{upk} e I_{lowk} , como no caso anterior. O primeiro exemplo α_{k1} é escolhido achando o máximo F_i em I_{upk} , também como no caso anterior. O segundo exemplo α_{k2} é selecionado minimizando a expressão (4.2) em I_{lowk} . Sendo b_{lowki} igual à função F_i , η_{ki} a segunda derivada parcial da função objetivo L em relação ao exemplo i e $\bar{x}_i$ a entrada do i -ésimo exemplo. Depois de selecionado o k -ésimo par, a decisão para determinar se será otimizado ou não é realizada avaliando a desigualdade (4.1).

$$\begin{aligned}
& -(b_{upk} - b_{lowki})^2 / (2\eta_{ki}) + \\
& \sum_{\substack{j=1 \\ j\text{-par a} \\ \text{otimizar}}}^{j=k-1} \{ (K(\bar{x}_{j1}, \bar{x}_{k1}) - K(\bar{x}_{j1}, \bar{x}_i) - K(\bar{x}_{j2}, \bar{x}_{k1}) + K(\bar{x}_{j2}, \bar{x}_i)) (b_{upj} - b_{lowj}) (b_{upk} - b_{lowki}) / (\eta_j \eta_{ki}) \}
\end{aligned} \tag{4.2}$$

4.3 TAREFA 2: OTIMIZAÇÃO DOS PARES DE COEFICIENTES

Sendo decidida a otimização de M pares de coeficientes entre os N selecionados, ela é realizada de forma independente para cada par, usando a solução analítica do algoritmo *SMO* (Algoritmo 3.3). Portanto, as otimizações dos pares podem ser realizadas de forma sequencial, otimizando o primeiro par, depois o segundo, etc. Alternativamente, podem ser realizadas em paralelo usando uma maior quantidade de recursos de *hardware*, mas em menor tempo.

4.4 TAREFA 3: ATUALIZAÇÃO DE DADOS GLOBAIS

A atualização das funções F_i é realizada somando dois termos por cada par de coeficientes otimizado, como mostrado na expressão (4.3). Estes termos incluem cálculos de funções *kernels*. A quantidade desses cálculos é duas vezes o número de pares otimizados, vezes o número de exemplos ($2 \times M \times T$).

$$\mathbf{F}_i^{new} = \mathbf{F}_i^{old} + \sum_{j=1}^{j=M} \{ \mathbf{y}_{j1} (\alpha_{j1}^{old} - \alpha_{j1}^{new}) K(\bar{\mathbf{x}}_{j1}, \bar{\mathbf{x}}_i) + \mathbf{y}_{j2} (\alpha_{j2}^{old} - \alpha_{j2}^{new}) K(\bar{\mathbf{x}}_{j2}, \bar{\mathbf{x}}_i) \} i = 1..T \quad (4.3)$$

Os valores dos Multiplicadores de Lagrange podem ser iniciados em zero, e os das funções F_i em y_i como no algoritmo *SMO*.

4.5 CONVERGÊNCIA MONOTÔNICA

Nesta seção, será demonstrada a convergência monotônica do algoritmo *MP-SMO*. Para isso, o desenvolvimento matemático realizado por Lin, para a demonstração da

convergência monotônica do algoritmo *SMO* será aplicado (37). No trabalho de Lin, o par a ser selecionado é parametrizado como nas expressões (4.4) e (4.5).

$$\alpha_{11}(t) = \alpha_{11}^{old} - t / y_{11} \quad (4.4)$$

$$\alpha_{12}(t) = \alpha_{12}^{old} + t / y_{12} \quad (4.5)$$

Dessa forma, a função objetivo L é também parametrizada. Por se tratar de uma função quadrática, pode ser expressa da seguinte maneira:

$$L(\alpha_{11}(t), \alpha_{12}(t)) = \Psi(t) = \Psi(0) + \Psi'(0)t + \Psi''(0)t^2 / 2 \quad (4.6)$$

Sendo $\Psi(t)$ a função objetivo dependente de t . Em (4.6), $\Psi(0)$ é o valor da função objetivo antes da otimização. $\Psi'(0)$ e $\Psi''(0)$ são a primeira e segunda derivadas da função objetivo em relação a t antes da otimização. Usando a K-heurística na seleção do par de coeficientes a otimizar, do trabalho de Lin (37) conclui-se, para o algoritmo *SMO*, que:

$$\Psi(t_{op}) - \Psi(0) = -(2\gamma - \gamma^2)(b_{up} - b_{low})^2 / (2\eta_1) < 0 \quad (4.7)$$

Sendo t_{op} o valor do parâmetro t depois de finalizar a tarefa de otimização. O parâmetro γ representa o quanto foi recortado o novo valor de α_{12} na otimização (passo iv do Algoritmo 3.3). γ vale 1 se o novo valor de α_{12} satisfaz: $Lo < \alpha_{12} < Hi$, ou seu valor satisfaz: $0 < \gamma < 1$ se o novo α_{12} é igual a Lo ou Hi . Como $b_{low} \neq b_{up}$ o valor da função objetivo é reduzida em cada iteração, portanto, a convergência é monotônica.

Se dois pares $(\alpha_{11}, \alpha_{12})$ e $(\alpha_{21}, \alpha_{22})$ são otimizados em cada iteração, um novo parâmetro h é introduzido para a parametrização do segundo par como nas expressões (4.8) e (4.9).

$$\alpha_{21}(h) = \alpha_{21}^{old} - h / y_{21} \quad (4.8)$$

$$\alpha_{22}(h) = \alpha_{22}^{old} + h / y_{22} \quad (4.9)$$

A função objetivo Ψ depende agora de dois parâmetros t e h como na equação (4.10).

$$\begin{aligned} \Psi(t, h) = & \Psi(0, 0) + \frac{\partial \Psi}{\partial t}(0, 0)t + \frac{\partial^2 \Psi}{\partial t^2}(0, 0)t^2 / 2 \\ & + \frac{\partial \Psi}{\partial h}(0, 0)h + \frac{\partial^2 \Psi}{\partial h^2}(0, 0)h^2 / 2 + \frac{\partial^2 \Psi}{\partial h \partial t}(0, 0)th \end{aligned} \quad (4.10)$$

A partir da equação (4.10), a igualdade (4.11) pode ser demonstrada.

$$\begin{aligned} \Psi(t_{op}, h_{op}) - \Psi(0, 0) = & -(2\gamma_1 - \gamma_1^2)(b_{up1} - b_{low1})^2 / (2\eta_1) - (2\gamma_2 - \gamma_2^2)(b_{up2} - b_{low2})^2 / (2\eta_2) \\ & + \gamma_1 \gamma_2 (K(\bar{x}_{11}, \bar{x}_{21}) - K(\bar{x}_{11}, \bar{x}_{22}) - K(\bar{x}_{12}, \bar{x}_{21}) + K(\bar{x}_{12}, \bar{x}_{22}))(b_{up1} - b_{low1})(b_{up2} - b_{low2}) / (\eta_1 \eta_2) \end{aligned} \quad (4.11)$$

Sendo t_{op} e h_{op} os valores dos parâmetros t e h depois de finalizar a tarefa de otimização. A igualdade (4.11) pode ser vista como a soma de três termos como na expressão (4.12).

$$\Psi(t_{op}, h_{op}) - \Psi(0, 0) = -T_1 - (2\gamma_2 - \gamma_2^2)T_2 + \gamma_1 \gamma_2 T_3 \quad (4.12)$$

Sendo $-T_1$ igual ao lado direito da igualdade da expressão (4.7) e que depende somente da otimização do primeiro par de coeficientes. Por sua vez, os segundo e terceiro termos provêm da otimização do segundo par de coeficientes.

O algoritmo converge monotonicamente se o lado direito da igualdade (4.12) é negativo. Esta condição é garantida pelo fato de que $0 < \gamma_1 \leq 1$, $b_{low1} \neq b_{up1}$ (como no algoritmo SMO), $0 < \gamma_2 \leq 1$, $T_2 > 0$, e o algoritmo SMO de dois pares (2P-SMO) impõe que $T_2 > T_3$ (passos v e vi do Algoritmo 4.1 e do ii ao v do Algoritmo 4.2).

A demonstração anterior pode ser generalizada para a otimização de M pares de coeficientes. Nesse caso, para cada par é usado um parâmetro t_n , e a função objetivo é igual a:

$$\begin{aligned} \psi(t_1, t_2, \dots, t_M) = \psi(0, 0, \dots, 0) + \sum_{n=1}^{n=M} \frac{\partial \psi}{\partial t_n} t_n \\ + \frac{1}{2} \sum_{n=1}^{n=M} \frac{\partial^2 \psi}{\partial t_n^2} t_n^2 + \frac{1}{2} \sum_{n=1}^{n=M} \sum_{j=1}^{j=M} \frac{\partial^2 \psi}{\partial t_n \partial t_j} t_n t_j \end{aligned} \quad (4.13)$$

A partir da equação (4.13), a igualdade (4.14) pode ser demonstrada.

$$\begin{aligned} \Psi(t_1, \dots, t_M) - \Psi(0, 0) = -(2\gamma_1 - \gamma_1^2)(b_{up1} - b_{low1})^2 / (2\eta_1) - \sum_{n=2}^{n=M} (2\gamma_n - \gamma_n^2)(b_{upn} - b_{lown})^2 / (2\eta_n) \\ + \sum_{n=2}^{n=M} \sum_{j=1}^{j=n-1} \left[\gamma_j \gamma_n (K(\bar{x}_{j1}, \bar{x}_n) - K(\bar{x}_{j1}, \bar{x}_{n2}) - K(\bar{x}_{j2}, \bar{x}_{n1}) + K(\bar{x}_{j2}, \bar{x}_{n2}))(b_{upj} - b_{lowj})(b_{upn} - b_{lown}) / (\eta_j \eta_n) \right] \end{aligned} \quad (4.14)$$

Como $0 < \gamma_k \leq 1$ para $k = 1..M$, $b_{up1} \neq b_{low1}$ e a condição (4.1) é satisfeita para cada par otimizado (a partir do segundo), o lado direito da igualdade (4.14) é negativa e a convergência monotônica é garantida.

4.6 CONFORMIDADE DAS RESTRIÇÕES DO PROBLEMA SVM

Pelo fato do algoritmo *MP-SMO* ter sido construído sem impor as restrições do problema SVM, é preciso mostrar que ele as satisfaz. Como as restrições dadas por (3.5) são triviais, é necessário demonstrar somente a restrição dada pela expressão (3.4).

$$\sum_{i=1..T} \alpha_i y_i = 0 \quad (3.4)$$

Será considerado o caso de otimização de dois pares de coeficientes $(\alpha_{11}, \alpha_{12})$ e $(\alpha_{21}, \alpha_{22})$. A otimização de cada um em separado satisfaz a restrição dada por (3.4), portanto, para o primeiro par são válidas as igualdades (4.15) e (4.16).

$$y_{11}\alpha_{11}^{old} + y_{12}\alpha_{12}^{old} = -y_{21}\alpha_{21}^{old} - y_{22}\alpha_{22}^{old} - \sum_{i=5}^{i=T} y_i \alpha_i \quad (4.15)$$

$$y_{11}\alpha_{11}^{new} + y_{12}\alpha_{12}^{new} = -y_{21}\alpha_{21}^{old} - y_{22}\alpha_{22}^{old} - \sum_{i=5}^{i=T} y_i \alpha_i \quad (4.16)$$

No entanto, a aplicação da restrição dada por (3.4) na otimização do segundo par de coeficientes implica que as igualdades (4.17) e (4.18) sejam válidas.

$$y_{21}\alpha_{21}^{old} + y_{22}\alpha_{22}^{old} = -y_{11}\alpha_{11}^{old} - y_{12}\alpha_{12}^{old} - \sum_{i=5}^{i=T} y_i \alpha_i \quad (4.17)$$

$$y_{21}\alpha_{21}^{new} + y_{22}\alpha_{22}^{new} = -y_{11}\alpha_{11}^{old} - y_{12}\alpha_{12}^{old} - \sum_{i=5}^{i=T} y_i \alpha_i \quad (4.18).$$

Das igualdades (4.15) e (4.16) resulta a igualdade (4.19).

$$y_{11}\alpha_{11}^{new} + y_{12}\alpha_{12}^{new} = -y_{11}\alpha_{11}^{old} - y_{12}\alpha_{12}^{old} \quad (4.19)$$

Substituindo $-y_{11}\alpha_{11}^{old} - y_{12}\alpha_{12}^{old}$ em (4.18), resulta a igualdade (4.20).

$$y_{21}\alpha_{21}^{new} + y_{22}\alpha_{22}^{new} + y_{11}\alpha_{11}^{new} + y_{12}\alpha_{12}^{new} = -\sum_{i=5}^{i=T} y_i \alpha_i \quad (4.20)$$

Essa igualdade implica que os novos valores dos dois pares de coeficientes satisfazem a restrição dada pela expressão (3.4) do problema SVM.

Uma demonstração análoga pode ser feita para um número M de pares de coeficientes.

4.7 RESUMO DO CAPÍTULO

Neste capítulo foi descrita nossa modificação do algoritmo *SMO*, que consiste em otimizar múltiplos pares em cada iteração com o propósito de diminuir o tempo de treinamento em relação ao algoritmo original. O novo algoritmo foi denominado de *MP-SMO*. Foram mostradas as tarefas executadas em cada iteração do algoritmo, em particular as adaptações das K-heurística e F-heurística. A convergência monotônica do algoritmo *MP-SMO* foi demonstrada e também a conformidade das restrições da *SVM*.

5 IMPLEMENTAÇÃO EM *HARDWARE* DO ALGORITMO *MP-SMO*

5.1 IMPLEMENTAÇÃO DO *KERNEL HARDWARE* AMIGÁVEL

5.1.1 Escolha do *kernel*

A função *kernel* escolhida para a implementação em *hardware* do algoritmo *MP-SMO* é a apresentada por Anguita et al. (38). A motivação do desenvolvimento desse *kernel* foi a quantidade limitada de recursos de *hardware* disponíveis para implementar a fase de teste da *SVM*. É possível ampliar essa motivação para a fase de treinamento quando realizada com o algoritmo *MP-SMO*. Na expressão (4.3) da atualização de dados globais do algoritmo *MP-SMO* (Seção 4.4), pode ser observado que para cada exemplo do conjunto de treinamento, são calculados $2 \times M$ *kernels*. Se esses cálculos são realizados em paralelo, o tempo de execução dessa tarefa é diminuído ao custo de maior área. Portanto, interessa reduzir o máximo possível a área da implementação em *hardware* da função *kernel* para o treinamento da *SVM*.

A expressão (5.1) é a do *kernel* escolhido. Trata-se de uma função exponencial de base 2, cujo expoente contém a função norma L1 como mostrada na expressão (5.2).

$$K(\bar{x}_i, \bar{x}_j) = 2^{-\beta \|\bar{x}_i - \bar{x}_j\|_1} \quad (5.1)$$

$$\|\bar{x}_i - \bar{x}_j\|_1 = \sum_{s=1}^{s=d} |x_{is} - x_{js}| \quad (5.2)$$

Sendo β um parâmetro dado pelo usuário e igual a: $\beta = 2^{\pm p}$, com $p=0, 1, 2, \dots$

5.1.2 Implementação do *kernel* para o caso de atributos categóricos

No trabalho de Anguita et al. a função *kernel* da seção anterior é calculada usando um algoritmo do tipo *CORDIC* (38). A arquitetura de *hardware* apresentada usa somente operações somas e deslocamentos, evitando o uso de multiplicadores que requerem quantidades importantes de área. Essa arquitetura pode ser usada para entradas com atributos categóricos ou de outro tipo (inteiros, reais). De acordo com os resultados experimentais reportados, a execução do algoritmo pode ser realizada em 16 ciclos de relógio para obter o resultado correto, somados aos necessários para calcular o expoente.

Se o expoente em (5.1) for um número inteiro, o *kernel* pode ser calculado realizando somente uma operação deslocamento. Será mostrado que se conjuntos de treinamento com atributos categóricos são considerados, como o do *benchmark Tic-tac-toe endgame*, essa simplificação é válida.

Quando os atributos são categóricos e são transformados com a codificação *1-of-N*, dois exemplos $\vec{x}_i$ e $\vec{x}_j$ diferirão em um número par de componentes. Por exemplo, na Figura 5.1 são mostradas duas entradas do *benchmark Tic-tac-toe endgame* que diferem em uma só posição, e quando codificadas duas componentes são diferentes. Portanto, o valor da função norma L1 ($\|\vec{x}_i - \vec{x}_j\|_1$) é par, maior que 0.

Figura 5.1 – Diferença entre dois exemplos com codificação *1-of-N*

Sendo que o valor da função norma L1 é par, e dado que β é uma potência de 2, se $\beta \geq 0,5$, o expoente da função *kernel* é um número inteiro negativo. Dessa maneira, a implementação da função exponencial é realizada com uma operação deslocamento à direita. Por sua vez, a função norma L1 é implementada usando somente operações somas e o expoente com outra operação deslocamento, à direita ou à esquerda dependendo do valor de β . É possível fazer as operações em 2 ciclos de relógio, somados aos necessários para calcular a norma L1. Na Figura 5.2 é mostrada a arquitetura para o cálculo da função *kernel* para entradas com atributos categóricos. Além de ter como entradas os vetores $\vec{x}_i$ e $\vec{x}_j$, o deslocamento final é realizado sobre $y_i(\alpha_i^{old} - \alpha_i^{new})$, dado que é um fator pelo qual o *kernel* é multiplicado na atualização de dados (expressão (4.3)).

Figura 5.2 – Arquitetura para cálculo da função *kernel*

5.2 IMPLEMENTAÇÃO DA OTIMIZAÇÃO

A função *kernel* é também calculada para a tarefa de otimização de um par de coeficientes. A realização do cálculo é necessária para achar a segunda derivada

parcial da função objetivo em relação ao segundo coeficiente a otimizar (linha 8 do Algoritmo 3.3), como mostrado na expressão (5.3).

$$\eta_1 = K(\bar{x}_{11}, \bar{x}_{11}) + K(\bar{x}_{12}, \bar{x}_{12}) - 2K(\bar{x}_{12}, \bar{x}_{11}) \quad (5.3)$$

Se for possível aproximar $K(\bar{x}_{12}, \bar{x}_{11}) \ll 1$, e dado que $K(\bar{x}_{11}, \bar{x}_{11}) + K(\bar{x}_{12}, \bar{x}_{12}) = 2$, a expressão (5.3) é aproximada por:

$$\eta_1 = 2 \quad (5.4).$$

Consequentemente, o cálculo do novo valor do segundo coeficiente α_{12}^{new} é simplificado (linha 9 do Algoritmo 3.3). O cálculo de α_{12}^{new} é então realizado como na expressão (5.5). Pode se observar que com a aproximação realizada, a divisão envolvida contém um divisor igual a 2, podendo ser realizada com um deslocamento para a direita.

$$\alpha_{12}^{new} = \alpha_{12}^{old} + y_{12} (b_{low} - b_{up}) / 2 \quad (5.5)$$

A aproximação $K(\bar{x}_{12}, \bar{x}_{11}) \ll 1$ depende da escolha do parâmetro β , quanto maior é β , melhor é a aproximação. Como o valor desse parâmetro na prática é determinado pela minimização do erro de teste, só poderá ser realizada uma verificação experimental da validade da aproximação e observar como afeta a convergência do algoritmo.

Portanto, a tarefa de otimização também pode ser implementada em *hardware* sem a necessidade de realizar multiplicações ou divisões que requerem importantes recursos de área. Na Figura 5.3 é mostrada a arquitetura *hardware* do otimizador, na qual todos os blocos são implementados sem realizar operações de multiplicação ou divisão.

Figura 5.3 – Arquitetura do otimizador de par de coeficientes

5.3 IMPLEMENTAÇÃO DA SELEÇÃO DO CONJUNTO DE TRABALHO E DA ATUALIZAÇÃO DE DADOS GLOBAIS

5.3.1 Implementação da seleção do conjunto de trabalho

Para implementar a seleção do conjunto de trabalho em *hardware*, é preciso escolher entre implementar a adaptação da K-heurística (Algoritmo 4.1 para dois pares) e a adaptação da F-heurística (Algoritmo 4.2 para dois pares). A adaptação da F-heurística realiza cálculos de maior complexidade na escolha dos segundos coeficientes de cada par que a adaptação da K-heurística (operações quadráticas e divisões). A complexidade computacional da K-heurística está na decisão de otimizar ou não os pares selecionados a partir do segundo (passos *v*, *vi* e *vii* do Algoritmo 4.1

para dois pares). Para simplificar essa tarefa, será usada a aproximação realizada na Seção 5.2, que para este caso pode ser expressa como: $K(\bar{x}_i, \bar{x}_j) \ll 1, i \neq j$. Dessa maneira o coeficiente *coeff1* do Algoritmo 4.1 (expressão (5.6)) pode ser também considerado $coeff1 \ll 1$, e *coeff2* (expressão (5.7)) negativo, e portanto, a condição para otimizar o segundo par de coeficientes é satisfeita.

$$coeff1 = (K(\bar{x}_{11}, \bar{x}_{21}) - K(\bar{x}_{11}, \bar{x}_{22}) - K(\bar{x}_{12}, \bar{x}_{21}) + K(\bar{x}_{12}, \bar{x}_{22})) \quad (5.6)$$

$$coeff2 = -(b_{up2} - b_{low2})^2 / (2\eta_2) + coeff1(b_{up1} - b_{low1})(b_{up2} - b_{low2}) / (\eta_1\eta_2) \quad (5.7)$$

A simplificação da adaptação da K-heurística, permite que possa ser considerada *hardware* amigável, dado que pode ser implementada realizando somente operações de comparação. Na Figura 5.4 é mostrada a arquitetura para realizar a seleção do primeiro exemplo do conjunto de trabalho.

Figura 5.4 – Arquitetura da seleção de um dos coeficientes do conjunto de trabalho

Nessa arquitetura, são realizadas comparações para cada exemplo para decidir se ele é o escolhido. Uma arquitetura análoga corresponde à seleção do segundo coeficiente do conjunto de trabalho.

5.3.2 Implementação da atualização de dados globais

A atualização de dados globais é realizada através da expressão (4.3). Os valores das funções F_i são atualizados com a soma de $2 \times M$ termos e os valores anteriores são previamente lidos de memória. Os $2 \times M$ termos são os calculados pelos blocos *kernel*. Cada um desses termos são representados como S_{j1} ou S_{j2} na expressão (5.8).

$$F_i^{new} = F_i^{old} + \sum_{j=1}^{j=M} \{y_{j1}(\alpha_{j1}^{old} - \alpha_{j1}^{new})K(\bar{x}_{j1}, \bar{x}_i) + y_{j2}(\alpha_{j2}^{old} - \alpha_{j2}^{new})K(\bar{x}_{j2}, \bar{x}_i)\} i = 1..T \quad (4.3)$$

$$F_i^{new} = F_i^{old} + \sum_{j=1}^{j=M} \{S_{j1} + S_{j2}\} i = 1..T \quad (5.8)$$

Na Figura 5.5 é mostrada a arquitetura *hardware* para realizar a tarefa de atualização de dados. Para cada exemplo, a função F_i é atualizada realizando somente operações somas.

Figura 5.5 – Arquitetura da atualização de dados globais

5.4 REPRESENTAÇÃO DE DADOS EM PONTO FIXO

Os valores dos Multiplicadores de Lagrange α_i , das funções F_i , das saídas dos blocos *kernel* e dos parâmetros b_{up} , b_{low} , e C são números reais. Devido à limitação em área, não é conveniente representá-los em ponto flutuante de alta precisão (38). Esses dados serão representados em ponto fixo (21), cujas quantidades de bits serão determinadas experimentalmente para que não afete a convergência do algoritmo e não aumentar de forma importante o erro de teste.

5.5 ARQUITETURA DO ALGORITMO MP-SMO

Na Figura 5.6 é mostrada a arquitetura do caminho de dados do algoritmo *MP-SMO* sem *pipelining* nem paralelismo. Ela possui 4 **blocos de memória** que contém as

entradas $\bar{x}_i$, as saídas y_i , os Multiplicadores de Lagrange α_i e as funções F_i . O **bloco seletor de conjunto de trabalho** escolhe os pares de coeficientes a serem otimizados em cada iteração. Na primeira seleção, esse bloco lê da memória os valores iniciais das funções F_i , dos multiplicadores α_i , e das saídas y_i . Os coeficientes escolhidos ingressam no **bloco otimizador**, que nesta arquitetura otimiza de a um par por vez. Depois da otimização, os novos valores dos coeficientes são guardados em memória e também ingressam no **bloco kernel**. São também ingressadas a esse bloco as entradas dos exemplos $\bar{x}_i$ que são lidas da memória. Para cada exemplo, são calculados $2 \times M$ *kernels*, que nesta arquitetura também são realizados de a um por vez. Por último, o **bloco atualizador de F_i** realiza essa tarefa, lendo da memória os valores F_i e posteriormente guardando os valores atualizados. Exceto no início do algoritmo, depois da otimização dos pares de coeficientes, para cada exemplo primeiro são executados os cálculos dos *kernels*, depois a atualização da função F_i e por último é executada a seleção do conjunto de trabalho. Posteriormente a mesma sequência é executada para o exemplo seguinte, e continua até percorrer todo o conjunto de treinamento. No final resultam os pares a serem otimizados, ou em função dos valores b_{up1} e b_{low1} determina-se a parada do algoritmo.

Figura 5.6 – Arquitetura VLSI do caminho de dados do algoritmo MP-SMO sem *pipelining* em paralelismo

Para acelerar a execução de cada iteração, *pipelining* pode ser introduzido no bloco *kernel*, como sugere o trabalho de Pedersen e Schoeberl (21). Também o bloco atualizador de dados globais pode ser realizado com *pipelining*. Esses blocos, junto com o seletor de conjunto de trabalho, que em nossa arquitetura é um estágio de todo o *pipelining*, são os que se executam durante mais tempo em cada iteração. Na Figura 5.7 é mostrada essa arquitetura, na qual o *pipelining* contém $17 \times M + 4$ estágios (diferenciados pela cor azul).

Figura 5.7 - Arquitetura VLSI do caminho de dados do algoritmo *MP-SMO* com *pipelining*

Outra arquitetura que acelera a execução de cada iteração é a que incorpora paralelismo (42). Nesse caso, as otimizações dos múltiplos pares podem ser paralelizadas, assim como o cálculo dos *kernels*. Na Figura 5.8 essa arquitetura é mostrada, a qual contém M blocos otimizadores e $2 \times M$ blocos *kernel*.

Figura 5.8 - Arquitetura VLSI do caminho de dados do algoritmo MP-SMO com paralelismo

A última arquitetura a ser implementada e testada é a que combina *pipelining* e paralelismo. Na Figura 5.9 essa arquitetura é exibida, na qual o *pipelining* de cada bloco *kernel* é realizado com nove estágios e o bloco atualização de dados com $M + 1$ estágios.

Figura 5.9 - Arquitetura VLSI do caminho de dados do algoritmo MP-SMO com *pipelining* e paralelismo

5.6 RESUMO DO CAPÍTULO

Neste capítulo foram apresentadas as arquiteturas *hardware* para realizar as diferentes tarefas executadas em cada iteração do algoritmo. Para o tipo de atributos do *benchmark* escolhido para a implementação, os blocos *kernel* e otimizador foram simplificados de maneira que não são usadas operações de multiplicação ou divisão que ocupam importante quantidade de área. Simplificou-se também a decisão de otimizar ou não os pares de coeficientes a partir do segundo. Foram também

apresentadas as diferentes arquiteturas *hardware* de todo o algoritmo, incluindo paralelismo e *pipelining* para reduzir o tempo de execução de cada iteração.

6 RESULTADOS EXPERIMENTAIS

6.1 RESULTADOS DA IMPLEMENTAÇÃO EM SOFTWARE DO ALGORITMO MP-SMO

Nesta seção são apresentados os resultados de executar a implementação em *software* do algoritmo *MP-SMO*. O algoritmo *MP-SMO* foi testado para sete *benchmarks* do repositório da *UCI* (22). Na Tabela 6.1, os *benchmarks* são listados na coluna 1. A coluna 2 exibe o número de exemplos de cada *benchmark*. Essas quantidades variam de poucas centenas (208) a milhares de exemplos (4177). Na coluna 3 é mostrado o número de atributos categóricos de cada *benchmark*, os quais foram codificados em atributos binários, e na coluna 4 o número de atributos reais. Pode ser observado que o número total de atributos varia entre 8 e 60, e que há dois *benchmarks* com somente atributos categóricos, dois com só atributos reais e 3 com os dois tipos de atributos.

Tabela 6.1 – Características dos *benchmarks* e parâmetros

<i>Benchmarks</i>	# exemplos	# Atributos categóricos	# Atributos reais	<i>C</i>	$2\sigma^2$
<i>Tic-tac-toe endgame</i>	958	9	0	1	3
<i>Adult</i>	1605	14	0	4	8
<i>Dermatology</i>	358	33	1	5	8
<i>Ionosphere</i>	351	1	33	10	0,25
<i>Sonar</i>	208	0	60	5	1
<i>Breast cancer</i>	569	0	30	7	1
<i>Abalone</i>	4177	1	7	10	0,1

O *kernel* empregado foi a função de base radial gaussiana (*RBF*) dada pela expressão (3.8). Esse *kernel* é um dos mais referenciados pela literatura especializada, satisfaz as condições de Mercer e transforma o hiperespaço das entradas em outro de Hilbert de dimensão infinita, aumentando dessa maneira a separabilidade dos exemplos de treinamento (5).

$$K_2(\bar{x}_i, \bar{x}_j) = e^{-\|\bar{x}_i - \bar{x}_j\|^2 / (2\sigma^2)} \quad (3.8)$$

O parâmetro σ do *kernel* RBF deve ser escolhido pelo usuário, assim como o parâmetro C da SVM. Essa escolha foi realizada minimizando o erro na fase de teste. Para estimá-lo, foi aplicado o método *ten-fold cross validation*⁸ (43). Os valores obtidos de C e de $2\sigma^2$ são mostrados nas colunas 5 e 6 da Tabela 6.1.

Os algoritmos foram executados de forma sequencial em um *core* de processador 8212 AMD Opteron™ de 1 MHz. Também foram escritos e executados usando o ambiente de programação OCTAVE.

Implementaram-se os algoritmos SMO e MP-SMO otimizando 2, 3 e 4 pares de Multiplicadores de Lagrange, e usando as técnicas *shrinking* e *caching*. Dado um *benchmark*, a mesma quantidade de memória foi usada para *caching* para todos os algoritmos. Foi usada uma quantidade de memória capaz de armazenar 10% do total de *kernels*.

O tempo de treinamento do algoritmo MP-SMO (T_{MP-SMO}) foi comparado com o tempo de treinamento do algoritmo SMO (T_{SMO}). Os resultados, usando a adaptação da K-heurística para a seleção do conjunto de trabalho, são apresentados na Figura 6.1. Nela, é mostrada a razão de tempo de treinamento para cada *benchmark* e para cada número de pares otimizados. Os melhores resultados foram obtidos com o algoritmo 4P-SMO, exceto para o *benchmark* Abalone, no qual o melhor resultado foi obtido com o algoritmo 2P-SMO. Para a maioria dos *benchmarks*, observa-se que quanto mais pares de coeficientes são otimizados, maior é a velocidade de treinamento obtida. Não obstante, a relação dessas duas quantidades é menor que proporcional.

O mesmo tipo de gráfico é exibido na Figura 6.2 usando a adaptação da F-heurística. Nesse caso, os melhores resultados foram obtidos otimizando dois pares de coeficientes para três dos *benchmarks* e otimizando três pares de coeficientes para quatro dos sete *benchmarks*. O melhor desempenho, quanto ao tempo de treinamento do algoritmo MP-SMO em relação ao algoritmo SMO, sem importar qual das heurísticas é usada, é evidenciado através das Figuras 6.1 e 6.2.

⁸ No método *ten-fold cross validation*, o conjunto de treinamento é dividido em dez conjuntos disjuntos de igual número de exemplos. Dez treinamentos e testes são realizados. Em cada um deles o treinamento é realizado com 90% dos exemplos e o teste com um dos conjuntos. Dessa maneira todos os exemplos são parte do conjunto de teste uma só vez. O erro é a média dos dez erros obtidos.

Figura 6.1 – Tempo de treinamento do algoritmo *MP-SMO* em relação ao tempo de treinamento do *SMO* (K-heurística)

Figura 6.2 - Tempo de treinamento do algoritmo *MP-SMO* em relação ao tempo de treinamento do *SMO* (F-heurística)

Na Figura 6.3 é representado o número de iterações do algoritmo *MP-SMO* (l_t) em relação ao número de iterações do algoritmo *SMO* (l_{t1}), usando a adaptação da K-heurística, e na Figura 6.4 usando a adaptação da F-heurística. Como esperado, em

todos os casos o número de iterações do algoritmo *MP-SMO* é menor que o número de iterações do algoritmo *SMO*.

Figura 6.3 - Número de iterações do algoritmo *MP-SMO* em relação ao número de iterações do *SMO* (*K*-heurística)

Figura 6.4 - Número de iterações do algoritmo *MP-SMO* em relação ao número de iterações do *SMO* (*F*-heurística)

Comparando os resultados das Figuras 6.1 e 6.2 com os resultados das Figuras 6.3 e 6.4, pode-se observar que a redução do número de iterações é maior que a redução do tempo de treinamento. Isso é devido à redução do tempo de treinamento (T_{SMO}) ser obtida reduzindo-se o número de iterações (I_t) a expensas de maior tempo de execução de cada iteração ($t_1+t_2+t_3$), devido ao aumento da complexidade computacional.

Apesar de que a velocidade de execução do treinamento é aumentada para todos os casos, o número de pares de coeficientes que mais convém otimizar depende de cada aplicação. Isso é devido a que a heurística para a seleção do conjunto de trabalho escolhe o primeiro par produzindo a maior redução da função objetivo, o segundo par, a segunda maior redução, etc. Portanto, a diminuição do número de iterações produzida pelos terceiro e quarto pares pode ser menor que o aumento do tempo de execução de cada iteração. Como resultado, em muitos casos a melhora do tempo de treinamento quando três ou quatro pares são otimizados é menor que quando dois pares são otimizados.

A Tabela 6.2 mostra o tempo de treinamento absoluto. A primeira coluna corresponde aos *benchmarks*, a segunda e a terceira ao tempo de treinamento do SMO (em segundos) usando a K-heurística e a F-heurística, respectivamente. O melhor tempo de treinamento para cada *benchmark* obtido com o algoritmo MP-SMO é apresentado na quarta coluna, e na quinta coluna, são mostrados a heurística e o número de pares otimizados por iteração.

A Tabela 6.2 mostra que somente para dois *benchmarks* o melhor tempo de treinamento do algoritmo MP-SMO é obtido com a F-heurística adaptada, os outros cinco melhores tempos são obtidos com a K-heurística adaptada. Pode ser observado também que para os *benchmarks Adult* e *Sonar*, o melhor tempo de treinamento para o algoritmo SMO foi obtido com a F-heurística. Não obstante, o melhor tempo para o algoritmo MP-SMO foi obtido com a K-heurística adaptada. Isso pode ser explicado porque a F-heurística adaptada apresenta maior complexidade computacional e portanto, o tempo de execução de cada iteração é maior que o do algoritmo MP-SMO com a K-heurística adaptada.

A porcentagem de diminuição do tempo de treinamento para os *benchmarks* testados e para os melhores resultados varia entre 22,5% e 42,8%.

Tabela 6.2 - Tempos de treinamento para o algoritmo SMO, melhor tempo de treinamento e melhor algoritmo

<i>Benchmarks</i>	Tempo de treinamento (s) K-heur. SMO	Tempo de treinamento(s) F-heur. SMO	Melhor tempo de treinamento (s)	Heurística- # pares
<i>Tic-tac-toe endgame</i>	414,5	424,7	237,0	K adaptada 4P
<i>Adult</i>	633,5	541,0	332,8	K adaptada 4P
<i>Dermatology</i>	28,72	32,30	19,58	K adaptada 4P
<i>Ionosphere</i>	42,57	45,97	28,145	K adaptada 4P
<i>Sonar</i>	20,02	18,28	12,46	K adaptada 4P
<i>Breast cancer</i>	14,98	8,470	5,931	F adaptada 2P
<i>Abalone</i>	10940	1122	870,1	F adaptada 2P

Na Tabela 6.3 o erro de teste é exibido para todos os *benchmarks* e cada algoritmo. Na primeira coluna são listados os *benchmarks*, na segunda o erro de teste para o algoritmo SMO, na terceira para o algoritmo 2P-SMO, na quarta para o algoritmo 3P-SMO e na quinta para o algoritmo 4P-SMO. Os resultados sem parênteses correspondem aos algoritmos usando a K-heurística (adaptada) e entre parênteses aos algoritmos usando a F-heurística (adaptada).

Tabela 6.3 – Erro de teste usando a K-heurística (F-heurística)

<i>Benchmarks</i>	Erro de teste (%) SMO	Erro de teste (%) 2P-SMO	Erro de teste (%) 3P-SMO	Erro de teste (%) 4P-SMO
<i>Tic-tac-toe endgame</i>	9,8 (9,8)	9,8 (9,8)	9,8 (9,8)	9,8 (9,8)
<i>Adult</i>	18,8 (18,6)	18,6 (18,8)	18,6 (18,7)	18,7 (18,7)
<i>Dermatology</i>	0,6 (0,6)	0,6 (0,6)	0,6 (0,6)	0,6 (0,6)
<i>Ionosphere</i>	12,1 (12,1)	12,1 (12,1)	12,1 (12,1)	12,1 (12,1)
<i>Sonar</i>	30,5 (30,5)	30,5 (30,5)	30,5 (30,5)	30,5 (30,5)
<i>Breast cancer</i>	3,9 (4,3)	3,4 (4,1)	6,6 (4,3)	6,2 (4,1)
<i>Abalone</i>	21,5 (22,1)	21,7 (21,5)	22,2 (22,9)	22,0 (21,3)

Interessa saber se a otimização de múltiplos pares introduz aumento no erro de teste. Os resultados experimentais indicam que para quase todos os *benchmarks* a diferença de erro de teste é pequena. Não obstante, para o *benchmark Breast cancer* usando três e quatro pares com a K-heurística adaptada, o erro aumenta de

forma significativa. Apesar disso, para esse *benchmark* o melhor tempo de treinamento foi obtido com a F-heurística adaptada.

Como foi visto no Capítulo 4, no algoritmo *MP-SMO* são selecionados N pares de coeficientes e otimizados M pares. Sendo que $1 \leq M \leq N$. A decisão de otimizar ou não os pares de coeficientes, obtida com a avaliação da desigualdade (4.1), garante que a função objetivo L decresça em cada iteração. Por sua vez, quando um par não é escolhido, o algoritmo não garante que seja a melhor decisão. Ou seja, se o par tivesse sido otimizado, a função objetivo L poderia ter decrescido mais. Foram realizados seis treinamentos nos quais foram contabilizadas as vezes que isso aconteceu. O resultado indicou que entre 0% e 5% de todas as iterações, decidiu-se não otimizar e o contrário teria como efeito que L decrescesse mais.

6.2 RESULTADOS DA IMPLEMENTAÇÃO EM *HARDWARE* DO ALGORITMO *MP-SMO*

Os algoritmos *SMO* e *MP-SMO* foram implementados em *hardware* para o *benchmark Tic-tac-toe endgame*, que pertence ao tipo de problema de classificação cujos atributos das entradas são do tipo categórico. O conjunto de treinamento desse *benchmark* contém 958 exemplos e as entradas codificadas com o código 1-*of-N* são formadas por 27 componentes binárias. As arquiteturas usadas foram as descritas no Capítulo 5, portanto, o *kernel* usado foi o da expressão (5.1), sendo que o valor do parâmetro β deve satisfazer: $\beta \geq 0,5$ para simplificar seu cálculo.

$$K(\bar{x}_i, \bar{x}_j) = 2^{-\beta \|\bar{x}_i - \bar{x}_j\|}, \quad (5.1)$$

Para determinar os valores de β e C a serem usados, o algoritmo *SMO* foi simulado no ambiente de programação *OCTAVE* em um computador de uso geral. A determinação foi realizada minimizando o erro de teste e usando o método *ten-fold cross validation* (43). O resultado obtido é o seguinte:

$\beta = 0,5$ $C = 1$

Com os dados obtidos realizou-se a implementação em *hardware* do algoritmo *SMO* com a ferramenta *Bluespec System Verilog (BSV)*, a qual permite fazer uma descrição comportamental do circuito e simulá-la.

Determinou-se experimentalmente a quantidade mínima de bits necessária para representar os dados em ponto fixo. A quantidade de bits necessária para representar a parte inteira é determinada pelo valor máximo das funções F_i . Das simulações realizadas no ambiente de programação *OCTAVE* do algoritmo *SMO*, resultou que **4 bits para a parte inteira** são suficientes. Das simulações no *BSV*, verificou-se que a precisão para representar os dados é crítica para a convergência do algoritmo. A quantidade mínima de bits para que o algoritmo convirja é de 15 para a parte fracionária. Não obstante, aumentando a quantidade de bits, a variação do erro de teste é pequena como mostrada na Tabela 6.4. Na primeira coluna dessa tabela, são exibidas as diferentes representações dos dados com que foi simulado o algoritmo, na segunda coluna o erro de teste do algoritmo *SMO* das simulações no *BSV* e na terceira para as simulações no *OCTAVE*. Como pode ser observado, o erro para as representações em ponto fixo é muito similar ao erro para a representação em ponto flutuante de precisão dupla. No caso da implementação no *BSV*, para a representação com 32 e 64 bits da parte fracionária, o erro é menor que na implementação no *OCTAVE*. Isso é devido a que na implementação *OCTAVE* são realizadas operações de multiplicação que aumentam o erro de cálculo numérico, enquanto que em *BSV* essas operações são evitadas. Considerando uma margem de segurança de 1 bit, em relação à convergência, e para minimizar a área ocupada pelo circuito, escolheu-se **16 bits para a parte fracionária**.

Tabela 6.4 – Erro de teste para diferentes representações dos dados

Representação dos dados	Erro de teste (%) <i>SMO (BSV)</i>	Erro de teste (%) <i>SMO (OCTAVE)</i>
Ponto fixo 4.16 b	10,3	-
Ponto fixo 4.32 b	9,8	-
Ponto fixo 4.64 b	9,8	-
Ponto Flutuante precisão dupla	-	10,2

Foram implementados os algoritmos *SMO*, e o *MP-SMO* para 2, 3 e 4 pares otimizados por iteração. Foram realizadas implementações com paralelismo como descritas na Seção 5.5. *Pipelining* foi também incorporado nos blocos *kernel*, atualização de dados e seleção do conjunto de trabalho. O *pipelining* e o paralelismo têm o propósito de diminuir o tempo de execução de cada iteração.

Os algoritmos foram implementados, simulados e testados na placa *ALTERA DE2*. Esta placa possui um dispositivo *FPGA* EP2C35F672C6 da família *Cyclone II* com 35000 elementos lógicos e 483Kb de memória *SRAM*. A frequência do relógio é de 50 MHz. A síntese lógica, mapeamento ao *FPGA*, simulações em nível lógico e programação do dispositivo foram realizadas com a ferramenta da *ALTERA*, *QUARTUS II*, versão 8.0.

Na Tabela 6.5 são mostrados os resultados de 16 arquiteturas implementadas e testadas. Elas correspondem aos algoritmos *SMO*, *2P-SMO*, *3P-SMO* e *4P-SMO*, sem *pipelining* nem paralelismo e com as combinações possíveis dessas duas técnicas.

Tabela 6.5 - Resultados da implementação em hardware dos algoritmos *SMO* e *MP-SMO* para o benchmark *Tic-tac-toe endgame*

Arquitetura	Número de iterações	fmax. (MHz)	Tempo de treinamento(s) @fmax.	Tempo de treinamento(s) @50MHz	Elementos lógicos (%)	SRAM (%)
<i>SMO</i>	2728	122,7	0,575	1,412	5	14
<i>SMO-pipe</i>	2728	111,8	0,048	0,106	10	14
<i>SMO-par</i>	2728	125,8	0,354	0,889	6	14
<i>SMO-pipepar</i>	2728	122,7	0,043	0,106	8	14
<i>2P-SMO</i>	1324	127,9	0,465	1,190	7	14
<i>2P-SMO-pipe</i>	1324	110,3	0,024	0,052	20	14
<i>2P-SMO-par</i>	1324	117,2	0,184	0,432	11	14
<i>2P-SMO-pipepar</i>	1324	115,1	0,022	0,052	13	14
<i>3P-SMO</i>	828	120,4	0,440	1,060	9	14
<i>3P-SMO-pipe</i>	828	117,6	0,014	0,033	31	14
<i>3P-SMO-par</i>	828	114,3	0,118	0,270	16	14
<i>3P-SMO-pipepar</i>	828	93,3	0,017	0,032	17	14
<i>4P-SMO</i>	584	122,5	0,402	0,985	11	14
<i>4P-SMO-pipe</i>	584	118,6	0,010	0,024	45	14
<i>4P-SMO-par</i>	584	107,8	0,094	0,202	21	14
<i>4P-SMO-pipepar</i>	584	109,8	0,010	0,023	23	14

Na primeira coluna são listadas as arquiteturas implementadas, na segunda o número de iterações resultante das simulações e dos testes em silício, na terceira a frequência máxima obtida com a síntese, na quarta o tempo de treinamento calculado para a frequência máxima, na quinta o tempo de treinamento obtido simulando a 50MHz, na sexta a porcentagem de elementos lógicos, e na sétima a porcentagem de memória *SRAM* usada. Como pode ser observado, a otimização de múltiplos pares diminui o número total de iterações e o tempo de treinamento, assim como também a incorporação de *pipelining* e/ou paralelismo, como previsto. Considerando o melhor caso (4 pares com paralelismo e *pipelining*), o tempo de treinamento é **1,6%** do tempo de treinamento do algoritmo *SMO* executado sem paralelismo nem *pipelining*, para a frequência da placa de desenvolvimento de 50MHz ao custo de um aumento de 5% para 23% de elementos lógicos do componente usados. Apesar disso, a quantidade de memória usada não varia, pois depende do número de exemplos, do número de dimensões das entradas e da

quantidade de bits para representar números reais e não da quantidade de pares otimizados por iteração.

Na Tabela 6.6 são mostradas a potência e o consumo de energia para as implementações dos algoritmos. Na primeira coluna são listadas as arquiteturas, na segunda a potência dinâmica a 50 MHz, na terceira a estática e na quarta a energia total consumida.

A otimização de múltiplos pares aumenta o consumo de potência dinâmica, assim como também o uso de paralelismo e/ou *pipelining*. Não obstante, a potência estática não varia, e o consumo de energia é diminuído, pois o tempo de treinamento diminui mais do que aumenta a potência dinâmica.

Na Figura 6.5 são representados graficamente os resultados de número de iterações, tempo de treinamento a 50MHz e à frequência máxima (f_{max}), para as arquiteturas sem *pipelining* nem paralelismo. Os gráficos exibem os resultados de treinar otimizando múltiplos pares, relativos aos do algoritmo SMO.

Tabela 6.6 - Potência e consumo de energia

Arquitetura	Potência dinâmica (mW)@50MHz	Potência estática (mW)	Energia total (mJ)@50MHz
SMO	21	80	42,6
SMO-pipe	38	80	12,5
SMO-par	22	80	89,8
SMO-pipepar	31	80	11,8
2P-SMO	22	80	121,4
2P-SMO-pipe	65	80	7,5
2P-SMO-par	25	80	50,5
2P-SMO-pipepar	38	80	6,1
3P-SMO	23	80	109,2
3P-SMO-pipe	93	80	5,7
3P-SMO-par	28	80	29,2
3P-SMO-pipepar	48	80	4,1
4P-SMO	24	80	102,4
4P-SMO-pipe	129	80	5,0
4P-SMO-par	32	80	22,6
4P-SMO-pipepar	53	80	3,1

Figura 6.5 - Resultados das arquiteturas sem *pipelining* em paralelismo para múltiplos pares otimizados, relativos aos do algoritmo SMO

A diminuição relativa do número de iterações é maior que a do tempo de treinamento quanto mais pares são otimizados. Isso decorre do aumento da complexidade computacional e do tempo de execução de cada iteração com o número de pares otimizados como acontece na implementação em *software*.

Nas Figuras 6.6, 6.7, e 6.8 são exibidos os gráficos do mesmo tipo que o da Figura 6.5 para as arquiteturas com *pipelining*, com paralelismo e com as duas técnicas combinadas, respectivamente. Os três gráficos mostram que a diminuição do tempo de treinamento a 50MHz é similar à do número de iterações. Isso é devido a que o *pipelining* e o paralelismo produzem tempos de execução de cada iteração que não variam quando é incrementada a quantidade de pares de coeficientes otimizados. No entanto, a diminuição relativa do tempo de treinamento à f_{max} é menor que a do número de iterações nas arquiteturas com paralelismo e combinada. Isso é devido a que a f_{max} é diferente para cada implementação, e nesses casos os resultados das sínteses mostraram diminuição dessa frequência para múltiplos pares otimizados.

Figura 6.6 - Resultados das arquiteturas com *pipelining* para múltiplos pares otimizados, relativos aos do algoritmo *SMO*

Figura 6.7 - Resultados das arquiteturas com paralelismo para múltiplos pares otimizados, relativos aos do algoritmo *SMO*

Figura 6.8 - Resultados das arquiteturas com *pipelining* e paralelismo para múltiplos pares otimizados, relativos aos do algoritmo SMO

Na Figura 6.9 é mostrado o efeito de acrescentar *pipelining* e/ou paralelismo na arquitetura do algoritmo SMO. É exibido o tempo de treinamento a 50 MHz e à f_{max} relativo ao do algoritmo SMO sem *pipelining* nem paralelismo. Como pode ser observado, as arquiteturas SMO com *pipelining* e SMO com *pipelining* e paralelismo reduzem mais o tempo de treinamento do que a arquitetura SMO com somente paralelismo.

No entanto, essa maior diminuição do tempo de treinamento é a expensas de maior quantidade de elementos lógicos e potência dinâmica como é mostrado na Figura 6.10, na qual são exibidos esses dois parâmetros de desempenho relativos aos obtidos com o algoritmo SMO sem *pipelining* nem paralelismo.

Desses dois gráficos conclui-se que a arquitetura do SMO com somente paralelismo é a que menos diminui o tempo de treinamento, mas também a que menos aumenta a quantidade de elementos lógicos e potência dinâmica. Por sua vez, as arquiteturas com somente *pipelining* e com *pipelining* e paralelismo diminuem aproximadamente na mesma proporção o tempo de treinamento, mas a última aumenta menos a quantidade de elementos lógicos e potência dinâmica. Isso porque a arquitetura implementada com somente *pipelining* possui maior quantidade de registradores que guardam resultados intermediários que são transferidos até o último estágio. Portanto, a arquitetura com somente *pipelining* não é conveniente de ser usada.

Figura 6.9 – Redução relativa do tempo de treinamento das arquiteturas quando incluído o *pipelining*, ou o paralelismo ou as duas técnicas

Figura 6.10 – Aumento relativo dos elementos lógicos e potência dinâmica das arquiteturas, quando incluído o *pipelining*, ou o paralelismo ou as duas técnicas

A partir dos dados das Tabelas 6.5 e 6.6, são selecionadas todas as arquiteturas que convêm ser usadas de acordo com o desempenho para a frequência de 50 Mhz. Na Tabela 6.7 são listadas as arquiteturas na primeira coluna, na segunda é mostrado o tempo de treinamento, na terceira a quantidade de elementos lógicos e na quarta a potência dinâmica. Pode ser observado que o uso de múltiplos pares é

conveniente quando implementado com paralelismo e *pipelining* devido a que somente o uso de múltiplos pares diminui em pouca proporção o tempo de treinamento.

Tabela 6.7 – Arquiteturas selecionadas e seus desempenhos

Arquitetura	Tempo de treinamento(s) @50MHz	Elementos lógicos (%)	Potência dinâmica (mW)@50MHz
<i>SMO</i>	1,412	5	21
<i>SMO-par</i>	0,889	6	22
<i>SMO-pipepar</i>	0,106	8	31
<i>2P-SMO-pipepar</i>	0,052	13	38
<i>3P-SMO-pipepar</i>	0,032	17	48
<i>4P-SMO-pipepar</i>	0,023	23	53

O erro de teste também foi achado para as arquiteturas *SMO*, *2P-SMO*, *3P-SMO* e *4P-SMO*, com o objetivo de conhecer a variação que poderia produzir a introdução de múltiplos pares. As arquiteturas foram simuladas em *BSV* e os Multiplicadores de Lagrange obtidos foram usados em *OCTAVE* para obter o erro. Os resultados são exibidos na Tabela 6.8, na qual, na primeira coluna são listadas as arquiteturas e na segunda o erro de teste.

Tabela 6.8 – Erro de teste de arquiteturas *SMO* e *MP-SMO*

Arquitetura	Erro de teste (%)
<i>SMO</i>	10,3
<i>2P-SMO</i>	10,3
<i>3P-SMO</i>	10,3
<i>4P-SMO</i>	10,3

Para o *benchmark Tic-tac-toe endgame*, a implementação em *hardware* do algoritmo *MP-SMO* não produz variações no erro de teste em relação à arquitetura *SMO*. Esta conclusão é a mesma à obtida para este *benchmark* com a implementação em *software* como mostrada na Tabela 6.3. Quando são incorporados *pipelining* e/ou paralelismo a qualquer uma dessas arquiteturas, os Multiplicadores de Lagrange não

variam, portanto o erro tampouco. A partir deste resultado e o da Tabela 6.4, na qual é comparado o erro de teste das implementações em *hardware* e *software* do algoritmo *SMO*, é possível aproximá-lo para a implementação em *hardware* (sem importar qual arquitetura será usada), realizando uma simulação em *software*.

6.3 RESUMO DO CAPÍTULO

Na primeira seção deste capítulo foram apresentados os resultados experimentais para a implementação em *software* do algoritmo *MP-SMO*. Foi testada para 7 *benchmarks* otimizando 2, 3 e 4 pares de coeficientes. Os resultados mostram que para todos os casos, o tempo de treinamento é reduzido em relação ao obtido com o algoritmo *SMO*.

Na segunda seção, os resultados experimentais apresentados corresponderam para à implementação em *hardware* do algoritmo *MP-SMO* e para um *benchmark*, o *Tic-tac-toe endgame*. Foram também otimizados 2, 3 e 4 pares e foram incorporadas as técnicas de *pipelining* e paralelismo. Os resultados mostram que o tempo de treinamento é reduzido a expensas de maior número de elementos lógicos usados e potência dinâmica.

7 CONCLUSÕES E TRABALHOS FUTUROS

7.1 CONCLUSÕES SOBRE O ALGORITMO *MP-SMO*

Neste trabalho foi apresentado o algoritmo *MP-SMO*, uma modificação do algoritmo *SMO* para o treinamento da *SVM*. Este algoritmo otimiza mais de um par de Multiplicadores de Lagrange a cada iteração, mantendo a solução analítica do algoritmo original. Também foi demonstrado que o algoritmo *MP-SMO* converge monotonicamente.

Os resultados experimentais mostraram que o tempo de treinamento da *SVM* diminuiu para todos os *benchmarks* testados, otimizando dois, três e quatro coeficientes quando comparado com o tempo de treinamento do algoritmo original. Estes testes foram realizados usando a *K-heurística* e a *F-heurística* adaptadas na seleção dos pares de coeficientes e as técnicas *shrinking* e *caching* em todos os algoritmos implementados. Para os melhores resultados obtidos com cada *benchmark* a diminuição do tempo de treinamento variou entre 22,5% e 42,8%.

7.2 TRABALHOS FUTUROS RELATIVOS AO ALGORITMO *MP-SMO*

Nesta seção são propostos os seguintes trabalhos futuros relativos ao algoritmo *MP-SMO*:

- 1) estudar diferentes formas de decidir a otimização ou não dos pares de coeficientes selecionados a partir do segundo par. Isto pelo fato de que em algumas iterações determina-se que algum dos pares não seja otimizado, sendo que pode ser conveniente sua otimização.

Uma possibilidade é a de otimizar todos os pares selecionados e depois com a informação resultante, determinar qual otimização deve ser desfeita.

Não obstante, deverá considerar-se o custo computacional de cada modificação realizada ao algoritmo, dado que a diminuição no número de iterações pode não compensar o aumento do tempo de execução de cada iteração.

2) Dado que para diferentes *benchmarks*, o melhor resultado foi obtido otimizando diferentes números máximos de pares de coeficientes, estudar a maneira de determinar essa quantidade para qualquer problema dado. Em particular, se a decisão pode ser realizada treinando com um subconjunto pequeno de exemplos. Com essa técnica, a determinação seria experimental e rápida.

3) Para aumentar a velocidade de execução do treinamento, desenvolver uma implementação em *software* usando programação paralela. Dado que a otimização de cada par é independente das outras, as mesmas podem ser executadas em paralelo em diferentes *cores* de processador. Também podem ser executados em paralelo os cálculos dos *kernels* para a atualização de dados. Dessa maneira, a diminuição relativa do tempo de treinamento é aproximada à do número de iterações (figuras 6.3 e 6.4).

7.3 CONCLUSÕES SOBRE A IMPLEMENTAÇÃO EM *HARDWARE* DO ALGORITMO *MP-SMO*

Neste trabalho foram apresentadas diferentes implementações em *hardware* dedicado dos algoritmos *SMO* e *MP-SMO* para o treinamento da *SVM*. As implementações foram realizadas para o *benchmark Tic-tac-toe endgame*, cujo conjunto de treinamento possui 958 exemplos com 9 atributos categóricos. O algoritmo *MP-SMO* foi implementado otimizando dois, três e quatro pares de coeficientes para diminuir o tempo de treinamento quando comparado com o algoritmo *SMO*. Foram realizadas diferentes arquiteturas, incluindo paralelismo e *pipelining*, para obter uma maior diminuição do tempo de treinamento. Segundo nosso conhecimento, estas são as primeiras implementações completas do algoritmo *SMO* em *hardware* dedicado.

Os resultados experimentais mostraram que a otimização de múltiplos pares, e o uso de paralelismo e *pipelining* aceleram o treinamento quando comparado com o da

arquitetura do algoritmo *SMO* sem a incorporação dessas técnicas. Para o melhor caso (4 pares, com *pipelining* e paralelismo), a velocidade de execução do treinamento aumentou em 61,4 vezes.

Uma comparação direta do tempo de treinamento destas arquiteturas com a da implementação anterior em *software* e *hardware* do algoritmo *SMO* de Pedersen e Schoeberl (21) não é possível. Isso pelo fato de que nesse trabalho os conjuntos de treinamento foram gerados artificialmente com atributos reais, e no nosso trabalho o conjunto de treinamento é o do *benchmark Tic-tac-toe*, de atributos categóricos. As famílias dos dispositivos *FPGAs* e as frequências de relógio também são diferentes. Não obstante, as implementações do cálculo dos *kernels*, a tarefa de maior tempo de execução, podem ser comparadas. No trabalho de Pedersen e Schoeberl (21), o cálculo de um *kernel* é realizado em 244 ciclos de relógio, enquanto em nossa arquitetura, devido à função *kernel* escolhida e às simplificações realizadas, é realizado em 12 ciclos de relógio (2 para a leitura dos dados e 10 para calcular o *kernel*). Como citado por Pedersen e Schoeberl, um problema da implementação em *software-hardware* é o fato de que a leitura dos dados domina a tarefa de cálculo do *kernel*, sendo realizada pelo *software*. Nesse trabalho, o cálculo dos *kernels* é realizado usando *pipelining*. Na nossa arquitetura do algoritmo *SMO*, além do *pipelining*, vários *kernels* são calculados em paralelo para diminuir mais o tempo de execução da sequência completa de cálculos de *kernels*. Nossa implementação inclui também a modificação do algoritmo, otimizando múltiplos pares para acelerar sua convergência.

Nas Tabelas 7.1 e 7.2 é apresentado um resumo das implementações em *hardware* do treinamento da *SVM*. Na segunda coluna da Tabela 7.1, são mostrados os algoritmos usados em cada implementação. Na terceira, o tipo de *kernel* usado, na quarta o número de ciclos de relógio para o cálculo do *kernel*. Na quinta coluna o conjunto de treinamento e na sexta o número de exemplos e o número e tipo de atributos. As implementações para exemplos com atributos reais são gerais, incluindo os casos de atributos binários.

Tabela 7.1: Resumo das implementações em *hardware* do treinamento da SVM (I)

Implementação	Algoritmo	<i>Kernel</i>	Cálculo do <i>kernel</i> (ciclos de relógio)	Conjunto de Treinamento	Número de exemplos – atributos
Anguita; Boni e Ridella (10)	Rede recorrente	Qualquer tipo	Não calcula	Equalização de canal de comunicações	32 - 2 Reais
Sheng Yu Peng; Minch e Hasler (20)	Rede de Projeção	Não linear, <i>RBF</i>	Cálculo analógico	Gerado artificialmente	4 - 2 Reais
Pedersen e Schoeberl (21)	<i>SMO</i>	Linear	244	Gerado artificialmente	60 – 2 Reais
4P-SMO-pipepar	MP-SMO	Não linear, Anguita et al. (38)	12	<i>Tic-tac-toe endgame benchmark</i>	958 – 9 categóricos

Na Tabela 7.2 são apresentadas as implementações na primeira coluna, na segunda o dispositivo usado, na terceira a relação entre área e o número de exemplos. Na quarta coluna é mostrada a frequência na qual foi testado o circuito, exceto para a primeira implementação, pelo fato de que a informação disponível é a frequência máxima. Na última coluna é exibido o melhor tempo de treinamento de cada implementação.

Tabela 7.2: Resumo das implementações em *hardware* do treinamento da SVM (II)

Implementação	Dispositivo	Área (Número de exemplos T)	Frequência do teste (MHz)	Tempo de treinamento (ms)
Anguita; Boni e Ridella (10)	FPGA – Virtex II	$A1 T^2 + A2 T + A3$	21,06 ($f_{\max}$)	6,7 ($@f_{\max}$)
Sheng Yu Peng; Minch e Hasler (20)	Analógico – simulado	$S1 T^2 + S2 T + S3$	-	0,4
Pedersen e Schoeberl (21)	FPGA – Cyclone	$P1 T + P2$	100	364
4P-SMO-pipepar	FPGA – Cyclone II	$H1 T + H2$	50	23

Pode ser observado que as duas primeiras implementações apresentam os melhores tempos de treinamento absolutos. Não obstante o número de exemplos é pequeno, 32 e 4 respectivamente, e a área nessas implementações possui uma relação quadrática com o número de exemplos. A implementação de Anguita; Boni e

Ridella (10) não inclui o cálculo de *kernels*, a tarefa de maior tempo de execução. Nossa implementação é a que possui o maior número de exemplos, não obstante pode ser aplicada somente em problemas com atributos categóricos. Apesar da impossibilidade de comparar diretamente as diferentes implementações, a nossa é a que possui a melhor relação tempo de treinamento sobre número de exemplos.

7.4 TRABALHOS FUTUROS RELATIVOS À IMPLEMENTAÇÃO EM *HARDWARE* DO ALGORITMO *MP-SMO*

Nesta seção são propostos os seguintes trabalhos futuros relativos ao algoritmo *MP-SMO* implementado em *hardware*:

1) Generalizar a implementação em *hardware* para atributos reais. Dessa maneira, o bloco *kernel* não pode ser simplificado da forma apresentada na Seção 3.1.2. Não obstante, pode ser usada a arquitetura proposta por Anguita et al. (38). Nessa arquitetura é usado um algoritmo do tipo *CORDIC* para o cálculo do *kernel*, usando somente operações soma e deslocamento. Segundo Anguita et al. para realizar esse cálculo são suficientes 16 ciclos de relógio (sem contar os ciclos de leitura dos dados), dessa maneira o erro de teste é mantido no valor mínimo. Não obstante, foram realizadas simulações em *OCTAVE* para o *benchmark Breast cancer* do repositório da *UCI* (22) para 7 ciclos de relógio, obtendo-se os valores mínimos do erro de teste. Para verificar que é possível implementar o *kernel* em *hardware* para essa quantidade de ciclos de relógio, uma implementação deste tipo foi realizada e simulada em *Bluespec*. No entanto, a leitura das componentes das entradas $\vec{x}_i$ pode implicar um número importante de ciclos de relógio.

2) Neste trabalho foi usada a condição $K(\vec{x}_i, \vec{x}_j) \ll 1, i \neq j$. A partir dessa aproximação foi simplificada a tarefa de otimização de pares de coeficientes e a avaliação da expressão para a decisão de otimizar ou não os pares de coeficientes a partir do segundo par (expressões (5.6) e (5.7) da Seção 5.3.1). Essas simplificações dependem do valor do parâmetro β do *kernel* (expressão (5.1)) escolhido pelo usuário. O valor do parâmetro β também determina o erro de teste.

Consequentemente, interessa poder ampliar a faixa de valores deste parâmetro para poder minimizar o erro de teste. Portanto, interessa generalizar a implementação em relação aos valores de β . Para isso duas situações devem ser estudadas:

- 2.1) estudar se a aproximação (5.4) pode seguir sendo aplicada. Caso contrário o cálculo do *kernel* deve ser incluído no cálculo de η_1 . Também deve ser realizada uma operação divisão na otimização, de acordo com a equação (5.5) para o cálculo do novo valor do segundo Multiplicador de Lagrange escolhido.

$$\eta_1 = 2 \quad (5.4)$$

$$\alpha_2^{new} = \alpha_2^{old} + y_2 (b_{low} - b_{up}) / 2 \quad (5.5)$$

De acordo com simulações realizadas para o *benchmark Breast cancer*, o algoritmo converge quando a aproximação (5.4) é usada mesmo para valores de $K(\bar{x}_{12}, \bar{x}_{11})$ próximos a 1.

- 2.2) quando a aproximação $K(\bar{x}_i, \bar{x}_j) \ll 1, i \neq j$ não é válida, a expressão (4.1) deve ser implementada. Para isso devem ser estudadas as formas de simplificar essa expressão. Por exemplo, eliminar o fator comum $(b_{upk} - b_{lowk})$ e aproximar os valores η_k e η_j a 2. As multiplicações podem ser inclusas nos cálculos dos *kernels* com o algoritmo *CORDIC* como proposto por Anguita et al. (38).

$$\begin{aligned} & -(b_{upk} - b_{lowk})^2 / (2\eta_k) + \\ & \sum_{\substack{j=1 \\ j\text{-par a} \\ \text{otimizar}}}^{j=k-1} \{ (K(\bar{x}_{j1}, \bar{x}_{k1}) - K(\bar{x}_{j1}, \bar{x}_{k2}) - K(\bar{x}_{j2}, \bar{x}_{k1}) + K(\bar{x}_{j2}, \bar{x}_{k2})) (b_{upj} - b_{lowj}) (b_{upk} - b_{lowk}) / (\eta_j \eta_k) \} < 0 \end{aligned} \quad (4.1)$$

3) na implementação em *software* foram usadas as técnicas de *shrinking* e *caching* para diminuir o tempo de treinamento. Como trabalho futuro propõe-se implementar essas técnicas em *hardware*. Essa nova diminuição do tempo de treinamento terá um novo custo em área pelo fato de que: 1) para o *shrinking* é

preciso incorporar comparadores (expressões (3.12) e (3.13)). 2) para o *caching* é preciso armazenar em memória os valores calculados da função *kernel* .

REFERÊNCIAS

- 1 HAYKIN, S. **Neural networks, a comprehensive foundation**. 2nd ed. Hamilton, Ontario, Canada: Prentice Hall, 1999. 823 p.

- 2 CORTES, C.; VAPNIK, V. **Support-vector networks**. Machine Learning, Netherlands, v. 20, n. 3, pp. 273-297, 1995.

- 3 GUYON, I., SVM application list, Available from: <http://www.clopinet.com/isabelle/Projects/SVM/applist.html>, Last access: 5 april 2009.

- 4 ROMERO, E.; ACOSTA, R.; STRUM, M; JIANG CHAU WANG. **Support Vector Machine Coverage Driven Verification for Communication Cores**. In: The 17th IFIP/IEEE International Conference on Very Large Scale Integration, 2009, Florianópolis, Brazil. **VLSISOC 2009, accepted**.

- 5 BURGESS, C. J. C. **A Tutorial on support vector machines for pattern recognition**. Data Mining and Knowledge Discovery, Hingham, MA, USA, v. 2, n. 2, pp. 121-167, 1998.

- 6 GENOV, R.; CAUWENBERGHS, G. **Kerneltron: support vector machine in silicon**. Neural networks, IEEE Transactions on, Piscataway, NJ, USA, v. 14, n. 5, pp. 1426-1434, 2003.

- 7 CAUWENBERGHS, G.; EDWARDS, R. T.; YUNBIN DENG; GENOV, R.; LEMONDS, D. **Neuromorphic processor for real-time biosonar object detection**. In: Acoustics, Speech, and Signal Processing. IEEE International Conference on, 2002, Orlando, Florida, USA. **ICASSP'02 Proceedings**. Orlando, Florida, USA: IEEE, 2002. pp. 3984-3987.

- 8 DUMAIS, S.; PLATT, J.; HECKERMAN, D.; SAHAMI, M. **Inductive learning algorithms and representations for text categorization**. In: Conference on Information and Knowledge Management, 1998, Bethesda, Maryland, USA. **Proceedings of the seventh international ACM-CIKM**. Bethesda, Maryland, USA: ACM, 1998. pp. 148-155.

- 9 VAPNIK, V. **Estimation of dependences based on empirical data - Empirical Inference Science**. 2nd ed. New York, NY, USA: Springer, Reprint of the 1982 edition - Afterword of 2006. 505 p.

10 ANGUITA, D.; BONI, A.; RIDELLA, S. **A digital architecture for support vector machines: theory, algorithm, and FPGA implementation.** Neural networks, IEEE Transactions on, Piscataway, NJ, USA, v. 14, n. 5, pp. 993-1009, 2003.

11 JING YANG; XUE YANG; JIANPEI ZHANG. **A parallel multi-class classification support vector machine based on sequential minimal optimization.** In: Computer and Computational Sciences. First International Multi-Symposiums on, 2006, Hangzhou, Zhejiang, China. **IMSCCS'06 Proceedings.** Washington, DC, USA: IEEE, 2006. pp. 443-446.

12 MANGASARIAN, O. L.; MUSICANT, D. R. **Lagrangian support vector machines.** The Journal of Machine Learning Research, Cambridge, MA, USA, v. 1, n. 1, pp. 161-177, 2001.

13 JOACHIMS, T. **Training linear SVMs in linear time.** In: International Conference on Knowledge Discovery and Data Mining, 2006, Philadelphia, PA, USA. **Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.** New York, NY, USA: ACM, 2006. pp.

14 PLATT, J. C. Fast training of support vector machines using sequential minimal optimization. In: SCHÖLKOPF, B.; BURGESS, C. J. C.; SMOLA, A. J. **Advances in kernel methods: support vector learning.** 1st. ed. Cambridge, MA, USA: MIT Press, 1999. Chap. 2, pp. 185-208.

15 RUSZCZYNSKI, A. **Nonlinear optimization.** 1 ed. Princeton, NJ, USA: Princeton University Press, 2006. 464 p.

16 KEERTHI, S. S.; SHEVADE, S. K.; BHATTACHARYYA, C.; MURTHY K. R. K. **Improvements to Platt's SMO algorithm for SVM classifier design.** Neural Computation, Cambridge, MA, USA, v. 13, n. 3, pp. 637-649, 2001.

17 RONG-EN FAN; PAI-HSUEN CHEN; CHIH-JEN LIN. **Working set selection using second order information for training support vector machines.** The Journal of Machine Learning Research, Cambridge, MA, USA, v. 6, n. 1, pp. 1889-1918, 2005.

18 DURDANOVIC, I.; COSATTO, E.; GRAF, H. P. Large-scale parallel SVM implementation. In: BOTTOU, L; CHAPELLE, O.; DECOSTE, D.; WESTON, J. **Large-Scale Kernel Machines.** 1st. ed. Cambridge, MA, USA; London, England: The MIT Press, 2007. Chap. 5, pp. 416.

19 BIASI, I.; BONI, A.; ZORAT, A. **A reconfigurable parallel architecture for SVM classification**. In: Neural Networks. 2005 IEEE International Joint Conference on, 2005, Montreal, QC, Canada. **IJCNN'05 Proceedings**. Piscataway, NJ, USA: IEEE, 2005. pp. 2867-2872.

20 SHENG-YU PENG; MINCH, B. A.; HASLER, P. **Analog VLSI implementation of support vector machine learning and classification**. In: Circuits and Systems. IEEE International Symposium on, 2008, Seattle, WA, USA. **ISCAS'08 Proceedings**. Stoughton, WI, USA: The Printing House, 2008. pp. 860-863 CD-ROM.

21 PEDERSEN, R.; SCHOEBERL, M. **An embedded support vector machine**. In: Intelligent Solutions in Embedded Systems, International Workshop on, 2006, Vienna, Austria. **WISES'06 Proceedings**. Austria: IEEE, 2006. pp. 1-11.

22 ASUNCION, A.; NEWMAN, D. J., UCI Machine Learning Repository, Available from: <http://www.ics.uci.edu/~mllearn/MLRepository.html>, Last access: December 2008.

23 CHUN F. HSU. **Design and implementation of hardware-SVM-based shot detection system**. 2007. 52 p. Thesis (Ph. D.) - National Taiwan University. Taipei, Taiwan, 2005.

24 DOMINGUEZ-LOPEZ, J. A.; DAMPER, R. I.; CROWDER, R. M.; HARRIS, C. J. **Hybrid neurofuzzy online learning for optimal grasping**. In: Machine Learning and Cybernetics, International Conference on, 2003, Xi-an, China. **Proceedings**. Piscataway, NJ USA: IEEE, 2003. pp. 803-808.

25 MENG JOO ER; CHANG DENG. **Obstacle avoidance of a mobile robot using hybrid learning approach**. Industrial Electronics, IEEE Transactions on, Piscataway, NJ USA, v. 52, n. 3, pp. 898-905, 2005.

26 XUE-NING WANG; WEI CHEN; DA-XUE LIU; TAO WU; HAN-GEN HE. **The optimality analysis of hybrid reinforcement learning combined with SVMs**. In: Intelligent Systems Design and Applications. Sixth International Conference on, 2006, Jinan, China. **ISDA '06 Proceedings**. Piscataway, NJ, USA: IEEE, 2006. pp. 936 - 941.

27 LAAKSONEN, J.; AKSELA, M.; OJA, E. **Adaptive local subspace classifier in on-line recognition of handwritten characters**. In: Neural Networks. International Joint Conference on, 1999, Washington, D. C. USA. **IJCNN'99 Proceedings**. [S.l.]: IEEE, 1999. pp. 2812-2815 CD-ROM.

28 MCINERNEY, S.; REILLY, R. B. **Hybrid multiplier/CORDIC unit for online handwriting recognition**. In: Acoustics, Speech, and Signal Processing. IEEE International Conference on, 1999, Phoenix, Arizona USA. **ICASSP'99 Proceedings**. Piscataway, NJ USA: IEEE, 1999. pp. 1909-1912.

29 MORE, J. J.; WRIGHT, S. J. **Optimization software guide**. 2nd ed. Philadelphia, PA, USA: SIAM, 1993. 154 p.

30 VANDERBEI, R. J. **LOQO: an interior point code for quadratic programming**. Optimization Methods and Software, Abingdon, England, v. 11, n. 1 - 4, pp. 451-484, 1999.

31 BOTTOU, L.; LIN, C. J. Support vector machine solvers. In: BOTTOU, L.; CHAPPELLE, O.; DECOSTE, D.; WESTON, J. **Large-Scale Kernel Machines**. 1st. ed. Cambridge, MA, USA; London, England: The MIT Press, 2007. Chap. 1, pp. 416.

32 OSUNA, E.; FREUND, R.; GIROSI, F. **An improved training algorithm for support vector machines**. In: Neural Networks for Signal Processing VII, 1997, Amelia Island Plantation, FL, USA. **NNSP '97 Proceedings**. Piscataway, NJ, USA: IEEE, 1997. pp. 276-285.

33 JOACHIMS, T. Making large-scale support vector machine learning practical. In: SCHÖLKOPF, B.; BURGESS, C. J. C.; SMOLA, A. J. **Advances in kernel methods: support vector learning**. 1st. ed. Cambridge, MA, USA: MIT Press, 1999. Chap. 2, pp. 169-184.

34 CHIH-CHUNG CHANG, CHIH-JEN LIN **LIBSVM: a Library for Support Vector Machines**, Department of Computer Science, National Taiwan University, Taipei, Taiwan.: 2008, <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.

35 GLASMACHERS, T.; IGEL, C. **Maximum-gain working set selection for SVMs**. The Journal of Machine Learning Research, Cambridge, MA, USA, v. 7, n. 1, pp. 1437 - 1466, 2006.

36 LIANG ZHOU; FEN XIA; YANWU YANG. **An improved caching strategy for training SVMs**. In: International Conference on Intelligent Systems and Knowledge Engineering, 2007, Chengdu, China. **ISKE 2007 Proceedings**. Paris, France - Amsterdam, Netherlands: Atlantis Press, 2007. pp. online and cd-rom.

37 CHIH-JEN LIN. **Asymptotic convergence of an SMO algorithm without any assumptions**. Neural networks, IEEE Transactions on, Piscataway, NJ, USA, v. 13, n. 1, pp. 248 - 250, 2002.

38 ANGUITA, D.; PISCHIUTTA, S.; RIDELLA, S.; STERPI, D. **Feed-forward support vector machine without multipliers**. Neural networks, IEEE Transactions on, Piscataway, NJ, USA, v. 17, n. 5, pp. 1328 - 1331, 2006.

39 ANGUITA, D.; RIDELLA, S.; ROVETTA, S. **Circuital implementation of support vector machines**. ELECTRONICS LETTERS, Stevenage, United Kingdom, v. 34, n. 16, pp. 1596-1597, 1998.

40 GOMES FILHO, J. **Aplicação de técnicas de reconfiguração dinâmica a projeto de algoritmo SVM (Support Vector Machine)**. São Paulo, Universidade de São Paulo, 13 de abril de 2009, Qualificação de mestrado.

41 HERNANDEZ, R. A.; STRUM, M.; WANG JIANG CHAU, GONZALEZ, J. A. Q. **The Multiple Pairs SMO: a modified SMO algorithm for the acceleration of the SVM training**. In: The 2009 International Joint Conference on Neural Networks, 2009, Atlanta, GA, USA. **IJCNN 2009 Conference Proceedings**. IEEE, pp. 1221-1228, CD.

42 HERNANDEZ, R. A.; STRUM, M.; WANG JIANG CHAU, GONZALEZ, J. A. Q. **A VLSI implementation of the Multiple Pairs SMO algorithm for the acceleration of the SVM training**. In: XV Workshop Iberchip, 2009, Buenos Aires, Argentina. **IWS'2009 Proceedings**. Fernando G. Tinetti, pp. 204-209.

43 KOHAVI, R.; **A study of cross-validation and bootstrap for accuracy estimation and model selection**. In: International Joint Conference on Artificial Intelligence, 1995, Montreal, Canada. **IJCAI 1995 Proceedings**. San Mateo, Ca, USA: Morgan Kaufmann Publishers, Inc., 1995. pp. 1137-1143.

44 BRONSHTEN, I.; SEMENDIAEV, K. **Manual de Matemáticas para Ingenieros y Estudiantes**. Segunda ed. Moscú: MIR, 1973. 694 p.

45 PEDREGAL, P. **Texts in Applied Mathematics: Introduction to Optimization**. 1st. ed. New York, USA: Springer-Verlag, 2004. 245 p.

ANEXO – DESENVOLVIMENTO MATEMÁTICO DA SVM

A.1 INTRODUÇÃO

Neste ANEXO é apresentado o desenvolvimento matemático do problema SVM, partindo da formulação geométrica para o caso linear e separável e chegando até o problema QP do caso geral, o não linear e não separável. A principal referência deste ANEXO é o tutorial sobre SVM de Burges (5). Esse tutorial apresenta o desenvolvimento matemático da SVM, sendo aqui replicado e complementado com a descrição detalhada de seus passos. Tais demonstrações complementares são baseadas em material bibliográfico de matemática geral (44) e de otimização em particular (15, 45).

A.2 CASO LINEAR E SEPARÁVEL

A.2.1 Problema SVM

Para introduzir a análise da SVM na fase de treinamento, é conveniente começar com o caso mais simples, o linear e separável (5). A aprendizagem é realizada com um conjunto de T exemplos de treinamento constituídos pelas entradas $(\bar{x}_i)$, e pelas saídas correspondentes $y_i \in \{-1, 1\}$. Quando na suposição de que todos os exemplos com saídas positivas podem ser separados de todos os exemplos com saídas negativas por um hiperplano, o caso é separável e linear, como mostrado na Figura A.1. Para um ponto X pertencente a esse hiperplano a seguinte igualdade é satisfeita:

$$\vec{w} \cdot \vec{x} - b = 0 \quad (\text{A.1})$$

Sendo $\vec{x}$ o vetor $\overrightarrow{OX}$, $\vec{w}$ um vetor normal ao hiperplano, e $|b| / \|\vec{w}\|$ a distância do hiperplano à origem. Isso implica que, conhecidos o vetor $\vec{w}$ e a constante b , é determinada a equação do hiperplano que separa os exemplos positivos dos negativos.

Figura A.1: Exemplos de treinamento e hiperplano de separação

O objetivo é achar aquele hiperplano que maximiza o parâmetro **margem**. A margem é definida como a soma das distâncias mínimas de cada classe a esse hiperplano ótimo, d_+ e d_- com d_+ igual a d_- . (5). Para começar a descrever um método que permita chegar ao objetivo anterior, consideremos que com uma escala apropriada os exemplos de treinamento $\{\vec{x}_i, y_i\}$ satisfazem:

$$\vec{w} \cdot \vec{x}_i - b \geq 1, \text{ para } y_i = 1 \quad (\text{A.2})$$

$$\vec{w} \cdot \vec{x}_i - b \leq -1, \text{ para } y_i = -1 \quad (\text{A.3})$$

As desigualdades anteriores podem ser expressas em uma só equivalente:

$$y_i(\vec{w} \cdot \vec{x}_i - b) - 1 \geq 0, \text{ para } i = 1..T \quad (\text{A.4})$$

Os exemplos positivos que satisfazem a igualdade em A.4 estão em um hiperplano cuja distância à origem é $|1+b| / \|\vec{w}\|$. Esses exemplos são os mais próximos ao hiperplano de separação e d_+ é igual a $1/\|\vec{w}\|$. Analogamente os exemplos negativos que satisfazem a igualdade estão em um hiperplano que define a distância d_- igual a $1/\|\vec{w}\|$. Portanto, a margem é igual a $2/\|\vec{w}\|$. Considera-se a margem como a função f_m das componentes de $\vec{w} = (w_1, w_2, \dots, w_d)$. Procura-se, portanto, maximizar $f_m(w_1, w_2, \dots, w_d)$, com restrições dadas por (A.4). O problema de obter $\vec{w}$ e b que determinam o hiperplano de separação com margem máxima é expresso como:

Problema 1:

$$\max \{f_m(w_1, w_2, \dots, w_d) = 2/\|\vec{w}\|\} \quad (\text{A.5})$$

com restrições:

$$g_i(w_1, w_2, \dots, w_d, b) = y_i(\vec{w} \cdot \vec{x}_i - b) - 1 \geq 0, \text{ para } i = 1..T \quad (\text{A.4})$$

A denominação Problema 1 é consequência de ser transformado sucessivamente em outros problemas equivalentes que serão denominados Problema 2, Problema 3...etc. Procura-se em cada transformação obter expressões mais simples. Maximizar f_m é equivalente a minimizar $\|\vec{w}\|$. Devido à conveniência posterior no desenvolvimento matemático das equações, o objetivo é minimizar a função:

$$f(w_1, w_2, \dots, w_d) = 1/2 \|\vec{w}\|^2 \quad (\text{A.6})$$

As restrições dadas por (A.4) são mantidas e pode ser observado que a função (A.6) é diferenciável para todo $\vec{w}$, enquanto que $2/\|\vec{w}\|$ não é diferenciável quando alguma das componentes de $\vec{w}$ é igual a zero. Essa propriedade é uma das vantagens que apresenta a nova função a minimizar, dado que os métodos para obter o mínimo dessa função (por exemplo SMO) incluem o uso de derivadas. Portanto, a solução do Problema 1 é a mesma que a do Problema 2.

Pelo fato de que cada restrição do Problema 2 relaciona todas as variáveis, a resolução apresenta importante dificuldade **(5)**. Para simplificar o problema é introduzido um conjunto de coeficientes chamados de Multiplicadores de Lagrange **(2, 5)**, usados para achar mínimos ou máximos com restrições.

Problema 2:

$$\min \{f(w_1, w_2, \dots, w_d) = 1/2 \|\vec{w}\|^2\} \quad (\text{A.7})$$

Restrições:

$$g_i(w_1, w_2, \dots, w_d, b) = y_i(\vec{w} \cdot \vec{x}_i - b) - 1 \geq 0, \text{ para } i = 1..T \quad (\text{A.4})$$

A.2.2 Multiplicadores de Lagrange

O Método dos Multiplicadores de Lagrange **(44)** proporciona uma condição necessária para poder achar mínimos ou máximos de uma função F de r variáveis $x, y, \dots, t$ para as quais são satisfeitas N restrições de igualdade. Dado o problema:

$$\min \text{ ou } \max \{F(x, y, \dots, t)\} \quad (\text{A.8})$$

Com N restrições:

$$G_i(x, y, \dots, t) = 0; i = 1..N \quad (\text{A.9})$$

A condição necessária para que $(x_1, y_1, \dots, t_1)$ seja solução do problema dado por (A.8) e (A.9) é que $(x_1, y_1, \dots, t_1, \alpha_{11}, \alpha_{21}, \dots, \alpha_{N1})$ seja mínimo, máximo ou ponto de inflexão da função Ψ das $r + N$ variáveis $x, y, \dots, t, \alpha_1, \alpha_2, \dots, \alpha_N$. Sendo Ψ :

$$\Psi(x, y, \dots, t, \alpha_1, \alpha_2, \dots, \alpha_N) = F(x, y, \dots, t) - \sum_{i=1}^{i=N} \alpha_i G_i(x, y, \dots, t) \quad (\text{A.10})$$

A função Ψ é chamada de **Lagrangeano** e $\alpha_1, \alpha_2, \dots, \alpha_N$, **Multiplicadores de Lagrange (44)**. Como pode ser visto em (A.10), há um multiplicador de Lagrange por cada restrição em (A9). No Problema 2 procura-se achar o mínimo de uma função com restrições de desigualdade no lugar de restrições de igualdade. Para este tipo de problema usaremos a função Lagrangeana e a equivalência do Problema 2 com os chamados **Problema Primal** e **Problema Dual (15)**.

A.2.3 Problemas Primal e Dual

Dado o seguinte problema de Programação Matemática (minimização):

$$\min\{f(x_1 \dots x_r)\} \quad (\text{A.11})$$

Com as restrições:

$$g_i(x_1 \dots x_r) \geq 0; i = 1 \dots m \quad (\text{A.12})$$

$$h_j(x_1 \dots x_r) = 0; j = 1 \dots p \quad (\text{A.13})$$

O Lagrangeano apresenta a forma:

$$L_1(x_1 \dots x_r, \alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p) = f(x_1 \dots x_r) - \sum_{i=1}^{i=m} \alpha_i g_i(x_1 \dots x_r) - \sum_{j=1}^{j=p} \mu_j h_j(x_1 \dots x_r) \quad (\text{A.14})$$

Os coeficientes α_i são positivos ou iguais a zero e considerando o Lagrangeano como função das variáveis $x_1 \dots x_r, \alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p$, define-se a **função Primal** associada ao problema anterior como **(15)**:

$$L_p(x_1 \dots x_r) = \sup_{\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p} L_1(x_1 \dots x_r, \alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p) \quad (\text{A.15})$$

Sendo $\sup_{\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p}$ o supremum⁹ de L_1 em relação a $\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p$. Define-se também a **função Dual** associada ao problema anterior como **(15)**:

$$L_D(\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p) = \inf_{x_1 \dots x_r} L_1(x_1 \dots x_r, \alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p) \quad (\text{A.16})$$

Sendo $\inf_{x_1 \dots x_r}$ o infimum¹⁰ de L_1 em relação a $x_1 \dots x_r$. Tendo estas duas funções, o **problema Primal**¹¹ é definido como **(15)**:

$$\min_{x_1 \dots x_r} L_p(x_1 \dots x_r) \quad (\text{A.17})$$

Enquanto que o **problema Dual** é definido como **(15)**:

$$\max_{\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p} L_D(\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p) \quad (\text{A.18})$$

Pode ser demonstrado que o problema de Programação Matemática dado por (A.11), (A.12) e (A.13) é equivalente ao seu problema Primal **(15)**. Por sua vez, se as funções $f(x_1 \dots x_r)$, $g_i(x_1 \dots x_r)$ são convexas¹² e, $h_j(x_1 \dots x_r)$ é uma função afim¹³, o problema Dual é equivalente ao problema Primal e portanto ao Problema dado por (A.11), (A.12) e (A.13) **(15)**. O problema Dual é expresso como a maximização do Lagrangeano em relação às variáveis $\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p$, anulando as primeiras derivadas de L_1 em relação às variáveis $x_1 \dots x_r$, para determinar o infimum de L_1 em relação a $x_1 \dots x_r$, e com a restrição de que os Multiplicadores de Lagrange sejam positivos **(15)**.

⁹ Supremum da função L em relação a $\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p$ é o máximo valor de L em relação às variáveis $\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p$ para um certo valor dado das variáveis $x_1 \dots x_r$.

¹⁰ Infimum da função L_1 em relação a $x_1 \dots x_r$ é o mínimo valor de L_1 em relação às variáveis $x_1 \dots x_r$ para um certo valor dado das variáveis $\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p$.

¹¹ Sobre a notação usada: $\min_{x_1 \dots x_r} L_p(x_1 \dots x_r)$, significa que o problema é achar os valores de $x_1 \dots x_r$ que minimizam L_p .

¹² Um conjunto $K \subset \mathbb{R}^n$ é convexo se dados dois pontos pertencentes a K , os pontos do segmento que os une também pertencem a K . Uma função $f: K \subset \mathbb{R}^n \rightarrow \mathbb{R}$ é convexa, se K é convexo e se dados dois pontos pertencentes a K , x e y : $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$, para todo $t \in [0, 1]$ **(45)**.

¹³ Uma função afim é uma função linear somada a um deslocamento dado por uma constante.

Problema Dual:

$$\max_{\alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p} L_1(x_1 \dots x_r, \alpha_1 \dots \alpha_m, \mu_1 \dots \mu_p) \quad (\text{A.19})$$

$$\partial L_1 / \partial x_i = 0; i = 1 \dots r \quad (\text{A.20})$$

$$\alpha_i \geq 0; i = 1 \dots m \quad (\text{A.21})$$

A.2.4 Aplicação dos Problemas Primal e Dual à SVM

Construindo o Lagrangeano para o Problema 2: a função $f(x, y, \dots, t)$ é substituída pela função $f(w_1, w_2, \dots, w_d, b) = \frac{1}{2} \|\vec{w}\|^2$. Enquanto que as funções $g_i(x, y, \dots, t)$, pelas funções $g_i(w_1, w_2, \dots, w_d, b) = y_i(\vec{x}_i \cdot \vec{w} - b) - 1$. Dessa maneira:

$$L_1 = \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{i=T} \alpha_i (y_i (\vec{x}_i \cdot \vec{w} - b) - 1) \quad (\text{A.22})$$

No interior da somatória em (A.22) há dois termos. Portanto, essa somatória pode ser dividida em duas e a nova expressão de L_1 é então:

$$L_1 = \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{i=T} \alpha_i y_i (\vec{x}_i \cdot \vec{w} - b) + \sum_{i=1}^{i=T} \alpha_i \quad (\text{A.23})$$

Aplicando a formulação do problema Primal o Problema 2 é transformado no Problema 3 (5):

Problema 3:

$$\min_{w_1 \dots w_d, b} \left\{ \sup_{\alpha_1 \dots \alpha_T} (L = 1/2 \|\vec{w}\|^2 - \sum_{i=1}^{i=T} \alpha_i y_i (\vec{x}_i \cdot \vec{w} - b) + \sum_{i=1}^{i=T} \alpha_i) \right\} \quad (\text{A.24})$$

$$\alpha_i \geq 0; i = 1 \dots T \quad (\text{A.25})$$

No Problema 2, a função objetivo $f(w_1, w_2, \dots, w_d)$ é convexa e as restrições dadas por (A.4) também. Portanto o Problema 3 é equivalente ao Problema Dual.

Dual do Problema 3 **(5)**:

$$\max_{\alpha_i} (L_1 = 1/2 \|\vec{w}\|^2 - \sum_{i=1}^{i=T} \alpha_i y_i (\vec{x}_i \cdot \vec{w} - b) + \sum_{i=1}^{i=T} \alpha_i) \quad (\text{A.26})$$

$$\partial L_1 / \partial w_t = 0; t = 1 \dots d \quad (\text{A.27})$$

$$\partial L_1 / \partial b = 0 \quad (\text{A.28})$$

$$\alpha_i \geq 0; i = 1 \dots T \quad (\text{A.25})$$

Depois de derivar L_1 em relação às componentes de $\vec{w}$ e igualar a zero como indica a equação (A.27), a seguinte expressão do vetor $\vec{w}$ é obtida **(5)**:

$$\vec{w} = \sum_{i=1}^{i=T} y_i \alpha_i \vec{x}_i \quad (\text{A.29})$$

Por sua vez, a anulação da derivada de L_1 em relação a b produz a seguinte igualdade **(5)**:

$$\sum_{i=1}^{i=T} y_i \alpha_i = 0 \quad (\text{A.30})$$

Substituindo (A.29) e (A.30) em (A.26), o problema dual pode ser formulado da seguinte maneira:

Problema 4:

$$\max_{\alpha_i} \left\{ L_1 = \sum_{i=1..T} \alpha_i - 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \bar{x}_i \cdot \bar{x}_j \right\} \quad (A.31)$$

Restrições:

$$\sum_{i=1}^{i=T} y_i \alpha_i = 0 \quad (A.30)$$

$$\alpha_i \geq 0; i = 1...T \quad (A.25)$$

Se a função objetivo em (A.31) for multiplicada por -1, o problema de achar o máximo é transformado no problema de achar o mínimo. Esse tipo de modificação é conveniente pelo fato de que os métodos encontrados na bibliografia para resolver problemas de otimização são desenvolvidos para achar mínimos, sendo inclusive sugerido na bibliografia especializada **(45)**. A transformação de (A.31) é realizada da seguinte maneira:

$$\begin{aligned} \max_{\alpha_i} \left\{ L_1 = \sum_{i=1..T} \alpha_i - 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \bar{x}_i \cdot \bar{x}_j \right\} = \\ - \min_{\alpha_i} \left\{ -L_1 = L_2 = 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \bar{x}_i \cdot \bar{x}_j - \sum_{i=1..T} \alpha_i \right\} \end{aligned} \quad (A.32)$$

Com essa transformação, os multiplicadores de Lagrange α_i que minimizam a função L_2 são os mesmos que maximizam a função L_1 . O Problema 4 é portanto, transformado no seguinte problema:

Problema 5:

$$\min_{\alpha_i} \{L_2 = 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \bar{x}_i \cdot \bar{x}_j - \sum_{i=1..T} \alpha_i\} \quad (A.33)$$

Restrições:

$$\sum_{i=1}^{i=T} y_i \alpha_i = 0 \quad (A.30)$$

$$\alpha_i \geq 0; i = 1...T \quad (A.25)$$

Esta formulação do problema possui a vantagem de que se procura minimizar L_2 com restrições sobre os Multiplicadores de Lagrange, sendo mais simples que as restrições originais sobre $\vec{w}$ e b . Outra vantagem é que os vetores correspondentes às entradas dos exemplos aparecem na expressão de L_2 na forma de produtos internos $\bar{x}_i \cdot \bar{x}_j$, a qual permitirá que o problema seja generalizado posteriormente para o caso não linear.

Portanto, para o caso linear e separável é necessário implementar um método que resolva o Problema 5 de forma tal que sejam obtidos os multiplicadores de Lagrange α_i . Depois de obter os Multiplicadores de Lagrange, é possível calcular $\vec{w}$ através de (A.29), e depois de obter $\vec{w}$, é possível obter b para qualquer exemplo cujo multiplicador de Lagrange associado não seja nulo, dado que só nesse caso vale a igualdade na expressão (A.4).

$$\vec{w} = \sum_{i=1}^{i=T} y_i \alpha_i \bar{x}_i \quad (A.29)$$

$$b = \vec{w} \cdot \bar{x}_k - y_k; \text{ para } \alpha_k > 0 \quad (A.34)$$

Obtidos $\vec{w}$ e b , o hiperplano que separa os exemplos positivos dos negativos com margem máxima é determinado, e portanto, finaliza a fase de treinamento da SVM.

Os Multiplicadores de Lagrange α_i que são nulos podem ser eliminados da igualdade (A.29), e correspondem a exemplos alocados a uma distância do hiperplano de separação maior ou igual que d_+ . Não obstante, os Multiplicadores de Lagrange α_i maiores a zero correspondem a exemplos alocados a uma distância igual a d_+ , e determinam o hiperplano de separação **(5)**. Os $\bar{x}_i$ associados a eles são denominados **vetores de suporte**, dado que são os que determinam $\bar{w}$ e b , portanto, o hiperplano de separação das duas classes. Essa denominação de vetores de suporte dá origem ao nome da máquina de aprendizado como **Máquina de Vetores de Suporte**. Terminada a fase de treinamento, na qual o hiperplano de separação caracterizado por $\bar{w}$ e b é obtido, a fase de teste é realizada, obtendo-se a saída para uma entrada $\bar{x}$, avaliando a expressão dada por (A.35) **(14)**.

$$\text{sgn}(u = \bar{w} \cdot \bar{x} - b) \quad (\text{A.35})$$

Na expressão (A.35) a função $\text{sgn}(u)$ é a função sinal. Em muitos casos reais, na fase de treinamento a separação de todos os exemplos positivos de todos os exemplos negativos não é possível, portanto, cabe agora estudar o caso linear e não separável.

A.3 CASO LINEAR E NÃO SEPARÁVEL

Na Figura A.2 é mostrado um caso, no qual, para qualquer hiperplano de separação, há exemplos alocados no semiespaço incorreto.

Figura A.2: Representação dos exemplos não separáveis.

Procura-se, portanto, o hiperplano de separação que maximize a margem como anteriormente e minimize a quantidade de exemplos no semiespaço incorreto, ou também chamada de erro de treinamento **(5)**. Para modelar esta situação acrescentam-se as variáveis ξ_i $i = 1..T$ nas restrições (A.2) e (A.3) **(5)**, de forma tal que:

$$\vec{w} \cdot \vec{x}_i - b \geq 1 - \xi_i, \text{ para } y_i = 1 \quad (\text{A.36})$$

$$\vec{w} \cdot \vec{x}_i - b \leq -1 + \xi_i, \text{ para } y_i = -1 \quad (\text{A.37})$$

$$\xi_i \geq 0; i = 1..T \quad (\text{A.38})$$

Como pode ser observado em (A.36) e (A.37), a variável ξ_i introduzida para cada exemplo, representa no caso de ser $\xi_i > 1$, que o exemplo correspondente aloca-se no semiespaço incorreto. Portanto, cada vez que ξ_i supera o valor 1, um erro ocorre no treinamento. Dessa maneira, $\sum_{i=1..T} \xi_i$ é o limite superior do número de erros no treinamento. O número de erros no treinamento influencia no número de erros na fase de teste de acordo com a teoria da estatística do aprendizado **(1, 5)**. O fato de aceitar certo número de erros no treinamento pode ser refletido pagando um incremento na função objetivo a ser minimizada. A nova função objetivo $f_2(w_1, \dots, w_d, \xi_1, \dots, \xi_T)$ é agora:

$$f_2(w_1, w_2, \dots, w_d, \xi_1, \dots, \xi_T) = 1/2 \|\vec{w}\|^2 + C \sum_{i=1}^{i=T} \xi_i \quad (\text{A.39})$$

O coeficiente C introduzido em (A.39) é chamado de custo. C é um parâmetro de entrada, cujo valor é determinado experimentalmente de maneira que o erro de teste seja minimizado **(1, 14)**. Pode-se observar que minimizar f_2 implica maximizar a margem e minimizar o erro de treinamento. O problema a resolver para o caso linear e não separável é análogo ao Problema 2 da Seção A.2.1 e pode ser expresso como:

Problema 6

$$\min \left\{ f_2(w_1, w_2, \dots, w_d, \xi_1, \dots, \xi_T) = 1/2 \|\vec{w}\|^2 + C \sum_{i=1}^{i=T} \xi_i \right\} \quad (\text{A.40})$$

Restrições:

$$g_{i1}(w_1, w_2, \dots, w_d, b, \xi_1, \dots, \xi_T) = y_i(\vec{w} \cdot \vec{X}_i - b) - 1 + \xi_i \geq 0; i = 1 \dots T \quad (\text{A.41})$$

$$\xi_i \geq 0; i = 1 \dots T \quad (\text{A.38})$$

Se as mesmas transformações realizadas para o Problema 2 da Seção A.2.1 são aplicadas ao Problema 6, nem as variáveis ξ_i , nem seus Multiplicadores de Lagrange associados aparecem nas expressões do problema análogo ao Problema 5 da Seção A.2.4. O análogo do Problema 5 é **(5)**:

Problema 7:

$$\min_{\alpha_i} \{L_2 = 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \bar{x}_i \cdot \bar{x}_j - \sum_{i=1..T} \alpha_i\} \quad (A.33)$$

Restrições:

$$\sum_{i=1}^{i=T} y_i \alpha_i = 0 \quad (A.30)$$

$$0 \leq \alpha_i \leq C; i = 1...T \quad (A.42)$$

Como pode ser observado, a única diferença entre o Problema 5 o Problema 7 é a desigualdade (A.42) que inclui o parâmetro C e substitui à desigualdade (A.25). Os Multiplicadores de Lagrange possuem neste caso o valor do custo C como limite superior. Os coeficientes iguais a C correspondem a exemplos cujas distâncias ao hiperplano de separação são menores ou iguais a d_+ ou correspondem a um erro **(5)**. Os coeficientes entre 0 e C correspondem a exemplos cujas distâncias ao hiperplano de separação são iguais a d_+ . Por último, os coeficientes iguais a 0 correspondem a exemplos cujas distâncias ao hiperplano de separação são maiores ou iguais d_+ .

Achados os multiplicadores de Lagrange, o vetor $\bar{w}$ e o escalar b são determinados da mesma forma que no caso linear e separável, mas no caso de b , usando um α_k diferente a 0 e C (A.29 e A.34). Obtidos $\bar{w}$ e b , o hiperplano que separa os exemplos positivos dos negativos com margem máxima e que minimiza a quantidade de erros é determinado, e portanto, finaliza a fase de treinamento da SVM. A fase de teste é realizada como na Seção A.2.4 avaliando a expressão (A.35).

Até agora, na fase de treinamento, a separação dos exemplos positivos dos negativos tem sido realizada com um hiperplano. Mas devido à não separabilidade, o número de erros de teste pode ser importante. Para diminuir esta quantidade, poderia ser conveniente usar uma superfície de separação não linear ao invés de um hiperplano.

A.4 CASO NÃO LINEAR E NÃO SEPARÁVEL

Para o caso em que a separação dos exemplos de treinamento de forma linear resulta em um índice de erro muito alto, é necessária uma separação não linear. Para obtê-la, realiza-se uma transformação do espaço R^d a outro espaço de Hilbert H em R^n , com a dimensão n sendo maior que d para aumentar a separabilidade **(5)**. Essa transformação é realizada através de uma função de mapeamento $\vec{\phi}$.

$$\vec{\phi} : R^d \rightarrow R^n \quad (\text{A.43})$$

Efetuada essa transformação, na fase de treinamento é realizada a separação dos exemplos positivos dos negativos através de uma superfície linear ou hiperplano. A obtenção de menor erro com essa transformação dependerá da aplicação e da função $\vec{\phi}$ escolhida **(5)**. A aplicação da transformação a cada entrada $\vec{x}_i$ gera um vetor $\vec{z}_i$ correspondente no espaço H com componentes: $(z_{i1}, z_{i2} \dots z_{in})$.

$$\vec{\phi}(\vec{x}_i) = \vec{z}_i \quad (\text{A.44})$$

O problema agora é linear e não separável no espaço H . O hiperplano de separação é obtido resolvendo o Problema 7 substituindo as entradas $\vec{x}_i$ por $\vec{z}_i$.

Problema 8:

$$\min_{\alpha_i} \{L_2 = 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \vec{z}_i \cdot \vec{z}_j - \sum_{i=1..T} \alpha_i\} \quad (A.45)$$

Restrições:

$$\sum_{i=1}^{i=T} y_i \alpha_i = 0 \quad (A.30)$$

$$0 \leq \alpha_i \leq C; i = 1..T \quad (A.42)$$

Dado que $\vec{z}_i \cdot \vec{z}_j = \vec{\phi}(\vec{x}_i) \cdot \vec{\phi}(\vec{x}_j)$, (A.45) é substituída por (A.46)

$$\min_{\alpha_i} \{L_2 = 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j \vec{\phi}(\vec{x}_i) \cdot \vec{\phi}(\vec{x}_j) - \sum_{i=1..T} \alpha_i\} \quad (A.46)$$

Como a transformação $\vec{\phi}$ é vetorial o problema é simplificado se é conhecida uma função $K(\vec{x}_i, \vec{x}_j)$ de forma tal que:

$$K(\vec{x}_i, \vec{x}_j) = \vec{\phi}(\vec{x}_i) \cdot \vec{\phi}(\vec{x}_j) \quad (A.47)$$

A igualdade (A.47) implica que não é necessário conhecer a função $\vec{\phi}$ para a determinação dos multiplicadores de Lagrange, uma vez que $\vec{\phi}$ desaparece das expressões matemáticas. Essa função $K(\vec{x}_i, \vec{x}_j)$ é conhecida como **kernel**. Substituindo $\vec{\phi}(\vec{x}_i) \cdot \vec{\phi}(\vec{x}_j)$ por $K(\vec{x}_i, \vec{x}_j)$ em (A.46) e mantendo as restrições (A.30) e (A.42) do Problema 8, é obtido o problema a ser resolvido para o caso não linear e não separável:

Problema 9:

$$\min_{\alpha_i} \{L_2 = 1/2 \sum_{i,j=1..T} y_i y_j \alpha_i \alpha_j K(\bar{x}_i, \bar{x}_j) - \sum_{i=1..T} \alpha_i\} \quad (A.48)$$

Restrições:

$$\sum_{i=1}^{i=T} y_i \alpha_i = 0 \quad (A.30)$$

$$0 \leq \alpha_i \leq C; i = 1..T \quad (A.42)$$

Se os exemplos são separáveis, o erro de treinamento é nulo, sendo um caso particular do Problema 9, que não modifica sua formulação. No caso não linear e não separável, pretende-se obter $\vec{w}$ e b no espaço H . Se a igualdade (A.29) mostrada na Seção A.2.4 é aplicada para os vetores $\vec{z}_i$, a expressão para $\vec{w}$ é obtida.

$$\vec{w} = \sum_{i=1}^{i=T} y_i \alpha_i \vec{z}_i = \sum_{i=1}^{i=T} y_i \alpha_i \vec{\phi}(\bar{x}_i) \quad (A.49)$$

Mas no caso em que $\vec{\phi}(\bar{x}_i)$ não é conhecida, $\vec{w}$ não pode ser achado. Apesar disso, utilizando sua expressão é possível achar b e a função utilizada na fase de teste. Substituindo $\vec{w}$ segundo a expressão (A.49), na expressão (A.34) para o cálculo de b :

$$b = \vec{w} \cdot \vec{z}_k - y_k = \sum_{i=1}^{i=T} y_i \alpha_i \vec{\phi}(\bar{x}_i) \cdot \vec{\phi}(\bar{x}_k) - y_k; \text{ para } 0 < \alpha_k < C \quad (A.50)$$

Substituindo $\vec{\phi}(\bar{x}_i) \cdot \vec{\phi}(\bar{x}_k)$ pela função *kernel*:

$$b = \vec{w} \cdot \vec{z}_k - y_k = \sum_{i=1}^{i=T} y_i \alpha_i K(\vec{x}_i, \vec{x}_k) - y_k; \text{ para } 0 < \alpha_k < C \quad (\text{A.51})$$

A fase de teste é realizada aplicando a equação (A.35) a uma entrada $\vec{z}$ e substituindo $\vec{w}$ por (A.49).

$$\text{sgn}(u = \sum_{i=1}^{i=T} (y_i \alpha_i K(\vec{x}_i, \vec{x})) - b) \quad (\text{A.52})$$

Como pode ser apreciado, apesar de não ser possível obter $\vec{w}$, o coeficiente b é obtido a partir das expressões (A.34) e (A.49). Na função usada no teste, (A.52), a expressão de $\vec{w}$ também está presente implicitamente na somatória dos multiplicadores de Lagrange multiplicados pelas saídas y_i e pela função *kernel*.

Neste Anexo foi apresentado o desenvolvimento matemático da SVM. Este material está presente em diferentes referências bibliográficas que se complementam. Neste trabalho juntou-se uma boa parte dele, deixando algumas demonstrações mais específicas para consulta nas referências bibliográficas. Primeiro mostrou-se o caso mais simples, o linear e separável. Depois foi incorporado o problema de não separabilidade e finalmente a não linearidade da superfície de separação. Foram mostradas as principais transformações do problema de treinamento da SVM e do teste.