UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
CENTRO ESTADUAL DE SENSORIAMENTO REMOTO E METEROLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM SENSORIAMENTO REMOTO

DIEGO BONESSO

Estimação dos Parâmetros do Kernel em um Classificador SVM na Classificação de Imagens Hiperespectrais em uma Abordagem Multiclasse

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Sensoriamento Remoto

Prof. PhD.
Vitor Haertel
Orientador

Porto Alegre, Agosto de 2013

CIP - Catalogação na Publicação

Bonesso , Diego

Estimação dos Parâmetros do Kernel em um Classificador SVM na Classificação de Imagens Hiperespectrais em uma Abordagem Multiclasse / Diego Bonesso. – 2013.

108 f.

Orientador: Vitor Haertel.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul, Centro Estadual de Pesquisas em Sensoriamento Remoto e Meteorologia, Programa de Pós-Graduação em Sensoriamento Remoto, Porto Alegre, BR-RS, 2013.

1. Máquinas de Vetores de Suporte. 2. Recozimento Simulado. 3. Dados de Imagens Hiperespectrais. 4. Parâmetros do Kernel. I. Haertel, Vitor, orient. II. Título.

Uma teoria matemática não deve ser considerada completa, até que você a tenha deixado tão clara, de modo que pode ser explicada para o primeiro homem que você encontrar na rua.

— DAVID HILBERT

AGRADECIMENTOS

À minha esposa e família pelo apoio, paciência e compreensão em todos os momentos. Especialmente ao Prof. Vitor Haertel, exemplo de competência e dedicação à ciência, pela orientação. À Universidade Federal do Rio Grande do Sul (UFRGS) e a todos que, de alguma maneira, contribuíram com o desenvolvimento desta dissertação.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

SVM	Support Vector Machine
RBF	Radial Basis Function
SVMBT-AMOEBASA	Support Vector Machine Binary Tree AMOEBA Simulated Annealing
CABSVM	Classificador em Árvore Binária
AVIRIS	Airbone Visible Infrared Imagining Spectrometer
ROSIS	Reflective Optics System Imaging Spectrometer
RFE	Recursion Feature Elimination

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Nessa dissertação é investigada e testada uma metodologia para otimizar os parâmetros do *kernel* do classificador Support Vector Machines (SVM). Experimentos são realizados utilizando dados de imagens em alta dimensão. Imagens em alta dimensão abrem novas possibilidades para a classificação de imagens de sensoriamento remoto que capturam cenas naturais. É sabido que classes que são espectralmente muito similares, i.e, classes que possuem vetores de média muito próximos podem não obstante serem separadas com alto grau de acurácia em espaço de alta dimensão, desde que a matriz de covariância apresente diferenças significativas. O uso de dados de imagens em alta dimensão pode apresentar, no entanto, alguns desafios metodológicos quando aplicado um classificador paramétrico como o classificador de Máxima Verossimilhança Gaussiana. Conforme aumenta a dimensionalidade dos dados, o número de parâmetros a serem estimados a partir de um número geralmente limitado de amostras de treinamento também aumenta. Esse fato pode ocasionar estimativas pouco confiáveis, que por sua vez resultam em baixa acurácia na imagem classificada. Existem diversos abordagens propostas na literatura para minimizar esse problema. Os classificadores não paramétricos podem ser uma boa alternativa para mitigar esse problema. O SVM atualmente tem sido investigado na classificação de dados de imagens em alta-dimensão com número limitado de amostras de treinamento. Para que o classificador SVM seja utilizado com sucesso é necessário escolher uma função de *kernel* adequada, bem como os parâmetros dessa função. O *kernel* RBF tem sido frequentemente mencionado na literatura por obter bons resultados na classificação de imagens de sensoriamento remoto. Neste caso, dois parâmetro devem ser escolhidos para o classificador SVM: (1) O parâmetro de margem (C) que determina um ponto de equilíbrio razoável entre a maximização da margem e a minimização do erro de classificação, e (2) o parâmetro γ que controla o raio do *kernel* RBF. Estes dois parâmetros podem ser vistos como definindo um espaço de busca. O problema nesse caso consiste em procurar o ponto ótimo que maximize a acurácia do classificador SVM. O método de Busca em Grade é baseado na exploração exaustiva deste espaço de busca. Esse método é proibitivo do ponto de vista do tempo de processamento, sendo utilizado apenas com propósitos comparativos. Na prática os métodos heurísticos são a abordagem mais utilizada, proporcionado níveis aceitáveis de acurácia e tempo de processamento. Na literatura diversos métodos heurísticos são aplicados ao problema de classificação de forma global, i.e, os valores selecionados são aplicados durante todo processo de classificação. Esse processo, no entanto, não considera a diversidade das classes presentes nos dados. Nessa dissertação investigamos a aplicação da heurística Simulated Annealing (Recozimento Simulado) para um problema de múltiplas classes usando o classificador SVM estruturado como uma arvore binária. Seguindo essa abordagem, os parâmetros são estimados em cada nó da arvore binária, resultado em uma melhora na acurácia e tempo

razoável de processamento. Experimentos são realizados utilizando dados de uma imagem hiperespectral disponível, cobrindo uma área de teste com controle terrestre bastante confiável.

Palavras-chave: Máquinas de Vetores de Suporte, Recozimento Simulado, Dados de Imagens Hiperespectrais.

ABSTRACT

In this dissertation we investigate and test a methodology to optimize the *kernel* parameters in a Support Vector Machines classifier. Experiments were carried out using remote sensing high-dimensional image data. High dimensional image data opens new possibilities in the classification of remote sensing image data covering natural scenes. It is well known that classes that are spectrally very similar, i.e., classes that show very similar mean vectors can notwithstanding be separated with an high degree of accuracy in high dimensional spaces, provided that their covariance matrices differ significantly. The use of high-dimensional image data may present, however, some drawbacks when applied in parametric classifiers such as the Gaussian Maximum Likelihood classifier. As the data dimensionality increases, so does the number of parameters to be estimated from a generally limited number of training samples. This fact results in unreliable estimates for the parameters, which in turn results in low accuracy in the classified image. There are several approaches proposed in the literature to minimize this problem. Non-parametric classifiers may provide a sensible way to overcome this problem. Support Vector Machines (SVM) have been more recently investigated in the classification of high-dimensional image data with a limited number of training samples. To achieve this end, a proper *kernel* function has to be implemented in the SVM classifier and the respective parameters selected properly. The RBF *kernel* has been frequently mentioned in the literature as providing good results in the classification of remotely sensed data. In this case, two parameters must be chosen in the SVM classification: (1) the margin parameter (C) that determines the trade-off between the maximization of the margin in the SVM and minimization of the classification error, and (2) the parameter γ that controls the radius in the RBF *kernel*. These two parameters can be seen as defining a search space, The problem here consists in finding an optimal point that maximizes the accuracy in the SVM classifier. The Grid Search approach is based on an exhaustive exploration in the search space. This approach results prohibitively time consuming and is used only for comparative purposes. In practice heuristic methods are the most commonly used approaches, providing acceptable levels of accuracy and computing time. In the literature several heuristic methods are applied to the classification problem in a global fashion, i.e., the selected values are applied to the entire classification process. This procedure, however, does not take into consideration the diversity of the classes present in the data. In this dissertation we investigate the application of Simulated Annealing to a multiclass problem using the SVM classifier structured as a binary tree. Following this proposed approach, the parameters are estimated at every level of the binary tree, resulting in better accuracy and a reasonable computing time. Experiments are done using a set of hyperspectral image data, covering a test area with very reliable ground control available.

Keywords: Support Vector Machines,Simulated Annealing,Hyperspectral Image Data.

1 INTRODUÇÃO

Dados em alta dimensionalidade, conhecidos como hiperespectrais normalmente contém centenas de bandas espectrais produzindo imagens com alta resolução espectral. Os dados hiperespectrais vem sendo utilizados com sucesso em aplicações de agricultura, mineralogia, física e vigilância, entre outros.

Sensores convencionais, como o Landsat e Spot, em geral conseguem discriminar a maioria das classes que ocorrem em cenas naturais como: florestas, culturas agrícolas, corpos de água, rochas e solos. Entretanto essa capacidade de discriminar é limitada quando estão presentes na cena classes espectralmente muito semelhantes, isto é, classes cujos vetores de médias são muito próximos entre si. Sensores hiperespectrais podem ser usados para auxiliar nesse problema. FUKUNAGA (1990) mostra que classes espectralmente muito semelhantes entre si, como, por exemplo, coberturas terrestres com diferenças espectrais sutis (vetoreAbe2005s de média muito próximos entre si) podem, não obstante, ser classificados com boa acurácia empregando dados de alta dimensionalidade, desde que as matrizes de covariância das diferentes classes difiram entre si.

Segundo LANDGREBE (1997), a utilização de sensores hiperespectrais, os quais produzem dados muito mais complexos que os sensores tradicionais, oferece uma maior possibilidade para extrair informações úteis do fluxo de dados produzido pelo sensor. No entanto, dados mais complexos requerem processos de classificação mais sofisticados para que o volume de informação produzido pelos sensores hiperespectrais possa ser utilizado em todo seu potencial. A alta dimensionalidade dos dados gerados pelos sensores hiperespectrais resulta em problemas, no caso de classificadores paramétricos, devido à dificuldade em estimar um número crescente de parâmetros a partir de um número geralmente limitado de amostras. Esta situação resulta no fenômeno conhecido na literatura como fenômeno de Hughes, que consiste em um incremento inicial na acurácia dos resultados na medida em que a dimensionalidade dos dados (número de bandas espectrais) aumenta. Para um determinado número de bandas espectrais adicionadas, a acurácia atinge um máximo para em seguida passar a decrescer, na medida em que bandas espectrais adicionadas são inseridas, refletindo a menor confiabilidade nos valores estimados para o crescente número de parâmetros.

Na literatura existem diversas abordagens propostas para mitigar o problema da classificação de dados hiperespectrais. Dentre essas abordagens podemos destacar o Support Vector Machines (SVM), o qual tem sido aplicado com sucesso na classificação de imagens hiperespectrais. Duas características do classificador SVM são fundamentais para o seu bom desempenho: a boa capacidade de generalização do classificador e a

baixa sensibilidade ao fenômeno de Hughes. A principal explicação para a baixa sensibilidade ao fenômeno de Hughes se deve ao princípio da margem máxima, no qual o SVM é baseado. O princípio de margem máxima torna desnecessário estimar explicitamente as distribuições estatísticas de classes no espaço de feições hiperdimensional para realizar a tarefa de classificação.

Apesar de o SVM apresentar bom poder de generalização (ABE, 2005), seu desempenho depende da seleção de parâmetros na função de *kernel* do classificador. A escolha de parâmetros inadequados pode resultar em decréscimo na acurácia dos resultados. Atualmente não existe um método universal para guiar a seleção de parâmetros do *kernel*.

Na literatura encontramos diversas abordagens para selecionar o modelo mais adequado do classificador SVM. Dentre essas abordagens podemos destacar heurísticas como o Recozimento Simulado (*Simulated Annealing*), os Algoritmos Genéticos, além de outras.

Nessa dissertação se investiga a seleção de parâmetros do *kernel* do classificador SVM implementado em estágios múltiplos e estruturados na forma de árvore binária. A estrutura binária em múltiplos estágios permite tratar pares de classes a cada estágio (nó). No presente trabalho é proposta a otimização dos parâmetros do Classificador SVM em cada nó da árvore, investigando-se o resultado da escolha dos parâmetros através de uma heurística, o Recozimento Simulado. Além disso, os resultados obtidos pelo método proposto são comparados com a busca em grade (*grid search*), a qual é uma busca exaustiva. Esta foi implementada de maneira a otimizar a escolha dos parâmetros do classificador em cada nó da árvore da mesma forma que a heurística. E, por fim, os resultados são comparados com a utilização de parâmetros globais para toda árvore do classificador. Nesse caso foram utilizados parâmetros que produziram bons resultados em trabalhos desenvolvidos anteriormente pelo grupo ANDREOLA; HAERTEL (2010).

1.1 Objetivos

Este estudo tem por objetivo geral investigar a contribuição da seleção dos parâmetros do *kernel* na acurácia da imagem produzida pelo processo de classificação. São investigados dois métodos de seleção de parâmetros, o Busca em Grade (busca exaustiva) e uma heurística, o Recozimento Simulado.

Tem-se, como objetivo específico, a implementação do classificador SVM em uma estrutura de árvore binária, na qual a seleção dos parâmetros do *kernel* é realizada em cada nó da árvore. Em geral os trabalhos na literatura abordam os parâmetros do *kernel* de forma global, ou seja, os mesmos valores para os parâmetros são utilizados durante todo processo de classificação. A utilização de uma árvore binária decompõe o problema de classificação de múltiplas classes em vários problemas de classificação binários. Deseja-se investigar se a seleção dos parâmetros do *kernel*, de maneira individual para cada problema de classificação binário, resulta em um aumento de acurácia global.

Deseja-se, além disso, investigar a validade da aplicação de uma heurística para estimar os parâmetros do *kernel* do classificador SVM, os quais são decisivos na obtenção de uma boa acurácia. Em geral, quando o pesquisador não possui conhecimento prévio sobre o conjunto de dados e utiliza o classificador SVM, os parâmetros do *kernel* são escolhidos através de tentativa e erro, nem sempre produzindo bons resultados. Ou ainda, é aplicado

um método exaustivo, como a busca em grade, que pode ser demorada e necessita da escolha de um espaço de busca que seja razoável.

A validação do algoritmo Support Vector Machines Amoebasa Binary Tree (SVMBT-AMOEBASA), baseado no classificador em árvore binária (CABSVM), proposto por ANDREOLA; HAERTEL (2010), será realizada através da confrontação dos resultados obtidos pela utilização de parâmetros fixos do *kernel* para todas as classes e a abordagem proposta. Além disso, a heurística proposta será confrontada com o método de Busca em Grade.

1.2 Reconhecimento de Padrões

Reconhecimento de padrões é a disciplina cientifica na qual o principal objetivo é a classificação de objetos em um número de categorias ou classes. Dependendo do tipo de aplicação, esses objetos podem ser imagens, textos, áudios ou qualquer outro tipo de informação que possa ser mensurada de alguma forma e que, também, possa ser classificada. Por "padrões" se entende como qualquer relação, regularidade ou estruturas inerentes de alguma fonte de dados como, por exemplo, pixels em uma imagem.

Espera-se, ao detectar padrões significativos nos dados disponíveis, que o classificador seja capaz de fazer previsões sobre os novos dados provenientes da mesma fonte. Nesse caso dizemos que o classificador possui poder de generalização sobre aquela fonte de dados, conseguindo classificar corretamente novos dados que são inseridos.

Um classificador que faz uso de amostras previamente definidas por um analista, através de técnicas como análise visual ou estudo de campo, é conhecido como classificador supervisionado. Um classificador supervisionado exige que sejam fornecidas amostras durante a fase de treinamento. Já os classificadores não supervisionados determinam de forma automática as classes existentes em um conjunto de dados.

O sensoriamento remoto é a ciência de obter informações sobre um objeto a partir de medições feitas a distância do alvo de interesse, ou seja, sem realmente entrar em contato com o objeto. A quantidade mais frequentemente mensurada pelos sistemas de sensoriamento remoto é a energia eletromagnética que emana do objeto de interesse, embora existam outras possibilidades como, por exemplo, ondas sísmicas, ondas sonoras e as forças gravitacionais.

É frequente haver o debate sobre o quão longe um objeto deve estar do dispositivo de medição para poder ser considerado remoto, mas este é apenas um debate semântico.

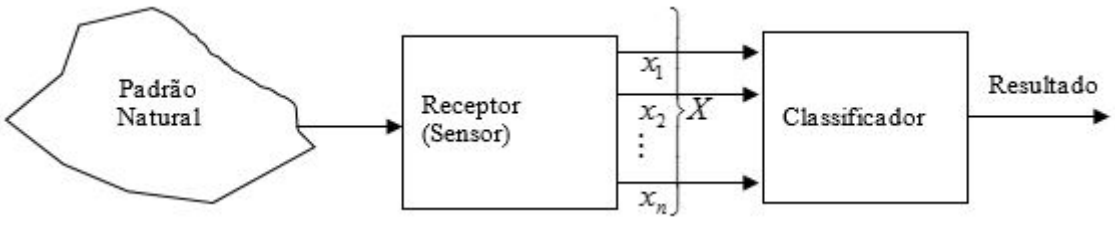

Figura 1.1: Modelo de Sistema de Reconhecimento de Padrões. Fonte: LANDGREBE (1997)

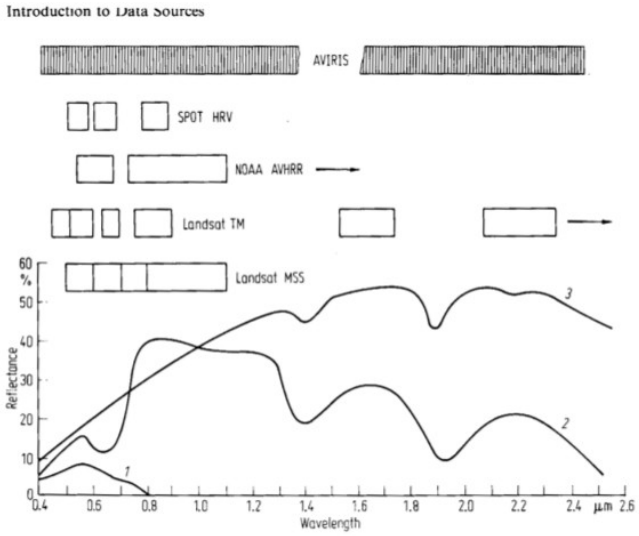

Figura 1.2: Características de refletância espectral de materiais comuns na superfície da terra no espectro visível e infravermelho próximo. Adaptado de: (RICHARDS; JIA, 2006).

O sensoriamento remoto pode envolver medições realizadas a centenas ou milhares de metros do objeto de interesse, como, também, podemos conceber aplicações em que o dispositivo de medição pode estar apenas a uma fração de um metro do objeto de interesse.

Nas fotografias aéreas, que se constituem em uma das formas originais de sensoriamento remoto, as informações são coletadas no espectro magnético, na porção visível e no infravermelho próximo. Porém, o avanço no desenvolvimento de sensores permitiu a aquisição de informações em outros comprimentos de ondas, tais como: infravermelho médio, termal e micro ondas. Sensores que coletam informações em um amplo número de bandas são conhecidos como multiespectrais ou hiperespectrais dependendo da quantidade de bandas espectrais presentes.

Uma banda espectral é definida como um intervalo discreto do espectro eletromagnético. Por exemplo, o intervalo de comprimento de onda de $\mu m0,4$ até $\mu m0,5$ (mícron) é uma banda espectral. Os satélites são projetados para mensurar a resposta em bandas espectrais particulares, permitindo a discriminação da maioria dos materiais da superfície da Terra.

A radiação solar incidente na superfície da Terra interage com os diferentes tipos de cobertura do solo (alvos). Neste processo, parte da energia incidente é absorvida, parte é transmitida e parte é refletida. A proporção da energia refletida varia conforme o comprimento de onda e da natureza do alvo. Essa variação, específica de cada material, é conhecida como assinatura espectral. Diferentes tipos de objetos ou alvos são encontrados na superfície da Terra tais como minerais, vegetais, solo exposto, água e neve, entre outros, cada qual apresentando uma assinatura espectral específica, conforme ilustra a Figura (1.2).

As regiões do espectro eletromagnético de maior interesse em sensoriamento remoto são a porção visível do espectro, o infravermelho próximo e médio (energia refletida), o infravermelho termal (energia emitida) e a região de micro ondas.

1.3 Estrutura da Dissertação

No primeiro capítulo é feita uma introdução aos objetivos da dissertação, bem como uma rápida exposição sobre o reconhecimento de padrões em imagens digitais,características dos dados multidimensionais e problemas encontrados na classificação de dados em alta dimensionalidade.

No segundo capítulo são apresentados os conceitos do classificador Support Vector Machines (*SVM*) bem como sua formulação matemática e sua abordagem em estágios-múltiplos implementada através de uma arvore binária. Além disso, é apresentada uma revisão bibliográfica sobre os problemas associados à seleção dos parâmetros do *kernel* do SVM.

No terceiro capitulo Metodologia, aborda as heurísticas envolvidas no processo de seleção de parâmetros do *kernel* e o algoritmo proposto, desenvolvido em forma de arvore binária, empregado nessa dissertação. Além disso, são abordados detalhes referentes à implementação.

No quarto capitulo são apresentados os experimentos realizados e os resultados obtidos visando a validação da metodologia proposta. No quinto capítulo são apresentadas as conclusões, análise final e sugestões.
O Apêndice A contém a listagem dos programas elaborados para viabilizar a pesquisa.
O Apêndice B apresenta as tabelas de contingência dos experimentos.

2 REVISÃO BIBLIOGRÁFICA

2.1 Seleção de Parâmetro do Kernel

O classificador SVM é baseado no princípio de margem máxima sendo desnecessário estimar distribuições estatísticas das classes hiperespectrais no espaço de característica. Outra propriedade do classificador SVM é o seu bom poder de generalização devido a sua representação esparsa da função de decisão. O SVM vem sendo utilizando com sucesso em sensoriamento remoto em particular na classificação de imagens hiperespectrais. Conforme mencionado anteriormente os sensores remotos hiperespectrais têm a particularidade de fornecer dados sobre asuperfície da Terra com uma resolução espectral muito alta, que geralmente resulta em dezenas de canais. A grande dimensão do espaço dos dados gerado pelo sensor introduz desafios metodológicos. No contexto dos classificadores supervisionados o maior desafio a ser mitigado é a maldição da dimensionalidade (MELGANI; BRUZZONE, 2004) (BAZI, 2006) (MANDAL, 2010).

Em (PAL; MATHER, 2005), o SVM e dois classificadores tradicionalmente utilizados em sensoriamento remoto são comparados (Máxima Verossimilhança Gaussiana e Redes Neurais de múltiplas camadas). O conjunto de dados utilizado para comparação dos resultados foi adquirido pelo sensor Landsat-7 (sensor multiespectral) e pelo sensor hiperespectral, *Digital Airborne Imaging Spectrometer* (DAIS). Em geral, o classificador SVM apresentou resultados superiores comparados ao classificador estatístico e ao classificar de Redes Neurais tanto na imagem multiespectral bem como na hiperespectral.

A seleção dos parâmetros do *kernel* do SVM são críticos para obter um bom desempenho. Inicialmente, (VAPNIK, 1998) recomenda a escolha manual dos parâmetros do *kernel* pelo especialista baseado no conhecimento a priori do conjunto de dados a ser avaliado. Porém o processo de seleção de parâmetros de maneira empírica pode resultar em uma acurácia inferior, existem diversos trabalhos na literatura que propõe métodos automáticos para a seleção dos valores dos parâmetros da função de *kernel*. Uma das técnicas mais utilizadas para seleção dos parâmetros de forma automática quando o conhecimento prévio do conjunto de dados não existe, é a Busca em Grade.

No contexto de aprendizagem de máquina, a busca em grade se refere ao processo de busca exaustiva sobre um subconjunto do espaço de trabalho. No caso especifico de seleção dos parâmetros do *kernel* RBF, a busca é realizada em um espaço formado pelos parâmetros gama γ e o parâmetro de custo C, cada coordenada desse espaço é formada por um par ordenado (γ, C). O objetivo da busca exaustiva é encontrar no espaço formado por (γ, C) pontos nos quais a acurácia do classificador seja a maior possível (LIEPERT,

2003).

O processo de busca exaustiva é bastante simples de ser programado e apresenta bons resultados na busca dos melhores parâmetros. Sua desvantagem é o tempo de processamento, já que faz uma busca linear no espaço de parâmetros. Outra questão fundamental no método de Busca em Grade é a delimitação do espaço a ser investigado quando não possuímos um conhecimento prévio dos dados a serem classificado. A busca em um espaço muito amplo resulta em um tempo de processamento alto e muitas vezes com resultados pouco efetivos.

Uma abordagem alternativa comumente encontrada na literatura é a utilização de heurísticas mais eficientes para obtenção de resultados precisos com um custo computacional menor. Na literatura podem ser encontrados diversos trabalhos aplicando heurísticas na escolha de parâmetros do *kernel* do SVM. Dentre as técnicas empregadas com frequências podemos destacar as seguintes heurísticas: Algoritmos Genéticos (CAROLINA; CARVALHO, 2008), Recozimento Simulado (BOARDMAN, 2006) e mais recentemente Colônias de Formigas (SAMADZADEGAN; HASANI; SHENK, 2012).

2.2 Meta-Heurística na Determinação de Parâmetros do Kernel

Nessa seção são analisados trabalhos relevantes encontrados na literatura sobre a seleção de parâmetros do *kernel* do SVM. Os artigos analisados tratam da utilização do *kernel RBF* e da otimização de seus parâmetros, que são o parâmetro gama γ e o parâmetro de custo *C*. Em geral, as abordagens encontradas na literatura de sensoriamento remoto combinam duas técnicas para melhorar a acurácia do classificador SVM para imagens hiperespectrais, a saber, a seleção dos parâmetros do *kernel* através de alguma heurística e a seleção de bandas espectrais. Uma das heurísticas mais frequentemente usadas para a solução de problemas de otimização são os Algoritmos Genéticos. O Algoritmo Genético (GA) é uma busca heurística inspirada no processo de seleção natural. A forma mais comum de algoritmo genético envolve os seguintes conjuntos de passos: Uma população inicial de cromossomos é gerada de maneira aleatória. A população gerada é avaliada por uma função de custo. O processo de avaliação do cromossomo permite que os melhores cromossomos isto é, aqueles que apresentarem melhores resultados, tenham maiores possibilidades de serem selecionados. Após o processo de seleção ser completado, o próximo passo é dedicado à reprodução da população, utilizando os operadores de cruzamento e mutação. O processo completo é iterado até que o critério de convergência definido pelo usuário seja atingido.

Nos trabalhos de (BAZI, 2006) e (CAROLINA; CARVALHO, 2008) a heurística de Algoritmos Genéticos e aplicados na seleção de modelo (seleção dos parâmetros da função de kernel) em uma abordagem de múltiplas classes. Em ambos os trabalhos utiliza-se o *Kernel* RBF para classificação.

BAZI (2006) propõe a seleção de modelo utilizando GA em uma abordagem de múltiplas classes na classificação de uma imagem hiperespectral. O método multiclasses adotado é o um-contra-todos *(one-agaist-all)*. Tomando **M** classes, construímos **M** superfície de decisão, $g^1, \ldots, g^M$, cada superfície de decisão é treinada para separar uma classe das classes restantes.O conjunto de dados utilizado para validar a proposta de implementação é o *Indian Pines*. O conjunto de dados foi adquirido pelo sensor *AVIRIS* sobre a

região de *Indian Pines* no noroeste do Estado de Indiana, EUA. A imagem possui dimensão espacial de 145 x 145 pixels, com resolução com 20 m/pixel. A tabela 2.1 mostra os resultados dos experimentos realizados.

	Com seleção de feições (SVM-GA-SV)	Sem seleção de feições (SVM)
ω_1	88.05	84.53
ω_2	88.52	89.79
ω_3	97.04	93.67
ω_4	97.76	96.92
ω_5	100	99.6
ω_6	91.89	89.60
ω_7	81.76	78.81
ω_8	94.82	91.90
ω_9	98.60	98.60
Acurácia Global	90.62	87.66
Acurácia Média	93.16	91.05
Tempo [H]	5.7	0.6

Tabela 2.1: A tabela apresenta os resultados obtidos em dois experimentos realizados no trabalho de (BAZI, 2006). AG é a acurácia global e AM é a acurácia média.

Foram utilizadas nove classes existentes na imagem. No primeiro resultado apresentado o classificador SVM convencional é aplicado ao conjunto de dados sem seleção de bandas espectrais. Os parâmetros γ e *C* variam no intervalo $[10^{-3}; 2]$ e $[10^{-3}; 200]$, respectivamente. A melhor acurácia obtida é de 87.66% para acurácia global e 91,05% para acurácia média.

O segundo resultado é obtido pela aplicação do algoritmo de SVM-GA-SV (*Support Vector Machines – Genetic Algorithm – Support Vector*), proposto por, (BAZI, 2006). O qual utiliza Algoritmos Genéticos aplicados ao classificador SVM para selecionar os parâmetros do Kernel. O número de Vetores do Suporte é utilizado como indicador do poder de generalização do hiperplano gerado pelo classificador. Além disso, as bandas espectrais são selecionadas através do algoritmo de RFE (*Recursive Feature Elimination*). O objetivo do algoritmo RFE é remover bandas que efetivamente pouco contribuem no processo de classificação, ou ainda, contribuem de forma negativa, diminuindo a acurácia.

É possível observar nos resultados apresentados um incremento na acurácia ao aplicar a heurística proposta. Podemos perceber que o tempo de processamento aumenta consideravelmente em relação ao primeiro experimento, onde o classificador é utilizado de forma convencional.

Já em (BOARDMAN, 2006), é proposta a utilização da heurística de Recozimento Simulado (*Simulated Annealing - SA*). O Recozimento Simulado é uma heurística para otimização que consiste numa técnica de busca local probabilística, fundamentada numa analogia com um processo físico estudado em termodinâmica. O processo percorre de maneira estocástica o espaço de parâmetros livres do SVM. É gerado um espaço utilizando uma escala logarítmica, variando os parâmetros γ e *C* sobre esse espaço.

A heurística de Recozimento Simulado é baseado na implementação do algoritmo encontrado no livro *Numerical Recipes in C*,(PRESS et al., 2007), o qual propõe um algoritmo de Recozimento Simulado para espaços discretos. Os autores empregam um gera-

dor de movimento aleatório simples, adicionado de um termo independente com pequeno valor numérico em torno da origem P_ϕ. O gerador de movimento percorre o espaço de soluções. Ocasionalmente, com baixa probabilidade, a busca reinicia em um ponto ótimo encontrado anteriormente. A função de custo, utilizada para estimar a acurácia nos pontos do espaço, é mostrada na equação 2.1.

$$E(f) = \frac{1}{2l}\sum_{i=1}^{l}|y_i - f(x_i)| + \lambda\left(\frac{\eta_{sv}}{l}\right)^{\tau} \tag{2.1}$$

Onde a função de custo $E(f)$ no ponto $P_i = C, \gamma$ do espaço de parâmetros é determinada não somente pelo erro de classificação, a primeira parcela da equação 2.1, mas, também, pela penalidade de complexidade, segunda parcela da equação 2.1. A penalidade de complexidade é definida pelo número de vetores de suporte η_{sv} dividido pelo número total de observações l. O parâmetro de regularização λ permite o controle sobre a compensação entre a acurácia de classificação e o modelo de generalização. No artigo, o parâmetro livre λ é fixado em 1 atribuindo igual peso para acurácia e para a complexidade. O parâmetro livre τ é fixado como $\frac{1}{2}$ visando penalizar as soluções que obtêm alta acurácia através de alta complexidade.

A ideia de relacionar a complexidade da superfície de decisão relacionada ao erro de classificação foi proposta por (VAPNIK, 1995), a equação a seguir expressa uma maneira de estimar o erro de classificação:

$$\varepsilon_{loo} \leq \frac{|S|}{M} \tag{2.2}$$

Onde ε_{loo} é a taxa de erro LOO (*leave-one-out*) para um conjunto de treinamento,$|S|$ é o número de vetores de suporte e $|M|$ é o número de dados de treinamento. Assumindo que seja removido um dado do conjunto de treinamento, que não é um vetor de suporte. Por exemplo, se pode obter uma estimativa do erro de teste removendo um ponto de treinamento e treinando novamente o classificador e testando com o ponto removido e, então, repetindo esse processo para todos os pontos de treinamento. Da solução do SVM sabemos que, removendo qualquer ponto de treinamento que não seja um vetor de suporte, não haverá qualquer efeito sobre o hiperplano de suporte, mesmo que esses pontos de treinamento sejam erros. Então, assumindo que todos os vetores de suporte estejam incorretamente classificados, obtemos a equação 2.2. A conclusão intuitiva é que um sistema com poucos vetores de suporte pode ter melhor desempenho quanto à acurácia.

Segundo (BURGES, 1998), existem situações onde o erro aumenta quando o número de vetores de suporte diminui, ou seja, nem sempre uma margem formada por poucos vetores de suporte pode apresentar melhor desempenho.

Em (CAROLINA; CARVALHO, 2008), o Algoritmo Genético é aplicado em problemas binários decompostos em múltiplas classes. O problema de seleção do modelo é formulado como uma busca de combinação de valores de parâmetros para minimizar a taxa de erro. Duas abordagens são comparadas, na primeira o algoritmo proposto utiliza valores comuns para todos os classificadores na decomposição, enquanto na segunda abordagem, para cada classificador é utilizado um conjunto distinto de parâmetros, ou seja, cada par de classes é tratado como um problema de classificação distinto.

O uso de abordagens de decomposição do classificador SVM em problemas de múltiplas classes aumenta o número de valores de parâmetros a serem estimados, já que cada classificador binário é um problema de classificação a parte, possuindo valores ótimos distintos a serem estimados para cada caso. Os experimentos são realizados utilizando conjuntos de dados de *benchmark* do repositório da *University of California, Irvine* (UCI) (BLAKE; MERZ, 1998). O repositório UCI é uma coleção de bases de dados usado pela comunidade de aprendizagem de máquina para realizar análise empírica de algoritmos. Os dados utilizados nos experimentos possuem múltiplas classes, sendo divididos em conjuntos de treinamento e de teste através da metodologia de validação cruzada (*k-fold*), que é uma metodologia de estratificação amplamente utilizada em reconhecimento de padrões. Segundo os autores, a utilização dos mesmos parâmetros de *kernel* em todas as superfície de decisão formadas pelo classificador SVM foi suficiente para obter bons resultados nos conjuntos de dados considerados, fato que também foi observado no trabalho de (LIEPERT, 2003). O trabalho de SAMADZADEGAN; HASANI; SHENK (2012), estuda o impacto da utilização da heurística de Colônia de Formigas (*Ant Colony Optimization - ACO*) na seleção dos parâmetros do *kernel*, simultaneamente é aplicado um método de seleção das bandas espectrais. A ACO é uma heurística baseada em probabilidade, criada para soluções de problemas que envolvem a procura por caminhos em grafos, o algoritmo é inspirado na observação do comportamento das formigas ao saírem de sua colônia para encontrar comida. São realizados experimentos com dois conjuntos de dados distintos, dos sensores AVIRIS (*Indian Pines*) e ROSIS. A aquisição do conjunto ROSIS foi realizada sobre a área do campus da Universidade de Pavia, Itália. Originalmente possui 115 bandas cobrindo o espectro eletromagnético na faixa de $0.43\mu m$ e $0.86\mu m$. Como métrica de validação da acurácia, os autores propõem a utilização da acurácia global além do coeficiente Kappa. Os experimentos analisados são divididos em três partes. Na primeira parte do experimento os parâmetros são determinados através do algoritmo proposto, BACO (*Binary Ant Colony Optimization*), utilizando a totalidade das bandas disponíveis. No segunda parte os parâmetros são computados pelo algoritmo de Busca em Grade e, consequentemente, fixado no algoritmo onde um subconjunto de feições espectrais é selecionado. Na terceira e última parte são considerados os processos de estimação de parâmetros ótimos e subconjunto de feições (bandas espectrais) ótimas. Também são realizados experimentos para comparar o desempenho do algoritmo proposto com as heurísticas de Recozimento Simulado, Busca Tabu e Algoritmos Genéticos. Segundo SAMADZADEGAN; HASANI; SHENK (2012), não existem diferenças práticas em termos da acurácia de classificação entre as várias heurísticas aplicadas, incluindo o método de Busca em Grade. A vantagem da metodologia proposta é o menor custo computacional se comparado a outras heurísticas. Os valores estimados para os parâmetros são utilizados de maneira constante em cada problema de classificação binária. Nessa seção foram apresentados vários trabalhos que fazem uso de heurísticas para seleção dos parâmetros do kernel do classificador SVM. Os trabalhos analisados utilizam o *kernel* RBF. Como abordagem multiclasse adotam métodos de decomposição como o um-contra-um ou todos-contra-todos. Os resultados mostram que heurísticas aplicadas à estimação de parâmetros ótimos do kernel podem apresentar bons resultados e diminuem consideravelmente o tempo de treinamento se comparado ao método de busca em grade, principalmente quando não existe um conhecimento prévio do conjunto de dados.

3 METODOLOGIA

O classificador SVM é fundamentado no princípio da minimização do Risco Estrutural (Structural Risk Minimization-SRM) proposto por (VAPNIK; CHERVONENKIS, 1974) e sua equipe. Formalmente podemos definir o SRM com a função $f(x)$ que minimiza o risco médio no conjunto de treinamento. O princípio da minimização do Risco Estrutural busca minimizar o erro com relação a um conjunto de treinamento (risco empírico), assim como o erro com relação ao conjunto de testes, isto é, o conjunto de amostras não empregado no treinamento do classificador (risco de generalização). O objetivo do SVM consiste em obter um equilíbrio entre esses erros, minimizando o excesso de ajustes (*overfitting*) que podem reduzir a capacidade de generalização do classificador. O problema de *overfitting* está relacionado ao fato do classificador memorizar os padrões de treinamento, gravando suas peculiaridades e ruídos, ao invés de extrair as características gerais que permitirão a generalização ou reconhecimento de padrões não utilizados no treinamento do classificador, de acordo com (SMOLA et al., 2000).

A questão da generalização pode ser mais bem avaliada para o caso de duas classes.Assumindo que as amostras de treinamento das duas classes não são linearmente separáveis, a função de decisão mais adequada é aquela para qual a distância entre os conjuntos das amostras de treinamento é maximizada. O SVM é um classificador que utiliza o princípio de maximização da margem. O princípio da margem máxima coloca a superfície de decisão exatamente entre o limite das duas classes e maximiza a distância do limite das classes. Segundo (HAMEL, 2009), essa abordagem reduz a probabilidade de erro de classificação.

Em relação ao SRM Vapnik (1998), mostra que, desde que o hiperplano de separação não cometa nenhum erro empírico, ou seja, separe corretamente todos os exemplos de treinamento, maximizar a margem equivale a minimizar o limite superior do risco esperado. Desta forma ao separar um hiperplano com margem máxima, o risco de generalização será minimizado.

A busca por uma superfície de decisão com critério ótimo como a margem, implica em um problema de otimização. Construir um classificador de margem máxima é um problema de otimização convexa que pode ser resolvido através de técnicas de programação quadrática. Problemas de otimização são problemas nos quais desejamos a melhor solução em um número de possíveis soluções ou soluções factíveis. A meta é encontrar a solução que maximiza ou minimiza o valor da função objetivo. A formulação matemática apresentada a seguir é baseada em (ABE, 2005).

3.1 SVM COM MARGENS RÍGIDAS

Seja $\mathbf{x}_i$ $(i = 1, \cdots, M)$ um conjunto de treinamento em um problema que consiste em duas classes linearmente separáveis (ω_1 *e* ω_2). Cada amostra fica associada aos rótulos $y_i = 1$ para a ω_1 e $y_i = -1$ para a ω_2. Se esses dados forem linearmente separáveis, a função de decisão pode ser determinada pela equação a seguir:

$$D(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{3.1}$$

Nesse caso, **w** é um vetor $m - dimensional$ e b é o termo independente. Onde *m* representa a dimensionalidade dos dados, e M é o número de amostras de treinamento, ou seja, os vetores **w** e **x** são representados por w_i e x_i para $i = 1, \cdots, m$ bandas e x_j para j= $1, \cdots, M$ amostras.

A função linear 3.1 divide o espaço em duas regiões $\mathbf{w}^T\mathbf{x} + b > 0$ e $\mathbf{w}^T\mathbf{x} + b < 0$ como podemos visualizar na Figura 3.1, gerando um hiperplano.

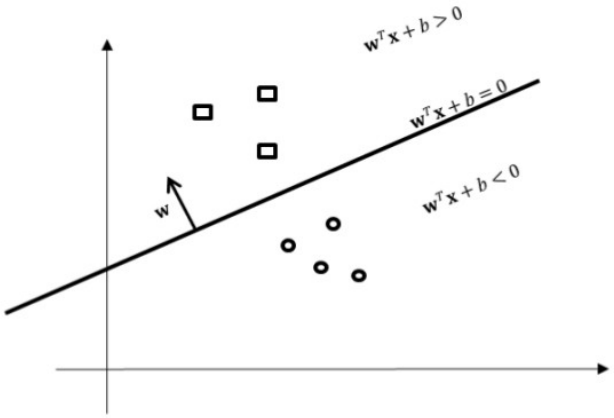

Figura 3.1: Um hiperplano que divide a região do espaço separando as amostras em duas regiões.

O processo de classificação das amostras pode ser formulado como se segue:

$$\mathbf{w}^T\mathbf{x}_i + b = \begin{cases} \geq a \ \mathbf{x}_i \in \omega_1(y_i = 1) \\ \leq -a \ \mathbf{x}_i \in \omega_2(y_i = -1) \end{cases} \tag{3.2}$$

sendo a uma constante $(a > 0)$. As inequações 3.2 podem ser rearranjadas, dividindo ambos os membros por a e aplicando uma transformação de escala em **w** **e** b e reescrevendo-a de forma mais compacta obtemos:

$$\mathbf{w}^T\mathbf{x}_i + b = \begin{cases} \geq 1 \ \mathbf{x}_i \in \omega_1(y_i = 1) \\ \leq 1 \ \mathbf{x}_i \in \omega_2(y_i = -1) \end{cases} \tag{3.3}$$

A equação 3.3 pode ser reescrita em uma só como se segue:

$$y_i(\mathbf{W}^T\mathbf{x}_i + b) \geq \ 1 \ para \ i = 1, ..., M \tag{3.4}$$

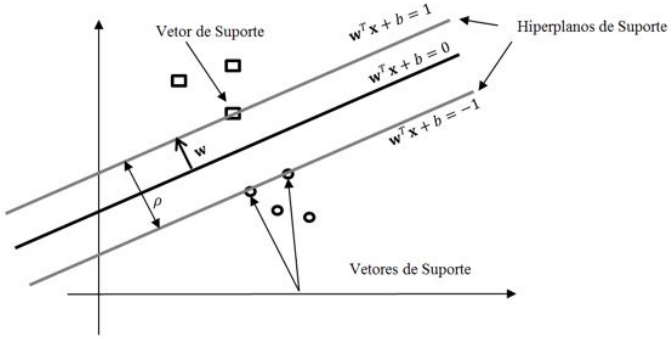

Figura 3.2: O hiperplano ótimo separando os dados com margem máxima ρ. Os vetores de suporte são as amostras que satisfazem as equações $D(\mathbf{x}) = 1$ e $D(\mathbf{x}) = -1$. Adaptado de (HAMEL, 2009).

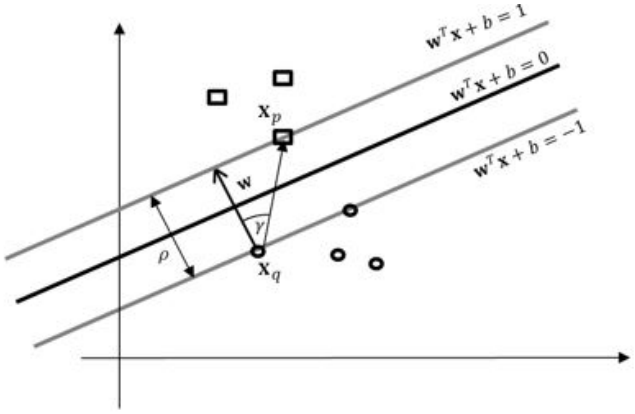

Figura 3.3: Margem ρ entre os dois hiperplanos de suporte. Adaptado de Adaptado de (HAMEL, 2009).

Existem infinitas funções lineares que separam as amostras de treinamento (Figura3.1). O objetivo do classificador SVM é encontrar um hiperplano ótimo que separe as duas classes (ω_1 e ω_2) de forma ótima. A equação $\mathrm{D(x)} = 0$ define um hiperplano a meia distância entre os hiperplanos $\mathrm{D(x)} = 1$ e $\mathrm{D(x)} = -1$. A distância entre esses hiperplanos é denominada de margem representada por ρ na Figura3.2. Supondo a existência de pelo menos uma amostra x para a qual $\mathrm{D\,(x)} = 1$, e outra amostra para a qual $\mathrm{D\,(x)} = -1$, então o hiperplano $\mathrm{D\,(x)} = 0$ representa a melhor superfície de separação dessas amostras. As amostras que se encontram sobre $\mathrm{D\,(x)} = 1$ e $\mathrm{D\,(x)} = -1$ são os vetores de suporte. O primeiro hiperplano é o hiperplano de suporte para a classe +1 e fica sobre a superfície de decisão, o segundo hiperplano é o hiperplano de suporte para a classe -1 e fica sob a superfície de decisão.

A largura da margem (ρ) pode ser calculada pela projeção da diferença entre os vetores de suporte em direção ao vetor normal da superfície de decisão, a construção pode ser visualizada na Figura3.3. A distância pode ser calculada através da seguinte

equação:

$$\begin{aligned}\rho &= \|x_p - x_q\| \cos\gamma \\ &= \frac{\mathbf{w} \bullet (x_p - x_q)}{\|w\|} \\ &= \frac{(b+1)(b-1)}{\|w\|} \\ &= \frac{2}{\|w\|}\end{aligned} \tag{3.5}$$

Como queremos maximizar a margem entre os hiperplanos de suporte isso implica em maximizar $\frac{2}{\|\mathbf{W}\|}$. No entanto podemos expressar esse problema de maximização com um problema de minimização $\frac{\|\mathbf{W}\|}{2}$, podendo ser reescrito como $\frac{\|\mathbf{W}\|^2}{2}$. Segundo (**?**), a otimização sobre valores positivos $\|\mathbf{W}\|$ é invariante sobre a transformação com funções quadráticas.

A partir das considerações anteriores, se verifica que a maximização da margem de separação das amostras de treinamento, em relação a $\mathbf{w}^T\mathbf{x} + b = 0$, pode ser obtida pela minimização de $\|\mathbf{W}\|$. assim, queremos minimizar uma função quadrática com restrições lineares. A seguir apresentamos a formulação do problema de otimização:

$$\begin{aligned}&\underset{\mathbf{W},b}{\text{minimizar}}\ \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{sujeito à } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1,\ 1 \leq i \leq m.\end{aligned} \tag{3.6}$$

Por ser um problema convexo, a resolução de 3.6 pode ser realizada recorrendo aos multiplicadores de Lagrange. Em 3.6, temos que $\mathbf{w} \in R^n$ com $i = 1, \ldots, m$ e a restrição $y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ é linear. A restrição é inserida para assegurar que não existam amostras de treinamento na região de separação entre as classes. A introdução da restrições no problema de minimização pode ser resolvida por meio da têcnica dos multiplicadores de Lagrange (α). A nova função objetivo é dada pela diferença entre a função objetivo anterior e o produto das respectivas restrições com os multiplicadores de Lagrange, segue a formulação do Lagrangiano primal:

$$Q(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{M} \alpha_i y_i (w^T x_i + b) + \sum_{i=1}^{M} \alpha_i \tag{3.7}$$

sendo $\alpha = (\alpha_1, \ldots, \alpha_M)$ os multiplicadores de Lagrange um vetor de dimensão m com $\alpha_i \geq 0$. A solução para este problema de extremos pode então ser obtida minimizando $Q(\mathbf{w}, b, \alpha)$ com relação a $\mathbf{w}, b$ e maximizando com relação a $\alpha_i \geq 0$

$$\frac{\partial Q(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^{M} \alpha_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{M} \alpha_i y_i x_i \tag{3.8}$$

$$\frac{\partial Q(w, b, \alpha)}{\partial b} = -\sum_{i=1}^{M} \alpha_i y_i = 0 \tag{3.9}$$

acrescidas das condições:

$$\alpha \left\{ \mathrm{y_i}(w^T x_i + b) - 1 \right\} \text{ para } i = 1, \ldots, M, \tag{3.10}$$

As equações 3.7, 3.8, 3.9 são conhecidas como as condições de Karush-Kunh-Tucker (KKT), (ABE, 2005). Substituindo as equações 3.8 e 3.9 em 3.7, obtém-se uma equação expressa em termos de α somente:

$$Q(w, b, \alpha) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i=1}^{M} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{3.11}$$

O problema de otimização, portanto, se torna maximizar (3.11) com respeito à α e sujeito as restrições:

$$\sum_{i=1}^{M} \alpha_i y_i \mathbf{x}_i = 0 \; e \; \alpha_i \geq 0 \; para \; i = 1, \ldots, M \tag{3.12}$$

Essa formulação é denominada forma dual. Segundo (HAMEL, 2009), a visão dual do problema tem consequências interessantes. Uma das consequências mais relevantes é que um classificador linear baseado em SVM pode ser facilmente estendido para um classificador não linear. Na formulação dual as amostras de treinamento aparecem na forma de produto interno $\mathbf{x}_i \cdot \mathbf{x}_j$.

A forma dual apresenta como coeficientes α_i diferentes de zero os vetores de suporte, isto é, para os exemplos de treinamento que verificam a igualdade na equação (3.4). Uma vez estimados os parâmetros que definem o hiperplano ótimo, constrói-se uma função de decisão:

$$D(x) = \sum_{i \in S} \alpha_i y_i x_i^T x + bD(x) = \sum_{i \in S} \alpha_i y_i x_i^T x + b \tag{3.13}$$

3.2 SVM MARGEM SUAVE

O problema de otimização (3.6) possui solução somente no caso das amostras $\mathbf{x}_i$ pertencerem a duas classes linearmente separáveis. Em situações reais é pouco provável que duas classes sejam separáveis por um hiperplano no seu espaço original. O caso das classes não linearmente separáveis é tratado de forma idêntica, sendo, porém, necessário introduzir uma penalização às observações que se encontram do lado errado do hiperplano. Os hiperplanos que introduzem essa penalização são conhecidos como hiperplanos de margem suave, conforme a Figura 3.8.

No caso não separável se permite que existam amostras de treinamento dentro da margem máxima. A nova definição dos hiperplanos de suporte é obtida introduzindo a variável de folga ξ_i para cada ponto:

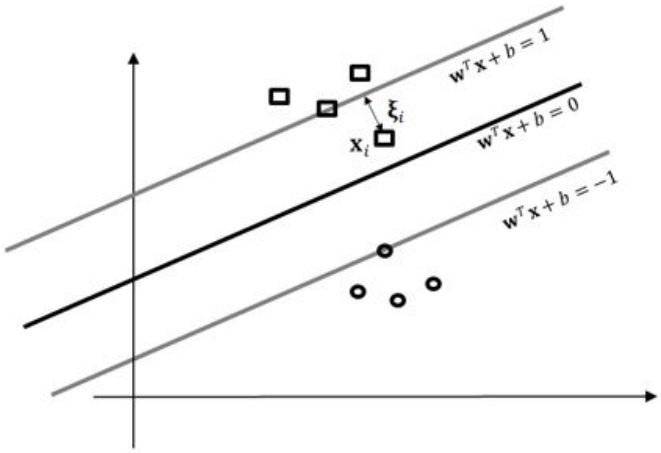

Figura 3.4: Um hiperplano separador com custos ξ associados a amostras de treinamento incorretamente classificadas. Adaptado de Adaptado de (HAMEL, 2009).

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi,\ \mathbf{x}_i \in \omega_1(y_i = 1) \tag{3.14}$$
$$\mathbf{w}^T\mathbf{x}_i + b \geq -1 + \xi,\ \mathbf{x}_i \in \omega_2(y_i = -1) \tag{3.15}$$
$$\xi_i \geq 0\ \forall\ i \tag{3.16}$$

O procedimento de suavização da margem do classificador linear permite que alguns dados de treinamento permaneçam entre os hiperplanos de suporte. Além disso, permite também a ocorrência de alguns erros de classificação.

Para o caso onde $0 < \xi_i < 1$, a amostra correspondente não terá margem máxima, mas será rotulada corretamente. No caso de $\xi_i \geq 1$, a amostra $\mathbf{x}_i$ será rotulada erroneamente.

Para que ocorra um erro é necessário que o respectivo ξ_i seja maior que 1, assim $\sum_i \xi_i$ é o limite superior para o número de erros de treinamento. Agora o problema de otimização deve minimizar os erros de treinamento. Assim se modifica a função objetivo para minimizar:

$$Q(\mathbf{w}, b, \alpha) = \frac{1}{2} \parallel \mathbf{w}\parallel^2 + C\sum_{i=1}^{M} \xi_i \tag{3.17}$$

Onde C é um parâmetro a ser escolhido pelo usuário, quanto maior o valor do parâmetro C, maior será a penalização associada aos erros cometidos. A constante C é conhecida como "parâmetro de margem" e estabelece a importância relativa das duas parcelas do lado direito da igualdade da equação (3.17) neste processo de minimização. A minimização de $\|\mathbf{w}\|^2$resulta na maximização da margem, enquanto que a minimização da segunda parcela $(\sum_i^M \xi_i)$ resulta na mimização do erro de classificação.

A solução de (3.17) é obtida de forma análoga ao caso separável e reescrevendo o problema primal Lagrangiano:

$$Q(w,b,\alpha) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{M}\xi_i - \sum_{i=1}^{M}\alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^{M}\beta_i\xi_i \quad (3.18)$$
$$\text{onde } \alpha = (\alpha_1, \ldots, \alpha_M)^T \ \ \beta = (\beta_1, \ldots, \beta_M)^T$$

Conforme Abe (2005), para a solução ótima as condições de KKT, apresentadas a seguir, devem ser satisfeitas.

$$\frac{\partial Q(w,b,\xi,\alpha,\beta)}{\partial w} = 0 \quad (3.19)$$
$$\frac{\partial Q(w,b,\xi,\alpha,\beta)}{\partial b} = 0 \quad (3.20)$$
$$\frac{\partial Q(w,b,\xi,\alpha,\beta)}{\partial \xi} = 0 \quad (3.21)$$
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0 \, para \, i = 1, \ldots, M \quad (3.22)$$
$$\beta_i\xi_i \geq 0 \ \ para \ \ i = 1, \ldots, M \quad (3.23)$$
$$\alpha_i \geq 0, \beta \geq 0, \xi \geq 0 \, para \, i = 1, \ldots, M. \quad (3.24)$$

Determinam-se os multiplicadores de Lagrange positivos $\alpha_i \geq 0$ e $\beta_i \geq 0$, igualando as derivadas em ordem $\mathbf{w}, \xi$ e b e, substituindo estas relações em (3.18), obtemos a seguinte formulação:

$$\begin{aligned} &\text{Maximizar} \quad \mathrm{Q(w,b,\alpha)} = \sum_{\mathrm{i=1}}^{\mathrm{M}}\alpha_\mathrm{i} - \tfrac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{M}}\alpha_\mathrm{i}\alpha_\mathrm{j}\mathrm{y_i y_j x_i^T x_j} \\ &\text{sujeito a} \quad \sum_{\mathrm{i=1}}^{\mathrm{M}}\mathrm{y_i}\alpha_\mathrm{i} = 0, \mathrm{C_i} \geq \ \alpha_\mathrm{i} \geq 0 \text{ para } \ 1 \leq \mathrm{i} \leq \mathrm{m} \end{aligned} \quad (3.25)$$

Ainda segundo Abe (2005), a única diferença entre o SVM de margem suave e o SVM de margem rígida é que $\alpha_i$não pode exceder C. A função de decisão é a mesma para o caso de margem rígida

$$D(x) = \sum_{i \in S}\alpha_i y_i x_i^T x + b \quad (3.26)$$

3.3 TRUQUE DO KERNEL

Conforme Abe (2005), o hiperplano ótimo do classificador SVM é determinado de maneira a maximizar a capacidade de generalização. Porém, se os dados de treinamento não forem linearmente separáveis, o hiperplano obtido pelo classificador pode ter baixo poder de generalização, mesmo que o hiperplano seja determinado de maneira ótima. Assim, para melhor a separabilidade linear, o espaço original é mapeado em um espaço de dimensão mais alta chamado de espaço de características.

O Truque do *Kernel* possibilita que o espaço original seja mapeando em um espaço de produto escalar de alta-dimensão chamado espaço de características, onde os dados

podem ser linearmente separáveis. Na Figura 3.5 é possível visualizar um exemplo de mapeamento de um espaço de baixa dimensão (bidimensional) para um espaço de dimensão mais alto (tridimensional). No espaço de dimensão mais alta é possível separar os dados com uma superfície linear. Na prática são utilizados espaços de características com dimensões muito altas.

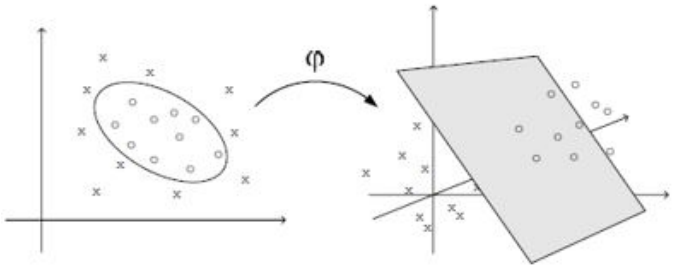

Figura 3.5: Mapeamento de dados para um espaço de características de mais alta dimensão através do truque do *kenel* – Adpatado de (PRESS et al., 2007).

Utilizando uma função vetorial não-linear $g\left(\mathbf{x}\right) = \left(g_1\left(\mathbf{x}\right), \ldots g_l\left(\mathbf{x}\right)\right)^T$, que mapeia um vetor de entrada m-dimensional para um espaço de característica l-dimensional, a função de decisão linear no espaço de características é dada por:

$$D(\mathbf{x}) = \mathbf{w}^T g(\mathbf{x}) + b \tag{3.27}$$

onde **w** é um vetor *l*-dimensional e b é o termo independente.

De acordo com a teoria de Hilbert-Schmidt, se uma função simétrica $H(\mathbf{x}, \mathbf{x}^{'})$ que satisfaz:

$$\sum_{i,j}^{M} h_i h_j H(x_i, x_j) \geq 0 \tag{3.28}$$

para todo M, $\mathbf{x}_i$ e h_i, onde M toma um número natural e h_i toma um número real, existe uma função de mapeamento que $g(\mathbf{x})$, que mapeia $\mathbf{x}$ em um espaço de característica de produto interno e $g(\mathbf{x})$ satisfaz:

$$H(\mathbf{x}_i, \mathbf{x}_j) = g^T(\mathbf{x}) g(\mathbf{x}') \geq 0 \tag{3.29}$$

Se a equação (3.29) é satisfeita, então:

$$\sum_{i,j}^{M} h_i h_j H(x_i, x_j) = \left(\sum_{i=1}^{M} h_i g^T(x_i)\right) \left(\sum_{i=1}^{M} h_i g(x_i)\right) \geq 0 \tag{3.30}$$

Em Abe (2005), as condições impostas por (3.28) e (3.30) são conhecidas como condições de Mercer e as funções que satisfazem essas condições são conhecidas como *kernel* positivos semi-definido ou *kernel* de Mercer.

A vantagem de utilizar um kernel é que não é necessário tratar o espaço de características em alta dimensão explicitamente. Essa têcnica é conhecida como truque do

kernel. Ao invés de utilizar a função $g(\mathbf{x})$ utilizamos $H\left(\mathbf{x}, \mathbf{x}^{'}\right)$, não sendo necessário explicitar a função $g(\mathbf{x})$.

Utilizando um kernel de Mercer, o problema dual no espaço característica é formulado como um problema de otimização conforme se segue:

$$\begin{aligned} &\text{Maximizar} \qquad \mathrm{Q}(\alpha) = \sum_{\mathrm{i}=1}^{\mathrm{M}} \alpha_{\mathrm{i}} - \tfrac{1}{2} \sum_{\mathrm{j,i}=1}^{\mathrm{M}} \alpha_{\mathrm{i}}\alpha_{\mathrm{j}}\mathrm{y_i y_j K(x_i, x_j)} \\ &\text{sujeito} \qquad \sum_{\mathrm{i}=1}^{\mathrm{M}} \mathrm{y_i}\alpha_{\mathrm{i}} = 0,\ 0 \leq \alpha_{\mathrm{i}} \leq \mathrm{C} \ \text{ para } \ \mathrm{i} = 1, ..., \mathrm{M}. \end{aligned} \tag{3.31}$$

Segundo Abe (2005), devido à $H\left(\mathbf{x}, \mathbf{x}^{'}\right)$, ser um *kernel* positivo semi-definido, o problema de otimização (3.30) é um problema côncavo quadrático. E, devido ao $\alpha = 0$ ser uma solução viável, o problema de otimização tem um mínimo global. As condições de complementariedade de KKT são dadas por

$$\alpha_i \left(y_i \left(\sum_{i \in S}^{M} y_i \alpha_i H(x_i, x_j) + b \right) - 1 + \xi_i \right) = 0 \tag{3.32}$$

$$para \ \ \mathrm{i} = 1, ..., \mathrm{M} \tag{3.33}$$

$$(C - \alpha_i) + \xi_i = 0 \text{ para } \mathrm{i} = 1, ..., \mathrm{M} \tag{3.34}$$

$$\alpha_i \geq 0, \xi_i \geq 0 \text{ para } \mathrm{i} = 1, ..., \mathrm{M} \tag{3.35}$$

A função de decisão assume a forma a seguir:

$$\mathbf{D(x)}=\sum_{i \in S} \alpha_i y_i H(\mathbf{x}_i, \mathbf{x}) + b \tag{3.36}$$

Onde b é dado pela equação:

$$b = y_j - \sum_{i \in S}^{M} \alpha_i y_i k(x_i, x_j) \tag{3.37}$$

De acordo com HUANG (2006), a função de *kernel* é fundamental na localização de limites de decisão entre classes que formam fronteiras complexas. Através do mapeamento dos dados de entrada, os quais no espaço original não possuem fronteiras lineares de decisão, em um espaço de dimensão mais alta em que os dados podem ser linearmente separáveis. Então nesse espaço as fronteiras de decisão são localizadas através de um algoritmo de otimização. No entanto a escolha de uma função de *kernel* adequada e os valores apropriados dos parâmetros desse *kernel* podem afetar consideravelmente a performance do SVM. Os *kernel* mais comumente utilizados na literatura são o *kernel* Linear e o *kernel* RBF são apresentados na tabela a seguir:

Na literatura, para a classificação de imagens hiperespectrais em geral, utiliza-se o *kernel RBF*. Em (MELGANI; BRUZZONE, 2004) e (ANDREOLA; HAERTEL, 2010), o kernel RBF apresentou acurácia superior se comparado ao *kernel* polinomial. No presente trabalho adotaremos o *kernel* RBF para classificação.

Nome	Função	Pârametros Livres
Kernel Linear	$K\left(\mathbf{x},\mathbf{x}^{'}\right)=\mathbf{x}^{T}\mathbf{x}'$	Nenhum
Kernel polinomial homogêneo	$K\left(\mathbf{x},\mathbf{x}^{'}\right)=\left(\mathbf{x}^{T}\mathbf{x}^{'}\right)^{d}$	d≥ 2
Kernel polinomial não homogêneo	$K\left(\mathbf{x},\mathbf{x}^{'}\right)=\left(\mathbf{x}^{T}\mathbf{x}^{'}+1\right)^{d}$	d≥ 2
Kernel RBF	$K\left(\mathbf{x},\mathbf{x}^{'}\right)=\exp(-\gamma\left\|\mathbf{x}-\mathbf{x}'\right\|^{2})$	$\sigma>0$

Tabela 3.1: Funções de kernel utilizadas com mais frequência na literatura e seus respectivos parâmetros livres.

O *kernel RBF*, conforme apresentado na Tabela 3.1 , possui um parâmetro Gama (γ) que é equivalente a $\gamma=\frac{1}{2\sigma^{2}}$. A função de *kernel*, considerando o parâmetro gama como $\frac{1}{2\sigma^{2}}$, é mostrada a seguir:

$$k(x,x')=\exp\left(-\frac{\left\|x-x^{'}\right\|_{2}^{2}}{2\sigma^{2}}\right) \tag{3.38}$$

Segundo (SHAWE-TAYLOR; CRISTIANINI, 2004), o parâmetro γ controla a flexibilidade da função de *kernel*, valores pequenos de γ permitem ao classificador ajustar todos os rótulos havendo risco de sobre ajustamento (*overfitting*). Nesse caso o kernel da matriz se aproxima da matriz de identidade. Por outro lado, valores grandes de gama reduzem o *kernel* para uma função constante, tornando impossível o processo de aprendizagem de qualquer classificador não-trivial.

3.4 SMO (SEQUENTIAL MINIMAL OPTIMIZATION)

A equação (3.31) é um problema de otimização quadrático, o qual envolve uma matriz que tem número de elementos igual ao quadrado do número de exemplos de treinamento. Ou seja, quando o número de exemplos de treinamento é grande o processo de treinamento tende a ser custoso quanto ao tempo. O algoritmo SMO, (PLATT, 1999), se propõe a resolver de forma eficiente o problema de otimização quadrática relacionado ao processo de treinamento do classificador SVM.

Segundo PLATT (1999), o SMO é um algoritmo conceitualmente simples, fácil de implementar e geralmente é mais rápido que a implementação tradicional do algoritmo de treinamento do SVM. O SMO decompõe o problema de programação quadrática (*Quadratic Programing* - QP) em subproblemas, utilizando o teorema de Osuna (OSUNA; FREUND; GIROSI, 1997) para assegurar a convergência. O teorema de Osuna prova que um problema QP pode ser decomposto em um série de pequenos subproblemas.

O SMO utiliza dois multiplicadores de Lagrange em cada iteração do algoritmo. A solução pode ser obtida de forma analítica. Existem dois componentes principais do algoritmo de SMO:

1. O método analítico, para resolver os dois multiplicadores de Lagrange.
2. A heurística, para escolher quais multiplicadores devem ser otimizados.

O SMO resolve sempre o menor problema de otimização possível em cada iteração.

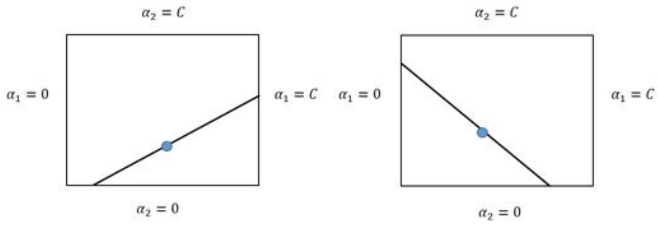

Figura 3.6: Espaço formado por (α_1, α_2) e as restrições $\mathbf{0} \leq \alpha_1, \alpha_2 \leq \mathbf{C}$

.

Supondo, sem perda de generalidade, que os multiplicadores de Lagrange escolhidos pela heurística são α_1 e α_2 e suas respectivas classes y_1e y_2. Devido à restrição linear da equação (3.31), $\sum_{i=1}^{M} y_i \alpha_i = 0$, ou seja, quando um multiplicador for atualizado, a condição linear obriga o ajuste para que o outro multiplicador mantenha a validade da restrição linear $\sum_{i=1}^{M} y_i \alpha_i = 0$. Os novos multiplicadores de Lagrange devem respeitar a igualdade a seguir:

$$\alpha_1^{novo} y_2 + \alpha_2^{novo} y_2 = \text{constante} = \alpha_1 y_1 + \alpha_2 y_2 \tag{3.39}$$

No espaço formado por (α_1, α_2) e as restrições $0 \leq \alpha_1, \alpha_2 \leq C$, os dois multiplicadores de Lagrange resultantes devem obedecer às restrições impostas pelo problema original. As desigualdades restringem os multiplicadores em um quadrado e a restrição linear obriga aos multiplicadores a ficarem em uma linha diagonal, conforme a figura a seguir:

O algoritmo primeiro computa o segundo multiplicador de Lagrange α_2 e computa o final da reta diagonal em termos de α_2. Se $y_1 \neq y_2$, então as seguintes restrições são aplicadas a α_2,

$$\begin{aligned} L &= \max(0, \alpha_2 - \alpha_1) \\ H &= \min(0, C - \alpha_2 + \alpha_1) \end{aligned} \tag{3.40}$$

e se y_1=y_2, então as seguintes restrições são aplicadas a α_2:

$$\begin{aligned} L &= \max(0, \alpha_2 + \alpha_1 - C) \\ H &= \min(0, \alpha_2 + \alpha_1) \end{aligned} \tag{3.41}$$

A segunda derivada da função objetivo ao longo da reta diagonal pode ser expressa conforme a equação a seguir:

$$\eta = k(x_1, x_1) + k(x_2, x_2) - 2k(x_1, x_2) \tag{3.42}$$

Segundo PLATT (1999), em circunstâncias normais, a função objetivo será positiva semi-definida, haverá um mínimo ao longo da direção da restrição linear de igualdade e η será maior que zero. Nesse caso, o algoritmo de SMO computa o mínimo na direção da restrição:

$$\alpha_2^{novo} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta} \tag{3.43}$$

O valor de E_i é a diferença entre $\mathrm{D}(\mathrm{x}_i)$ e é definido em (3.36), e y_i é o rótulo de x_i, ou seja, E_i é o erro no i-ésimo exemplo de treinamento $E_i = D(\mathrm{x}_i) - y_i$.

O máximo da função objetivo será obtido com o valor:

$$\alpha_2^{novo*} = \begin{cases} H, \text{ se } \alpha_2^{novo*} \geq H \\ \alpha_2^{novo*}, \text{ se } \alpha_2^{novo*} < H \\ L, \text{ se } \alpha_2^{novo*} \leq L \end{cases} \tag{3.44}$$

O valor de α_1^{novo} é computado a partir de α_2^{novo*}, conforme a equação a seguir,

$$\alpha_1^{novo} = \alpha_1 + y_1 y_2(\alpha_1 - \alpha_2^{novo}) \tag{3.45}$$

3.5 HEURÍSTICA DE ESCOLHA DOS MULTIPLICADORES

Têm-se duas heurísticas separadas, uma para o primeiro multiplicador de Lagrange e outra para o segundo. No laço externo do algoritmo responsável pela primeira heurística, o algoritmo busca, dentre todos os exemplos de treinamentos, aqueles que violam as condições de KKT, conforme (3.32),(3.33), (3.34) e (3.35), e escolhe qualquer dos pontos que violam as condições de KKT para atualizar. Quando tal ponto é encontrado, se utiliza uma segunda heurística para selecionar o segundo ponto, e o valor dos respectivos multiplicadores são atualizados. Em seguida, o algoritmo retorna ao laço externo buscando por novas violações nas restrições de KKT. A fim de aumentar a chance de encontrar violações nas condições de KKT, o laço externo busca pontos para os quais os correspondentes α_i satisfazem a seguinte restrição $0 < \alpha_i < c$. Satisfazer a restrição anterior implica que esses valores não estão na fronteira da região factível. Somente quando todos os pontos satisfizerem as condições de KKT, para um determinado nível de tolerância, o algoritmo será finalizado.

De acordo com NELLO; SHAWE-TAYLOR (2000), na segunda heurística, o segundo ponto x_2 deve ser escolhido de tal forma que, ao atualizar o par α_1, α_2, a atualização cause uma grande mudança na função objetivo dual. De maneira a encontrar um bom ponto, sem ser custoso computacionalmente, se escolhe um ponto x_2 que maximize a quantidade $|E_1 - E_2|$. Se E_1 é positivo, o SMO escolhe uma amostra de treinamento E_2 cujo erro seja mínimo. Se E_1 é negativo, o SMO escolhe uma amostra de treinamento com máximo erro E_2. A lista adicional de erros de todos os pontos de treinamento é mantida na memória para reduzir cálculos adicionais.

Caso não aconteça um aumento significativo na função objetivo dual, o algoritmo SMO tenta cada ponto, que satisfaz $0 < \alpha_i < c$, por sua vez. Se ainda não houver progressos significativos, o SMO busca através de todo o conjunto de treinamento por um ponto adequado.

Se ocorrerem alterações de valores, a heurística retorna para escolher outros pontos $\mathbf{x}_1$ e $\mathbf{x}_2$, até que todos os pontos estejam obedecendo às condições de KKT. Quando todos os pontos estiverem obedecendo às condições de KKT, termina o processo.

3.6 CLASSIFICAÇÃO MÚLTIPLAS CLASSES

Os problemas de classificação reais apresentam normalmente várias classes, sendo esse tipo de problema denominado "classificação multiclasses". Existem duas abordagens básicas para estender o SVM para classificação de múltiplas classes. A primeira consiste em reduzir o problema multiclasses a uma série de problemas de classificação binária. Os métodos mais utilizados para realizar essa decomposição em múltiplas classes são as abordagem um-contra-todos, um-contra-um e a decomposição em árvore binária. A segunda abordagem é a generalização do SVM para mais de duas classes, um exemplo dessa abordagem é o método de *error-correcting codes* proposto por DIETTERICH; BAKIRU (1995).

Na formulação um-contra-todos, a classe é determinada pela i-ésima função de decisão $g_i\left(x\right)\left(i=1,\ldots,n\right)$, onde *n* é o número de classes, de modo que quando $\mathbf{x}$ pertence à classe i,

$$g_i(x) > 0 \tag{3.46}$$

e quando $\mathbf{x}$ pertence a qualquer uma da classes restantes,

$$g_i(x) < 0 \tag{3.47}$$

Podem, ainda, ocorrer regiões inclassificáveis quando mais de uma função de decisão é positiva ou nenhuma das funções de decisão é positiva, (ABE, 2005), gerando uma região sem classificação. Segundo ABE (2005) o método de um-contra-um constrói $n(n-1)/2$ funções de decisão, onde n é o número de classes. Cada função de decisão é treinada com duas classes. Uma vez construídas todas as superfícies de decisão, se pode classificar $\mathbf{x}$ aplicando cada uma das $n(n-1)/2$ superfícies de decisão, contando quantas vezes x foi classificado em uma determinada classe. A classe que obtiver o maior número de contagens é considerada o rótulo para $\mathbf{x}$. Caso aconteça um empate entre as classes, uma das têcnicas utilizadas para resolver esse impasse é escolher a classe com menor índice.

A formulação seguinte é baseada em uma árvore de decisão. Pode ser considerada como uma variação do método um-contra-todos. Determina-se a i-ésima função de decisão $g_i\left(x\right)\left(i=1,\ldots,n-1\right)$, de modo que quando x pertence à classe i

$$g_i(x) > 0 \tag{3.48}$$

e quando $\mathbf{x}$ pertence a uma das demais classes $\{i=1,\ldots,n-1\}$,

$$g_i(x) < 0 \tag{3.49}$$

Dada uma amostra $\mathbf{x}$, inicia-se a busca pelo nó raiz. A amostra $\mathbf{x}$ é avaliada pela função de decisão desse nó, se movendo para um dos dois nós filhos, dependendo do valor

de saída. O processo se repete em cada nó, até que se atinja um nó folha indicando a qual classe x pertence.

3.7 CLASSIFICADOR DE DECISÃO EM ÁRVORE

O classificador de Decisão em árvore (CDA) é um grafo direcionado acíclico e deve satisfazer algumas propriedades:

1. Possui apenas um nó chamado raiz, no nível zero e sem arestas de chegadas;
2. O nó raiz contém todos os padrões de todas as n classes a serem classificadas pela árvore;
3. Cada um dos demais nós contém uma e apenas uma aresta de chegada, existindo um único caminho do nó raiz aos demais nós;
4. Considera-se como nó filho, os nós que são originados por determinado nó que será chamado de pai;
5. Os nós que não possuem nós filhos são chamados de nós folhas ou terminais. Em tais nós, o padrão discriminado recebe o rótulo da classe do nó.

A Figura 3.7 apresenta um classificador estruturado em forma de árvore onde, no nível 0, temos o nó raiz do classificador. No nó raiz um critério direciona para qual lado da árvore um padrão deve seguir, para o nó filho da direita, se $\mathrm{g_1\,(x) > 0}$ ou da esquerda, se $\mathrm{g_1\,(x) < 0}$. No nível m os nós folhas contendo padrões que foram discriminados e receberam rótulos de classe, $\omega_1 \ldots \omega_\mathrm{n}$.

Existem vários métodos heurísticos para construção de árvores de decisão, entre eles o *bottom-up*. Nesta abordagem, a árvore binária é construída a partir de um conjunto contendo todas as classes, dispostas no nó raiz. Através de um conjunto de características das amostras de treinamento das classes é estimada uma medida de separabilidade com o objetivo de identificar o par de classes que apresenta a maior distância entre suas componentes. Estas duas classes são então utilizadas na definição das regras de decisão que irão caracterizar os dois nós descendentes. Este procedimento se repete a cada nó, até que os nós folhas (constituídos por apenas uma classe) sejam atingidos.

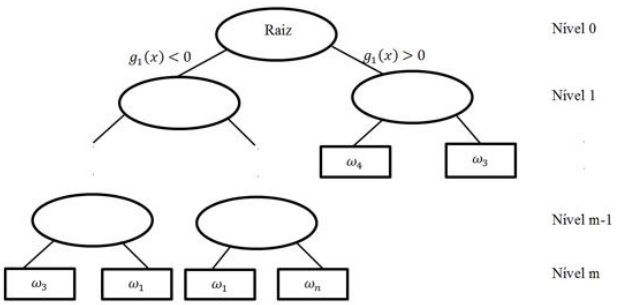

Figura 3.7: Estrutura de um classificador de decisão em árvore binária. Adaptado de (SAVAVIAN; LANDGREBE, 1991).

O objetivo do presente trabalho é combinar a construção da árvore de decisão com uma heurística para selecionar os parâmetros do kernel do classificador SVM em cada nó da árvore. Supõe-se que a seleção de parâmetros ótimos em cada nó da árvore podem apresentar uma melhora na acurácia geral do classificador. A seleção dos parâmetros do kernel é realizada através de uma heurística baseada no Recozimento Simulado, Support Vector Machines *Binary Tree Amoebasa Binary* (SVMBT-AMOEBASA). O algoritmo de Recozimento Simulado é implementado conforme a proposta de R. L. SALCEDO (1996). Essa heurística é descrita na próxima seção.

3.8 RECOZIMENTO SIMULADO (SIMULATED ANNEALING)

A ideia básica do algoritmo de Recozimento Simulado para problemas de otimização *N*-dimensionais contínuos é encontrar um valor para uma função $f(x)$ (*função objetivo*), o qual idealmente deve ser o mínimo global. A heurística é aplicada em um espaço onde existem diversos mínimos locais e x é um vetor *N*-dimensional.

O algoritmo de Recozimento Simulado é um método de busca estocástica modelado de acordo com o processo físico de recozimento oriundo da termodinâmica. O recozimento se refere ao processo de um sistema têrmico inicialmente fundido em alta temperatura e então lentamente resfriado até atingir uma temperatura baixa e um estado estável (estado de menor energia).

No trabalho de (KIRKPATRICK; JR.; GELATT, 1983) foi proposta uma conexão entre o processo de recozimento simulado (*simulated annealing*) e problemas de otimização combinatória.

Segundo FLOUDAS; PARDALOS (2008), a principal vantagem do algoritmo de recozimento simulado (SA) é sua capacidade de "escapar" de ótimos locais usando um mecanismo, o qual permite a deterioração de valores da função objetivo. Ou seja, permite que durante o processo de otimização sejam aceitos valores piores da função objetivo, no caso de problemas de minimização valores maiores. A aceitação é controlada probabilisticamente através da temperatura T.

Particularmente nos primeiros estágios do SA, onde T é relativamente alta, o espaço de soluções é largamente explorado, identificando distintas direções de soluções e aceitando soluções piores com alta probabilidade. À medida que a temperatura vai diminuindo, a probabilidade de aceitar soluções com valores piores de função objetivo vai diminuindo. Com a aceitação controlada de movimentos de subida (*"uphill"*), ou seja, aumentando o valor da função objetivo em um processo de minimização ou diminuindo o valor da função objetivo em um processo de maximização, se pode evitar armadilhas de ótimos locais.

Considerando um processo de minimização e tomando $\triangle E$ como a variação da função objetivo, a variação corresponde à diferença entre o estado atual e o estado anterior. Em analogia com o processo físico de recozimento, a variação de $\triangle E$ é a mudança no nível de energia. A probabilidade P de aceitar uma solução de subida é igual a $e^{-\triangle E/(k_B T)}$, onde k_B é a constante de *Boltzmann*. A seguir é apresentado o algoritmo de SA.

Procedure SIMULATED_ANNEALING
Begin
Gera aleatoriamente uma solução inicial;
Inicializa *T;*
FAÇA $T > 0$
FAÇA enquanto o equilíbrio termal não for atingido
Gera um estado vizinho aleatoriamente;
avalia $\triangle E$;
atualiza o estado atual;
Se ($\triangle E < 0$) com novo estado;
Se ($\triangle E \geq 0$) com novo estado com probabilidade $e^{-\triangle E/(k_B T)}$;
End
Diminuir *T conforme o cronograma de recozimento definido;*
End
End

A solução de um problema de otimização através do algoritmo de recozimento simulado envolve os seguintes passos:

1. A definição de uma função objetivo a ser minimizada.
2. A adoção de um cronograma de resfriamento, que representa os parâmetros do algoritmo a serem configurados. A temperatura inicial, o número de configurações a serem gerado em cada temperatura e um método para diminuir a temperatura em cada iteração.
3. Em cada temperatura no cronograma de resfriamento, o algoritmo deve produzir uma mudança na configuração do sistema de maneira estocástica.
4. Um mecanismo que aceita ou rejeita o estado atual do sistema.
5. Adoção de um critério adequado de parada para o algoritmo.

O algoritmo de recozimento, conforme apresentado anteriormente, é aplicado com sucesso em problemas discretos, como o problema clássico do caixeiro viajante, o qual tenta determinar a menor rota entre uma sárie de cidades. A ideia básica do algoritmo pode ser aplicada a problemas n-dimensionais contínuos.

Segundo PRESS et al. (2007), um mecanismo que produza mudanças na configuração do sistema, levando x até $\mathrm{x} + \triangle \mathrm{x}$, ou seja, que produza mudanças no valor da funçãoo objetivo de formas aleatórias é ineficiente se, quando sai de um movimento de "descida", quase sempre propõe um movimento de subida, isto no caso de um problema de minimização. Além disso, o mecanismo de movimento também deve ser capaz de escapar de vales estreitos e profundos, onde pode encontrar mínimos locais e atingir regiões mais baixas.

R. L. SALCEDO (1996) propõe a implementação do algoritmo de recozimento simulado para espaços contínuos combinado ao algoritmo *downhill* simplex. O método de *downhill* foi proposto por Nelder e Mead (1965).

O método de *downhill* simplex ou método de amoeba é uma têcnica de otimização não-linear multidimensional. O método usa o conceito de simplex, que é uma figura geométrica (polítopo) em que o número de vértices é igual ao número de variáveis mais um.

Conforme PRESS et al. (2007), o método amoeba executa uma série de passos, a maioria das etapas apenas se movendo para o ponto do simplex onde a função é a maior ("ponto mais alto"), partindo de uma face oposta do simplex, um ponto mais baixo. Estes passos são chamados reflexões e são construídos de modo a conservar o volume do simplex. O algoritmo, quando possível, expande o simplex em uma ou outra direção para que passos maiores sejam realizados.

Quando o algoritmo atinge um "vale profundo", o método faz uma contração no sentido transversal e tenta se infiltrar pelo vale. Se houver uma situação em que o simplex está tentando "passar através de um buraco de agulha"(vale estreito e profundo), se contrai em todas as direções, e em torno do ponto mais baixo (o melhor ponto). Este método é chamado de amoeba (ameba, que realiza movimento ameboide (contrações rítmicas)) devido a esse tipo de comportamento.

A figura a seguir mostra a representação de um problema de otimização onde são utilizadas três possíveis soluções, círculos coloridos em azul quadriculado. Cada solução possui um valor de função objetivo associado, a solução pior é aquela, por exemplo, em que um problema de minimização possui um valor mais alto.

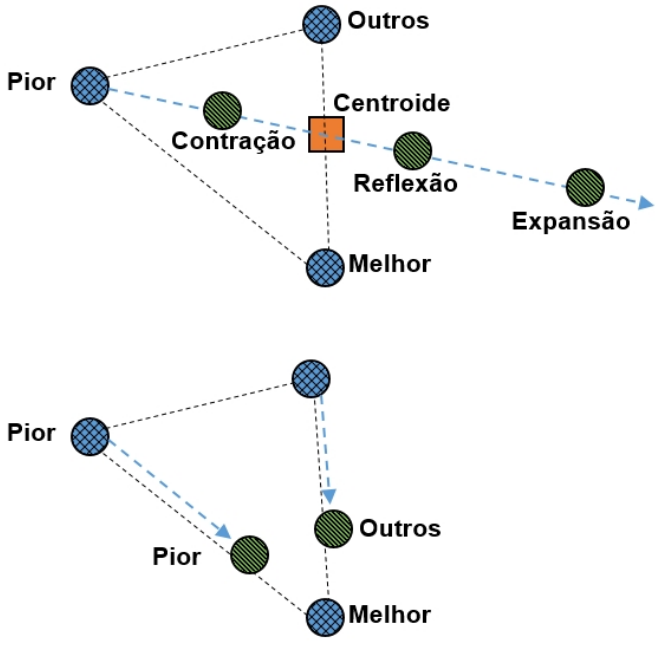

Figura 3.8: Representação de possíveis movimentos realizados pelo algoritmo amoeba. Baseado em (MCCAFFREY, 2013).

O algoritmo AMOEBA geralmente é empregado na solução de problemas multidimensionais. Uma das principais questões nesse tipo de algoritmo é o critério de parada. Uma alternativa é encerrar o processo quando a distância do vetor movimentado naquele passo é menor que um valor de tolerância definido.

R. L. SALCEDO (1996) propõe o algoritmo SIMPSA que combina o algoritmo de Recozimento Simulado como o algoritmo simplex de Nelder e Mead (NELDER; MEAD, 1965). O cronograma de resfriamento proposto para o algoritmo envolve escolher a temperatura inicial conforme a equação a seguir:

$$X = \frac{m_1 + m_2 \exp\left(\frac{\triangle f^+}{T_i}\right)}{m_1 + m_2} \tag{3.50}$$

Onde X representa a taxa de aceitação, que foi configurada para 95%. Para um universo de configurações m, que é calculado como $100 \times N$, onde N é o número de dimensões do problema. O parâmetro m_1 representa o número de movimentos realizados com sucesso, m_2 representa o número de movimentos que pioraram os resultados e $\triangle f^+$ representa a média no incremento na função de custos para m_2.

Para que a equação (3.50) possa ser aplicada, uma temperatura inicial bastante alta é configurada, por exemplo, 10^5. Essa temperatura permite que todos os movimentos sejam inicialmente aceitos e, apenas um número m de movimentos, a temperatura T_i seja utilizada para o cronograma de resfriamento.

O cronograma de resfriamento é implementado através da equação:

$$T^{j+1} = \frac{T^j}{1 + \frac{T^j ln(1+\ln)}{3\sigma}} \tag{3.51}$$

Onde j é a iteração corrente. O parâmetro δ controla a taxa de resfriamento. Pequenos valores (< 1) produzem convergência lenta e valores (> 1) podem produzir convergência pobre para ótimos locais. O σ e o desvio padrão relacionado aos valores da função de custos produzidos em todas as interações.

O algoritmo de simplex é acoplado ao recozimento simulado de forma a produzir o movimento pela espaço de variáveis, conforme proposto por PRESS et al. (2007), utilizando uma variável com distribuição logarítmica proporcional ao parâmetro de temperatura T, essa variável é associada com todos os vértices do simplex. Um valor similar a variável aleatória é subtraído da função em cada novo ponto substituído, conforme a seguir:

$$(F_{perturbao}) = F_k - T \times \ln(rnd), k = 1, N + 1 \tag{3.52}$$

$$(F_{perturbao})^{novos} = F_k + T \times \ln(rnd) \tag{3.53}$$

Onde F_k é o valor da função no vértice k, F_{novo} é o valor da função no ponto substituído e $F_{perturbao}$ é o valor de perturbação da função. Para um problema de minimização, *N+1* vértices são perturbados com valores baixos, de acordo com a equação (3.52). E os pontos de substituição são perturbados com valores altos, de acordo com a equação (3.53)

Os critérios de parada são um ponto fundamental na utilização de um processo heurístico. O algoritmo implementado no presente trabalho utiliza quatro critérios de parada:

1. A diferença entre o melhor e o pior valor da função de avaliação do algoritmo AMOEBA é menor que um valor de tolerância.
2. A máxima diferença entre as coordenadas dos vértices do simplex é menor que um valor de tolerância definido.
3. Valor máximo de iterações definidas do algoritmo foi atingido.
4. Tempo máximo de execução definido foi atingido.

4 ALGORITMO PROPOSTO

A metodologia proposta no presente trabalho é a construção de um classificador SVM para problemas que envolvem múltiplas classes, baseado em uma árvore binária, onde cada nó da árvore é formada por um hiperplano cujos parâmetros do kernel devem ser selecionados através de uma heurística de maneira a se obter a maior acurácia global possível. Em geral, na literatura, os trabalhos que propõem uma metodologia para incrementar a acurácia abordam a seleção dos parâmetros do kernel de forma global, ou seja, utilizam os mesmos parâmetros para todos os pares de classes. Nesta seção é apresentada a metodologia proposta para fins de seleção de parâmetros ótimos em cada nó do classificador estruturado em forma de árvore binária. Os experimentos visando avaliar a metodologia proposta serão descritos na próxima seção.

Conforme mencionado anteriormente, uma abordagem ao problema de classificação de múltiplas classes foi investigada, decompondo o problema em uma série de problemas de classificações binárias. Para tanto se utilizou uma estrutura em árvore binária com abordagem do tipo *bottom-up*, conforme a proposta de ANDREOLA; HAERTEL (2010). Nesta abordagem se empregou a distância de *Bhattacharyya* para identificar em cada nó da árvore binária o par de classes que apresenta a maior separabilidade.

Segundo DUDA; HART; STORK (2000), a distância de *Bhattacharyya* é uma distância estatística que pode ser usado na estimação da separabilidade entre um par de classes. Assumindo uma distribuição Normal multivariada para os dados, esta distância estatística assume a seguinte forma:

$$B = \frac{1}{8}(\mu_1 - \mu_2)^T \left(\frac{\sum_1 + \sum_2}{2}\right)^{-1} (\mu_1 - \mu_2) + \frac{1}{2}\ln\left(\left|\frac{\frac{\sum_1 + \sum_2}{2}}{|\sum_1|^{1/2}|\sum_2|^{1/2}}\right|\right) \quad (4.1)$$

Onde μ_1e μ_2 são vetores de média e Σ_1 e Σ_2 são matrizes de covariância das classes.

O algoritmo foi implementado durante o desenvolvimento dessa dissertação em MATLAB (2012), utilizando paradigma orientado a objeto. O arquivo svmTree.m, mostrado no código em anexo 7.4, constrói a árvore binária conforme os seguintes critérios:

1. Todas as amostras de treinamento de todas as classes são atribuídas ao nó raiz;
2. Supondo-se que os dados são normalmente distribuídos pelo critério da distância de *Bhattacharyya*, equação (4.1), escolhe-se o par de classes que, apresentando a maior distância estatística, darão origem aos nós filhos;

3. Determinadas as classes que darão origem aos nós filhos, se faz uso do algoritmo de Recozimento Simulado que tem por objetivo, em cada nó, selecionar o conjunto de parâmetros com maior poder discriminante para este par de classes;

4. Neste ponto se usa a regra de decisão do classificador SVM, equações(3.36) e (3.37);

5. Utilizando-se as respectivas funções de decisão, se classificam as amostras de treinamento das demais classes em um dos dois nós filhos;

6. Neste ponto se utiliza o conceito de Limiar de Verossimilhança (LV), proposto por (MORAES, 2005).Segundo MORAES (2005) o LV é um parâmetro introduzido no estudo com a finalidade de definir o critério de pertinência de uma classe aos nós descendentes. O LV estipula a fração dos indivíduos na amostra de treinamento que devem ser alocados em um nó, para que a classe em questão seja alocada unicamente a este nó. Caso a porcentagem de amostras de treinamento de uma dada classe classificada em um dos nós filhos seja maior que o LV determinado pelo usuário – entre 0 e 100% – todas as amostras serão atribuídas a este nó filho. Caso contrário, as amostras de treinamento desta classe são replicadas em ambos os nós filhos.

7. Esse processo será repetido ao longo da árvore binária até que cada n contenha apenas uma classe (nó terminal, ou nó folha).

8. Na etapa final a árvore gerada é salva em disco. Cada nó da árvore contém a superfície de decisão gerada com seus respectivos parâmetros.

O algoritmo de recozimento simulado, conforme descrito na seção 3.8, foi implementado no arquivo “amoebasa.m”, o algoritmo é responsável por otimizar os parâmetros do classificado SVM. O parâmetro gama do kernel RBF, conforme apresentado na Tabela 3.1, e o parâmetro C, que controla a compensação dos erros cometidos na fase de treinamento, e a margem rígida, permitindo que alguns erros de classificação sejam cometidos. O anexo 7.2 mostra o código fonte da implementação. A função de custo do algoritmo é implementada no arquivo “AmoebaSaFunction.m”, conforme o anexo 7.3. A função de custo calcula a acurácia relacionada a um par de classes e os parâmetros *gama* e C, fornecidos pelo processo de otimização.

O classificador SVM foi incialmente implementado em MATLAB utilizando a rotina “quadprog” para resolver o problema de programação quadrática, conforme a equação 3.32 e formulação de (ABE, 2005). Devido à performance foi posteriormente substituído pela biblioteca desenvolvida em C libsvm, (CHIH-CHUNG; LIN, 2011), conforme a formulação de (PLATT et al., 2000). A biblioteca libsvm foi instalada em conjunto com o MATLAB, sendo utilizada para realizar o treinamento do classificador. O arquivo “SVMABE” 7.1 implementa o classificador SVM, conforme mostrado na ver nos anexos 7.1.

O método svmtrain da biblioteca libsvm é usado para treinar o classificador, recebendo como parâmetros o tipo de kernel, o parâmetro gama e o parâmetro C. O kernel utilizado é o kernel RBF e os parâmetros são gerados pela heurística de recozimento simulado, método de busca em grade ou parâmetros fixos. O svmtrain desenvolve um modelo que posteriormente é utilizado para o método de classificação, nesse modelo é armazenada a superfície de decisão gerada pelo classificador, os vetores de suporte e o termo

independente *b*. Nos casos da busca em grade e do recozimento simulado, ao obter o melhor conjunto de parâmetros para aquele par de classes, ou seja, nó da árvore, o modelo é salvo para posteriormente ser utilizado no processo de classificação.

No diagrama, Figura 4.1 é mostrado o algoritmo de treinamento do classificador proposto SVMBT-AMOEBASA.

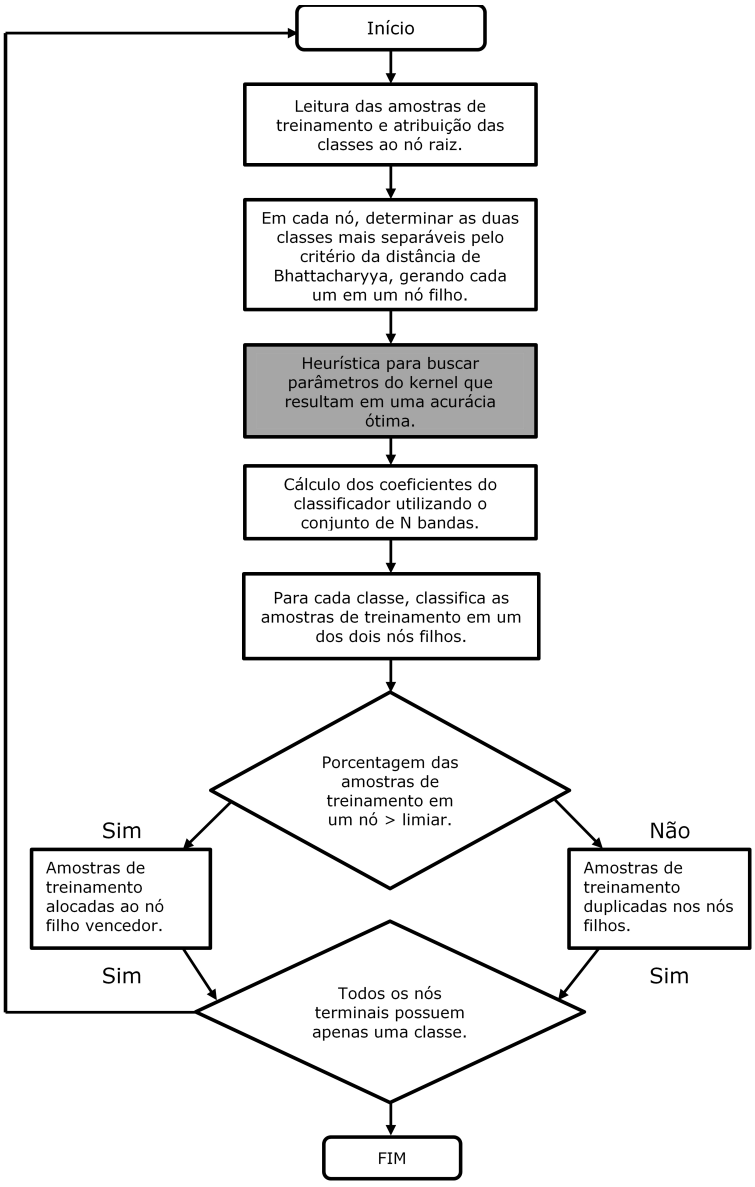

Figura 4.1: Fluxograma do algoritmo de treinamento do classificador.

Para o algoritmo de teste do classificador, se pode observar o fluxograma da Figura 4.2 , que segue os seguintes passos:

1. Todas as amostras de teste são atribuídas ao nó raiz;
2. Com base na superfície de decisão, calculada no processo de treinamento, em cada nó, se decide em qual nó filho a amostra de teste será classificada;
3. Este processo é repetido para cada amostra, ao longo de vários níveis na árvore

binária, até que o nó terminal seja atingido, atribuindo desta forma um rótulo em cada uma das amostras.

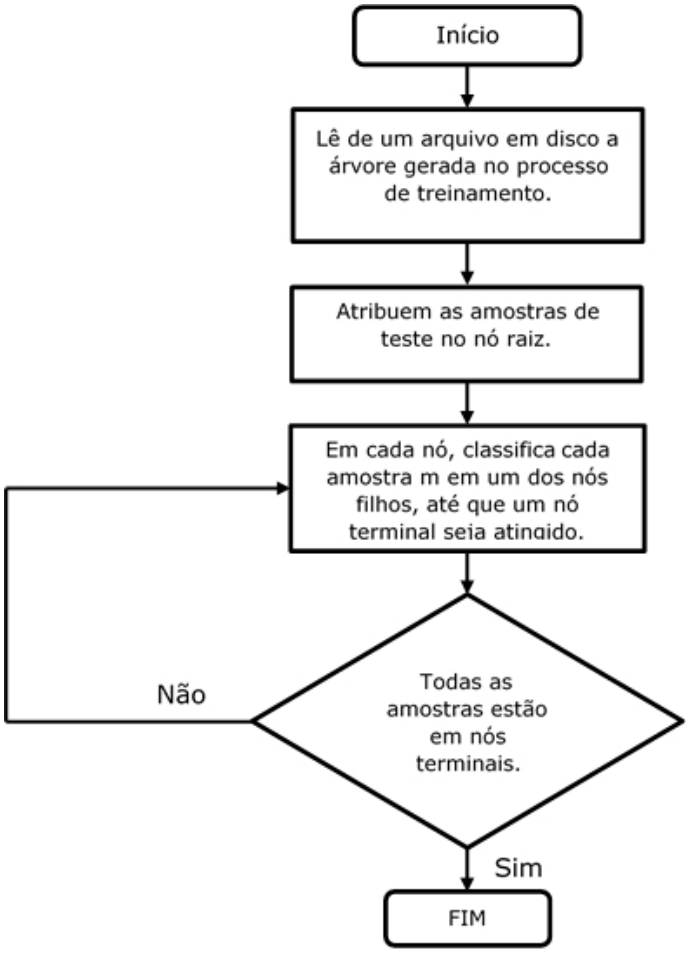

Figura 4.2: Fluxograma do algoritmo de treinamento do classificador.

Uma das maiores dificuldades na seleção dos parâmetros ótimos do SVM é a determinação do espaço inicial de busca. Um espaço de busca muito amplo implica em um maior número de iteração, principalmente em algoritmos de busca exaustiva, como a Busca em Grade.

O espaço inicial de busca dos parâmetros foi selecionado de acordo com resultados obtidos em ANDREOLA; HAERTEL (2010), além disso, foi realizado um processo de inspeção dos resultados de vários testes realizados na busca de parâmetros ótimos. Duas ferramentas foram utilizadas para inspecionar os valores dos testes realizados e estimar um espaço inicial de busca mais relevante.

Visando auxiliar na visualização dos árvore gerada pelo software foi desenvolvida uma rotina para gerar um grafo de visualização da árvore construída pelo classificador. A visualização foi é gerada pela biblioteca JGRAPHX 2.2.0.0 (2012),a biblioteca é inteiramente escrita em linguagem JAVA. Foi desenvolvida uma classe em (MATLAB, 2012) utilizando a biblioteca JGRAPHX 2.2.0.0 (2012) para gerar uma árvore a partir dos nós criados pelo classificador.

Em cada nó é possível visualizar a classes que geraram a superfície de decisão daquele nó, todas as classes oriundas do nó pai, o valor do parâmetro gama obtido pelo

processo de otimização daquele nó, o valor do parâmetro C, além da acurácia obtida ao classificar as duas classes utilizando os parâmetros do kernel ótimos. Ao inspecionar a árvore é possível visualizar a variação dos parâmetros, se obtendo uma boa intuição de um espaço inicial de busca.

Na figura 4.3 em destaque o nodo pai e dois nodos filhos. Em cada nó temos: ParentID (identificador do nó pai), NodeID (Identificador do nó), Classe1 e Classe 2 (classes utilizadas para construir a superficie de decisão). Além disso o valor de e C otimizado para a superfície de decisão do nó e a lista de classes que compõe o nodo. No case de existirem apenas duas classes o algoritmo irá produzir dois nós folhas.

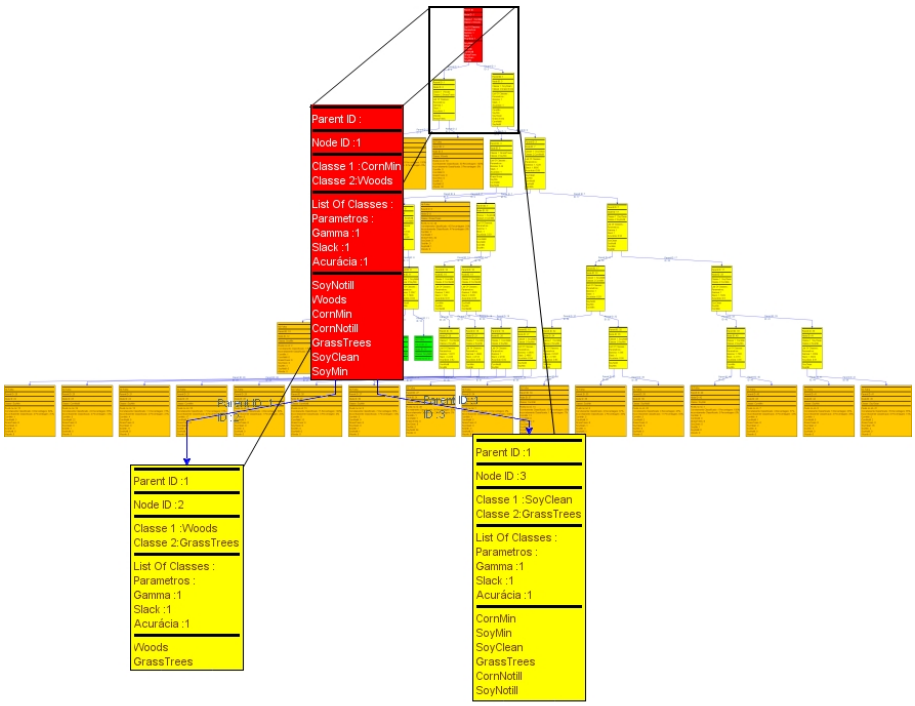

Figura 4.3: Nós gerados com seus respectivos parâmetros, o grafo da árvore é gerado durante o processo de teste utilizando a a implementação da biblioteca Jgraph (2012) em conjunto com o MATLAB.

Visando auxiliar a inspeção dos valores obtidos pelo processo de otimização foi criado um conjunto de tabelas em um banco de dados relacional MYSQL (2012). O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL - Linguagem de Consulta Estruturada, *(*Structured Query Language) como interface. O MYSQL possui amplo suporte a linguagem JAVA possuindo interface de conexão nativa. Isso permite que o MATLAB conecte com o banco de dados realizando operações como : consulta de dados, inserção e exclusão, além de outros. Na implementação SVMBT-AMOEBASA foram criadas rotinas que salvam no banco de dados os parâmetros obtidos em cada execução do processo de otimização.

Na Figura 4.4 é possível visualizar o diagrama das tabelas criadas no banco de dados que armazena os experimentos. A tabela de "testes" armazena o data e hora de início do teste, a data e hora do fim do teste. A tabela "parâmetro" armazena os parâmetros obtidos em cada iteração do algoritmo de otimização γ, C, a acurácia obtida por aquele par de classes, além do melhor conjunto de parâmetro obtido durante o processo de otimização

de um par de classes. A tabela “matrixcon” armazena a matriz de confusão cada teste realizado.

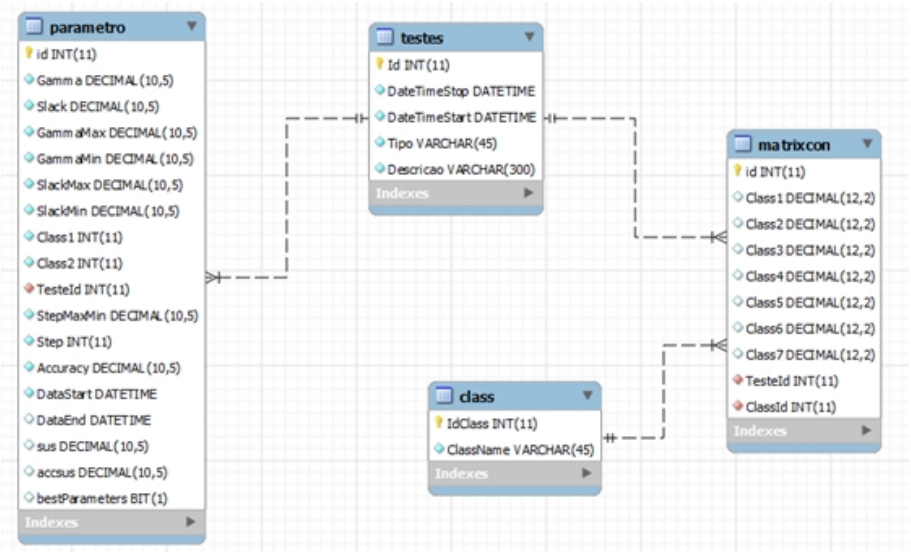

Figura 4.4: Diagrama entidade relacionamento do banco de dados criado para armazenar os experimentos executados.

5 EXPERIMENTOS

Nesta seção são descritos e discutidos os materiais e os experimentos que tem por finalidade validar a metodologia descrita na seção anterior. Para esta finalidade, foi desenvolvido no presente trabalho o aplicativo SVMBT-AMOEBASA, o qual tem por objetivo classificar dados em alta dimensionalidade. O aplicativo implementa a heurística de Recozimento Simulado construindo um classificador SVM em árvore binária. A heurística é implementada visando um incremento da acurácia do classificador, o aplicativo foi desenvolvido em ambiente MATLAB utilizando paradigma orientado a objetos, o classificador utiliza a biblioteca libsvm (CHANG, 2011). Além do MATLAB foram utilizadas duas bibliotecas desenvolvidas em Java, a MYSQL-CONNECTOR-JAVA que conecta o aplicativo com banco de dados MYSQL responsável por armazenar os resultados dos experimentos e parâmetros obtidos pelos métodos heurísticos, e a biblioteca JGRAPH que gera a visualização da árvore binária de treinamento, permitindo verificar visualmente o processo de classificação. Nestes experimentos são empregados dados em alta dimensionalidade coletados pelo sistema sensor AVIRIS sobre uma área agrícola de testes, desenvolvida pela *Purdue University*, denominada *Indian Pines*, localizada no noroeste do Estado de Indiana - EUA. A imagem é formada por dezesseis classes espectrais - Figura 5.1.

Figura 5.1: Cena captura pelo sensor AVIRIS utilizada na realização dos experimentos.

Da cena *Indian Pines*, foi selecionado de um segmento de imagem de (435x435) um recorte de (145x118), num total de 17110 pixels.

As classes escolhidas, que fazem parte da cena, são compostas por culturas que

possuem assinatura espectral bastante similar (vetores de médios muito próximos entre si) sendo, portanto, de difícil discriminação. As classes como soja cultivo mínimo e soja plantio direto diferem apenas pela técnica de cultivo, como as classes composta de milho. Além disso, a cena foi capturada no início da época de crescimento das culturas de soja e milho. Nesta etapa apenas aproximadamente 5% da área está efetivamente coberta pela vegetação, sendo os restantes 95% composto por solo exposto e resíduo de colheitas anteriores. Estas condições resultam em classes espectralmente muito semelhantes entre si, se constituindo, por esta razão, em um desafio o processo de classificação. Por se constituir um problema desafiador de classificação, diversos trabalhos encontrados na literatura utilizam esse conjunto de dados como base comparativa de desempenho.

Do conjunto de 220 bandas que dispõe a cena AVIRIS, foram removidas as bandas ruidosas causadas por problemas atmosféricos (vapor de água, CO2, O3). O número final de bandas espectrais a ser usada, isto é, a dimensionalidade dos dados é de 190, resultando em vetores com 190 valores de contador digital para cada pixel da imagem. Para fins dos experimentos foram selecionadas sete classes, conforme a Tabela 5.1.

CLASSES	**DESCRIÇÃO**	**AMOSTRAS DISPONÍVEIS**
ω_1 - Corn Min	Milho Cultivo Mínimo	834 x 190
ω_2 - Corn Notill	Milho Plantio Direto	1434 x 190
ω_3 - Grass Tress	Pastagem e árvores	747 x 190
ω_4 - Soybean Clean	Soja Cultivo Convencional	614 x190
ω_5 - Soybean Min	Soja Cultivo Mínimo	2468 x 190
ω_6 - Soybean Notill	Soja Plantio Direto	968 x 190
ω_7 – Woods	Mata	997 x 190

Tabela 5.1: Relação das classes utilizadas nos experimentos

Das sete classes, duas são espectralmente distintas de todas as outras, as classes *grass trees* e *woods*, sendo facilmente separáveis das demais classes, servindo de referência no processo de classificação.

5.1 AMOSTRAS DE TREINAMENTO

Do conjunto das amostras disponíveis para cada classe foram extraídos dois subconjuntos: um com amostras de treinamento e um segundo com amostras de teste. Com a finalidade de capturar as variáveis naturais que ocorrem ao longo da área coberta pela imagem, as amostras em ambos os subconjuntos foram extraídas alternadamente do conjunto das amostras disponíveis nos dados de verdade terrestre. Para tornar os resultados obtidos para as várias classes comparáveis entre si, foram utilizados subconjuntos de treinamento e de teste de mesmo tamanho para todas as classes em estudo: inicialmente foram tomadas 50 amostras por classe para treinamento e 300 amostras por classe para teste; em um segundo momento, 100 amostras para treinamento e 300 amostras para teste; em seguida se coletaram 200 amostras para treinamento e outras 300 para fins de teste; e finalmente um quarto conjunto, com 300 amostras de treinamento e 300 amostras de teste.

As amostras de treinamento e teste foram tomadas a intervalos regulares do conjunto total de amostras para cada classe, ou seja, não necessariamente as 50 primeiras amostras estão contidas no conjunto de 100 amostras e estas não necessariamente estão contidas nas 200 amostras seguintes e assim sucessivamente. As 300 amostras de teste são coletadas da mesma forma e são diferentes para cada classificador em árvore binária.

Os experimentos foram desenvolvidos com o objetivo de quantificar numericamente os resultados e desempenho da metodologia proposta, especialmente em relação à acurácia no processo de classificação das imagens hiperespectrais. O primeiro experimento utiliza valores de parâmetro de *kernel* que apresentam uma acurácia média alta, servindo de base de comparação com o método proposto para as três abordagens distintas quanto à seleção de diferentes parâmetros do *kernel* RBF. Foram projetados três experimentos:

1. O primeiro experimento constrói uma árvore binária contendo em cada nó uma superfície de decisão gerada por um par de classes. Em todos os nós são utilizados os mesmos parâmetros do *kernel*, os quais sabidamente apresentam bons resultados. O objetivo desse experimento é obter uma base de comparação com a metodologia proposta de selecionar o parâmetro mais adequado para cada par de classes.

2. O segundo experimento constrói uma árvore binária onde em cada nó os parâmetros do kernel são selecionados de maneira a produzir a maior acurácia possível. O método utilizado para selecionar os melhores parâmetros é a busca em grade. A grade gerada é formada pelos parâmetros gama e *C*, ambos variando a intervalos regulares entre $0,5$ e 10.

3. No terceiro experimento, a construção da árvore com parâmetros otimizados é realizada através de uma heurística baseada em Recozimento Simulado. O objetivo desse experimento é mostrar numericamente que uma heurística pode ser tão eficiente quando o método de busca em grade, apresentando menor tempo de processamento. Além disso, não exige do usuário conhecimento prévio do conjunto de dados, ou seja, parâmetros que podem apresentar boa acurácia. O espaço de busca é formado pelos parâmetros gama e C, iniciando no intervalo 0.1 até 10. O algoritmo realiza uma busca contínua dentro desse intervalo. Além disso, foi realizado um experimento adicional onde o espaço de busca foi ampliado até 100, ou seja, o espaço de busca é formado pelos parâmetros gama e *C* iniciando em 0.1 até 100. Esse último experimento visa mostrar a eficiência do algoritmo de recozimento simulado mesmo ampliando o espaço de busca.

O valor da dimensionalidade dos dados nos quatro experimentos, isto é, o número de bandas espectrais empregadas, variou entre 20 e 180. Desta maneira se objetiva analisar o comportamento da acurácia produzida pelo classificador SVM em função da dimensionalidade dos dados e dos parâmetros escolhidos.

Todos os experimentos apresentam como resultado a Matriz de Confusão e um grafo da árvore gerada. Os valores de acurácia em cada experimento foram estimados a partir de matrizes de contingência, ou matrizes de confusão. Neste processo, a terminologia empregada é definida a seguir:

1. Acurácia do Produtor: é estimada pela fração do número total das amostras de teste fornecidas ao classificador que foram rotuladas corretamente pelo classificador. Esta acurácia estima, portanto, a capacidade de o classificador reconhecer corretamente uma amostra.

2. Acurácia do Usuário: é estimada pela fração das amostras de teste rotuladas pelo classificador em cada uma das classes e que efetivamente pertencem a esta classe. Esta acurácia estima, portanto, o grau de confiança que se pode ter na imagem temática produzida pelo classificador.

3. Acurácia Média: é estimada pela razão do total de amostras classificadas corretamente em cada classe, ou seja, a soma dos valores na diagonal principal na matriz de contingência, pelo número total das amostras de teste.

5.2 Experimento 1

No primeiro experimento, em todos os casos, o parâmetro C foi fixado em 1 e o parâmetro gama variou 0.5, 1, 1.5 e 2. Os parâmetros são mantidos fixos em todos os nós da árvore. Nas tabelas 4,5,6 e 7 e nas figuras 16, 17,18 e 19 são apresentadas as acurácias médias obtidas pelo parâmetro C=1 e o parâmetro gama variando 0,5, 1, 1,5 e 2, respectivamente. O objetivo desse experimento à servir como base de comparação dos resultados obtidos pelo algoritmo proposto SVMBT-AMOEBASA. Esses valores de parâmetros foram escolhidos com base no trabalho de ANDREOLA; HAERTEL (2010).

Bandas	50 Amostras		100 Amostras		200 Amostras		300 Amostras	
20	72,71%	**00:00:06**	78,19%	**00:00:04**	81,05%	**00:00:05**	84,48%	**00:00:04**
40	74,67%	**00:00:03**	80,48%	**00:00:03**	83,05%	**00:00:05**	86,14%	**00:00:08**
60	75,52%	**00:00:04**	81,76%	**00:00:03**	85,00%	**00:00:08**	87,24%	**00:00:08**
80	76,48%	**00:00:04**	81,38%	**00:00:04**	84,52%	**00:00:08**	87,76%	**00:00:12**
100	75,76%	**00:00:06**	81,71%	**00:00:04**	84,38%	**00:00:07**	87,62%	**00:00:14**
120	76,76%	**00:00:05**	81,00%	**00:00:03**	84,10%	**00:00:10**	87,14%	**00:00:17**
140	76,71%	**00:00:05**	79,71%	**00:00:05**	83,67%	**00:00:09**	86,71%	**00:00:23**
160	75,38%	**00:00:06**	78,95%	**00:00:05**	83,00%	**00:00:10**	86,24%	**00:00:22**
180	75,81%	**00:00:05**	79,38%	**00:00:05**	82,57%	**00:00:13**	86,10%	**00:00:29**

Tabela 5.2: Experimento parâmetro fixo: gama = 0.5 e $C = 1$. A tabela mostra a acurácia obtida para o conjunto amostras treinamento x bandas ao classificar 2100 pixels com o número respectivos de banda, além disso é mostrado o tempo de treinamento.

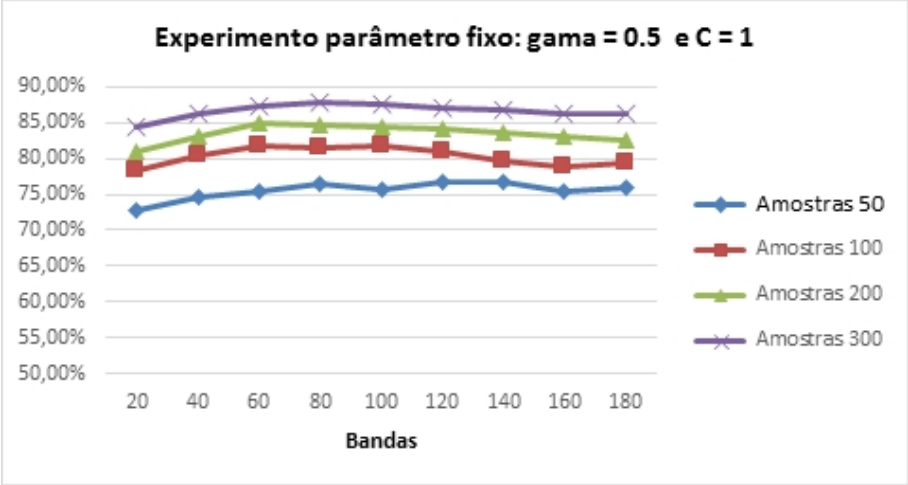

Figura 5.2: Acurácia média.

Bandas	50 Amostras		100 Amostras		200 Amostras		300 Amostras	
20	73,00%	**00:00:03**	78,43%	**00:00:03**	81,00%	**00:00:06**	84,57%	**00:00:06**
40	74,05%	**00:00:04**	79,19%	**00:00:04**	82,52%	**00:00:06**	85,57%	**00:00:09**
60	75,05%	**00:00:03**	78,57%	**00:00:04**	83,19%	**00:00:07**	85,90%	**00:00:12**
80	74,24%	**00:00:05**	77,62%	**00:00:06**	81,38%	**00:00:08**	85,33%	**00:00:13**
100	73,14%	**00:00:05**	78,10%	**00:00:05**	80,76%	**00:00:10**	85,00%	**00:00:16**
120	72,24%	**00:00:04**	76,71%	**00:00:05**	79,81%	**00:00:07**	84,14%	**00:00:16**
140	68,86%	**00:00:07**	74,57%	**00:00:08**	79,05%	**00:00:10**	82,95%	**00:00:18**
160	65,86%	**00:00:05**	73,29%	**00:00:07**	78,24%	**00:00:12**	82,05%	**00:00:18**
180	66,10%	**00:00:06**	71,43%	**00:00:07**	77,57%	**00:00:12**	81,48%	**00:00:22**

Tabela 5.3: Experimento parâmetro fixo: gama = 1 e C = 1. A tabela mostra a acurácia obtida para o conjunto amostras treinamento x bandas ao classificar 2100 pixels com o número respectivos de banda, além disso é mostrado o tempo de treinamento.

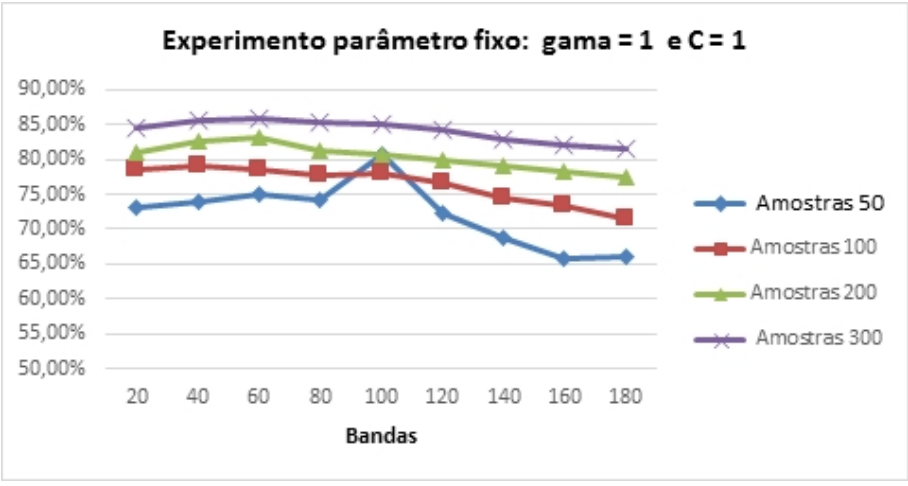

Figura 5.3: Acurácia média.

Bandas	50 Amostras		100 Amostras		200 Amostras		300 Amostras	
20	72,24%	**00:00:02**	77,86%	**00:00:03**	81,19%	**00:00:05**	84,14%	**00:00:07**
40	72,90%	**00:00:04**	77,19%	**00:00:04**	80,71%	**00:00:06**	84,24%	**00:00:09**
60	72,67%	**00:00:04**	76,48%	**00:00:04**	80,33%	**00:00:07**	84,10%	**00:00:11**
80	70,14%	**00:00:03**	75,24%	**00:00:04**	78,90%	**00:00:08**	83,00%	**00:00:12**
100	68,05%	**00:00:05**	73,29%	**00:00:07**	77,29%	**00:00:08**	82,29%	**00:00:11**
120	64,57%	**00:00:06**	71,62%	**00:00:06**	75,76%	**00:00:07**	80,57%	**00:00:11**
140	59,62%	**00:00:05**	68,05%	**00:00:06**	75,33%	**00:00:09**	79,71%	**00:00:17**
160	57,81%	**00:00:05**	65,76%	**00:00:06**	74,38%	**00:00:12**	78,67%	**00:00:19**
180	55,71%	**00:00:08**	63,29%	**00:00:07**	72,52%	**00:00:10**	78,57%	**00:00:17**

Tabela 5.4: Experimento parâmetro fixo: gama = 1.5 e C = 1. A tabela mostra a acurácia obtida para o conjunto amostras treinamento x bandas ao classificar 2100 pixels com o número respectivos de banda. Além disso, é mostrado o tempo de treinamento.

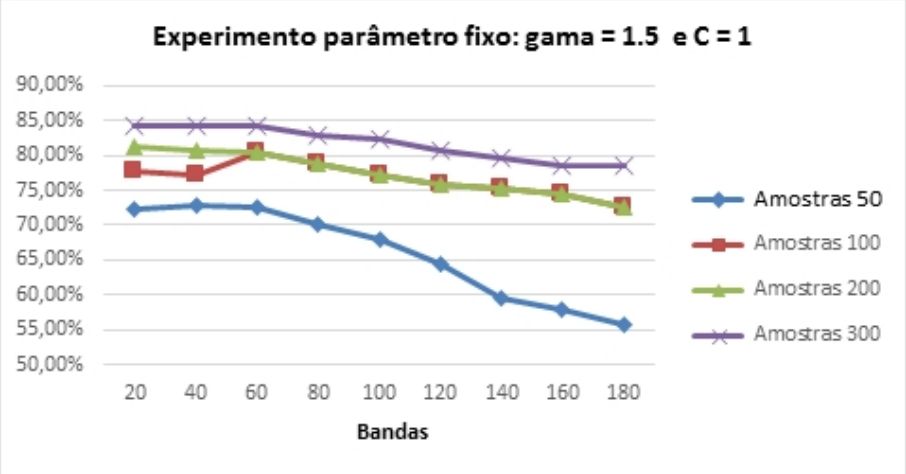

Figura 5.4: Acurácia média.

Bandas	50 Amostras		100 Amostras		200 Amostras		300 Amostras	
20	71,14%	**00:00:02**	77,05%	**00:00:03**	80,24%	**00:00:04**	83,38%	**00:00:05**
40	70,38%	**00:00:02**	75,52%	**00:00:04**	78,90%	**00:00:07**	82,48%	**00:00:09**
60	70,90%	**00:00:02**	73,71%	**00:00:04**	78,14%	**00:00:06**	82,43%	**00:00:10**
80	64,19%	**00:00:03**	71,33%	**00:00:05**	75,43%	**00:00:06**	80,00%	**00:00:09**
100	62,67%	**00:00:04**	67,81%	**00:00:07**	74,05%	**00:00:06**	79,48%	**00:00:12**
120	59,00%	**00:00:05**	67,38%	**00:00:06**	72,76%	**00:00:07**	77,71%	**00:00:13**
140	51,57%	**00:00:06**	61,33%	**00:00:06**	70,62%	**00:00:08**	76,81%	**00:00:16**
160	52,14%	**00:00:04**	58,52%	**00:00:06**	68,76%	**00:00:11**	76,05%	**00:00:17**
180	50,52%	**00:00:05**	55,29%	**00:00:06**	66,95%	**00:00:11**	75,29%	**00:00:14**

Tabela 5.5: Experimento parâmetro fixo: gama = 2 e C = 1. A tabela mostra a acurácia obtida para o conjunto amostras treinamento x bandas ao classificar 2100 pixels com o número respectivos de banda, além disso é mostrado o tempo de treinamento.

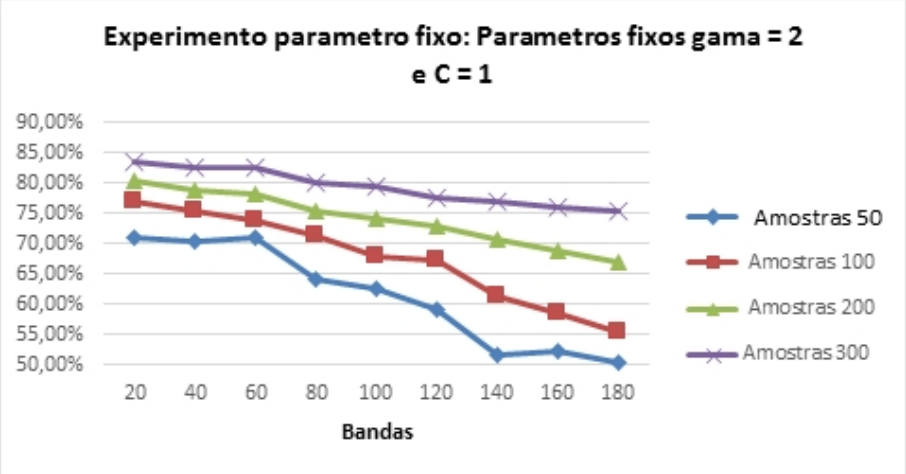

Figura 5.5: Acurácia média.

5.3 Experimento 2

No segundo experimento, em cada nó da árvore são selecionados os melhores parâmetros através do método de busca em grade, ou seja, que produzem a maior acurácia. O espaço de variáveis é formado por uma grade de pontos onde o valor do parâmetro C varia de 1 até 10 e o valor do parâmetro gama variando de 1 até 10, sendo selecionado o par de parâmetros (C, γ), que produz a maior acurácia média para cada par de classes. No primeiro experimento o passo utilizando é de 0.5 e no segundo é de 0.1.

A Tabela 5.6 e o gráfico 5.6 a seguir apresentam os resultados obtidos pelo método de busca em grade iniciando em 1 variando de 0.5 até 10 para γ e C.

Bandas	50 Amostras		100 Amostras		200 Amostras		300 Amostras	
20	75,00%	**00:13:17**	80,86%	**17:46:43**	83,81%	**01:37:38**	87,48%	**00:34:41**
40	75,48%	**00:11:19**	80,86%	**17:55:32**	84,00%	**01:28:48**	87,62%	**00:43:46**
60	75,57%	**00:10:34**	80,19%	**17:57:40**	84,67%	**01:17:05**	88,10%	**00:59:07**
80	75,05%	**00:10:13**	79,33%	**17:50:28**	82,67%	**01:11:56**	86,81%	**00:52:09**
100	73,52%	**00:08:30**	79,00%	**17:30:15**	81,95%	**01:27:21**	86,29%	**01:00:58**
120	72,24%	**00:07:52**	76,81%	**17:12:22**	80,71%	**01:13:42**	85,33%	**01:00:12**
140	69,90%	**00:08:37**	74,90%	**16:38:01**	79,57%	**01:20:37**	83,57%	**01:07:29**
160	67,67%	**00:08:54**	73,90%	**16:05:30**	78,67%	**01:38:44**	83,10%	**01:11:44**
180	66,43%	**00:08:50**	71,67%	**15:47:44**	78,33%	**01:24:16**	82,10%	**01:14:06**

Tabela 5.6: Experimento Grid: espaço de busca iniciando em 1 variando 0.5 até 10 para gama e C. A tabela mostra a acurácia obtida para o conjunto amostras treinamento x bandas ao classificar 2100 pixels com o número respectivos de banda, além disso é mostrado o tempo de treinamento.

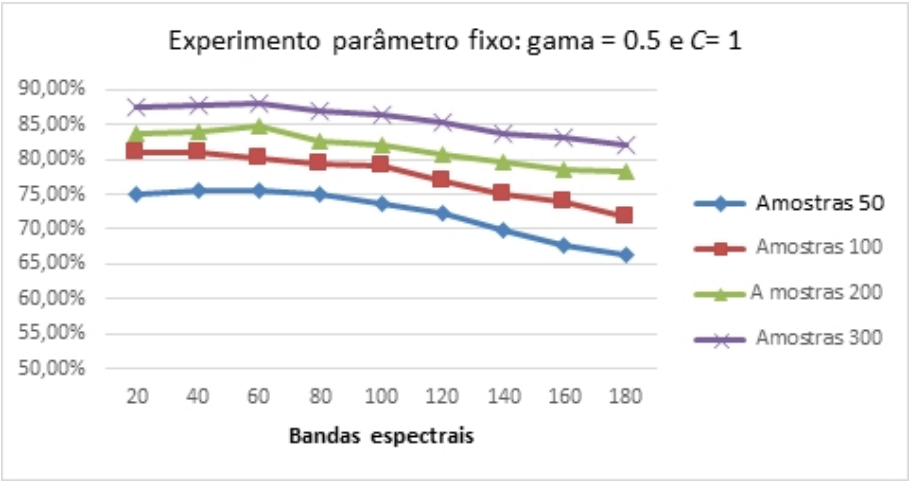

Figura 5.6: Acurácia média.

A Tabela 5.7 e Figuras 5.7 é possível verificar os resultados para o espaço de busca ampliado, iniciando em 1 variando de 0.1 até 10 para γ e C. Devido ao tempo de processamento elevado foram realizados testes com 50 amostras de treinamento.

Bandas	50 Amostras	Tempo
20	66,43%	03:28:35
40	67,62%	03:33:21
60	69,62%	03:13:56
80	72,48%	03:05:32
100	73,52%	03:19:54
120	74,24%	03:52:33
140	75,43%	03:58:00
160	75,00%	04:16:55
180	75,48%	04:27:48

Tabela 5.7: Experimento Grid: espaço de busca iniciando em 1 variando 0.1 até 10 para gama e C. A tabela mostra a acurácia obtida para o conjunto amostras treinamento x bandas ao classificar 2100 pixels com o número respectivos de banda, além disso é mostrado o tempo de treinamento.

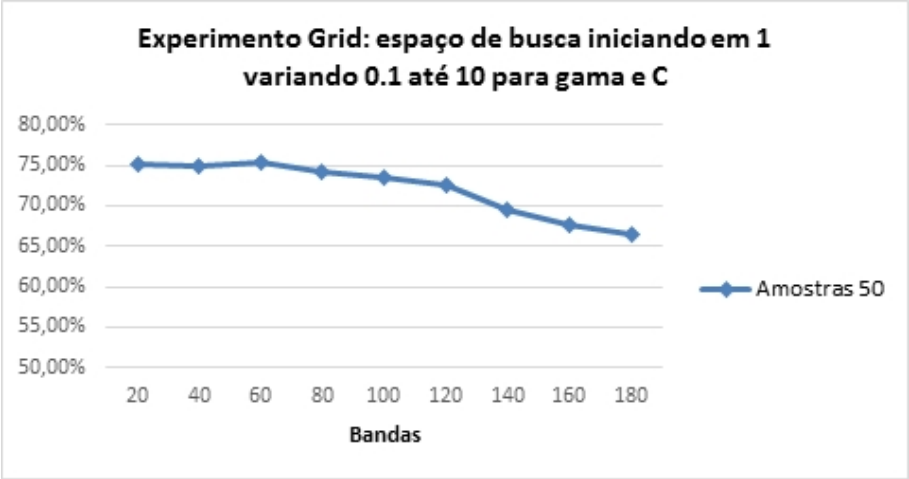

Figura 5.7: Acurácia média.

Conforme é possível perceber pela Figura 5.8, um passo menos discreto produz um incremento considerável no tempo de processamento. Na Figura 5.9 podemos perceber que, ao diminuir a discretização no espaço de busca, não implica em um aumento considerável da acurácia. A determinação do espaço de busca é justamente o maior problema da aplicação do método de busca em grade, já que o tempo de processamento é diretamente relacionado ao passo utilizado. O espaço de busca utilizado foi previamente determinado pelo Experimento 1, ou seja, exigiu conhecimento prévio do espaço de busca, porém, a granularidade do espaço de busca só pode ser determinada pela execução do método.

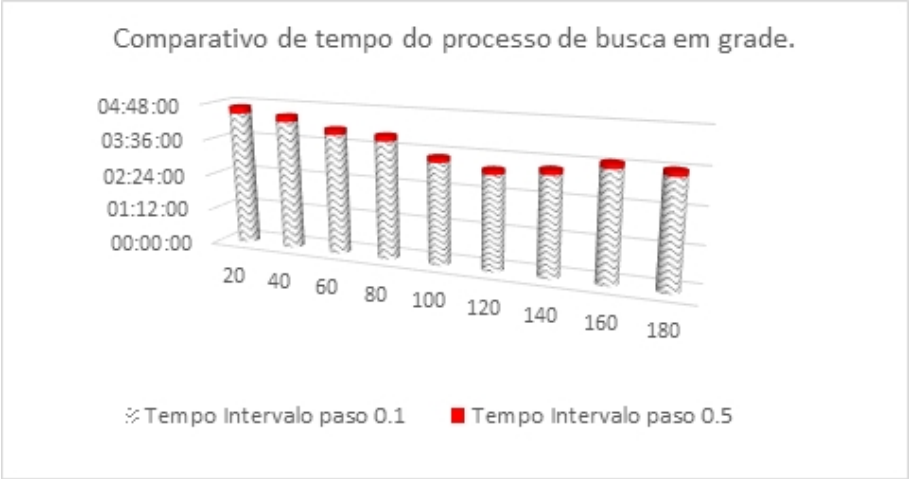

Figura 5.8: Comparativo de tempo do processo de busca em grade iniciando o espaço de busca em 1 e variando 0.5 até 10 para gama e *C*. O gráfico compara tempo do mesmo processo quando é utilizado um passo de 0.1. Foram usadas 50 amostras de treinamento.

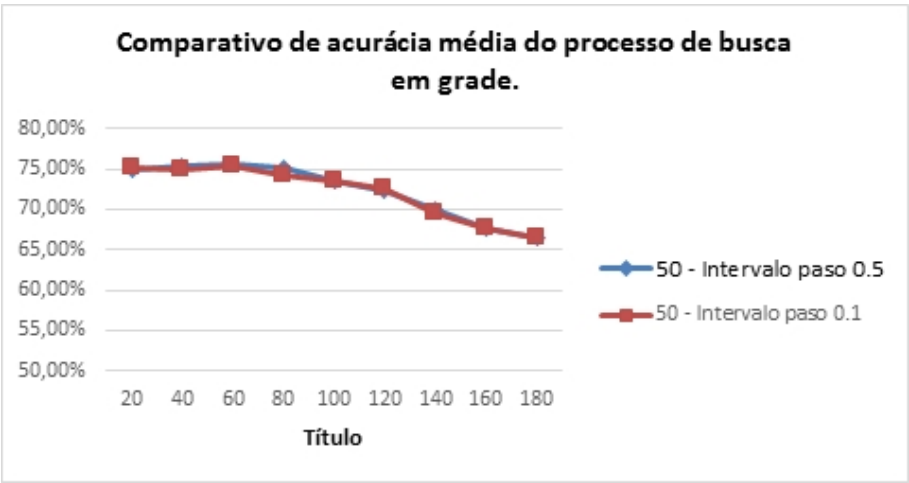

Figura 5.9: Comparativo de acurácia média do processo de busca em grade iniciando o espaço de busca em 1 e variando 0.5 até 10 para gama e *C*. O gráfico compara tempo do mesmo processo quando é utilizado um passo de 0.1. Foram usadas 50 amostras de treinamento.

Os resultados obtidos pelo método de busca em grade, o qual realiza uma busca exaustiva no espaço de variáveis, são compatíveis aos resultados obtidos utilizando os parâmetros fixos, confomre pode-se perceber na figura a seguir:

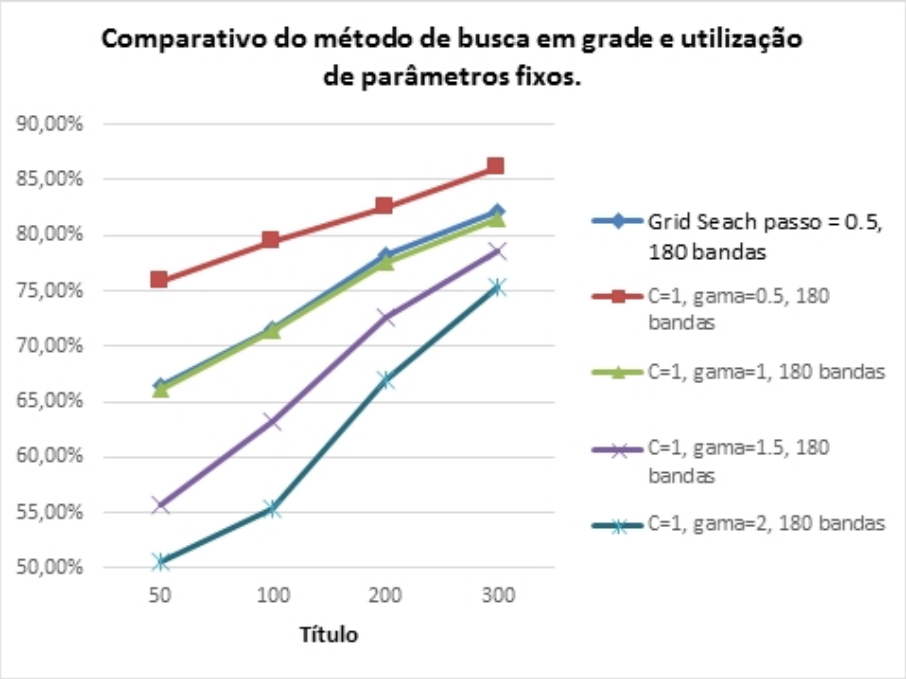

Figura 5.10: Comparativo de acurácia média do processo de busca em grande e utilização de parâmetros fixos.

5.4 EXPERIMENTO 3

No terceiro experimento, em cada nó da árvore são selecionados os melhores parâmetros, ou seja, os que produzem a maior acurácia em cada nó da árvore. O algoritmo proposto SVMBT-AMOEBASA é utilizado para otimizar os parâmetros. O espaço de variáveis é formado por uma grade de pontos onde o valor do parâmetro C varia de 1 até 10, como a busca é contínua, o intervalo entre os pontos $\in R$. O valor do parâmetro gama varia de 1 até 10, podendo assumir qualquer valor $\in R$ dentro desse intervalo. O par de parâmetros selecionado, (C, γ), é aquele que produz a maior acurácia naquele nó considerado. A Tabela 5.8 mostra os resultados obtidos:

Bandas	50 Amostras		100 Amostras		200 Amostras		300 Amostras	
20	77,52%	**00:02:59**	83,71%	**00:04:08**	85,81%	**00:07:52**	88,24%	**00:19:23**
40	80,19%	**00:03:23**	83,38%	**00:05:27**	88,10%	**00:12:30**	90,24%	**00:25:30**
60	79,33%	**00:03:09**	85,67%	**00:06:12**	88,71%	**00:14:34**	91,00%	**00:38:35**
80	80,95%	**00:04:25**	85,52%	**00:07:34**	89,14%	**00:17:14**	91,48%	**00:41:51**
100	66,67%	**00:03:45**	86,48%	**00:08:11**	89,62%	**00:26:40**	91,71%	**00:45:59**
120	80,57%	**00:04:42**	85,14%	**00:09:03**	89,14%	**00:29:08**	91,57%	**00:59:23**
140	80,52%	**00:04:38**	86,62%	**00:09:09**	89,43%	**00:34:04**	91,14%	**00:54:12**
160	79,14%	**00:04:56**	86,10%	**00:10:03**	89,33%	**00:42:24**	77,76%	**00:55:35**
180	79,90%	**00:05:11**	86,33%	**00:09:25**	89,52%	**00:43:02**	92,00%	**01:09:26**

Tabela 5.8: Experimento de recozimento simulado.

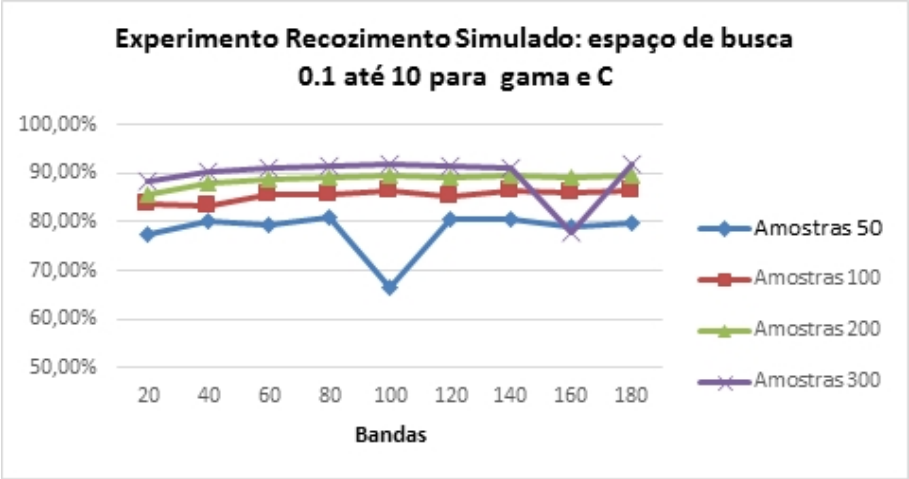

Figura 5.11: Acurácia média.

O gráfico a seguir, Figura 5.12, compara os resultados obtidos pela utilização de parâmetros fixos, o método de busca em grade e a heurística de recozimento simulado. O gráfico comparativo mostra a acurácia média obtida ao utilizar 180 bandas e 50 amostras, 100 amostras, 200 amostras e 300 amostras de treinamento na classificação de 2100 pixels através do classificador SVM, com parâmetros otimizados pelo método de recozimento simulado.

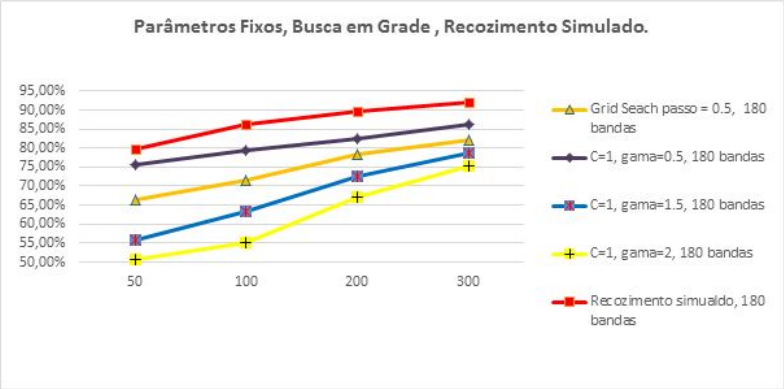

Figura 5.12: Acurácia média obtida pela utilização de parâmetros fixos, busca em grade e recozimento simulado.

O gráfico Figura 5.13, compara o tempo de processamento do método de busca em grade e recozimento simulado foram utilizadas 50 amostras, 100 amostras, 200 amostras e 300 amostras. O eixo X representa o número de bandas utilizadas nos experimentos.

Conforme é possível observar nos resultados apresentados na Tabela 5.8 e nas Figura 5.11 e Figura 5.12, a heurística de recozimento simulado obteve resultados superiores em termos de acurácia em relação à utilização dos parâmetros fixos e ao método de busca em grade. Além disso, conforme apresentado na Figura 5.13, o tempo do método de recozimento simulado foi inferior ao método de busca em grade.

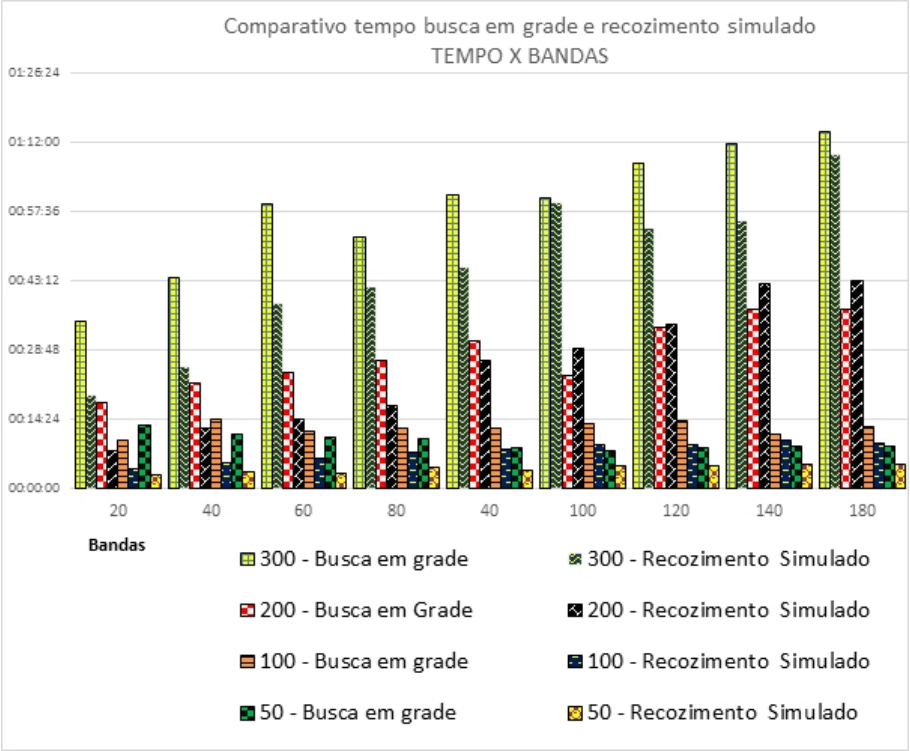

Figura 5.13: Comparativo de tempo entre o método de busca em grade com espaço de busca iniciando em 1 variando de 0.5 até 10 para γ e C e o mesmo espaço de busca utilizando algoritmo de recozimento simulado. Foram utilizadas 50, 100, 200 e 300 amostras para os métodos de busca em grade e recozimento simulado.

Quanto ao tempo de processamento, se compararmos o método de busca em grade e o algoritmo de recozimento simulado em um espaço mais amplo, vemos que o recozimento simulado apresenta tempo de processamento bastante inferior, conforme a Figura 5.14. Nesse caso, é comparado um espaço de busca com menor granularidade para o método de busca em grade com um espaço de busca mais amplo para o algoritmo de recozimento simulado.

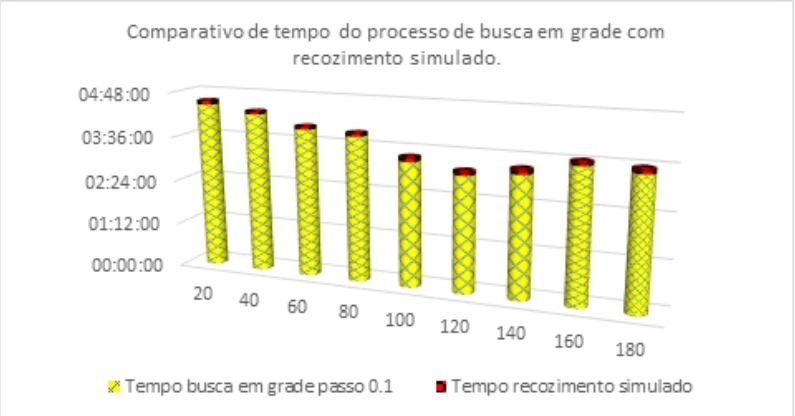

Figura 5.14: Acurácia média obtida pela utilização de parâmetros fixos, busca em grade e recozimento simulado.

Conforme é possível perceber na Tabela 5.9 a seguir, a adoção de um espaço de

busca mais amplo para o método de recozimento simulado não interferiu significativamente na acurácia obtida.

Bandas	50 Amostras		100 Amostras		200 Amostras		300 Amostras	
20	77,57%	**00:05:53**	83,19%	**00:05:24**	86,05%	**00:11:21**	88,95%	**00:26:25**
40	78,24%	**00:04:07**	85,05%	**00:06:26**	87,76%	**00:17:14**	90,05%	**00:34:20**
60	79,95%	**00:03:54**	86,29%	**00:06:33**	88,86%	**00:18:22**	91,05%	**00:41:11**
80	80,48%	**00:04:35**	83,95%	**00:07:53**	87,90%	**00:21:11**	90,19%	**00:39:09**
100	80,48%	**00:06:03**	85,38%	**00:11:25**	88,67%	**00:24:17**	90,81%	**00:57:03**
120	79,00%	**00:05:35**	71,38%	**00:09:00**	87,38%	**00:25:31**	90,90%	**01:00:37**
140	79,29%	**00:07:18**	72,86%	**00:10:55**	88,24%	**00:35:25**	89,33%	**01:07:35**
160	78,76%	**00:05:33**	83,10%	**00:14:16**	86,95%	**00:34:40**	87,57%	**01:02:55**
180	77,38%	**00:06:42**	81,05%	**00:08:28**	87,43%	**00:40:23**	88,33%	**01:05:31**

Tabela 5.9: Recozimento simulado variando de 0.1 até 100 para γ e C .

6 *CONCLUSÃO*

Nessa dissertação é proposta uma metodologia para selecionar os parâmetros do *kernel* de um classificador SVM para a classificação múltiplas classes em uma imagem hiperespectral. O classificador é construído em uma árvore binária de estágios múltiplos, em cada nó da árvore. Os parâmetros do *kernel* são otimizados através do algoritmo de recozimento simulado buscando aumentar a acurácia do classificador.

Na abordagem proposta nesta dissertação, a estrutura da árvore binária, isto é, a caracterização dos dois nós descendentes é definida a cada estágio com o auxílio da distância de *Bhattacharyya*. Esta distância estatística serve para estimar, em cada nó, o par de classes que apresenta a maior separabilidade e que darão origem aos dois nós descendentes no nível seguinte da árvore binária.

As duas classes selecionadas para formar o nó atual da árvore serão utilizadas para, no processo de treinamento, construir uma superfície de decisão, a qual seja capaz de classificar um conjunto de *pixels* com a maior acurácia possível. Nesse caso em particular, a classificação consiste em direcionar o pixel para um dos nós filhos descendentes até que o pixel atinge um nó terminal. No processo de construir a superfície de decisão, os parâmetros do *kernel*, no caso o RBF, o parâmetro gama e o parâmetro de suavização da margem C são de fundamental importância para que uma boa acurácia seja atingida.

Conforme os experimentos apresentados na seção anterior para um mesmo conjunto de dados e mesma quantidade de bandas, distintos parâmetros do *kernel* podem produzir diferenças consideráveis quanto à acurácia media do classificador (ver Tabela 5.2, 300 amostras, 180 bandas e Tabela 47, 300 amostras, 180 bandas).

Primeiro foi construído um conjunto de experimentos com parâmetros fixos em todos os nós da árvore, utilizando parâmetros de *kernel* que foram obtidos através de tentativa e erro. Esses resultados com parâmetros fixos servem como base comparativa ao método proposto.

Dois métodos foram implementados para otimizar os parâmetros do *kernel*. O primeiro, a busca em grade, que é basicamente uma busca através do espaço de pontos formados pelos parâmetros (γ, C). O segundo, uma heurística, o recozimento simulado combinado ao método *Nelder-Mead* simplex.

Para fins de teste da metodologia proposta, foi empregada uma imagem coletada pelo sistema sensor AVIRIS, o qual disponibiliza 190 bandas espectrais cobrindo as porções do visível, infravermelho próximo e infravermelho médio $(0.4\mu m \breve{} 2.4\mu m)$ no espectro eletromagnético. A cena utilizada consiste de uma área teste coberta por classes

espectralmente muito semelhantes, se constituindo assim em um desafio ao classificador. Com o intuito de investigar o comportamento da metodologia proposta, os experimentos foram realizados utilizando quatro conjuntos de dados, empregando respectivamente 50, 100, 200 e 300 amostras para treinamento. Para estimar a acurácia, os resultados produzidos pela metodologia proposta, foram utilizados 300 amostras para teste por classe em cada caso.

Os experimentos desenvolvidos confirmam a eficácia da metodologia proposta. Os experimentos mostraram que a metodologia de selecionar parâmetros do $kernel$ distintos em cada nó da árvore produzem um aumento na acurácia no caso do recozimento simulado. Por outro lado o processo de busca em grade produziu um valor aproximado à utilização dos parâmetros fixos. A metodologia proposta pode ser bastante útil em situações onde o pesquisador não conhece a natureza dos dados a serem classificados, sendo obrigado a empregar um método de tentativa e erro para configurar os parâmetros do classificador SVM.

O método de recozimento simulado se mostrou mais vantajoso, pois é capaz de percorrer um espaço de busca maior sem um grande aumento de tempo. Isso se deve ao fato de um dos critérios de parada do recozimento simulado ser o número de iterações. No caso de um espaço muito amplo, o algoritmo de recozimento simulado pode fornecer um resultado inferior em termos de acurácia, porém, de qualquer maneira, fornece uma boa estimativa inicial de onde os parâmetros que produzem uma boa acurácia se encontram. Já o algoritmo de busca em grade termina apenas quando o fim do espaço de busca é atingido, tornando o algoritmo proibitivo para espaços grandes. A Figura 6.1 mostra o gráfico de dispersão dos melhores parâmetros obtidos para os nodos da árvore binária do classificador SVM. O gráfico mostra o processo de busca realizado em um espaço variando 0.1 até 100 para gama e C. É possível perceber que os parâmetros não ficam limitados a regiões especificas. Em alguns casos o espaço terá de ser percorrido quase que completamente até que um parâmetro adequado seja atingido.

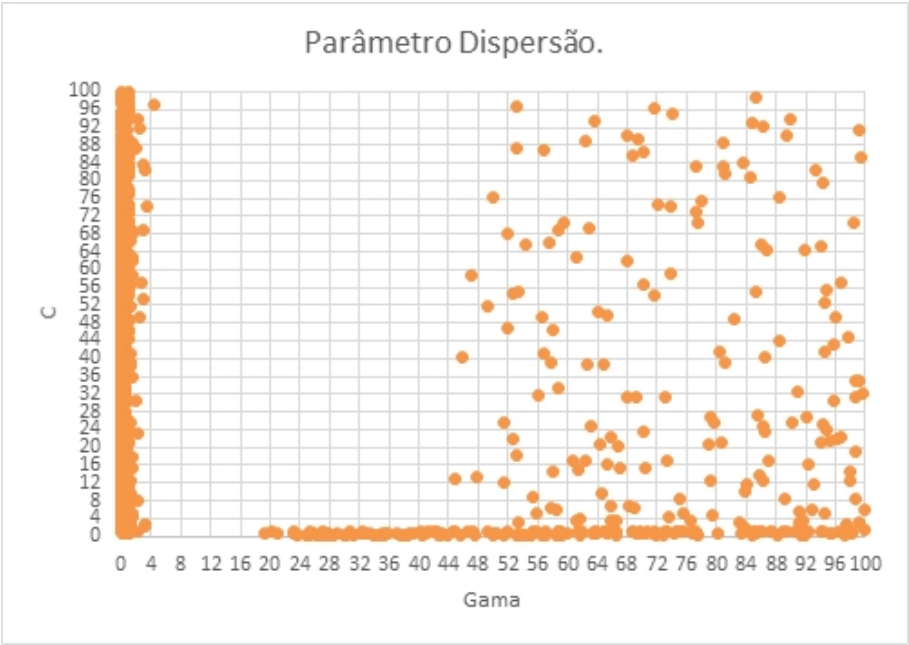

Figura 6.1: Gráfico de dispersão do processo de otimização do método de busca em grade em um variando 0.1 até 100 para gama e *C*.

Outra questão é a granularidade do algoritmo de busca em grade. A granularidade está diretamente relacionada ao tamanho do passo escolhido. Um passo muito pequeno resulta em um tempo de execução maior. Ao utilizar um passo grande, os valores do parâmetros serão truncados dentro do limite do passo. Conforme é possível verificar na Tabela 7.1, vários parâmetros ótimos obtidos possuem mais de duas casas decimais. Nesse caso, ao utilizar um passo grande, bons parâmetros de *kernel* podem ser perdidos.

O método proposto, SVMBT-AMOEBASA, através dos experimentos executados mostra sua viabilidade, obtendo uma acurácia superior à obtida utilizando parâmetros fixos. Comparando com o melhor resultado obtido pela utilização de parâmetros fixos 5.2 para 300 amostras e 180 bandas, o classificador SVM obteve a acurácia de 86,10 %, enquanto que o método proposto, Tabela 5.8 para 300 amostras e 180 bandas, obteve acurácia de 92,00 %.

Apesar dos resultados promissores, o classificador proposto tem limitações, principalmente no que diz respeito ao tempo de processamento. Aprimorar o algoritmo para reduzir o tempo de processamento pode ser abordado em trabalhos futuros. O algoritmo construído em forma de árvore repete em diversos nós os mesmos pares de classes. Nesse caso, armazenar os parâmetros para que o algoritmo não necessite otimizar pares de classe que já foram otimizados, recuperar esses parâmetros pode ser uma boa abordagem.

Outra abordagem para mitigar a questão do tempo de processamento seria utilizar para otimização de parâmetros apenas os pares de classes menos separáveis. Conforme os experimentos, ao olharmos a matriz de confusão, é possível perceber que as classes mais separáveis, conforme o critério de Bhattacharyya, são pouco sensíveis á otimização dos parâmetros do *kernel*. Seria interessante investigar se as classes mais separáveis necessitam passar pelo processo de otimização.

7 ANEXOS

7.1 TABELA

Gama	C	Gama	C	Gama	C	Gama	C	Gama	C
1	1	2,88421	83,63256	0,22812	7,00245	61,66515	0,15708	94,60257	41,56989
1	1	0,14959	61,67055	1	1	1,21235	2,87378	0,37578	6,8435
0,66637	10,1193	0,25513	15,36697	84,84996	0,15595	0,41875	13,44087	0,35871	78,49469
1	1	4,31275	97,00245	54,13816	0,11108	1	4,10163	0,31083	5,40819
0,65387	40,95502	1	45,82359	0,71224	13,8481	0,20068	17,5727	0,12218	27,32927
0,44659	21,47832	0,20215	2,75523	0,5892	62,74721	0,43809	94,76795	0,18727	3,26146
0,10926	51,8868	0,4689	78,06563	0,11173	6,37724	0,10275	50,64437	43,24099	0,69466
0,32421	10,99728	0,16309	2,71485	55,19678	0,14481	0,12174	6,79524	1	1
0,10736	56,63214	1	1	0,12006	39,64192	0,6621	35,87801	0,21726	2,52925
1	1	0,20241	0,71907	0,17093	8,25013	0,10552	95,29058	0,17352	24,65457
0,28703	24,33403	0,21894	68,07962	0,54742	49,87415	1	25,45142	0,18258	37,64301
1	1	0,20801	98,68982	0,15388	16,25917	0,18257	14,81573	1	1
0,23655	66,36545	0,84366	24,52047	1,06593	55,2931	0,15552	50,47792	0,76715	17,21892
0,20055	24,28901	1	99,90166	0,4353	15,39888	0,23104	95,41976	1	1
0,38771	69,53786	0,17945	34,69462	1,2877	40,92901	1	6,80513	0,12831	43,62559
0,65285	0,73692	0,55359	77,62819	0,56071	51,21031	0,1201	57,47997	1	82,49741
0,28895	17,10221	1	1	0,47504	33,54963	0,20126	7,94923	0,26669	78,57058
1	1	0,56692	50,09215	0,44533	16,6123	0,11367	22,98121	0,10897	37,00089
0,16863	82,17767	0,42172	6,08745	0,12273	12,91008	0,16817	34,06367	0,26955	52,92415
1	1	0,32495	66,41114	0,36826	5,54371	0,45202	46,51259	0,16858	25,43722
1	1	1	44,31498	1,34727	12,60966	0,1505	24,75637	0,45856	32,63536
0,53125	5,06847	0,255	13,36868	0,12543	10,76377	0,10757	51,71718	1	66,85283
0,41883	4,82916	0,16607	7,83928	1	46,62031	0,15318	2,42437	0,14614	49,08173
2,8893	53,31266	0,43373	26,42355	0,41407	5,42285	0,12031	45,63625	0,32781	62,86432
0,26694	36,18093	0,25485	6,24429	0,10822	25,64559	0,27217	27,16465	0,5669	19,55736
2,53892	49,29502	0,41322	92,42036	1,17196	1,78792	0,2425	12,29625	0,27186	32,57013
1	63,59412	0,18998	11,09711	0,57272	27,70837	0,63563	1,53704	0,5391	1,6185
0,10499	38,03504	0,10013	59,57061	0,19979	5,84654	0,11859	10,59805	1	1
2,62677	57,01263	1	2,53333	0,69973	91,46988	0,11759	76,5652	1	1
0,6224	63,18025	0,1364	10,39411	0,11097	49,08453	0,11075	23,48011	0,1048	14,06642
2,51614	91,88329	1	1	0,56905	22,85324	0,21183	40,10986	0,18071	69,72626
0,34028	24,8244	0,16538	57,30978	73,36487	0,93474	0,17106	1,93488	0,28603	44,43749
0,30685	5,1797	0,29194	42,21792	0,4778	5,53355	0,15024	33,50108	0,10086	49,4821
0,11249	15,42166	1,92994	30,38682	0,63941	4,85718	1	1	1	1
0,83753	44,84997	0,31257	34,49686	0,39641	39,95636	0,12171	94,50149	0,2525	55,40411
1	1	0,15888	7,14286	0,68542	74,18395	0,10256	98,52867	0,20683	15,43011
1	82,87013	0,1227	48,74831	0,45037	58,4951	94,83017	24,1201	0,32388	12,63003
2,88432	69,06457	1	1	0,17122	81,9026	91,97773	26,88909	1	3,39836
0,3656	9,95592	0,56493	2,55332	0,77369	63,9128	0,10464	62,65059	1	1

Tabela 7.1: A tabela mostra os melhores pares de parâmetros encontrados através do método de recozimento simulado. Cada par γ C foi encontrado para um par de classes.

7.2 CÓDIGO FONTE

Listing 7.1: Código SVM MATLAB

```
1 classdef SVMABE < handle
2     % This implementation is based on dissertaion SUPPORT
    VECTOR MACHINES
3     % IN THE CLASSIFICATION HYPERSPECTRAL IMAGING HAERTEL
    VITOR , PHD
4     % RAFAELA ANDREOLA function implements the svm
    classifier as in Abe.
5     % the constrained maximization problem as in Abe eq.
    (2.61) and (2.62)
```

```
6       % makes use of the "quadprog.m" function the kernel
        adopted here is the
7       % RBF kernel (ABE eq. (2.74)) with the parameter gamma
        INPUT: 1- full
8       % training and test samples for the two classes, in
        ERDAS 7.4 format:
9       %     "class_1_train", "class_2_train", "class_1_test",
        "class_2_test"
10      % 2- number of training samples to train the svm
        classifier:
11      % "n_samples_train" 3- number of testing samples to
        test the svm
12      % classifier: "n_samples_test" 4- data dimensionality
        to be used:
13      % "n_bands" 5- parameters slack ("C" in Abe) and "gamma
        " required by
14      % the RBF kernel (Abe eq. 2.74) 6- targets as in Abe's
        notation (+1,
15      % -1)
16      %
17      % NOTATION: N1: number of training samples
18      %
19      % Initial step consists in extrating the samples for
        training and for
20      % testing from the full set of available samples
21      %
22      properties
23          kernelMatrix = [];
24          dataSetNormalized = [];
25          dataSet;
26          labelDataSet;
27          biasmy;
28          alpha;
29          slack;
30          b;
31          %%x;
32          supportVector;
33          uSupportVector;
34          custFunction;
35          classSvm;
36          naccuracy;
37          accuracy;
38          dataSetTrain;
39          labelDataSetTrain;
40          dataSetTest;
41          labelDataSetTest;
42          tfolds;
43          trainDataCV;
```

```
44          labelTrainValidationCV;
45          gamma;
46          confMatrix;
47          melhor;
48          melhorC;
49          melhorGamma;
50          numberSamplesTrain;
51          numberBands;
52          kernelType;
53
54          trainDataSA     =   '';
55          trainLabelSA    =   '';
56          testDataSA      =   '';
57          testLabelSA     =   '';
58
59          Graphic
60          Conn;
61          Id;
62          sa;
63
64          number;
65      end
66
67      methods
68          function sgs = SVMABE(conn,id,kernel)
69              sgs.kernelType      =    kernel;
70              sgs.biasmy          =    0;
71              sgs.trainDataSA     =   '';
72              sgs.trainLabelSA    =   '';
73              sgs.testDataSA      =   '';
74              sgs.testLabelSA     =   '';
75              sgs.Conn            =   conn;
76              sgs.Id              =   id;
77              sgs.sa              =   SA();
78
79              sgs.number          =   0;
80              sgs.Graphic         =   0;
81          end
82
83
84          %%GridSearch Class esse m todo     chamado pelo
        treeSearch
85          function  result = gridSearch(sgs,class1,class2,
        Cost,Gama)
86
87              custoFinal        = max(Cost);
88              custoInicial      = min(Cost);
89              gamaFinal         = max(Gama);
```

```
90              gamaInicial         = min(Gama);
91
92              sgs.melhor = 100;
93              parameters= data();
94              count = 0;
95
96              sgs.trainDataSA   = [class1.getTrainData() ;
    class2.getTrainData()];
97              sgs.trainLabelSA = [ ones(size(class1.
    getTrainData(),1),1) ; -ones(size(class2.getTrainData()
    ,1),1)];
98              sgs.testDataSA    = [class1.getTestData() ;
    class2.getTestData()];
99              sgs.testLabelSA   = [ones(size(class1.
    getTestData(),1),1) ; -ones(size(class2. getTestData()
    ,1),1)];
100
101             fprintf('Inciando um novo par de classes');
102
103             for iterS0=1:size(Gama,2)
104                 count = count + 1;
105                 for iterS1=1:size(Cost,2)
106
107                     sgs.slack = Cost(iterS1);
108                     sgs.gamma = Gama(iterS0);
109                     inicio = datestr(now,'yyyy-mm-dd HH:MM:
    SS.FFF');
110                     %%sgs.classify(trainData,testData,
    trainLabel);
111                     %%sgs.validityAccuracy(trainLabel ,sgs.
    classSvm);
112                     %%sgs.confusionMatrix(testLabel ,sgs.
    classSvm);
113                     X = [ sgs.slack  sgs.gamma];
114                     [Z, model,tradeoff,acctreadeoff,
    svstreadoff] = AmoebaSAFunction(sgs,X,0);
115                     finalAlpha                          = 0;
116                     finalSupportVector                  = 0;
117                     finalUSupportVector                 = 0;
118                     finalC                              = X(1);
119                     finalGamma                          = X(2);
120                     finalAccuracy                       = Z;
121                     finalB                              = 0;
122                     sgs.accuracy                        = Z;
123
124
125
```

```
126                     fim = datestr(now,'yyyy-mm-dd HH:MM:SS.
    FFF')
127                     count = count + 1;
128
129
130                     fprintf('Acuracia : %f  \n ',sgs.
    accuracy * 100);
131                     fprintf('Gamma :  %f \n', sgs.gamma);
132                     fprintf('Cost  : %f  \n',  sgs.slack );
133
134
135                     if (sgs.accuracy < sgs.melhor)
136                         sgs.melhor =   sgs.accuracy;
137                         parameters.setData(finalGamma,
    finalC,finalAccuracy,finalAlpha,finalSupportVector,
    finalUSupportVector,finalB,sgs.trainLabelSA,sgs.
    trainDataSA,model);
138                         if (~isempty(sgs.Conn))
139                             datainsert(sgs.Conn,'parametro'
    ,{'Gamma','Slack','GammaMax','GammaMin','SlackMax','
    SlackMin','Class1','Class2','TesteID','StepMaxMin','Step
    ','Accuracy','DataStart','DataEnd'},{finalGamma,finalC,
    gamaFinal,gamaInicial,custoFinal,custoInicial,class1.
    idClass,class2.idClass,sgs.Id,0.5,count,sgs.accuracy,
    inicio,fim});
140                         end
141                     else
142                         if(~isempty(sgs.Conn))
143                             datainsert(sgs.Conn,'parametro'
    ,{'Gamma','Slack','GammaMax','GammaMin','SlackMax','
    SlackMin','Class1','Class2','TesteID','StepMaxMin','Step
    ','Accuracy','DataStart','DataEnd'},{finalGamma,finalC,
    gamaFinal,gamaInicial,custoFinal,custoInicial,class1.
    idClass,class2.idClass,sgs.Id,0.5,count,sgs.accuracy,
    inicio,fim});
144                         end
145                     end
146                     sgs.gamma           = '';
147                     sgs.slack           = '';
148                     sgs.accuracy        = '';
149                     sgs.alpha           = '';
150                     sgs.supportVector   = '';
151                     sgs.uSupportVector  = '';
152                     sgs.b               = '';
153                     sgs.kernelMatrix    = [];
154
155                 end
156             end
```

```
157             fprintf('Fim de um novo par de classes');
158             datainsert(sgs.Conn,'parametro',{'Gamma','Slack
    ','GammaMax','GammaMin','SlackMax','SlackMin','Class1','
    Class2','TesteID','StepMaxMin','Step','Accuracy','
    DataStart','DataEnd','bestParameters'},{parameters.bestG
    ,parameters.bestC,gamaFinal,gamaInicial,custoFinal,
    custoInicial,class1.idClass,class2.idClass,sgs.Id,0.5,
    count,parameters.bestAcc,inicio,fim,'true'});
159             result = parameters;
160             return;
161         end
162
163         %%Roda o codigo
164         function result = fixParameter(sgs,class1,class2,
    gammaFix,slackFix)
165             %Executa busca com parametros fixos
166             % Class1 - recebe a classe 1
167             % Class2 - recebe a classe 2
168             % parametro gama
169             % parametro fixo
170
171             parameters= data();
172
173             sgs.trainDataSA  = [class1.getTrainData() ;
    class2.getTrainData()];
174             sgs.trainLabelSA = [ ones(size(class1.
    getTrainData(),1),1) ; -ones(size(class2.getTrainData()
    ,1),1)];
175             sgs.testDataSA   = [class1.getTestData() ;
    class2.getTestData()];
176             sgs.testLabelSA  = [ones(size(class1.
    getTestData(),1),1) ; -ones(size(class2. getTestData()
    ,1),1)];
177
178             sgs.slack                   = slackFix;
179             sgs.gamma                   = gammaFix;
180
181             inicio = datestr(now,'yyyy-mm-dd HH:MM:SS.FFF')
    ;
182 %             sgs.classify(trainData,testData,trainLabel);
183 %             sgs.validityAccuracy(trainLabel ,sgs.classSvm
    );
184 %             sgs.confusionMatrix(testLabel ,sgs.classSvm);
185             X = [ sgs.slack  sgs.gamma];
186             [Z, model,tradeoff,acctreadeoff,svstreadoff] =
    AmoebaSAFunction(sgs,X,0);
187             finalAlpha                  = 0;
188             finalSupportVector          = 0;
```

```
189             finalUSupportVector                 = 0;
190             finalC                              = X(1);
191             finalGamma                          = X(2);
192             finalAccuracy                       = Z;
193             finalB                              = 0;
194             sgs.accuracy                        = Z;
195
196             fim = datestr(now,'yyyy-mm-dd HH:MM:SS.FFF');
197
198
199             fprintf('Acuracia : %f  \n ',sgs.accuracy *
    100);
200             fprintf('Gamma  :  %f \n', sgs.gamma);
201             fprintf('Cost   :  %f  \n',  sgs.slack );
202             %%fprintf('Alphas      : %f  \n',  svcount );
203             if (~isempty(sgs.Conn))
204                 datainsert(sgs.Conn,'parametro',{'Gamma','
    Slack','GammaMax','GammaMin','SlackMax','SlackMin','
    Class1','Class2','TesteID','StepMaxMin','Step','Accuracy
    ','DataStart','DataEnd'},{finalGamma,finalC,0,0,0,0,
    class1.idClass,class2.idClass,sgs.Id,0,0,sgs.accuracy,
    inicio,fim});
205             end
206             parameters.setData(finalGamma,finalC,
    finalAccuracy,finalAlpha,finalSupportVector,
    finalUSupportVector,finalB,sgs.trainLabelSA,sgs.
    trainDataSA,model);
207
208             sgs.gamma           = 0;
209             sgs.slack           = 0;
210             sgs.accuracy        = 0;
211             sgs.alpha           = 0;
212             sgs.supportVector   = 0;
213             sgs.uSupportVector  = 0;
214             sgs.b               = 0;
215             sgs.kernelMatrix    = [];
216
217
218             result = parameters;
219             return;
220
221         end
222
223
224
225         %%Simulated Annealing
226         function  result = SA(sgs,class1,class2,cost,gama,
    tfinal,tinicial)
```

```
227
228             sgs.number =  sgs.number + 1;
229             parameters= data();
230             sgs.trainDataSA   = [class1.getTrainData() ;
    class2.getTrainData()];
231             sgs.trainLabelSA = [ ones(size(class1.
    getTrainData(),1),1) ; -ones(size(class2.getTrainData()
    ,1),1)];
232             sgs.testDataSA    = [class1.getTestData() ;
    class2.getTestData()];
233             sgs.testLabelSA   = [ones(size(class1.
    getTestData(),1),1) ; -ones(size(class2. getTestData()
    ,1),1)];
234             if (~isempty(sgs.Conn))
235                 saveData =1;
236             else
237                 saveData =0;
238             end
239             tradeoff = 1;
240             sgs.sa = AmoebaSA(sgs,class1,class2,saveData,
    sgs.Id,tradeoff);
241             %%Mostra no formato n o cientifico
242             %%X = SIMPSA('MyFunction',x0,[-10 -10],[10 10])
    ;
243             %%runNedel(sgs,funcOpt,X0,lowerBound,
    upperBound,functionObjective)
244
245             X0 = [1 1];
246             lowerBound = [0.1 0.1]
247             upperBound = [100 100]
248             inicio = datestr(now,'yyyy-mm-dd HH:MM:SS.FFF')
    ;
249             [X] = sgs.sa.runNedel('AmoebaSAFunction',X0,
    lowerBound, upperBound)
250
251             %%Valida os parametros e devolve o modelo
252             [Z, model,tradeoff,acctreadeoff,svstreadoff] =
    AmoebaSAFunction(sgs,X,tradeoff);
253
254             fim = datestr(now,'yyyy-mm-dd HH:MM:SS.FFF');
255
256             %%sgs.slack =  X(1);
257             %%sgs.gamma =  X(2);
258             %%sgs.classify( sgs.trainDataSA, sgs.testDataSA
    , sgs.trainLabelSA);
259             %%sgs.validityAccuracy( sgs.trainLabelSA , sgs.
    classSvm);
```

```
260             %%sgs.confusionMatrix ( sgs.testLabelSA    , sgs
    .classSvm);
261             finalAlpha                          = 0;
262             finalSupportVector                  = 0;
263             finalUSupportVector                 = 0;
264             finalC                              = X(1);
265             finalGamma                          = X(2);
266
267
268             %count          = sgs.supportVector;
269             %sv             = length(count);
270             %fator          = sqrt((sv / length(sgs.
    trainLabelSA)));
271             %finalAccuracy =  1 - ( (sgs.accuracy/2) + (
    fator/2) );
272             finalAccuracy    = Z;
273             finalB          =    0;
274
275 %             display('Acuracia Final Simples');
276 %             display(1 - sgs.accuracy);
277 %
278 %
279 %              display('Acuracia Z');
280 %              display(Z);
281 %             if ( ((1-sgs.accuracy) - Z)  ~=0 )
282 %              pause;
283 %             end
284
285             if (~isempty(sgs.Conn))
286                 if(tradeoff == 1)
287                     datainsert(sgs.Conn,'parametro',{'Gamma',
    'Slack','GammaMax','GammaMin','SlackMax','SlackMin','
    Class1','Class2','TesteID','StepMaxMin','Step','Accuracy
    ','DataStart','DataEnd','sus','accsus','bestParameters'
    },{finalGamma,finalC,lowerBound(1), upperBound(1),
    lowerBound(2), upperBound(2),class1.idClass,class2.
    idClass,sgs.Id,0,0,finalAccuracy,inicio,fim,svstreadoff,
    acctreadeoff,'true'});
288                 else
289                     datainsert(sgs.Conn,'parametro',{'Gamma',
    'Slack','GammaMax','GammaMin','SlackMax','SlackMin','
    Class1','Class2','TesteID','StepMaxMin','Step','Accuracy
    ','DataStart','DataEnd', 'bestParameters'},{finalGamma,
    finalC,lowerBound(1), upperBound(1),lowerBound(2),
    upperBound(2),class1.idClass,class2.idClass,sgs.Id,0,0,
    finalAccuracy,inicio,fim,'true'});
290                 end
291             end
```

```
292             parameters.setData(finalGamma,finalC,
    finalAccuracy,finalAlpha,finalSupportVector,
    finalUSupportVector,finalB,sgs.trainLabelSA,sgs.
    trainDataSA,model);
293             result = parameters;
294
295
296         end
297
298
299
300
301
302         function classlist = classifyNode(sgs,class1,
    biasClassify,alphaClassify,supportVectorClassify,
    gammaClassify,cClassify,trainLabel,trainData,model)
303             %%disp('**********nomde da classe');
304             %%disp(class1.getClassName()); disp
    ('**********nomde
305             %%da classe');
306             trainDataClass  = [class1.getTrainData()];
307             classlist       = [];
308             classifySvm=[];
309
310             [predictLabel,acc,decvalues] = svmpredict(zeros
    (size(trainDataClass,1),1),trainDataClass,model);
311
312             for sample = 1:size(trainDataClass,1)
313                 classifySvm(sample,1)=predictLabel(sample);
314             end
315
316 %               for sample = 1:size(trainDataClass,1)
317 %                   dx=0;
318 %                   for i =1:size(supportVectorClassify,1)
319 %                       if(strcmp(sgs.kernelType,'poly'))
320 %                           dx = dx + alphaClassify(
    supportVectorClassify(i)) * trainLabel(
    supportVectorClassify(i))* ( trainDataClass(sample,:) *
    trainData(supportVectorClassify(i),:)' + 1) ^
    gammaClassify;
321 %                       elseif(strcmp(sgs.kernelType,'rbf'))
322 %                           dif = trainDataClass(sample,:) -
    trainData(supportVectorClassify(i),:);
323 %                           dx = dx + alphaClassify(
    supportVectorClassify(i)) * trainLabel(
    supportVectorClassify(i))* double(exp(-1/2 * ((dif*dif')
     / (gammaClassify*gammaClassify)) ));
324 %
```

```
325 %                       elseif(strcmp(sgs.kernelType,'sam'))
326 %                           multi =  trainDataClass(sample,:)
     * trainData(supportVectorClassify (i),:)';
327 %                       dx = dx + alphaClassify(
    supportVectorClassify (i)) * trainLabel(
    supportVectorClassify(i)) * acos(multi/(multi*multi));
328 %                       end
329 %                   end
330 %                   dx = dx + biasClassify;
331 %                   if  dx>0
332 %                       classifySvm(sample,1)=1;
333 %                   else
334 %                       classifySvm(sample,1)=-1;
335 %                   end
336 %               end
337             classlist = classifySvm;
338
339         end
340
341
342         function classifyPixel= classifyPixel(sgs,test,
    biasClassify,alphaClassify,supportVectorClassify,
    gammaClassify,cClassify,trainLabel,trainData)
343             disp('no deveria passar aqui');
344             pause;
345             for sample = 1:size(test,1)
346                 dx=0;
347                 for i =1:size(supportVectorClassify,1)
348                     dif = test(sample,:) - trainData(
    supportVectorClassify(i),:);
349                     dx = dx + alphaClassify(
    supportVectorClassify(i)) * trainLabel(
    supportVectorClassify(i))*  double(exp(-1/2 * ((dif*dif
    ') / (gammaClassify*gammaClassify)))) ;
350                 end
351                 dx = dx + biasClassify;
352                 if  dx>0
353                     classifyPixel=1;
354                 else
355                     classifyPixel=-1;
356                 end
357             end
358
359
360
361             return
362
363
```

```
364         end
365
366
367
368         function classify(sgs,data,test,label)
369             disp('no deveria passar aqui');
370             pause;
371             sgs.generateKernelMatrix(data);
372             sgs.quadraProg(data,label);
373             %%sgs.calculateBias(data);
374             %%%%%%%%%%%%%%%%%%%%Calculate Bias
375             bias=0;
376             for k = 1:size(sgs.uSupportVector,1)
377                 sum_i=0;
378                 for l=1:size(sgs.supportVector,1)
379                     %%display (sgs.alpha(sgs.supportVector(
    l))*label(sgs.supportVector(l))*sgs.kernelMatrix (sgs.
    supportVector(l),sgs.uSupportVector(k)));
380                     sum_i = sum_i + sgs.alpha(sgs.
    supportVector(l))*label(sgs.supportVector(l))*sgs.
    kernelMatrix (sgs.supportVector(l),sgs.uSupportVector(k)
    );
381                 end
382                 bias = label(sgs.uSupportVector(k))-sum_i;
383             end
384             %%disp('Aqui vai o B'); disp(bias);
385             bias=bias/size(sgs.uSupportVector,1);
386             sgs.b= bias;
387             sgs.generateHyperPlane(data,test,label);
388         end
389
390         function validityAccuracy(sgs,label,classifiedData)
391             correct   = 0;
392             incorrect = 0;
393             for j=1:size(label,1)
394                 % first column (ground truth for class # 1)
395                 if ((label(j)==1 & classifiedData(j)==1) ||
     (label(j)==-1 & classifiedData(j)==-1))
396                     correct=correct+1;
397                 else
398                     incorrect=incorrect+1;
399                 end
400             end
401             sgs.accuracy  = (correct / size(label,1));
402             sgs.naccuracy = (incorrect / size(label,1));
403         end
404
405         function confusionMatrix(sgs,class,target)
```

```
406             confmatrix =zeros(2,2);
407             %
408             % calculate the confusion matrix, columnwise
    ground truth along
409             % the columns thematic image along the rows
410             %
411             for sample=1:size(target,1)
412                 % first column (ground truth for class # 1)
413                 if target(sample)==1 & class(sample)==1
414                     confmatrix(1,1)=confmatrix(1,1)+1;
415                 end
416                 if target(sample)==1 & class(sample)==-1
417                     confmatrix(2,1)=confmatrix(2,1)+1;
418                 end
419                 % second column (ground truth for class #
    2)
420                 if target(sample)==-1 & class(sample)==1
421                     confmatrix(1,2)=confmatrix(1,2)+1;
422                 end
423                 if target(sample)==-1 & class(sample)==-1
424                     confmatrix(2,2)=confmatrix(2,2)+1;
425                 end
426             end
427             sgs.confMatrix=confmatrix;
428         end
429
430         function generateKernelMatrix(sgs,data)
431             disp('no deveria passar aqui');
432             pause;
433
434             sizeData = size(data,1);
435             for i = 1:sizeData
436                 for j = 1:sizeData
437                     %%TO-DO Trocar isso pela fun  o de
    kernel selecinada
438                     %%anteriormente no m todo
    generateCrossValidation
439                     if(strcmp(sgs.kernelType,'poly'))
440                         sgs.kernelMatrix(i,j)=(data(i,:)*
    data(j,:)' + 1)^ sgs.gamma;
441                     elseif(strcmp(sgs.kernelType,'rbf'))
442                         dif = data(i,:)-data(j,:);
443                         %%sgs.kernelMatrix(i,j)= double(exp
    (-sgs.gamma*(dif*dif')));
444                         sgs.kernelMatrix(i,j)= double(exp
    (-1/2 * ((dif*dif') / (sgs.gamma*sgs.gamma)) ));
445                     elseif(strcmp(sgs.kernelType,'sam'))
446                         multi =  data(i,:)*data(j,:)';
```

```
447                             sgs.kernelMatrix(i,j) = double(acos
    (multi/(multi*multi)));
448                         end
449                     end
450                 end
451             end
452
453
454
455
456
457
458
459
460
461 %           function quadraProg(sgs,data,label)
462 %               sizeData = size(data,1);
463 %               options = optimset('Algorithm','interior-
    point-convex','Display','off');
464 %               %%disp(sgs.slack);
465 %               %%disp(sgs.gamma);
466 %
467 %
468 %               [alpha,fval,exitflag,output,lambda] =
    quadprog( diag(label)* sgs.kernelMatrix*diag(label), -
    ones(1,sizeData),zeros(1,sizeData), 1, label', 0, zeros(
    sizeData,1),sgs.slack*ones(sizeData,1),[],options);
469 %               %%sgs.alpha = quadprog( diag(label)* sgs.
    kernelMatrix*diag(label), -ones(1,sizeData),zeros(1,
    sizeData), 1, label', 0, zeros(sizeData,1),sgs.slack*
    ones(sizeData,1));
470 %
471 %               sgs.alpha = alpha;
472 %               sgs.custFunction = fval;
473 %               sgs.supportVector = find (sgs.alpha>0);
474 %               sgs.uSupportVector = find ((sgs.alpha>0) & (
    sgs.alpha < sgs.slack));
475 %
476 %           end
477 %
478
479
480
481
482         function quadraProg(sgs,data,label)
483             disp('no␣deveria␣passar␣aqui');
484             pause;
485
```

```
486             sizeData = size(data,1);
487
488             mu  = 1e-12;
489             eps = 1e-12;
490
491             H = diag(label)* sgs.kernelMatrix*diag(label);
492             H = H + mu*eye(size(H));
493             f =  -ones(1,sizeData);
494
495             Aeq = label';
496             beq = 0;
497
498             LB = zeros(sizeData,1);
499             x0 = zeros(sizeData,1);
500
501
502             UB = sgs.slack*ones(sizeData,1);
503
504             qp_options =optimset('Algorithm','interior-
    point-convex','Display','off','LargeScale','off','
    MaxIter',2000);
505             [Alpha,fval,exitflag] = quadprog(H, f, [],[],
    Aeq, beq, LB, UB, x0, qp_options);
506
507
508             sgs.alpha               =  Alpha;
509             sgs.custFunction        = fval;
510             sgs.supportVector       = find (Alpha>eps);
511             sgs.uSupportVector      = find ((Alpha > eps) & (
    Alpha < (sgs.slack -eps)));
512
513         end
514
515
516
517         function generateHyperPlane(sgs,data,test,label)
518             disp('no␣deveria␣passar␣aqui');
519             pause;
520             sgs.classSvm=[];
521             for sample = 1:size(data,1)
522                 dx=0;
523                 for i =1:size(sgs.supportVector ,1)
524                     if(strcmp(sgs.kernelType,'poly'))
525                         dx = dx + sgs.alpha(sgs.
    supportVector (i)) * label(sgs.supportVector(i))* ( test
    (sample,:) * data(sgs.supportVector (i),:)' + 1) ^ sgs.
    gamma;
526                     elseif(strcmp(sgs.kernelType,'rbf'))
```

```
527                         dif = test(sample,:) - data(sgs.
    supportVector(i),:);
528                         dx = dx + sgs.alpha(sgs.
    supportVector(i)) * label(sgs.supportVector(i))* double(
    exp(-1/2 * ((dif*dif') / (sgs.gamma*sgs.gamma)) ));
529                     elseif(strcmp(sgs.kernelType,'sam'))
530                         %                               multi =
    test(sample,:) * data(sgs.supportVector(i),:)';
531                         %                               dx = dx +
     sgs.alpha(sgs.supportVector(i)) * label(sgs.
    supportVector(i))* acos(multi/ulti*mutli));
532                     end
533                 end
534                 dx = dx + sgs.b;
535                 if  dx>0
536                     sgs.classSvm(sample,1)=1;
537                 else
538                     sgs.classSvm(sample,1)=-1;
539                 end
540             end
541
542         end
543
544         function normalizeDataSet(sgs,data)
545             maxData = max(max(data));
546             minData = min(min(data));
547             [M N] = size(data);
548             for i = 1:M
549                 for j = 1:N
550                     sgs.dataSetNormalized(i,j)=(( data(i,j)
     - minData ) /  ( maxData - minData ));
551                 end
552             end
553
554         end
555
556
557         function calculateBias(sgs,label)
558             disp('no deveria passar aqui');
559             pause;
560             sgs.b=0;
561             for j = 1:size(sgs.uSupportVector,1)
562                 sum_i=0;
563                 for i=1:size(sgs.supportVector,1)
564                     sum_i = sum_i + sgs.alpha(sgs.
    supportVector(i))*label(sgs.supportVector(i))*sgs.
    kernelMatrix(sgs.supportVector(i),sgs.uSupportVector(1))
    ;
```

```
565                     end
566                     sgs.b = label(sgs.uSupportVector(j))-sum_i;
567                 end
568
569                 sgs.b=sgs.b/size(sgs.uSupportVector,1);
570             end
571         end
572 end
```

Listing 7.2: Código Recozimento Simulado Amoeba MATLAB

```
 1 classdef AmoebaSA < handle
 2
 3     properties
 4         Class1          = '';
 5         Class2          = '';
 6         Class1ID        = '';
 7         Class2ID        = '';
 8
 9         numberOfEvaluations = 0;
10         FuncOpt = '';
11
12         %%pontos de acuracia
13
14         costEvaluated_ith;
15         bestCostEvaluated_ith;
16         temperature_ith;
17         z_ith;
18         bestZ_ith;
19
20
21
22
23
24
25         %%%Optimazion Variables
26         TEMP_START                  ='';              % starting temperature (if none provided, an optimal one will be estimated)
27         TEMP_END                    =0;               % end temperature
28         COOL_RATE                   =0;               % small values (<1) means slow convergence, large values (>1) means fast convergence
29         INITIAL_ACCEPTANCE_RATIO    =0;               % when initial temperature is estimated, this will be the initial acceptance ratio in the first round
30         MIN_COOLING_FACTOR          =0;               % minimum cooling factor (<1)
```

```
31          MAX_ITER_TEMP_FIRST         =0;                 % number
    of iterations in the preliminary temperature loop
32          MAX_ITER_TEMP_LAST          =0;                 % number
    of iterations in the last temperature loop (pure simplex
    )
33          MAX_ITER_TEMP               =0;                 % number
    of iterations in the remaining temperature loops
34          MAX_ITER_TOTAL              =0;                 % maximum
    number of iterations tout court
35          MAX_TIME                    =0;                 % maximum
    duration of optimization
36          MAX_FUN_EVALS               =0;                 % maximum
    number of function evaluations
37          TOLX                        =0;                 % maximum
    difference between best and worst function evaluation in
     simplex
38          TOLFUN                      =0;                 % maximum
    difference between the coordinates of the vertices
39          TEMP_LOOP_NUMBER            =0;                 %
    temperature loop
40
41
42          bestZ                       =0;
43          bestX                       =0;
44          lowerBound                  =0;
45          upperBound                  =0;
46          saveData                    =0;
47          FtesteID                    =0;
48          tradeoff
49      end
50
51      methods
52
53          function sgs = AmoebaSA(funcOpt,class1,class2,
    savedata,testeID,tradeoff)
54
55              sgs.Class1          = class1.getClassName();
56              sgs.Class2          = class2.getClassName();
57
58              sgs.Class1ID        =  class1.idClass;
59              sgs.Class2ID        =  class2.idClass;
60              sgs.FtesteID        =  testeID;
61
62              sgs.saveData = savedata;
63              display(sprintf('%s  %s',sgs.Class1, sgs.Class2
    ));
64              display(sprintf('%s  %s',sgs.Class1, sgs.Class2
    ));
```

```
65              %%allow acess to SVMABE methods
66              sgs.FuncOpt          = funcOpt;
67
68
69              sgs.TEMP_START                  ='';                %
     starting temperature (if none provided, an optimal one
     will be estimated)
70              sgs.TEMP_END                    =1;                 %
     end temperature
71              sgs.COOL_RATE                   =10;                %
     small values (<1) means slow convergence,large values
     (>1) means fast convergence
72              sgs.INITIAL_ACCEPTANCE_RATIO =0.95;
     % when initial temperature is estimated, this will be
     the initial acceptance ratio in the first round
73              sgs.MIN_COOLING_FACTOR          =0.9;
     % minimum cooling factor (<1)
74              sgs.MAX_ITER_TEMP_FIRST         =50;                %
     number of iterations in the preliminary temperature
     loop
75              sgs.MAX_ITER_TEMP_LAST          =50;                %
     number of iterations in the last temperature loop (pure
     simplex)
76              sgs.MAX_ITER_TEMP               =10;                %
     number of iterations in the remaining temperature loops
77              sgs.MAX_ITER_TOTAL              =2500;
     % maximum number of iterations tout court
78              sgs.MAX_TIME                    =2500;
     % maximum duration of optimization
79              sgs.MAX_FUN_EVALS               =2500;
     % maximum number of function evaluations
80              sgs.TOLX                        =1e-6;
     % maximum difference between best and worst function
     evaluation in simplex
81              sgs.TOLFUN                      =1e-3;
     % maximum difference between the coordinates of the
     vertices
82              sgs.TEMP_LOOP_NUMBER            =0;
     %temperature loop
83              sgs.tradeoff                    =    tradeoff;
84          end
85
86
87          function   [X] = runNedel(sgs,func,X0,lowerbound,
     upperbound)
88
89              ndimension   =     length(X0);
90
```

```
91
92
93
94
95                SUCCESSFUL_MOVES    = 0;
96                UNSUCCESSFUL_MOVES  = 0;
97                UNSUCCESSFUL_COSTS  = 0;
98
99              %%incialize array to save
100             sgs.costEvaluated_ith        = zeros(sgs.
    MAX_ITER_TOTAL,ndimension+1);
101             sgs.bestCostEvaluated_ith    = zeros(sgs.
    MAX_ITER_TOTAL,1);
102             sgs.temperature_ith          = zeros(sgs.
    MAX_ITER_TOTAL,1);
103             sgs.z_ith                    = zeros(ndimension
    +1,ndimension,sgs.MAX_ITER_TOTAL);
104             sgs.bestZ_ith                = zeros(sgs.
    MAX_ITER_TOTAL,ndimension);
105
106             %%set lower bound
107             sgs.lowerBound  = lowerbound;
108
109             %%set upper bound
110             sgs.upperBound  = upperbound;
111
112
113             %%MATLAB\Functions\Mathematics\Arrays and
    Matrices\Elementary Matrices and Arrays\rng
114             %%save the current generator settings in s:
115             s = rng;
116
117             %%empty simplex matrix of candidate points
118             candidatePoints  = zeros(ndimension + 1 ,
    ndimension);
119
120             %%empty cost vector
121             Z                = zeros(ndimension + 1 , 1);
122
123             %%set best vertex of inial simplex equal to
    guess parameter
124             sgs.bestX   = X0;
125             %% evaluate cost function with simplex guess
    parameter
126             %% call evaluation function objectiveFunction(
    sgs,fun,point,lb,up)
127             %% with parameters
128             %%objective function
```

```
129             %% bestX in this case guess point
130             %% lowerBound
131             %% upperBound
132             sgs.bestZ    = sgs.objectiveFunction(func,sgs.
    bestX,sgs.lowerBound,sgs.upperBound);
133
134             sgs.TEMP_LOOP_NUMBER = 1;
135
136             %%set temperature
137             TEMPERATURE = abs(sgs.bestZ)*1e5;
138
139
140
141             sgs.bestX = X0;
142
143
144             %%start iterations
145             %%control stopping
146             EXITFLAG = 0;
147             %count iteration number
148             numberIteration = 0;
149             %start timer
150             tic
151
152             while EXITFLAG == 0
153
154               candidatePoints(1,:) = sgs.bestX;
155               Z(1)                 = sgs.objectiveFunction(
    func,sgs.bestX,sgs.lowerBound,sgs.upperBound);
156
157               if sgs.TEMP_LOOP_NUMBER == 1
158                  MAXITERTEMP = sgs.MAX_ITER_TEMP_FIRST *
    ndimension;
159               end
160
161               %%test remaining vertices of simplex
162               for k = 1:ndimension
163                       % copy first vertex in new vertex
164                        candidatePoints(k+1,:) =
    candidatePoints(1,:);
165                       % alter new vertex
166                        candidatePoints(k+1,k) = lowerbound(k
    )+rand*(upperbound(k)-lowerbound(k));
167                       % calculate the value of obejtive
    function at new vertex
168                       Z(k+1)  = sgs.objectiveFunction(func,
    candidatePoints(k+1,:),lowerbound,upperbound);
169               end
```

```
170
171
172             % initialize vector COSTS, needed to
    calculate new temperature using cooling
173             % schedule as described by Cardoso et al.
    (1996)
174             costs = zeros((ndimension+1)*MAXITERTEMP,1);
175
176
177                     iterationTemperature = 0;
178                     % Press and Teukolsky (1991) add a
    positive logarithmic distributed variable,
179                     % proportional to the control
    temperature T to the function value associated with
180                     % every vertex of the simplex.
    Likewise,they subtract a similar random variable
181                     % from the function value at every new
     replacement point.
182                     % Thus, if the replacement point
    corresponds to a lower cost, this method always
183                     % accepts a true down hill step. If,
    on the other hand, the replacement point
184                     % corresponds to a higher cost, an
    uphill move may be accepted, depending on the
185                     % relative COSTS of the perturbed
    values.
186                     % (taken from Cardoso et al.,1996)
187                     % this for corresponde to NelderMead
    method
188                     inicio = datestr(now,'yyyy-mm-dd␣HH:MM
    :SS.FFF');
189                     for iterationTemperature = 1:
    MAXITERTEMP
190                             numberIteration =
    numberIteration + 1;
191                             % add random fluctuations to
    function values of current vertices
192                             fluctuationZ = Z + TEMPERATURE
    * abs(log(rand(ndimension+1,1)));
193
194                             %reorder fluctuationZ, Z and P
    so the first row corresponds to the lowest vertice value
195                             tempMatrix =  sortrows([
    fluctuationZ,Z,candidatePoints],1);
196                             fluctuationZ = tempMatrix(:,1);
197                             Z = tempMatrix(:,2);
198                             candidatePoints = tempMatrix
    (:,3:end);
```

```
199                             sgs.temperature_ith(
    iterationTemperature) = TEMPERATURE;
200
201
202                             % store information about
    simplex at the current iteration
203                             sgs.z_ith(:,:,numberIteration)
                        = candidatePoints;
204                             sgs.bestZ_ith(numberIteration
    ,:)                   = sgs.bestX;
205
206                             % store cost function value of
    best vertex in current iteration
207                             sgs.costEvaluated_ith (
    numberIteration,:)          = Z;
208                             sgs.bestCostEvaluated_ith(
    numberIteration)        =   sgs.bestZ;
209
210
211                             % end the optimization if one
    of the stopping criteria is met
212                             %% 1. difference between best
    and worst function evaluation in simplex is smaller than
     TOLFUN
213                             %% 2. maximum difference
    between the coordinates of the vertices in simplex is
    less than TOLX
214                             %% 3. no convergence, but
    maximum number of iterations has been reached
215                             %% 4. no convergence, but
    maximum time has been reached
216
217                             if (abs(max(Z)-min(Z)) < sgs.
    TOLFUN) && (sgs.TEMP_LOOP_NUMBER ~= 1),
218                               disp('Change in the objective
    function value less than the specified tolerance (
    TOLFUN).');
219                               EXITFLAG = 1;
220                               break;
221                             end
222
223                             if (max(max(abs(candidatePoints
    (2:ndimension+1,:)-candidatePoints(1:ndimension,:)))) <
    sgs.TOLX) && (sgs.TEMP_LOOP_NUMBER ~= 1),
224                               disp('Change in X less than
    the specified tolerance (TOLX).');
225                               EXITFLAG = 1;
226                               break;
```

```
227                             end
228
229                             if (numberIteration >= sgs.
    MAX_ITER_TOTAL*ndimension) || (sgs.numberOfEvaluations
    >= sgs.MAX_FUN_EVALS*ndimension*(ndimension+1)),
230                                 disp('Maximum number of
    function evaluations or iterations reached.');
231                                 EXITFLAG = 1;
232                                 break;
233                             end
234
235                             if toc/60 > sgs.MAX_TIME
236                                 disp('Exceeded maximum time.'
    );
237                                 EXITFLAG = 1;
238                                 break;
239                             end
240                             %%disp(sprintf('%5.0f      %5.0
    f      %12.6g      %15.6g      %5.0f      %5.0f   %12.6g
        ',numberIteration,sgs.numberOfEvaluations,Z(1),sgs.
    bestZ,candidatePoints(1),candidatePoints(2),TEMPERATURE)
    );
241
242                     % begin a new iteration
243                     %% first extrapolate by a factor -1
    through the face of the simplex
244                     %% across from the high point, i.e.,
    reflect the simplex from the high point
245                 [zFlutuationTrial,zTrial,pointTrial] =
     Amoeba(sgs,func,candidatePoints,-1,sgs.lowerBound,sgs.
    upperBound,ndimension,TEMPERATURE);
246
247                     %% check the result
248                             if zFlutuationTrial <=
    fluctuationZ(1),
249                                 %% gives a result better
    than the best point, so try an additional
250                                 %% extrapolation by a
    factor 2
251                                 [zFlutuationTrialExtra,
    zTrialExtra,pointTrialExtra] = Amoeba(sgs,func,
    candidatePoints,-2,sgs.lowerBound,sgs.upperBound,
    ndimension,TEMPERATURE);
252                                 if zFlutuationTrialExtra <
    zFlutuationTrial,
253                                     candidatePoints(end,:)
    = pointTrialExtra;
254                                     Z(end)    = zTrialExtra;
```

```
255                                 else
256                                     candidatePoints(end,:)
    = pointTrial;
257                                     Z(end)    = zTrial;
258
259                                 end
260                             elseif zFlutuationTrial >=
    fluctuationZ(ndimension),
261                                 %% the reflected point is
    worse than the second-highest, so look
262                                 %% for an intermediate
    lower point, i.e., do a one-dimensional
263                                 %% contraction
264                                 [zFlutuationTrialContra,
    zTrialContra,pointTrialContra] = Amoeba(sgs,func,
    candidatePoints,-2,sgs.lowerBound,sgs.upperBound,
    ndimension,TEMPERATURE);
265                                 if zFlutuationTrialContra <
     fluctuationZ(end),
266                                     candidatePoints(end,:)
    = pointTrialContra;
267                                     Z(end)    = zTrialContra
    ;
268                                 else
269                                     %% can't seem to get
    rid of that high point, so better contract
270                                     %% around the lowest (
    best) point
271                                     X = ones(ndimension,
    ndimension)*diag(candidatePoints(1,:));
272                                     candidatePoints(2:end
    ,:) = 0.5*(candidatePoints(2:end,:)+X);
273                                     for k=2:ndimension,
274                                         Z(k) = sgs.
    objectiveFunction(func,candidatePoints(k,:),lowerbound,
    upperbound);
275                                     end
276                                 end
277                             else
278                                 %% if YTRY better than
    second-highest point, use this point
279                                 candidatePoints(end,:) =
    pointTrial;
280                                 Z(end) = zTrial;
281                             end
282
283                              % the initial temperature is
    estimated in the first loop from
```

```
284                                 % the number of successfull
    and unsuccesfull moves, and the average
285                                 % increase in cost associated
    with the unsuccessful moves
286
287                                 if sgs.TEMP_LOOP_NUMBER == 1 &&
     isempty(sgs.TEMP_START),
288                                     if Z(1) > Z(end),
289                                         SUCCESSFUL_MOVES =
    SUCCESSFUL_MOVES+1;
290                                     elseif Z(1) <= Z(end),
291                                         UNSUCCESSFUL_MOVES =
    UNSUCCESSFUL_MOVES+1;
292                                         UNSUCCESSFUL_COSTS =
    UNSUCCESSFUL_COSTS+(Z(end)-Z(1));
293                                     end
294                                 end
295
296                                 if(sgs.saveData == 1)
297                                   fim = datestr(now,'yyyy-mm-dd
    HH:MM:SS.FFF');
298                                   datainsert(sgs.FuncOpt.Conn,'
    parametro',{'Gamma','Slack','GammaMax','GammaMin','
    SlackMax','SlackMin','Class1','Class2','TesteID','
    StepMaxMin','Step','Accuracy','DataStart','DataEnd'},{
    candidatePoints(1),candidatePoints(2),sgs.lowerBound(1),
    sgs.upperBound(1),sgs.lowerBound(2), sgs.upperBound(2),
    sgs.Class1ID ,sgs.Class2ID ,sgs.FtesteID ,0,
    numberIteration,sgs.bestZ,inicio,fim});
299                                 end
300
301                             end
302
303
304                             if iterationTemperature < MAXITERTEMP
    ,
305                                 break;
306                             end
307                 % store cost function values in COSTS vector
308                 costs((iterationTemperature-1)*ndimension+1:
    iterationTemperature*ndimension+1) = Z;
309
310                             % calculated initial temperature or
     recalculate temperature
311                             % using cooling schedule as
    proposed by Cardoso et al. (1996)
312                             %
```

```
313
314                             if sgs.TEMP_LOOP_NUMBER == 1 &&
    isempty(sgs.TEMP_START)
315                                 TEMPERATURE = -(
    UNSUCCESSFUL_COSTS/(SUCCESSFUL_MOVES+UNSUCCESSFUL_MOVES)
    )/log(((SUCCESSFUL_MOVES+UNSUCCESSFUL_MOVES)*sgs.
    INITIAL_ACCEPTANCE_RATIO-SUCCESSFUL_MOVES)/
    UNSUCCESSFUL_MOVES);
316                             elseif sgs.TEMP_LOOP_NUMBER ~= 0,
317                                 STDEV_Y = std(costs);
318                                 COOLING_FACTOR = 1/(1+
    TEMPERATURE*log(1+sgs.COOL_RATE)/(3*STDEV_Y));
319                                 TEMPERATURE = TEMPERATURE*
    min(sgs.MIN_COOLING_FACTOR,COOLING_FACTOR);
320                             end
321
322                             % add one to temperature loop
    number
323                             sgs.TEMP_LOOP_NUMBER = sgs.
    TEMP_LOOP_NUMBER+1;
324
325                 end
326                 X = sgs.bestX;
327 %
328             end
329
330
331             function [zFlutuationTrial,zTrial,pointTrial] =
    Amoeba(sgs,fun,point,fac,lb,ub,ndimension,TEMPERATURE)
332                 % Extrapolates by a factor fac through the face
     of the simplex across from
333                 % the high point, tries it, and replaces the
    high point if the new point is
334                 % better.
335
336                 % calculate coordinates of new vertex
337                 psum = sum(point(1:ndimension,:))/ndimension;
338                 pointTrial = psum*(1-fac)+point(end,:)*fac;
339
340                 % evaluate the function at the trial point.
341                 zTrial = objectiveFunction(sgs,fun,pointTrial,
    lb,ub);
342                 % substract random fluctuations to function
    values of current vertices
343                 zFlutuationTrial = zTrial-TEMPERATURE*abs(log(
    rand(1)));
344
```

```
345             return
346         end
347
348
349
350         %% Cost Function Evaluation
351         function [Zevaluation] =  objectiveFunction(sgs,fun
    ,point,lb,ub)
352             ndimension  =    length(point);
353             for i = 1:ndimension
354                 if point(i) < lb(i)
355                     Zevaluation = 1e12 +(lb(i)-point(i))
    *1e6;
356                     return;
357                 end
358                 if point(i) > ub(i)
359                     Zevaluation = 1e12 +(point(i)-ub(i))
    *1e6;
360                     return;
361                 end
362
363             end
364
365             %%calculate cost associated with point
366             Zevaluation  = feval(fun,sgs.FuncOpt,point,sgs.
    tradeoff);
367
368             sgs.numberOfEvaluations = sgs.
    numberOfEvaluations + 1;
369
370             %%save the best point ever
371             if Zevaluation < sgs.bestZ
372                 sgs.bestZ = Zevaluation;
373                 sgs.bestX = point;
374             end
375         end
376
377
378     end
379 end
```

Listing 7.3: Função de Custo

```
1 %%Primeiro parametro deve ser slack e o segundo gama
2 function [Z,model,tradeoff,acctreadeoff,svstreadoff] =
    AmoebaSAFunction(sgs,X,tradeoff)
3
4                                   acctreadeoff
        = 0;
```

```
5                                       svstreadoff
        = 0;
6                                       sgs.slack =  X(1);
7                                       sgs.gamma =  X(2);
8                                       params = sprintf('-s 0 -t 2
    -c %.8f -g %.8f',sgs.slack,sgs.gamma);
9                                       model  = svmtrain(sgs.
    trainLabelSA,sgs.trainDataSA,params);
10                                      [predictLabel,acc,decvalues
    ] = svmpredict(sgs.testLabelSA,sgs.testDataSA,model);
11
12
13                                      %%display('Esse      o
     n mero  de vetores de suporte ');
14                                      %display(sv);
15
16                                      fator = sqrt(( model.
    totalSV / length(sgs.trainLabelSA)));
17                                      %%display('Esse      o fator
     de vetores de suporte ');
18                                      %%display(fator);
19
20                                      if(tradeoff == 1)
21                                              Z
                = (100 - ( (acc(1)/2) + (fator/2)))/100 ;
22                                              acctreadeoff
                = acc(1);
23                                              svstreadoff
                = model.totalSV;
24                                      else
25                                              Z
                 = 1 - (acc(1)/100);
26                                      end
27
28
29 end
```

Listing 7.4: Constroi a arvore binária

```
1 %%Classe que cria um lista encadeada.
2 %% prev[]  <- node[] ->  next[]
3 %%http://stackoverflow.com/questions/1894846/printing-bfs-
    binary-tree-in-level-order-with-specific-formatting
4 classdef svmTree < handle
5     properties
6         Data;
7         limiar;
8         DistanceList;
9     end
```

```
10      properties (SetAccess = private)
11          Root                    =  [];
12          nodeRet                 =  [];
13          count                   =  0;
14          nodeID                  =  0;
15          listOfLeaf              = '';
16          numberCorrectCount      = 0;
17          numberIncorrectCount    = 0;
18          numberTotalCount        = 0;
19          svm                     ='';
20          type                    ='';
21          player                  ='';
22          gammaFix                =0;
23          slackFix                =0;
24          Conn                    =0;
25          Id                      =0;
26
27
28          TFinal                  = 0;
29          TInicial                = 0;
30          iterations              = 0;
31          Cost                    = 0;
32          Gama                    = 0;
33          saveDataBase            = 0;
34
35
36      end
37      methods
38          function tree = svmTree(type,gammafix,slackfix,conn
        ,id,cost,gama,tfinal,tinicial,iterations,kernel,
        savedatabase)
39              tree.limiar = 99;
40              tree.nodeID = 0;
41              set(0,'RecursionLimit',2500);
42              tree.listOfLeaf  = dlNode();
43
44              tree.Conn                   = conn;
45              tree.Id                     = id;
46              tree.gammaFix               = gammafix;
47              tree.slackFix               = slackfix;
48              tree.Cost                   = cost;
49              tree.Gama                   = gama;
50              tree.TFinal                 = tfinal;
51              tree.TInicial               = tinicial;
52              tree.iterations             = iterations;
53              tree.saveDataBase           = savedatabase;
54
55
```

```
56              if (tree.saveDataBase == 0 )
57                  tree.Conn = '';
58                  tree.Id   = '';
59              end
60              tree.svm = SVMABE(tree.Conn ,tree.Id,kernel);
61              tree.type = type;
62          end
63
64          function addNodeSVM(tree,classes,parent)
65              if~isempty(parent)
66                  if ~isempty(parent.classes)
67                      parent.classes = '';
68                  end
69              end
70              if(classes.getListCount() > 1)
71                  [class1 class2] = tree.DistanceList.
    calculateDistanceBetwwenClass(classes);
72                  if(strcmp(tree.type,'sa'))
73                      params = tree.svm.SA(class1,class2,tree
    .Cost,tree.Gama,tree.TFinal,tree.TInicial);
74                  elseif(strcmp(tree.type,'gr'))
75                      params = tree.svm.gridSearch(class1,
    class2,tree.Cost,tree.Gama);
76                  elseif(strcmp(tree.type,'fx'))
77                      params = tree.svm.fixParameter(class1,
    class2,tree.gammaFix,tree.slackFix);
78                  end
79
80                  newList         = dlNode();
81                  for i=1:classes.getListCount()
82                      dataClasses = classes.getItem(i);
83                      if( strcmp(class1.getClassName(),
    dataClasses.getClassName()) || strcmp(class2.
    getClassName(),dataClasses.getClassName()) )
84
85                      else
86                          newList.insert(dataClasses);
87                      end
88                  end
89                  classes = newList;
90
91                  for k=1:classes.getListCount()
92                      dataClasses = classes.getItem(k);
93                  end
94                      listLeft        = dlNode();
95                      listRight       = dlNode();
96                      listRight.insert(class1);
97                      listLeft.insert(class2);
```

```
98                  for i=1:classes.getListCount()
99                      dataClasses = classes.getItem(i);
100                     classlist = tree.svm.classifyNode(
    dataClasses,params.bias,params.alpha,params.
    supportVector,params.bestG,params.bestC,params.
    trainLabel,params.trainData,params.modelData);
101                     sizeOFClasses = size(classlist,1);
102                     positive = find(classlist > 0);
103                     percentPositive =  (size(positive,1) *
    100) / sizeOFClasses;
104                     negative = find(classlist < 0);
105                     percentNegative =  (size(negative,1) *
    100) / sizeOFClasses;
106                     if ((percentPositive > percentNegative)
     && (percentPositive > tree.limiar))
107                         listRight.insert(dataClasses);
108                     elseif((percentPositive <
    percentNegative) && (percentNegative > tree.limiar))
109                         listLeft.insert(dataClasses);
110                     else
111                         listRight.insert(dataClasses);
112                         listLeft.insert(dataClasses);
113                     end
114                 end
115                 for k=1:listRight.getListCount()
116                     dataClasses = listRight.getItem(k);
117                 end
118
119                 for k=1:listLeft.getListCount()
120                     dataClasses = listLeft.getItem(k);
121                 end
122             else
123                 listLeft        = dlNode();
124                 listRight       = dlNode();
125                 listRight.insert(classes.getItem(1));
126             end
127
128             if isempty(tree.Root)
129                 listofClass         = dlNode();
130                 listofClass         = classes;
131                 listofClass.insert(class1);
132                 listofClass.insert(class2);
133                 tree.nodeID = tree.nodeID +1;
134                 newNode    = NodeSvm(params,class1,class2,
    listofClass,'',parent,tree.nodeID);
135                 parent = newNode;
136                 tree.Root = newNode;
137             end
```

```
138                 if(listLeft.getListCount() > 0)
139                     if(listLeft.getListCount()>1)
140                         [class1 class2] = tree.DistanceList
    .calculateDistanceBetwwenClass(listLeft);
141                         if(strcmp(tree.type,'sa'))
142                             params = tree.svm.SA(class1,
    class2,tree.Cost,tree.Gama,tree.TFinal,tree.TInicial);
143                         elseif(strcmp(tree.type,'gr'))
144                             params = tree.svm.gridSearch(
    class1,class2,tree.Cost,tree.Gama);
145                         elseif(strcmp(tree.type,'fx'))
146                             params = tree.svm.fixParameter(
    class1,class2,tree.gammaFix,tree.slackFix);
147                         end
148                         tree.nodeID = tree.nodeID +1;
149                         newnode = NodeSvm(params,class1,
    class2,listLeft,listLeft,parent,tree.nodeID);
150                         parent.left = newnode;
151                     else
152                         class1 = listLeft.getItem(1);
153                         tree.nodeID = tree.nodeID +1;
154                         parent.left = NodeSvm(params,class1
    ,'','','',parent,tree.nodeID);
155                     end
156                 end
157
158                 if(listRight.getListCount() > 0)
159                     if(listRight.getListCount()>1)
160                         [class1 class2] = tree.DistanceList
    .calculateDistanceBetwwenClass(listRight);
161                         if(strcmp(tree.type,'sa'))
162                             params = tree.svm.SA(class1,
    class2,tree.Cost,tree.Gama,tree.TFinal,tree.TInicial);
163                         elseif(strcmp(tree.type,'gr'))
164                             params = tree.svm.gridSearch(
    class1,class2,tree.Cost,tree.Gama);
165                         elseif(strcmp(tree.type,'fx'))
166                             params = tree.svm.fixParameter(
    class1,class2,tree.gammaFix,tree.slackFix);
167                         end
168
169
170                         tree.nodeID = tree.nodeID +1;
171                         newnode = NodeSvm(params,class1,
    class2,listRight,listRight,parent,tree.nodeID);
172                         parent.right = newnode;
173                     else
174                         class1 = listRight.getItem(1);
```

```
175                         tree.nodeID = tree.nodeID +1;
176                         parent.right = NodeSvm(params,
    class1,'','','',parent,tree.nodeID);
177                     end
178                 end
179
180
181
182
183         end
184
185         function tree = generateTree(tree,classes)
186             leaf          = 'false';
187             noderet       = '';
188             tree.DistanceList = Distance();
189             node            = '';
190             parent          = '';
191             tree.Root       = '';
192             tree.nodeRet    = 'start';
193             while(~isempty(tree.nodeRet))
194                 tree.insertClassInTree(classes,parent);
195                 tree.nodeRet    = '';
196                 tree.checkTree(tree.Root);
197                 if ~isempty(tree.nodeRet)
198                     parent    =  tree.nodeRet;
199                     classes   =  tree.nodeRet.classes;
200                 else
201                     %%play(tree.player);
202
203                     tree = tree;
204                     return
205                 end
206             end
207             tree.nodeRet    = 'stop';
208         end
209
210         function checkTree(tree,root)
211
212             if isempty(root)
213
214             else
215                 if (isempty(tree.nodeRet))
216                     tree.checkTree(root.left);
217                 end
218                 if (~isempty(root.class2) && ~isempty(root.
    classes))
219                     tree.nodeRet = root;
220                 end
```

```
221                 if (isempty(tree.nodeRet))
222                     tree.checkTree(root.right);
223                 end
224             end
225 
226         end
227 
228         function checkLeaf(tree,root)
229             if isempty(root)
230             else
231                 if (~isempty(root.left))
232                     tree.checkLeaf(root.left);
233                 end
234                 if (strcmp(root.isLeaf,'true') )
235                     tree.listOfLeaf.insert(root);
236                 end
237                 if (~isempty(root.right))
238                     tree.checkLeaf(root.right);
239                 end
240             end
241         end
242 
243         function printTree(tree,root,grapho,parent)
244             if isempty(root)
245             else
246                 %%disp(root);
247                 parent = grapho.addNode(root,parent);
248                 if (~isempty(root.left))
249                     tree.printTree(root.left,grapho,parent)
    ;
250                 end
251                 if (~isempty(root.right))
252                     tree.printTree(root.right,grapho,parent
    );
253                 end
254             end
255 
256         end
257 
258 
259         function sumTree(tree,root,matrix,parent)
260             if isempty(root)
261             else
262                 %%disp(root);
263                 parent = matrix.addNode(root,parent);
264                 if (~isempty(root.left))
265                     tree.sumTree(root.left,matrix,parent);
266                 end
```

```
267                 if (~isempty(root.right))
268                     tree.sumTree(root.right,matrix,parent);
269                 end
270             end
271
272         end
273
274
275         function [treeWithPixel list] = runPixelTree(tree,
    svmTree,list)
276             debug        = 'false';
277             debugStarted = 'false';
278             reply        = '';
279             for classes=1:list.getListCount()
280                 dataClasses = list.getItem(classes);
281                 sample      = dataClasses.getTestData();
282                 classname   = dataClasses.getClassName();
283                 node        = svmTree.Root;
284
285                  for i = 1:size(sample,1)
286
287                     pixel = classifiedPixels(classname,
    sample(i,:));
288                     while(strcmp(node.isLeaf,'false') )
289
290                         %%classPixel = tree.svm.
    classifyPixel(sample(i,:),node.params.bias,node.params.
    alpha,node.params.supportVector,node.params.bestG,node.
    params.bestC,node.params.trainLabel,node.params.
    trainData);
291                         [classPixel,acc,decvalues] =
    svmpredict(0,sample(i,:),node.params.modelData);
292
293                         if (classPixel > 0)
294                             node =  node.right;
295                         else
296                             node =  node.left;
297                         end
298
299                         if ( strcmp(node.isLeaf,'true') )
300                             if ( strcmp(node.class1Name,
    pixel.className) )
301                                 tree.numberCorrectCount   =
     tree.numberCorrectCount   +1;
302                                 set(pixel,'isCorrect',1);
303                             else
304                                 tree.numberIncorrectCount =
     tree.numberIncorrectCount +1;
```

```
305                                     set(pixel,'isCorrect',0);
306                                 end
307
308                                 tree.numberTotalCount =  tree.
    numberTotalCount +1;
309                                 node.addClassifiedPixel(pixel);
310                                 break;
311                             end
312
313                         end
314                         node = svmTree.Root;
315                     end
316
317                     tree.numberCorrectCount   = 0;
318                     tree.numberIncorrectCount = 0;
319                     tree.numberTotalCount     = 0;
320
321
322                 end
323                 treeWithPixel = svmTree;
324                 tree.checkLeaf(svmTree.Root)
325                 list          = tree.listOfLeaf;
326                 return;
327
328             end
329
330             function nodeReturn = insertClassInTree(tree,
    classes,parent)
331                 addNodeSVM(tree,classes,parent);
332                 return
333             end
334
335         end
336 end
```

REFERÊNCIAS

ABE, S. **Support Vector Machines for Pattern Classification**. London,: Springer-Verlang, 2005.

ANDREOLA, R.; HAERTEL, V. Classificação de Imagens Hiperespectrais Empregando Support Vector Machines. , [S.l.], v.16, p.210–231, 2010.

BAZI, F. M. Y. Toward an Optimal SVM Classification System for Hyperspectral Remote Sensing Images. , [S.l.], v.44, p.3374–3385, Novembro 2006.

BLAKE, C.; MERZ, C. 1998.

BOARDMAN, T. T. M. A Heurist for Free Parameter Optimization with Support Vector Machines. , [S.l.], July 2006.

BURGES, C. J. A Tutorial on Support Vector Machines for Pattern Recognition. , [S.l.], p.121–167, 1998.

CAROLINA, L. A.; CARVALHO, C. A. D. Evolutionary tuning of SVM parameter values in multiclass problems. , [S.l.], p.3326–3334, 2008.

CHIH-CHUNG; LIN, C.-J. C. LIBSVM : a library for support vector machines. , [S.l.], p.27:1–27:27, 2011.

DIETTERICH, T. G.; BAKIRU, G. Solving multiclass learning problems via error-correcting output codes. , [S.l.], p.263–86, 2 1995.

DUDA, O. R.; HART, P. E.; STORK, D. G. **Pattern Classification**. [S.l.]: Wiley-Intercience Publication, 2000.

FLOUDAS, C. A.; PARDALOS, P. M. **Encyclopedia of Optimization**. [S.l.]: Springer, 2008.

FUKUNAGA, K. **Introduction to Statical Pattern Recognition**. [S.l.]: Academic Press, 1990.

HAMEL, L. **Knowledge Discovery with Support Vector Machines**. New Jersey,: John Wiley & Sons, 2009.

HUANG, C.-J. W. C.-L. A GA-basead feature selection and parameters optimization for support vector machines. , [S.l.], v.31, p.231–240, 2006.

JGraphX 2.2.0.0. Lodon, 145-157 St John Street: JGraph Ltd 2006-2012., 2012.

KIRKPATRICK, S.; JR., C. D.; GELATT, M. P. V. Optimization by simulated annealing. , [S.l.], p.671–680, 1983.

LANDGREBE, D. **On Information Extraction Principles for Hyperspectral Data**. [S.l.: s.n.], 1997.

LIEPERT, M. Topological fields chunking for German with SVM's: optimizing SVM-parameters with GA's. In: INTERNATIONAL CONFERENCE ON RECENT ADVANCES IN NATURAL LANGUAGE PROCESSING (RANLP). **Proceedings...** [S.l.: s.n.], 2003.

MANDAL, J. W. H. G. Mrinal k. Classification of Hyperspectral Image with Feature Selection and Parameter Estimation. , [S.l.], 2010.

MATLAB. **MATLAB (R2012a)**. Natick, Massachusetts: The MathWorks Inc., 2012.

MCCAFFREY, J. **MSDN Magazine - Amoeba Method Optimization using C Sharp**. 2013.

MELGANI, F.; BRUZZONE, L. Classification of hyperspectral remote sensing images with support vector machines. , [S.l.], v.42, p.1778–1790, Agosto 2004.

MORAES, D. A. O. **Extração de Feições em dados Imagem com Alta Dimensão por otimização da Distância de Bhattacharrya em um classificador de Decisão em Árvore.** Porto Alegre,: [s.n.], 2005.

MYSQL. **MYSQL (5.6)**. [S.l.]: Oracle Corporation., 2012.

NELDER, J. A.; MEAD, R. A simplex method for function minimization. , [S.l.], p.308–313, January 1965.

NELLO; SHAWE-TAYLOR, J. C. **An introduction to support Vector Machines**: and other kernel-based learning methods. New York,: Cambridge University Press, 2000.

OSUNA, E.; FREUND, R.; GIROSI, F. An improved training algorithm for support vector machines. In: NEURAL NETWORKS FOR SIGNAL PROCESSING [1997] VII. PROCEEDINGS OF THE 1997 IEEE WORKSHOP. **Anais...** [S.l.: s.n.], 1997. p.276–285.

PAL, M.; MATHER, P. M. Support vector machines for classification in remote sensing. , [S.l.], p.1007–1011, Março 2005.

PLATT, J. C. Advances in kernel methods. In: SCHöLKOPF, B.; BURGES, C. J. C.; SMOLA, A. J. (Ed.). . Cambridge, MA, USA: MIT Press, 1999. p.185–208.

PLATT, J. et al. Large margin DAGSVM's for multiclass classification. , [S.l.], p.547–553, 2000.

PRESS, W. H. et al. **Numerical Recipes 3rd Edition**: the art of scientific computing. 3.ed. New York, NY, USA: Cambridge University Press, 2007.

R. L. SALCEDO, F. M. F. C. The simplex-simulated annealing approach to continuous non-linear optimization. , [S.l.], v.21, p.1065–1080, Setembro 1996.

RICHARDS, J. A.; JIA, X. **Remote Sensing Digital Image Analysis An Introduction**. Berlin Heidelberg 2006,: Springer-Verlang, 2006.

SAMADZADEGAN, F.; HASANI, H.; SHENK, T. Simultaneous feature selection and SVM parameter determination in classification of hyperspectral imagery using Ant Colony Optimization. , [S.l.], p.139–156, 2012.

SAVAVIAN, S. R.; LANDGREBE, D. A Survey of Decision Trees Classifier. , [S.l.], p.660–674, 1991.

SHAWE-TAYLOR, J.; CRISTIANINI, N. **Kernel Methods for Pattern Analysis**. New York, NY, USA: Cambridge University Press, 2004.

SMOLA, A. et al. **Advances in large margin classifiers**. London,: MIT Press, 2000.

VAPNIK, V. **Statistical Learning Theory**. NY,: Wiley, 1998.

VAPNIK, V.; CHERVONENKIS, A. **Theory of Pattern Recognition**. Berlin,: Akademie Verlang, 1974.

VAPNIK, V. N. **The nature of statistical learning theory**. New York, NY, USA: Springer-Verlag New York, Inc., 1995.