JANETE DA SILVA BORGES

CLASSIFICADORES E MÁQUINAS DE SUPORTE VECTORIAL EM IMAGENS MULTI-ESPECTRAIS

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

DEPARTAMENTO DE MATEMÁTICA APLICADA
FACULDADE DE CIÊNCIAS DA UNIVERSIDADE DO PORTO
MAIO DE 2003

JANETE DA SILVA BORGES

CLASSIFICADORES E MÁQUINAS DE SUPORTE VECTORIAL EM IMAGENS MULTI-ESPECTRAIS

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Estatística

DEPARTAMENTO DE MATEMÁTICA APLICADA
FACULDADE DE CIÊNCIAS DA UNIVERSIDADE DO PORTO
2003

Aos meus pais...

Agradecimentos

Quero agradecer a um conjunto de pessoas que me ajudaram a desenvolver o trabalho aqui apresentado, todas elas parcialmente responsáveis pelo seu conteúdo final.

Ao Doutor Joaquim P. Costa e Doutor André Marçal agradeço a orientação ao longo deste trabalho, a disponibilidade, os bons conselhos e incentivos.

À Associação Florestal do Vale do Sousa, particularmente ao Eng. Alexandre Gomes, pela disponibilização da informação relativa ao Vale do Sousa, sem a qual este trabalho seria impossível.

A Machelle Wilson pela proveitosa troca de impressões acerca das SVMs e pela bibliografia disponibilizada.

Aos amigos pelo apoio e conselhos que me deram.

E porque os últimos são os primeiros, os agradecimentos finais aos meus pais, ao Óscar e ao meu irmão... por tudo!

Resumo

O método de classificação por máquinas de suporte vectorial surge na sequência do desenvolvimento da Teoria de Aprendizagem Estatística introduzida por Vapnik no final dos anos 70. Este método de classificação, desenvolvido a partir dos anos 90, produz fronteiras lineares num espaço de dimensão superior ao original usando funções *kernel* de Mercer.
As máquinas de suporte vectorial têm sido aplicadas em diversas áreas com resultados positivos. São frequentes os casos em que este método ultrapassa outros mais conceituados. Esta é uma das motivações para este trabalho: explorar a capacidade das máquinas de suporte vectorial na classificação de imagens multi-espectrais.

Inicialmente são apresentadas algumas ideias da Teoria da Aprendizagem Estatística, avançando-se depois para o método de Máquinas de Suporte Vectorial. A sua aplicação numa imagem de satélite ASTER, com 9 bandas espectrais, é feita de uma forma especial. A classificação é orientada a objectos e não a pixels como tradicionalmente. Esta é uma abordagem que reflecte melhor a realidade. Os objectos são o resultado da segmentação da imagem. Existem vários trabalhos de classificação de imagens de satélite com máquinas de suporte vectorial, mas estes usam apenas a informação espectral do pixel. A classificação de imagens segmentadas permite ter em conta aspectos relativos à textura, morfologia e vizinhança dos pixels que doutra forma seriam ignorados pela classificação tradicional. Para além de que, permite a redução do número de indivíduos a classificar.
Também o método dos K vizinhos mais próximos e discriminação logística são utilizados neste trabalho. É feita uma breve descrição destes métodos e uma análise dos resultados. A introdução destes métodos visa a comparação com os resultados obtidos pelas máquinas de suporte vectorial.
Estes três métodos de classificação foram testados nos dados originais, nas suas componentes principais e nas variáveis canónicas. Sendo que, estas últimas demonstraram ser as mais adequadas.

Finalmente, é apresentado o método de análise de componentes principais usando funções *kernel* de Mercer. Este método permite calcular uma versão não linear das componentes principais. À semelhança das máquinas de suporte vectorial, isto é feito usando funções *kernel*.

Abstract

The Statistical Learning Theory developed by Vapnik at the end of the 70's, gave arise to support vector machines. Developped in the 90's, these techniques produces nonlinear boundaries by constructing a linear boundary in a large, transformed version of the feature space by means of Mercer's kernel functions.
There are many fields were the support vector machines have been applied with positive results. In most of these cases, this method outperforms more accredited ones. This is one of the motivations for this work: explore the ability of support vector machines in classification of multi-spectral data.

In this work some of the main ideas of the Statistical Learning Theory are presented initially, and then move on to the support vector machines method. Its application to an ASTER image, with 9 spectral bands, is made in a special way. Object oriented classification is made instead of traditional pixel classification. This approach is more adequate to extract real world objects. Objects are the result of a segmentation task. There are some works using support vector machines in multi-spectral images, but they use only the pixel spectral information. The classification of segmented images takes into account aspects relative to texture, morphology and neighborhood of pixels. The traditional pixel classification ignores these characteristics. Furthermore, the segmentation allows the reduction of the number of elements to classify.

The method of the K nearest neighbors and logistic discrimination are also used in this work. One brief description of these methods and an analysis of the results is made. The aim of the introduction of these methods is to compare their results with the results from the support vector machines. These three classification methods have been tested in the original data, its principal components and in the canonical variates. These last variables produced the best results.

Finally, Kernel Principal Component Analysis is presented. This is a non linear form of principal component analysis. By the use of Mercer kernels, one can efficiently compute principal components in high-dimensional feature spaces, related to input space by some nonlinear map.

Conteúdo

Capítulo 1

Introdução

Nas últimas décadas têm-se dado avanços extraordinários em tecnologias de informação e computação. Na sequência destes avanços, cresce também a quantidade de informação nas mais variadas áreas. A necessidade de estudar, compreender e aprender com estes dados levou ao desenvolvimento de novas ferramentes estatísticas que resultaram em novas áreas, sendo uma delas a aprendizagem estatística (*statistical learning*). A construção de máquinas capazes de aprender a partir de experiências tem vindo a ser desenvolvida à medida que se avança a nível tecnológico.

Uma das áreas da aprendizagem estatística é a classificação. Existem imensas áreas onde a classificação é usada, por exemplo, detecção remota, em medicina, processamento de sinais digitais, reconhecimento de caracteres, biologia, etc.

Neste trabalho é apresentado um método de classificação relativamente recente: Máquinas de Suporte Vectorial (SVM - Support Vector Machines). Este método foi desenvolvido por Vapnik [24] na sequência da sua teoria de Minimização do Risco Estrututral. Antes de proceder à descrição do método, pode ler-se no capítulo introdutório algumas noções da Teoria da Regularização e da Aprendizagem Estatística, úteis para perceber alguns dos fundamentos das Máquinas de Suporte Vectorial. A descrição do método de SVMs é feita no capítulo 2. Nesta sequência, é apresentada a aplicação deste método numa imagem multiespectral no capítulo 3, com a particularidade de se trabalhar a nível de objectos e não a nível de pixels. No capítulo seguinte são comparados os resultado obtidos com o método dos k vizinhos mais próximos e com discriminação logística. No capítulo 5 descreve-se brevemente a análise de componentes principais usando funções *kernel*, isto é, um método não linear de transformação dos dados. No capítulo final são apresentadas as conclusões.

No trabalho foi usado o *eCognition* [9] (uma ferramenta de análise de imagens multi-espectrais direcionada a objectos) para o pre-processamento da imagem multi-espectral (segmentação), o *PCI* [16] para a análise das classes consideradas (distância de Jeffries-Matusita), e o *R-project* [18] para o estudo e comparação dos métodos de classificação.

1.1 Metodologia de Aprendizagem

O uso de exemplos para sintetizar programas é conhecido como metodologia de aprendizagem, no caso particular de os exemplos serem pares entrada/saída denomina-se análise discriminante (aprendizagem supervisionada). Esses exemplos entrada/saída são chamados conjunto de treino [6]. No caso de classificação supervisionada, as entradas correspondem a características de um indivíduo (ou observação) e a saída à classe a que esse indivíduo pertence. A relação entre as entradas e saídas pode, normalmente, ser descrita por uma função; no caso de classificação essa denomina-se a função de decisão. O algoritmo que toma o conjunto de treino e com ele constrói a função de decisão é chamado de algoritmo de aprendizagem (ou máquina de aprendizagem). Um algoritmo de aprendizagem com saídas binárias é referido como um problema de classificação binária, um com um número finito de categorias como um problema de classificação multi-classe, enquanto que para valores reais o problema é conhecido como regressão.

O objectivo da aprendizagem estatística é então construir uma função que identifique correctamente as classes dos elementos do conjunto de treino. Existem dois problemas associados a este objectivo: o primeiro é que a função em causa pode não ter uma representação muito simples e portanto não ser fácil verificar se os resultados são os correctos; o segundo tem a ver com o ruído frequentemente presente no conjunto de treino não garantindo da existência de uma função que represente correctamente esse conjunto.

1.2 Teoria da Regularização e Teoria da Aprendizagem Estatística

1.2.1 Teoria da Regularização

Na teoria de Regularização Estatística consideram-se técnicas que conduzam a soluções da forma:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{l} c_i K(\mathbf{x}, \mathbf{x}_i) \tag{1.1}$$

onde $\mathbf{x}_i$ são as observações de entrada, K uma determinada função simétrica positiva chamada *kernel*[1], e c_i um conjunto de l parâmetros a ser determinados a partir das observações. A função $\hat{f}$ é calculada minimizando funcionais do tipo:

$$\psi[f] = \frac{1}{l} \sum_{i=1}^{l} V(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_K^2 \tag{1.2}$$

[1] O termo correcto a atribuir a esta função seria núcleo. No entanto, para evitar confusões com as funções núcleo usadas para estimar funções de densidade de probabilidade, opta-se, neste trabalho, por usar o termo inglês *kernel* para fazer referência a estas funções.

onde f pertence a um espaço de Hilbert H, V é uma função de perda que mede a qualidade da saída prevista, $||f||_K^2$ um termo de amaciamento que pode ser visto como uma norma no espaço de Hilbert definido pela kernel K e parâmetro positivo λ que controla o peso relativo entre o amaciamento e a qualidade das previsões.

A escolha de V conduz a diferentes técnicas de aprendizagem, cada uma levando a diferentes algoritmos para o cálculo dos coeficientes c_i. Doravante, considerar-se-á

$$V(y_i, f(\mathbf{x}_i)) = \frac{1}{2}|y_i - f(\mathbf{x}_i)|$$

por ser esta a função de perda usada para máquinas de suporte vectorial. A inclusão do termo de amaciamento assegura que a solução tem boas capacidades predictivas. Este assunto precisa, contudo, de um tratamento probabilístico que não é estudado pela Teoria da Regularização[11].

1.2.2 Teoria da Aprendizagem Estatística

Considere-se um conjunto de l observações i.i.d.. Cada observação consiste num par: um vector $\mathbf{x}_i \in X, i = 1, .., l$ e a classe associada $y_i \in Y$ relacionados por uma relação probabilística $P(\mathbf{x}, y)$. O problema da aprendizagem consiste em definir uma função $f : X \to Y$ que prevê o valor de y para um determinado $\mathbf{x}$.

Na Teoria da Aprendizagem Estatística, a forma de resolver este problema consiste em definir uma função de risco que meça o erro médio ou o risco associado com o estimador e depois encontrar o estimador com menor risco.

A dimensão de Vapnik Chervonenkis. Uma máquina de aprendizagem pode ser vista como um conjunto de funções (que a máquina tem à sua disposição), um princípio de indução, e um algoritmo para implementar o princípio de indução no conjunto de funções dado. Várias vezes o termo máquina de aprendizagem é usado para referir o seu conjunto de funções - neste sentido, fala-se na capacidade ou dimensão de Vapnik Chevonenkis (VC) das máquinas de aprendizagem.

A dimensão de VC é uma propriedade de um conjunto de funções $\{f(\mathbf{x}, \alpha)\}$ que permite medir a sua complexidade. A dimensão de VC da classe de funções $\{f(\mathbf{x}, \alpha)\}$ é definida como o maior número de pontos (em qualquer disposição) que pode ser separado por membros de $\{f(\mathbf{x}, \alpha)\}$. Um conjunto de pontos diz-se separável por uma classe de funções se, qualquer que seja a atribuição binária de uma classe a cada ponto, um membro da classe de funções é capaz de os separar.

Note-se que se a dimensão de VC é h, então existe pelo menos um conjunto de h pontos que são separáveis, mas de uma forma geral, não é verdade que todos os conjuntos de h pontos possam ser separáveis. Do teorema 1.1 tem-se um corolário que permite definir a dimensão de VC de uma família de hiperplanos em R^n.

Teorema 1.1 ***(Burges, 1998)*** *Considere-se um conjunto de m pontos em R^n. Escolha-se qualquer um dos pontos como origem. Então os m pontos podem ser separados por hiperplanos orientados se e só se os restantes pontos são linearmente independentes.*

Corolário: A dimensão de VC de um conjunto de hiperplanos orientados em R^n é $n+1$, uma vez que se pode sempre escolher $n+1$ pontos, e escolher um dos pontos como origem, de forma a que os restantes n pontos sejam linearmente independentes. No entanto é impossível escolher $n+2$ pontos uma vez que é impossível ter-se $n+1$ pontos linearmente independentes em R^n (ver exemplo em R^2 na fig.1.1).

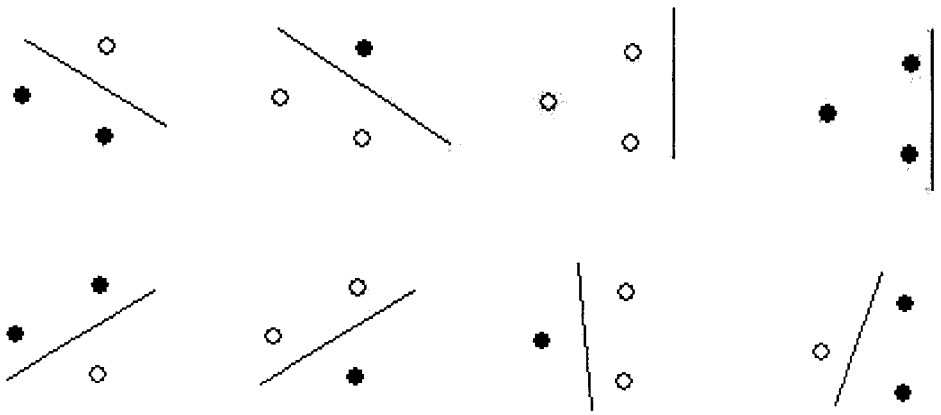

Figura 1.1: Um exemplo da dimensão de VC em R^2: a dimensão de um conjunto de hiperplanos orientados em R^2 é 3. *Figura adaptada de [3]*

Minimização do Erro Empírico. Suponha-se que se tem uma máquina cuja função é "aprender" a relação $\mathbf{x}_i \longmapsto y_i$. A máquina é definida por um conjunto de funções $\{f(\mathbf{x},\alpha)\}$ que descrevem essa relação, onde cada $f(\mathbf{x},\alpha)$ é determinística e ajustável pelo parâmetro α. Uma escolha particular de α resulta numa "máquina treinada". O valor esperado para o erro de teste é então:

$$R(\alpha) = \int \frac{1}{2}|y - f(\mathbf{x},\alpha)|dP(\mathbf{x},y) \tag{1.3}$$

Na prática, não é possível calcular esta função uma vez que se desconhece $P(\mathbf{x},y)$. $R(\alpha)$ é chamado risco (ou erro) esperado, ou erro teórico. Uma solução para este problema é o príncipio de minimização do risco empírico (ERM - Emprirical Risk Minimization) [24]. O método ERM consiste em usar o conjunto de treino para construir uma aproximação do risco esperado $R(\alpha)$. Essa aproximação é o chamado risco emprírico, $R_{emp}(\alpha)$, e não é mais do que o erro médio no conjunto de treino:

$$R_{emp}(\alpha) = \frac{1}{2l}\sum_{i=1}^{l}|y_i - f(\mathbf{x}_i,\alpha)| \tag{1.4}$$

A minimização directa do risco empírico pode ser problemática por duas razões. Primeiro, porque é um problema *"ill-posed"*[2][11], uma vez que admite um número infinito de soluções. Segundo, porque apesar do erro empírico poder ser muito próximo de zero, o erro esperado pode ser muito grande (note-se que $R_{emp}(\alpha)$ é calculado apenas com os valores de treino).

A Teoria de Aprendizagem Estatística apresenta limites probabilísticos para a distância entre o erro empírico e o erro esperado. Este limite é função do número de elementos do conjunto de treino l e da capacidade da máquina h. A capacidade da máquina é uma medida da complexidade do espaço onde se encontram as funções de decisão, e é dada pela dimensão de VC. O limite é da forma:

$$R(\alpha) \leq R_{emp}(\alpha) + \varphi\left(\sqrt{\frac{h}{l}}, \eta\right) \tag{1.5}$$

com probabilidade $1 - \eta$, onde φ é uma função crescente de $\frac{h}{l}$ e η. Intuitivamente, se a capacidade da máquina (h) for muito grande e o número de elementos de treino pequeno, a distância entre o risco empírico e o risco esperado pode ser grande. Torna-se assim necessário encontrar um compromisso entre o risco empírico e a dimensão do espaço. Se h for conhecido, o risco limite (membro direito de Eq. (1.5)) é facilmente calculado; assim, dadas várias máquinas de aprendizagem, i.e., vários conjuntos $\{f(\mathbf{x}, \alpha)\}$ e fixando um η suficientemente pequeno, escolhe-se a máquina que resulta num menor majorante do erro esperado. Esta é a ideia da minimização do erro estrutural.

Minimização do Erro Estrutural. Como já foi referido, no lugar de procurar minimizar apenas o risco empírico, torna-se necessário encontrar um compromisso entre o risco empírico e a complexidade do espaço para obter um limite mais realista para o risco esperado.
A ideia da minimização do erro estrutural (SRM - Structural Risk Minimization) é definir uma sequência de espaços encaixados

$$H_1 \subset H_2 \subset \ldots \subset H_M \tag{1.6}$$

em que cada um deles tem capacidade finita $h_1 \leq h_2 \leq \ldots \leq h_m$ e escolher a função $f(\mathbf{x}, \alpha)$ minimizadora do risco em H_{M^*} para o qual o limite do risco estrutural (lado direito da Eq. (1.5)) é minimizado.
Conforme se pode observar na figura 1.2, uma máquina de aprendizagem com maior complexidade, i.e, com maior conjunto de funções H_n, permite um erro de treino menor; uma máquina menos complexa, com um H_i menor tem a dimensão de VC menor e portanto resulta num termo de confiança φ menor. A minimização do erro estrutural encontra o melhor compromisso entre estes dois casos escolhendo a função da máquina de aprendizagem $f(\mathbf{x}, \alpha)$ tal que o limite do erro definido na Eq.(1.5) é

[2]Um problema *well-posed*, é um problema cuja solução (a) existe, (b) é única e (c) depende continuamente dos dados. Um problema para o qual pelo menos uma destas condições não é verificada é *ill-posed*.

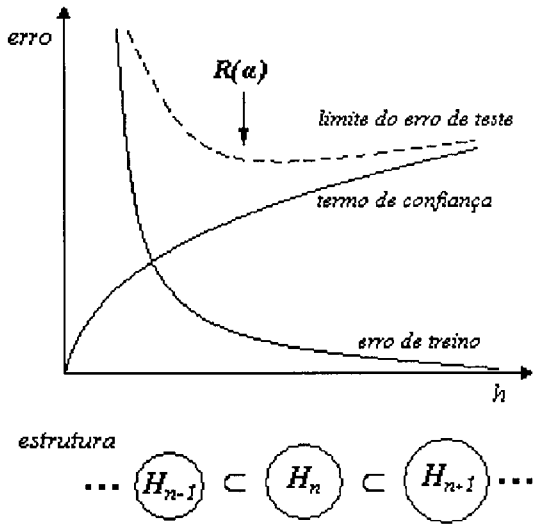

Figura 1.2: Descrição gráfica da Eq. (1.5), para um l fixo. *Figura adaptada de [22]*

mínimo.
As máquinas de suporte vectorial, apresentadas no capítulo seguinte, são construídas tendo em conta esta teoria de minimização do erro estrutural. (Ver Proposição??)

1.3 Funções *Kernel* de Mercer

Nesta secção são introduzidas as funções *kernel* de Mercer, essenciais para as máquinas de Suporte Vectorial e para a Análise em Componentes Principais com funções *kernel* de Mercer.

A complexidade da função a ser aprendida depende da forma em que está representada, e a dificuldade da tarefa de aprendizagem pode variar de acordo com essa representação. Idealmente, deve ser escolhida uma representação que se adapte ao problema de aprendizagem. Assim, uma tarefa comum de pré-processamento envolve modificar a representação dos dados:

$$\mathbf{x} = (x_1, \ldots, x_d) \longmapsto \phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_N(\mathbf{x}))$$

Este passo é equivalente a transformar o espaço original X num novo espaço, $Y = \{\phi(x)|\mathbf{x} \in X\}$ - o espaço final.
O objectivo da introdução das funções *kernel* de Mercer é permitir a transformação dos dados no espaço original num espaço de dimensão superior (até infinito) onde a separação linear é possível.
Por definição, uma função *kernel* K é tal que, para todo o $\mathbf{x}, \mathbf{z} \in X$

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \tag{1.7}$$

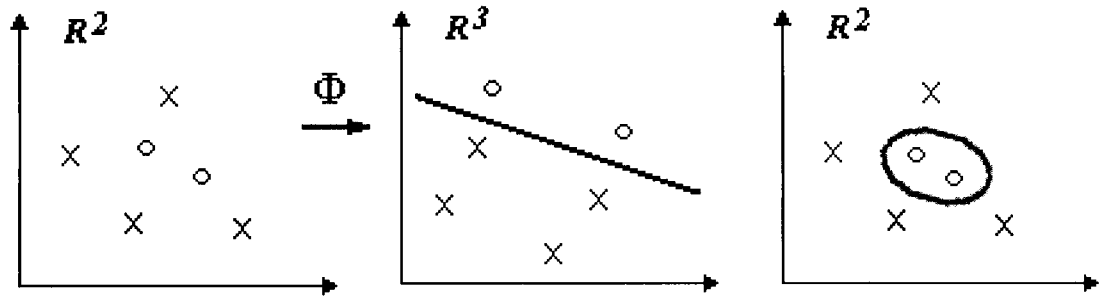

Figura 1.3: Fazendo uma transformação não linear (via ϕ) dos dados originais (esq.) num espaço de dimensão superior F (neste exemplo $\mathbf{R}^3$), e construíndo um hiperplano lá, tem-se no espaço original uma fronteira não linear.

onde $\phi : X \longmapsto Y$ é uma função não linear do espaço original X no espaço final Y onde está definido um produto interno [6].

Condição de Mercer. A questão que se coloca é, para que funções *kernel* K existe o par $\{Y, \phi\}$? Isto é, será que, para qualquer função *kernel*, existe sempre uma função ϕ que transforme o espaço X num espaço Y? A resposta é dada pela condição de Mercer [3]:

Existe uma função ϕ *e uma expansão* $K(\mathbf{x}, \mathbf{y}) = \sum_i \phi(\mathbf{y})_i \cdot \phi(\mathbf{x})_i$ *se e só se, para qualquer* $g(\mathbf{x})$ *com norma* L_2 *finita* $\left(\int g(\mathbf{x})^2 d\mathbf{x} < \infty\right)$, *se tem*

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq \mathbf{0} \tag{1.8}$$

Esta condição nem sempre é fácil de se verificar. Contudo, prova-se que potências inteiras da forma $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^p$ verificam sempre a condição de Mercer. Prova-se também que qualquer Kernel que possa ser expressa da forma

$$K(\mathbf{x}, \mathbf{y}) = \sum_{p=0}^{\infty} c_p (\mathbf{x} \cdot \mathbf{y})^p$$

onde c_p são coeficientes reais e a série uniformemente convergente, satisfaz a condição de Mercer [3].

Mas o que é que acontece quando uma kernel não satisfaz a condição de Mercer? Por vezes, para funções *kernel* que não satisfazem a condição de Mercer é possível encontrar conjuntos de treino nos quais essa função converge perfeitamente [3].

Exemplos de funções de *kernel*. A funções *kernel* mais usadas em máquinas de suporte vectorial para tarefas de classificação são do tipo:

$$\textit{Kernel Polinomial: } K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p \tag{1.9}$$

$$\textit{Kernel Gaussiana: } K(\mathbf{x},\mathbf{y}) = e^{-||\mathbf{x}-\mathbf{y}||^2/2\sigma^2} \tag{1.10}$$

$$\textit{Kernel Sigmoidal: } K(\mathbf{x},\mathbf{y}) = \tanh(\kappa\mathbf{x}\cdot\mathbf{y} - \delta) \tag{1.11}$$

Como já foi visto, a *kernel* polinomial verifica sempre a condição de Mercer para todo o p inteiro positivo, assim como a *kernel* Gaussiana (ou RBF). A *kernel* sigmoidal satisfaz a condição de Mercer apenas para certos valores de κ e δ; contudo, actualmente já se conhecem condições necessárias sobre estes parâmetros para que se verifique a condição de Mercer [3].

Capítulo 2

Classificadores e Máquinas de Suporte Vectorial

O algoritmo de Máquinas de Suporte Vectorial é baseado nos resultados da Teoria de Aprendizagem mencionada na secção 1.2. Usando as funções *kernel* (Secção 1.3) tem-se diferentes tipos de classificadores.

Este capítulo descreve o método de máquinas de suporte vectorial (SVM - Support Vector Machines). A descrição deste método será feita considerando primeiro, o caso de classificadores de suporte vectorial lineares, tendo em conta o caso separável e não separável, avançando depois para classificadores de suporte vectorial não lineares (máquinas de suporte vectorial), com duas classes. Mais tarde será feita a generalização para o caso de classificação com mais do que duas classes.

O problema de classsificação por SVMs resume-se a um problema de optimização; assim, será feita referência a alguns métodos de resolução de problemas de optimização.

2.1 Classificadores de Suporte Vectorial

Como base para o método SVM, é necessário descrever o algoritmo de suporte vectorial (SV) com algum detalhe. Este método aplica-se a observações linearmente separáveis. O algoritmo de suporte vectorial pode ser descrito em quatro passos. Primeiro é criada uma estrutura para as funções de decisão suficientemente simples, de forma a ser possível a formulação de uma fronteira no espaço correspondente à respectiva dimensão de VC. Baseando-se neste resultado, o algoritmo de *margem óptima* minimiza a dimensão de VC para esta classe de funções de decisão. Este algoritmo é então generalizado em dois passos de forma a obter as máquinas de suporte vectorial: o caso dos problemas de classificação não separáveis e o caso de funções de decisão não lineares, tendo em conta o limite da dimensão de VC.

2.1.1 Estrutura das funções de decisão

A escolha particular de uma estrutura, idêntica à definida na expressão (1.6), resulta num algoritmo de aprendizagem. Este é obtido a partir da aplicação da teoria da minimização do erro estrutural (Secção 1.2) num dado conjunto de funções. O algoritmo de vectores de suporte é baseado numa estrutura construída com base num conjunto de funções hiperplanas.

Considere-se uma amostra em que a cada observação está associada uma classe: -1 (negativa) ou 1 (positiva):

$$\{\mathbf{x}_i, y_i\}, i = 1, ..., l, y_i \in \{-1, 1\}, \mathbf{x} \in \mathbf{R}^d.$$

O objectivo do classificador de suporte vectorial é encontrar um hiperplano óptimo que separe estas duas classes, isto é, um hiperplano que maximize a separação entre as classes. Este hiperplano é formado pelos pontos $\mathbf{x}$ que satisfazem:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

onde $\mathbf{w}$ é o vector normal ao hiperplano, $|b|/||\mathbf{w}||$ a distância perpendicular do hiperplano à origem, e $||\mathbf{w}||$ a norma euclidiana de $\mathbf{w}$ (fig. 2.1).

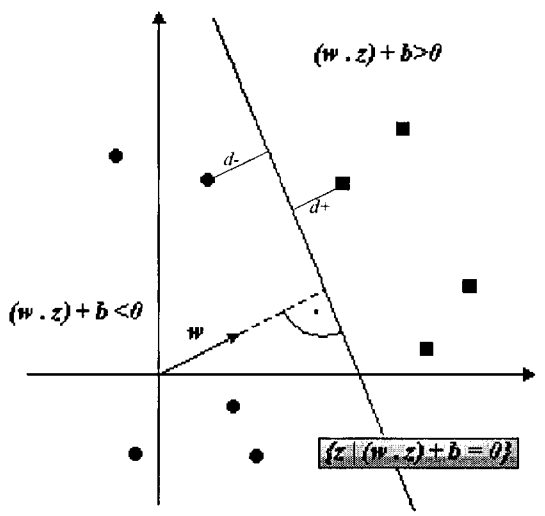

Figura 2.1: Um hiperplano separador, definido em termos de um vector director $\mathbf{w}$ e um desvio b. *Figura adaptada de [22]*

Seja d_+ (d_-) a distância mais curta do hiperplano separador à observação positiva (negativa) mais próxima. A margem do hiperplano separador será definida por $d_+ + d_-$. O algoritmo de suporte vectorial procura o hiperplano que maximiza esta margem.

Este problema pode ser formulado da seguinte forma: suponha-se que as observações satisfazem as condições[1]

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \quad para \quad y_i = +1 \tag{2.1}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \quad para \quad y_i = -1 \tag{2.2}$$

Estas duas condições podem ser descritas numa só:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \forall i \tag{2.3}$$

Sejam H_1 e H_2 os hiperplanos definidos pelos pontos para os quais as inequações (2.1) e (2.2) verificam a igualdade, isto é,

$$H_1 : \mathbf{x}_i \cdot \mathbf{w} + b = 1$$

com distância perpendicular à origem $|1 - b|/||\mathbf{w}||$ e

$$H_2 : \mathbf{x}_i \cdot \mathbf{w} + b = -1$$

com distância perpendicular à origem $| - 1 - b|/||\mathbf{w}||$. Ambos têm como vector normal $\mathbf{w}$. Assim, $d_+ = d_- = 1/||\mathbf{w}||$ e portanto, a margem é $2/||\mathbf{w}||$ (fig. 2.2). Note-se que H_1 e H_2 são paralelos (têm o mesmo vector normal) e que não existem

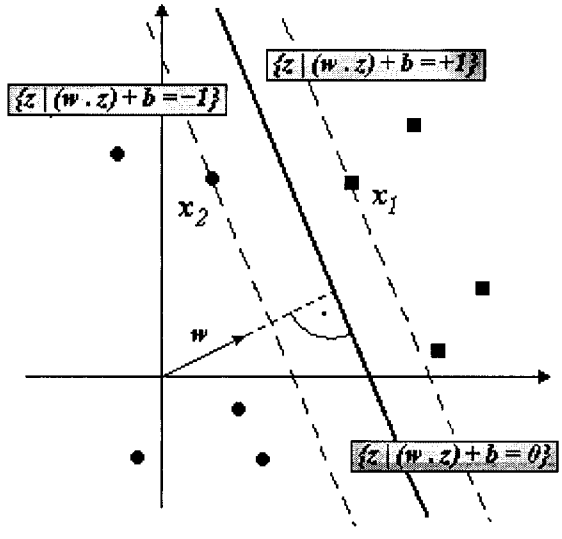

Figura 2.2: Um hiperplano separador com margem $2/||\mathbf{w}||$. *Figura adaptada de [22]*

pontos de treino entre eles. Então o problema de encontrar o par de hiperplanos que maximiza a margem resume-se a um problema de optimização. A introdução de uma estrutura do tipo 1.6 no conjunto de funções de hiperplanos é possível graças ao seguinte resultado [24]:

[1]É comum formular-se um problema de SVMs com estas condições[12]. O facto de ser usar a constante 1 para definir os hiperplanos em nada restringe o problema. Poderia usar-se uma qualquer constante γ desde que para isso se estabelecesse $||\mathbf{w}|| = 1$.

Proposição 2.1 ***(V. Vapnik)*** *Seja R o raio da bola mais pequena*

$$B_R(\mathbf{a}) = \{x \in X : ||\mathbf{x} - \mathbf{a}|| < R\} \ (\mathbf{a} \in X)$$

que contém os pontos $\mathbf{x}_1, \ldots, \mathbf{x}_r$, e sejam

$$f_{\mathbf{w},b} = sgn((\mathbf{w} \cdot \mathbf{x}) + b) \tag{2.4}$$

hiperplanos canónicos que representam funções de decisão para esses pontos. Então o conjunto $\{f_{\mathbf{w},b} : ||\mathbf{w}|| \leq A\}$ tem dimensão de VC igual a h que satisfaz

$$h < R^2A^2 + 1. \tag{2.5}$$

Note-se que, devido à proporcionalidade inversa da margem e $||\mathbf{w}||$, o resultado anterior mostra que estabelecendo para a margem um limite inferior maior (i.e. um A pequeno), obtem-se uma dimensão de VC mais baixa. Inversamente, permitindo uma separação com uma margem mais pequena, pode-se potencialmente separar uma maior classe de problemas (i.e. aumenta o número de diferentes classificações do conjunto de treino) [22]. Tendo em conta a que Eq.(1.5) aconselha a manter tanto o erro de treino como a dimenção de VC baixa de forma a obter uma máquina com uma boa capacidade de generalização, conclui-se que as funções de decisão hiperplanos devem ser construídas de forma a maximizar a margem, e que ao mesmo tempo separem os dados de treino tanto quanto possível.

2.1.2 Algoritmo de Margem Óptima

A tarefa de encontar o hiperplano separador óptimo resume-se a um problema de optimização convexo: minimizar uma função quadrática restricto a um conjunto de inequações lineares. Assim, tendo em conta as Eqs. (2.1) e (2.2), o problema pode ser formulado da seguinte forma:

$$\begin{aligned} &\min \ ||\mathbf{w}||^2 \\ &\text{s.a. } y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \ \ \forall i, \ i = 1, \ldots, l \end{aligned} \tag{2.6}$$

Os vectores de suporte são os pontos que verificam a igualdade na inequação e cuja remoção provoca uma alteração da solução.

Por ser um problema convexo, a resolução de (2.6) é feita recorrendo aos multiplicadores de Lagrange. Esta reformulação do problema traz vantagens: primeiro, porque simplifica o manuseamento das restrições do problema, uma vez que passam a ser apenas sobre os multiplicadores de Lagrange; segundo, porque nesta nova formulação, os dados de treino aparecem apenas na forma de produto escalar entre vectores. Esta é uma propriedade crucial que permite generalizar para o caso não linear.

Introduzindo então os multiplicadores de Lagrange[2] $\alpha_i \geq 0$, $i = 1, \ldots, l$, um para cada uma das restrições na Eq.(2.3), e procedendo às devidas transformações (a

[2]Para restrições do tipo $c_i \geq 0$ a regra é multiplicar as equações das restrições por multiplicadores de Lagrange *positivos* e subtraí-los à função objectivo.

nova função objectivo é dada pela diferença entre a função objectivo anterior e o produto das respectivas restrições com os mutiplicadores de Lagrange), tem-se o primal Lagrangiano:

$$L_P \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{l} \alpha_i \tag{2.7}$$

O dual de (2.7) é obtido derivando L_P relativamente a $\mathbf{w}$ e b e igualando essas derivadas a zero:

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \Longrightarrow \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{2.8}$$

$$\frac{\partial L_P}{\partial b} = -\sum_i \alpha_i y_i = 0 \tag{2.9}$$

Substituindo as Eqs.(2.8) e (2.9) na expressão do primal, obtem-se o dual Lagrangiano:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{2.10}$$

Ora, minimizar o primal é equivalente a maximizar o dual, impondo (2.9) e que os multiplicadores de Lagrange sejam não negativos. Assim, o problema a resolver é:

$$\begin{array}{l} \max\ L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{s.a. } \sum_i \alpha_i y_i = 0 \\ \qquad \alpha_i \geq 0,\ i = 1, \ldots, l \end{array} \tag{2.11}$$

A existência de solução num problema de optimização convexo é garantido pelas condições de Karush-Kuhn-Tucker [6]. Assim, um ponto que verifique as condições de KKT (com $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ e $\mathbf{w} = (w_1, w_2, \ldots, w_d)$):

$$\frac{\partial L_P}{\partial w_v} = w_v - \sum_i \alpha_i y_i x_{iv} = 0 \quad v = 1, \ldots, d \tag{2.12}$$

$$\frac{\partial L_P}{\partial b} = -\sum_i \alpha_i y_i = 0 \tag{2.13}$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \qquad \forall i \tag{2.14}$$

$$\alpha_i \geq 0 \qquad \forall i \tag{2.15}$$

$$\alpha_i (y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1) = 0 \qquad \forall i \tag{2.16}$$

é solução do problema (2.6). Ou seja, resolver um problema de suporte vectorial é equivalente a encontrar solução das condições de KKT.

Tem-se assim que a solução é da forma:

$$\mathbf{w} = \sum_i^l \alpha_i y_i \mathbf{x}_i \tag{2.17}$$

com coeficientes α_i diferentes de zero para os vectores de suporte, isto é, para as observações i que verificam a igualdade na Eq.(2.3). Esses valores de α_i são determinados durante a resolução do problema de maximização. O valor de b é obtido a partir da condição de KKT (2.16).

Uma vez estimados os parâmetros que definem o hiperplano óptimo, constrói-se a função de decisão:

$$f(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.18}$$

Assim, dado um novo indivíduo $\mathbf{x}$, a classe que lhe é atribuída é 1 ou -1 dependendo de $f(\mathbf{x})$ ser positiva ou negativa, respectivamente. De acordo com a condição (2.16), os vectores de suporte ($\mathbf{x}_i$ para os quais $\alpha_i > 0$), encontram-se exactamente na margem. As restantes observações do conjunto de treino são irrelevantes: a condição imposta pela Eq.(2.3) é satisfeita automaticamente, e não aparecem na expansão (2.17). Isto conduz a um limite superior da capacidade de generalização dos hiperplanos de margem óptima: suponha-se que usamos o método *leave-one-out* para estimar o erro de teste. Se se deixar de fora a observação $\mathbf{x}_{i^*}$ e se construir a solução com as restantes observações pode acontecer uma de quatro possibilidades (cf. Eq.(2.3)):

1. $y_{i^*} \cdot ((\mathbf{x}_{i^*} \cdot \mathbf{w}) + b) > 1$, i.e., o objecto é classificado correctamente e não está dentro da margem. Este é o tipo de observações que nunca se transformarão em vectores de suporte [22].

2. $y_{i^*} \cdot ((\mathbf{x}_{i^*} \cdot \mathbf{w}) + b) = 1$, i.e., $\mathbf{x}_{i^*}$ verifica Eq.(2.3). Neste caso, a solução $\mathbf{w}$ não muda, apesar de os coeficientes α_i na formulação dual do problema de optimização poderem mudar: aliás, $\mathbf{x}_{i^*}$ pode tornar-se num vector de suporte (i.e, $\alpha_{i^*} > 0$) se se mantiver no conjunto de treino.

3. $1 > y_{i^*} \cdot ((\mathbf{x}_{i^*} \cdot \mathbf{w}) + b) > 0$, i.e., $\mathbf{x}_{i^*}$ encontra-se dentro da margem, mas do lado correcto da fronteira de decisão. Neste caso, a solução é diferente da obtida caso $\mathbf{x}_{i^*}$ estivesse incluído no conjunto de treino (aí, $\mathbf{x}_{i^*}$ iria satisfazer Eq.(2.3) depois do treino), contudo, a classificação é correcta.

4. $y_{i^*} \cdot ((\mathbf{x}_{i^*} \cdot \mathbf{w}) + b) < 0$, neste caso $\mathbf{x}_{i^*}$ está mal classificado.

Note-se que os casos 3 e 4 correspondem a exemplos que se transformariam em vectores de suporte caso fossem incluídos no conjunto de treino; o caso 2 incluí potencialmente esses casos. Contudo, apenas o caso 4 conduz a erro no procedimento *leave-one-out*. Consequentemente tem-se o seguinte resultado para a generalização de classificadores de margem óptima [24]:

Proposição 2.1 ***(Vapnik and Chervonenkis)*** *O número de vectores de suporte obtidos durante o treino num conjunto de tamanho l, dividido por $l - 1$, é um limite superior para o erro de teste esperado.*

2.1.3 Hiperplanos com margem amaciada

Na realidade, é muito improvável que duas classes sejam separáveis por um hiperplano no seu espaço original. Quando isto não acontece, irão existir erros. Assim, o caso não separável é tratado de forma idêntica ao separável mas introduzindo uma penalização às observações que se encontram do lado errado do hiperplano. Os hiperplanos separadores nestas condições são conhecidos por hiperplanos com margem amaciada. No

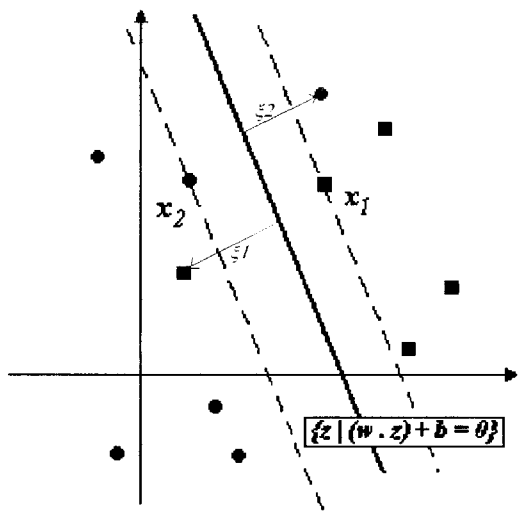

Figura 2.3: Um hiperplano separador com custos (ξ) associados às observações mal classificadas.

caso separável, a definição dos hiperplanos de suporte (H_1 e H_2) foi feita de forma a separar completamente as duas classes. No caso não separável permite-se que existam pontos no lado errado da margem, sendo-lhes associado um custo $\xi_i \geq 0$, $i = 1, \ldots, l$ (figura 2.3).

Tem-se então uma nova definição destes hiperplanos de suporte:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i \quad para \quad y_i = +1 \tag{2.19}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + \xi_i \quad para \quad y_i = -1 \tag{2.20}$$

$$\xi_i \geq 0 \quad \forall i \tag{2.21}$$

Adicionou-se assim o custo de errar às restrições do problema no caso separável. É necessário ainda incluir estes custos na função objectivo.

Para que exista um erro, é necessário que o respectivo ξ_i seja maior do que 1, assim $\sum_i \xi_i$ é um limite superior para o número de erros de treino; obviamente pretende-se minimizar este número. Logo, modifica-se a função objectivo a minimizar para

$$||\mathbf{w}||^2/2 + C \sum_i \xi_i \tag{2.22}$$

onde C é um parâmetro a ser escolhido pelo utilizador; quanto maior for C, maior será a penalização associada aos erros.

Procedendo de forma análoga ao caso separável, isto é, reescrever o problema no primal Lagrangiano:

$$L_P = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i - \sum_i \alpha_i \{y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} - \sum_i \mu_i \xi_i \tag{2.23}$$

onde $\alpha_i \geq 0$ são os multiplicadores de Lagrange e $\mu_i \geq 0$ os multiplicadores de Lagrange introduzidos para forçar a positividade de ξ_i[3].

Determina-se o respectivo dual igualando as derivadas em ordem a w, ξ e b, e substituindo estas relações em (2.23):

$$\begin{array}{l} \max\ L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \\ \text{s.a.}\ \ 0 \leq \alpha_i \leq C \\ \quad \sum_i \alpha_i y_i = 0 \end{array} \tag{2.24}$$

Novamente a solução é dada por:

$$\mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \mathbf{x}_i \tag{2.25}$$

onde N_S é o número de vectores de suporte.

As condições de Karush-Kunhn-Tucker para o problema primal permitem a determinação de b. As condições de complementaridade de KKT são:

$$\alpha_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} = 0\ \forall i \tag{2.26}$$

e são estas as que permitem o cálculo de b.

2.2 Máquinas de Suporte Vectorial

Os casos tratados até agora dão conta de como se constrói uma fronteira linear no espaço original. Em problemas reais, é quase certo que fronteiras lineares não são capazes de separar completamente as duas classes. A fronteira obtida com métodos lineares concerteza que classifica um maior número de objectos de forma errada o que conduz a um erro mais elevado.
A introdução de funções *kernel* permite construir fronteiras mais flexíveis uma vez que transforma o espaço inicial num novo espaço, de dimensão superior ao original, onde será possível construir uma fronteira linear que separe as duas classes. No espaço original tem-se então uma fronteira não linear.

2.2.1 Máquinas de suporte vectorial não lineares

Note-se que no problema (2.24) as observações de treino aparecem apenas na forma de produto interno $\mathbf{x}_i \cdot \mathbf{x}_j$. Substituindo esse produto interno por funções *kernel*, transformam-se o espaço original num espaço de dimensão superior (eventualmente infinito). Ora, nesse espaço já é possível construir uma fronteira linear usando a metodologia descrita nas secções anteriores, como se verá de seguida.

Suponha-se então que se transforma o espaço inicial num espaço euclidiano de dimensão superior Y através da função $\phi : \mathbf{R}^d \longrightarrow Y$. Então, o algoritmo de aprendizagem depende apenas do produto interno em Y, isto é, de $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$. Assim, se existir uma função *kernel* K tal que

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$$

o algoritmo de aprendizagem passa a usar apenas $K(\mathbf{x}_i, \mathbf{x}_j)$, sem necessidade de explicitar ϕ.
No novo espaço, a função a maximizar será da forma:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{2.27}$$

que, analogamente ao processo apresentado na secção anterior, tem como solução:

$$\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i) \tag{2.28}$$

Tem-se assim a solução no espaço Y. No entanto pretende-se saber como usar a máquina no espaço de dimensão original. Ora, a função de decisão no espaço original tem de incluir as funções *kernel* usadas para aumentar a dimensão do espaço.
Assim, tem-se uma função de decisão não linear:

$$f(\mathbf{x}) = sgn\left(\sum_{i=1}^{N_S} \alpha_i y_i \phi(\mathbf{s}_i) \cdot \phi(\mathbf{x}) + b\right) = sgn\left(\sum_{i=1}^{N_S} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b\right) \tag{2.29}$$

onde $\mathbf{s}_i$ são os vectores de suporte.
Para calcular b, tem-se em conta que dadas as Eqs.(2.19) e (2.20), para vectores de suporte $\mathbf{x}_j$ para os quais $\xi_j = 0$, tem-se

$$\sum_{i=1}^{l} y_i \alpha_i \cdot k(\mathbf{x}_j, \mathbf{x}_i) + b = y_j.$$

Então, o desvio pode ser calculado avaliando

$$b = y_j - \sum_{i=1}^{l} y_i \alpha_i \cdot k(\mathbf{x}_j, \mathbf{x}_i)$$

para todos os vectores de suporte $\mathbf{s}_i$.
Esta transformação dos dados traz vantagens na fase de teste da máquina uma vez que torna o processo mais rápido.

Na figura (2.4) pode ver-se um exemplo binário simples de uma fronteira não linear obtida com máquina de suporte vectorial.

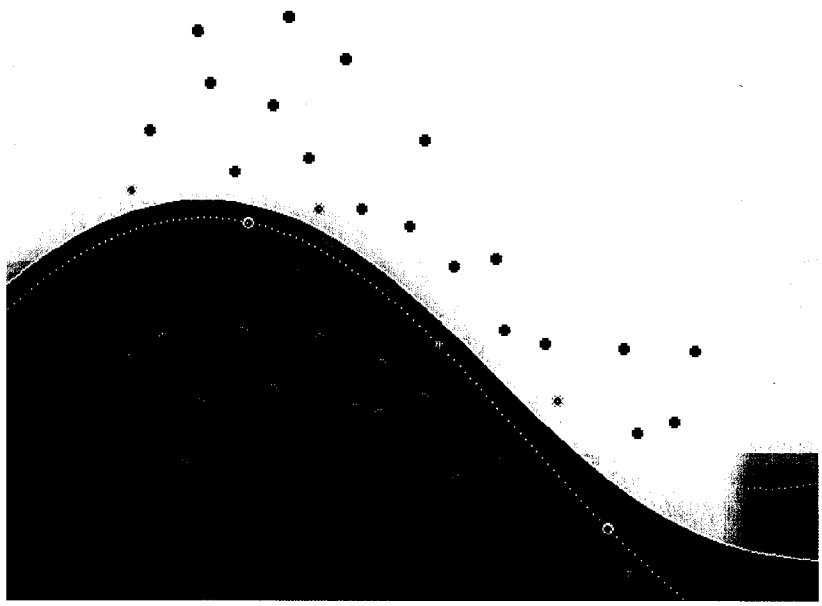

Figura 2.4: Exemplo de um classificador de suporte vectorial determinado usando uma função *kernel* RBF $k(\mathbf{x}, \mathbf{y}) = exp(-||\mathbf{x} - \mathbf{y}||^2)$.

2.2.2 Classificação Multi-Classe

As máquinas de suporte vectorial descritas anteriormente para problemas de 2 classes podem ser extendidas para o caso de k classes, com $k > 2$.

Uma forma de expandir este método é construir $k(k-1)/2$ classificadores, um para cada par de classes diferente - método *"one-against-one"*. Cada classificador resolve um problema de classificação binária, cada resultado dessa classificação é considerada um voto. O classificador final é o número de votos, isto é, uma observação $\mathbf{x}$ vai pertencer à classe que tem maior número de votos. Se houver empate, escolhe-se a classe de maior índice. A primeira utilização desta estratégia foi feita por Krebel [14].

Um outro método, *"one-against-all"*, para classificação com $k > 2$ é construir k modelos SVM cada um deles treinado para separar uma classe das restantes. Dado um objecto a classificar, atribui-se a classe que maximiza a função de decisão (Eq. 2.29) antes de lhe aplicar a função *sgn*, [22]. Platt [17] mostrou que o método *"one-against-one"* tem melhores resultados do que este último.

Neste trabalho foi usado o método *"one-against-one"* uma vez que é este o que está implementado no *R-project* (o software usado na construção das SVMs). Tendo em conta que na bibliografia este método é referido como sendo o que tem melhores resultados, não se optou por usar outro.

2.2.3 Complexidade das Máquinas de Suporte Vectorial

Na fase de treino. Uma propriedade muito importante das SVMs é que tanto as funções de treino, como a de teste dependem apenas dos dados a partir das funções *kernel* $K(\mathbf{x}_i, \mathbf{x}_j)$. Apesar desta função ser um produto escalar no espaço Y, o que implicaria muita complexidade na fase de cálculos, o cálculo de K pode-se tornar bastante simples. Por exemplo, o cálculo de $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p$, um produto interno que em Y necessitaria de operações de ordem $\binom{dx+p-1}{p}$; em $\mathbf{R}^d$ necessita apenas de operações na ordem de d, onde d é a dimensão do espaço original dos dados.
Assim, as SVMs ultrapassam o problema da maldição da dimensão: a proliferação dos parâmetros que provoca o aumento da complexidade e o *"overfitting"*.

Na fase de teste. Nesta fase basta calcular o valor da função de decisão (2.29) que requer operações na ordem de MN_S, onde N_S é o número de vectores de suporte e M é o número de operações necessárias para para calcular a função *kernel*. Para funções *kernel* RBF, M é $O(d)$, a dimensão dos vectores de dados [3].

Capítulo 3

Aplicação de SVMs a Imagens Multi-Espectrais

Neste capítulo é apresentada a aplicação do algoritmo de máquinas de suporte vectorial a imagens multi-espectrais. É feita uma classificação sobre os objectos resultantes da segmentação de uma imagem do sensor ASTER no lugar de uma classificação por pixels. A substituição de pixels por objectos, tem a ver com a necessidade de tornar o processo mais rápido e mais adequado ao objectivo desta classificação: o estudo do tipo de cobertura do solo da zona do Vale de Sousa.

De forma a tentar diminuir ainda mais o tempo de processamento da classificação, foram aplicados dois métodos de redução dos dados: componentes principais e variáveis canónicas.

3.1 A classificação de imagens multi-espectrais

Uma imagem multi-espectral é um conjunto de imagens obtidas simultaneamente num determinado número de bandas (secções ou faixas) do espectro electromagnético[1]. Cada pixel mede a luminosidade nas bandas do espectro electromagnético. Sendo assim, cada pixel é representado por um vector multidimensional, em que cada uma das suas componentes corresponde à intensidade da radiação numa determinada banda do espectro (figura 3.1).

A classificação de imagens é uma das ferramentas mais poderosas no processamento de imagens digitais multiespectrais. Permite que se converta um grande número de dados das bandas espectrais num produto de grande valor que nos dá a indicação do tipo de cobertura de cada pixel do local em estudo. A classificação de imagens é o processo de criar mapas de classes (thematic maps) a partir das imagens de satélite.

[1]No ApêndiceA encontra-se um breve resumo acerca do espectro electromagnético e da informação que se pode retirar de imagens multi-espectrais.

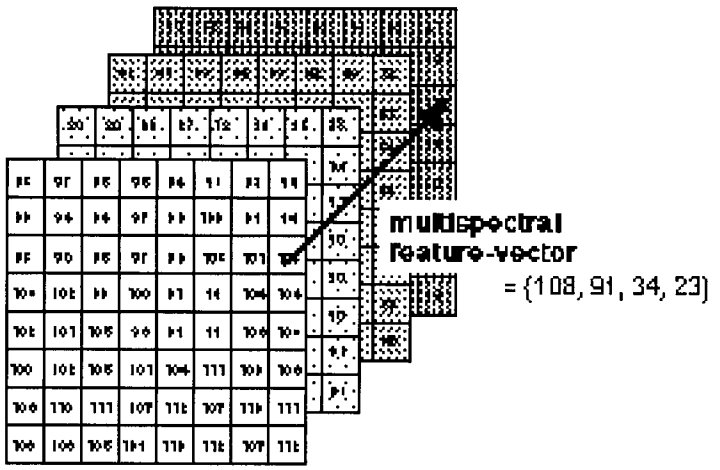

Figura 3.1: Ilustração de uma imagem multi-espectral (*fonte: MicroImages, Inc. TNTmips Reference Manual*).

Um mapa de classes (thematic map) é uma representação informativa de uma imagem que mostra a distribuição espacial de uma classe particular.

As classes podem ser tão diversificadas como as suas áreas de interesse. Alguns exemplos de classes são solo, vegetação, profundidade da água e atmosfera. Dentro de uma classe podem estar definidas subclasses, e assim é necessário tornar o processo de classificação mais refinado.

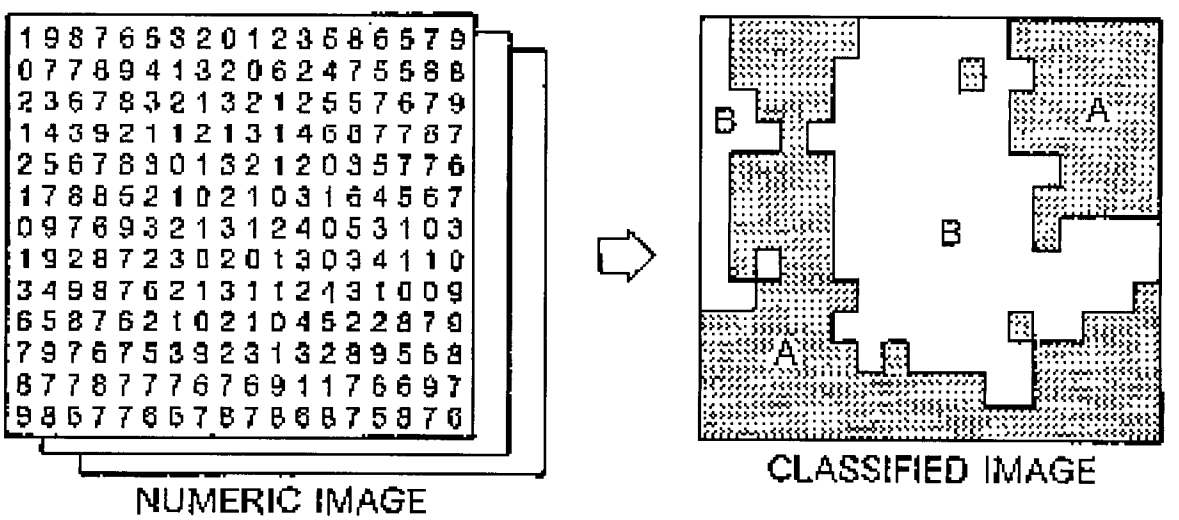

Figura 3.2: Classificação de uma imagem multiespectral (adaptada de [4]).

Habitualmente a classificação é feita com base na informação do pixel. Neste caso, o objectivo do processo de classificação é categorizar todos os pixels duma imagem numa das diversas classes. Estes dados categorizados podem então ser usados para produzir mapas temáticos (ou de classes) dos tipos de cobertura presentes na imagem.

O objectivo da classificação de imagens é identificar e representar, com um único nível de cinzento (ou cor), determinadas características que numa imagem definem um objecto ou um tipo de cobertura.

3.2 Segmentação de Imagens

A classificação automática de imagens multi-espectrais de satélites de observação da Terra é uma acção fundamental para a criação e actualização de informação geográfica. Com o desenvolvimento de sensores com melhores resoluções espaciais o problema de pixels de mistura diminui, mas a variabilidade interna e o ruído dentro das classes de ocupação do solo aumenta [21]. Como consequência, a classificação tradicional com base na informação de pixel gera demasiadas classes ou classes não muito bem definidas porque os seus *"clusters"* são definidos tendo em conta apenas a homogeneidade espectral.
Uma nova abordagem do problema de classificação é a classificação orientada a objectos que tem a ver com a segmentação da imagem em objectos ou segmentos. Um objecto pode ser visto como um conjunto de pixels vizinhos com características espectrais semelhantes. Uma das motivações para esta abordagem/técnica é o facto de o resultado esperado de muitas tarefas de processamento de imagens ser a extracção tanto da forma como da classificação de objectos do mundo real.

A cada tarefa de classificação está associada uma escala. É portanto, importante que a resolução média dos objectos da imagem possa ser adaptada à escala de interesse. A informação da imagem pode ser representada em diferentes escalas baseadas no tamanho médio dos objectos. A mesma imagem pode ser segmentada em objectos maiores ou menores, com impacto considerável em praticamente toda a informação que pode ser retirada dos objectos. Antes de avançar, note-se a diferença entre escala e resolução: resolução expressa o tamanho médio de área que um pixel cobre no terreno, escala descreve a magnitude ou nível de abstracção na qual um determinado fenómeno pode ser descrito [2]. Assim, estudar a imagem a partir de diferentes escalas em vez de diferentes resoluções pode facilitar a sua análise.

A abordagem orientada a objectos é, em princípio, independente das técnicas de segmentação e classificação. Contudo, a escolha acertada dos métodos de processamento pode melhorar os resultados, e os métodos de treino e classificação correctos podem dar ao utilizador grandes vantagens sobre o potencial desta abordagem.

Segmentação. Segmentação é a subdivisão de uma imagem em regiões separadas. Uma forte motivação para desenvolver técnicas de extracção de objectos de imagens advém do facto que muitas imagens exibem características ligadas à textura que são negligenciadas nos métodos de classificação baseados na informação de pixel. Existem muitos métodos de segmentação de imagens, cada um com as suas vantagens e desvantagens. Uns são completamente automáticos e outros semi-automáticos. Os métodos

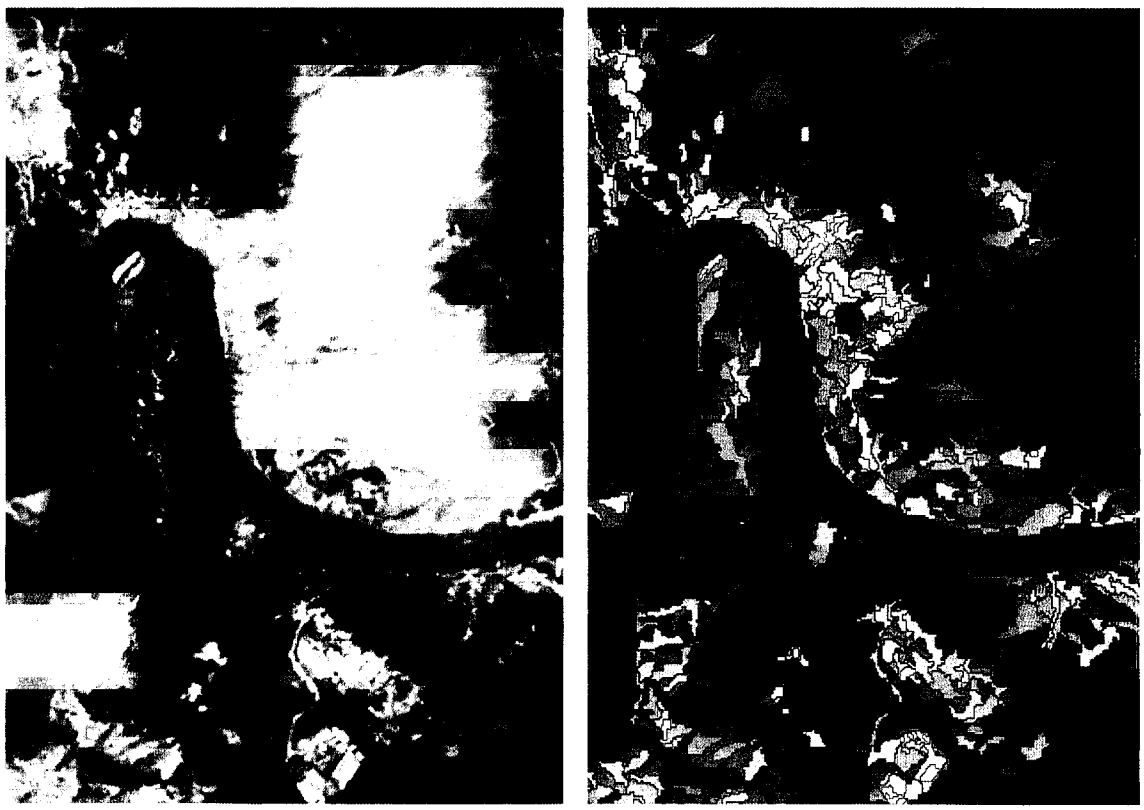

Figura 3.3: Uma parte da imagem original (composição RGB de 3 bandas do ASTER, à esquerda) e à direita a segmentação dessa imagem.

de segmentação de imagens podem ser divididos em dois grupos: métodos orientados pelo conhecimento (*"top-down"*) vs. métodos orientados pelos dados (*"bottom-up"*). Na abordagem *"top-down"* o utilizador já conhece quais os objectos que quer extrair da imagem, mas não sabe como fazê-lo. O sistema tenta encontrar o melhor método de processamento de imagem para os extrair formulando um modelo para esses objectos. Na abordagem *"bottom-up"* os segmentos são gerados com base num conjunto de métodos e parâmetros estatísticos para processar toda a imagem. Neste último caso, tal como nos métodos de "*clustering*", os segmentos gerados inicialmente não têm qualquer significado. Cabe ao utilizador determinar que tipo de objectos reais a imagem de objectos representa. A principal diferença entre estes dois métodos é que o *"top-down"* conduz a resultados locais porque apenas marca os pixels ou regiões que verificam as condições do modelo, por sua vez o método *"bottom-up"* segmenta toda a imagem, ou seja, agrupa pixels em "*clusters*" que verificam certos critérios de homogeneidade e heterogeneidade [9].

Neste trabalho o método de segmentação usado foi o *"bottom-up"*. Este processo começa num nível inicial onde um objecto é um pixel. Em passos consequentes, objectos menores são agregados em outros maiores. A área representada por um determinado objecto é definida pela soma das áreas dos seus sub-objectos. Introduzindo a noção de sub-objectos tem-se que cada nível é construído com base nos seus sub-objectos directos, isto é, os sub-objectos fundem-se em objectos maiores no

próximo nível. A segmentação resulta então numa estrutura hierárquica de objectos que representa a informação da imagem em diferentes resoluções espaciais (níveis). O número de objectos em cada nível é inversamente proporcional à dimensão dos objectos. Quanto mais refinada for a segmentação, menor será a dimensão dos objectos ao passo que o seu número será maior.

3.3 Descrição do problema

3.3.1 Os dados

Neste trabalho foi usada uma imagem multi-espectral da zona do Vale do Sousa com 2060x3340 pixels e nove bandas proveniente do sensor ASTER. Esta imagem foi segmentada usando o *eCognition*[9]: uma ferramenta de análise de imagens de satélite orientada a objectos. A segmentação foi feita em 5 níveis. Na tabela 3.1 estão apresentadas as características de cada um dos diferentes níveis: o número e tamanho[2] médio dos objectos que os compõem.

Nível:	1	2	3	4	5
No.Objs:	51186	14133	4857	2269	651
Tam.Médio (pxs):	134.92	486.83	1416.59	3032.35	10568.97

Tabela 3.1: Os diferentes níveis de segmentação.

Tem-se assim que a segmentação no nível 1 é a mais refinada, e foi esta a usada neste trabalho (fig. 3.3). Note-se na drástica redução do número de objectos a classificar (6880400 seria o número de dados a considerar no caso de classificação com base no pixel). Esta é uma das grandes vantagens da segmentação de imagens: a redução do número de indivíduos a classificar. No entanto, com a segmentação perde-se alguma informação espectral. Isto é, numa classificação com base no pixel tem-se em conta a informação espectral em cada pixel, com a segmentação essa informação é absorvida no novo objecto. A informação espectral disponível passa a ser a média de todos os pixels que compõem o novo objecto em cada uma das bandas espectrais.
Como resultado da segmentação o *eCognition* fornece várias características dos novos objectos. Essas características vão servir como novas variáveis no processo de classificação da imagem segmentada. Existem dois grupos de características: as do próprio objecto e as relacionadas com a classe dos seus vizinhos. Neste contexto, os objectos vizinhos são aqueles adjacentes a um determinado objecto no mesmo nível. Uma vez que os objectos de treino estão dispersos na imagem, isto é, não são vizinhos entre si, não faz sentido usar a informação dos vizinhos. Neste trabalho vai ter-se em conta

[2]O tamanho de um objecto é o número de pixels que o compõe.

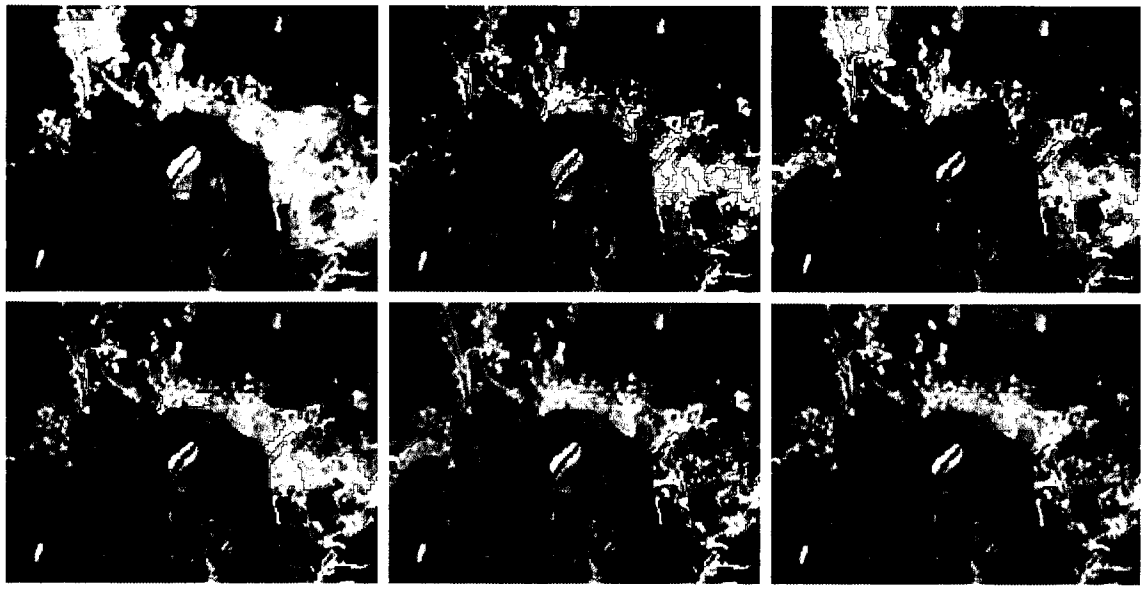

Figura 3.4: Um pormenor da imagem em vários níveis de segmentação. Em cima à esquerda a imagem por segmentar; em cima ao centro, o primeiro nível de segmentação; em cima à direita, o segundo nível; em baixo à esquerda, terceiro nível; em baixo ao centro, quarto nível; e finalmente, em baixo à direita o quinto nível de segmentação.

apenas as características do próprio objecto. As características de um objecto podem dividir-se em quatro grupos:

- **Valores espectrais** - Estas características dizem respeito aos valores dos pixels de um objecto em cada canal espectral.
- **Forma** - Com estas características a forma de um objecto pode ser descrita usando o próprio objecto ou os seus sub-objectos (usar-se-á apenas as relativas ao próprio objecto)
- **Textura** - Estas características avaliam a textura de um objecto baseando-se nos seus sub-objectos.
- **Hierarquia** - As características de hierarquia fornecem informação acerca dos objectos envolventes de um determinado objecto na hierarquia dos objectos da imagem.

Mais uma vez, não será usada a informação disponibilizada pelo *eCognition* relativa aos objectos vizinhos nem a informação relativa aos sub-objectos. Assim, os atributos a serem usados dizem respeito aos valores espectrais do próprio objecto e à sua forma. Quanto aos primeiros, tomou-se em conta a média espectral de cada objecto (a média da resposta espectral dos pixels de um determinado objecto) em cada uma das bandas e respectivos desvios padrões. Daqui resultam então 18 variáveis (9 médias e 9 desvios padrões). Os valores das médias variam entre 0 e 255.
Quanto às características que descrevem a forma de um objecto, existem várias a

considerar: área, comprimento, largura, razão comprimento/largura, comprimento da fronteira, índice da forma, densidade, direcção principal, assimetria, e outros. No entanto, não é necessário usar toda esta informação, isto porque muitos destes atributos têm informação redundante. Asim sendo, considerou-se apenas o atributo densidade. A escolha deste atributo deve-se ao facto de se considerar que este contém informação suficiente acerca do objecto; inclusivé outras características já enumeradas acabam por estar implícitas neste atributo. Para além disso, testou-se a inclusão de algumas dessas características e o resultado não foi vantajoso. A densidade d de um objecto descreve a compacidade de um objecto. Pode ser expressa pela área coberta pelo objecto dividida pelo seu raio. O *eCognition* usa a seguinte implementação [9]:

$$d = \frac{\sqrt{n}}{1 + \sqrt{Var(X) + Var(Y)}}$$

onde n é o número de pixels que formam o objecto; X e Y são o conjunto de coordenadas em x e y, respectivamente, que formam o objecto. Quanto mais próxima de um quadrado for a forma de um objecto, maior é a sua densidade. Os valores deste atributo são positivos e variam com a forma do objecto.

Estas passam a ser as 19 variáveis a considerar no problema de classificação: as nove médias espectrais dos pixels que formam o objecto em cada uma das 9 bandas espectrais; os 9 respectivos desvios padrões; e a densidade de cada objecto. Inicialmente foram incluídos outros atributos dos objectos relativos tanto às características espectrais como à sua forma mas considerou-se que a inclusão deles não trazia melhorias significativas à classificação, alguns chegavam até a ser prejudiciais uma vez que "confundiam" a máquina de aprendizagem.

3.3.2 As classes

Com estes dados pretende-se estudar a composição da cobertura do solo da região do Vale do Sousa. Para tal, a Associação Florestal do Vale do Sousa estabeleceu algumas classes de ocupação do solo definindo uma estratificação com base nas Normas de Estratificação e Fotointerpretação utilizadas no inventário florestal nacional. Resumidamente, a estratificação pode ser feita baseada em critérios hierarquicamente relacionados que reflectem: a utilização do solo (Nível I); as ocupações principal e secundária (Nível II); e quando necessário, a caracterização adicional das ocupações principal e secundária (Nível III).
Neste trabalho serão consideradas nove classes. Cada uma dessas classes tem identificada, na imagem multi-espectral, áreas de treino que foram definidas pela Associação Florestal do Vale do Sousa (ver fig. 3.5). Tem-se assim um problema de classificação supervisionada. As classes a considerar são: Social (SC), Água (HH), Fogo (FG), Floresta Mista (FLmista), Floresta de Folhosas Diversas (FLFdFd), Floresta de Pinheiros Bravos (FLPbPb), Floresta de Eucaliptos (FLEcEc), Inculto (IC), Agrícola (AG). Note-se que quatro das nove classes são vários tipos de floresta. Este é um facto que deve tornar a clasificação mais difícil, isto porque é obvio que será mais complicado

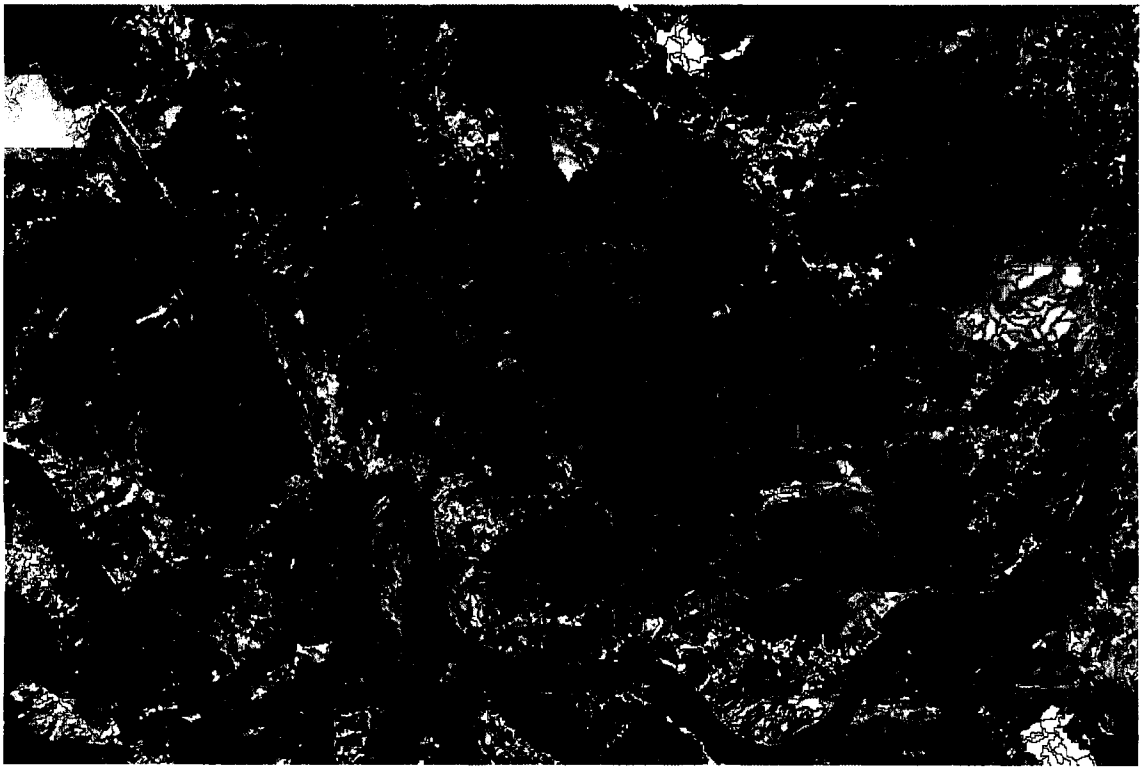

Figura 3.5: Um pormenor da imagem com áreas de treino.

distinguir estas classes. Certamente que este problema já não acontecerá com a classe HH uma vez que é bem definida espectralmente. Para melhor avaliar a separabilidade entre as classes recorreu-se à distância de Jeffries-Matusita (JM) [19].

Distância de Jeffries-Matusita (JM). A distância JM entre um par de distribuições probabilísticas (classes espectrais) é definida por

$$J_{ij} = \int_x \left\{ \sqrt{p(x|\omega_i)} - \sqrt{p(x|\omega_j)} \right\}^2 dx$$

que é uma medida da distância média entre duas funções de densidade. Para classes normalmente distribuídas tem-se que:

$$J_{ij} = 2(1 - e^{-B})$$

onde

$$B = \frac{1}{8}(m_i - m_j)^t \left\{ \frac{\Sigma_i + \Sigma_j}{2} \right\}^{-1} (m_i - m_j) + \frac{1}{2} ln \left\{ \frac{|(\Sigma_i + \Sigma_j)/2|}{|\Sigma_i|^{1/2}|\Sigma_j|^{1/2}} \right\}$$

é a distância de Bhattacharyya. Se se representar a distância de JM como função da distância entre médias de classes ela mostra um comportamento de saturação para 2(fig. 3.6). A distância de JM varia entre 0 e 2. Uma distância de 2 entre duas classes espectrais implica a classificação de um pixel numa dessas duas classes (assumindo que eram apenas duas) com 0% de erro.

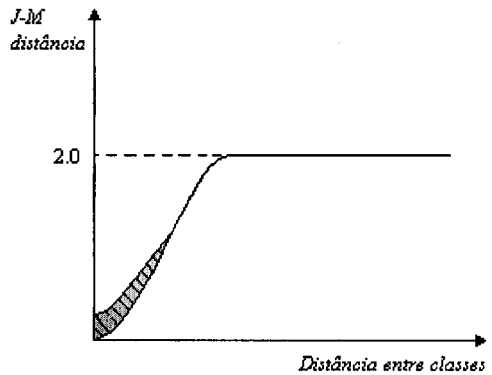

Figura 3.6: Distância de Jeffries-Matusita como função da separação das médias espectrais das classes. *Figura adaptada de [19]*

	SC	HH	FG	FLEcEc	FLmista	FLFdFd	FLPbPb	IC
HH	2.00							
FG	1.68	2.00						
FLEcEc	1.95	2.00	1.95					
FLmista	1.89	2.00	1.88	0.42				
FLFdFd	1.85	2.00	1.91	1.25	0.84			
FLPbPb	1.90	2.00	1.82	1.49	1.02	1.63		
IC	1.82	2.00	1.75	1.40	0.96	1.06	1.36	
AG	1.57	2.00	1.91	1.92	1.74	1.58	1.88	1.71

Tabela 3.2: As distâncias de Jeffries-Matusita entre as classes SC, HH, FG, FLEcEc, FLmista, FLFdFd, FLPbPb, IC e AG.

A tabela 3.2 mostra os valores da distância de Jeffries-Matusita para todos os pares das nove classes, pressupondo a normalidade das classes. Como se pode verificar, a classe HH é facilmente separável das restantes classes. Como já se esperava, o mesmo não se passa com as classes Flmista, FLFdFd, FLPbPb e FLEcEc, isto porque são classes com comportamento espectral muito próximo (fig.3.7). Todas são subclasses de floresta daí que se poderiam agrupar todas estas classes numa só. Provavelmente isto melhoraria os resultados da classificação, mas fazendo essa agregação o objectivo inicial deixaria de fazer sentido. Relembre-se que o que se pretende é identificar diferentes tipos de cobertura do solo, o que inclui diferentes tipos de vegetação. No entanto, por uma questão de curiosidade, calculou-se novamente as distância de JM para seis classes: agregou-se as classes Flmista, FLFdFd, FLPbPb e FLEcEc numa só classe Floresta (FL), as restantes cinco mantêm-se. As distâncias para estas seis classes estão na tabela-3.3.

Repare-se que ainda assim existem classes que não estão muito separadas espectralmente: o caso de FL e IC. Se se tiver em conta a definição da classe IC isto faz todo o sentido. Segundo [7] terrenos incultos (IC) são aqueles *"com cobertura vegetal com porte arbustivo, lenhosas ou herbáceas (...)"* daí a distinção entre esta classe e

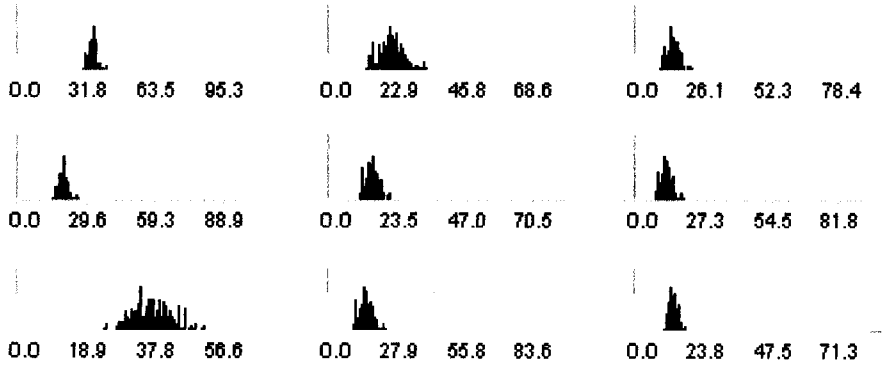

Figura 3.7: Comportamento espectral das classes FLmista (a preto) e FLEcEc (a azul) nas nove bandas espectrais consideradas. Note-se que a sobreposição é muita tal como indicia a distância de JM para estas duas classes.

	SC	HH	FG	IC	AG
HH	2.00				
FG	1.68	2.00			
IC	1.82	2.00	1.75		
AG	1.57	2.00	1.91	1.71	
FL	1.90	2.00	1.87	1.03	1.77

Tabela 3.3: As distâncias de Jeffries-Matusita entre as classes SC, HH, FG, IC, AG e FL (floresta).

a classe de floresta ser difícil. Outro factor que diminui a pureza das classes tem a ver com o facto de os objectos serem resultado de agregação de pixels. Por si só, um pixel já contém uma mistura de classes (dificilmente um pixel é completamente puro), tratando-se de um conjunto de pixels, é óbvio que essa mistura aumenta.
O facto de algumas classes não estarem espectralmente bem definidas é de ter em conta aquando da análise dos resultados da classificação. As áreas de treino, teoricamente, deveriam ser muito bem definidas espectralmente; mas na prática, são estas as classes que se pretendem identificar na imagem e portanto é com estas áreas de treino que a máquina irá ser treinada. Tem-se 582 áreas de treino cuja composição é a que se encontra na tabela 3.4.

Classes:	SC	HH	FG	FLEcEc	FLmista	FLFdFd	FLPbPb	IC	AG
N.Objs:	222	13	65	35	70	12	16	58	91

Tabela 3.4: Composição das áreas de treino.

3.3.3 A classificação com SVMs

Neste trabalho usaram-se apenas os dados de treino: 582 objectos resultantes da segmentação da imagem. O objectivo é encontrar a melhor máquina de suporte vectorial. Por melhor máquina de suporte vectorial entende-se aquela que tem menor erro no conjunto de teste.
O erro é estimado usando o método de validação cruzada nos 582 objectos de treino. Isto é, o conjunto de treino é aleatoriamente dividido em 10 conjuntos, sendo que o i-ésimo conjunto é deixado de fora (como conjunto de teste) e é determinada uma máquina com os restantes 9 conjuntos; no conjunto i a máquina é testada, calculando-se aí uma estimativa do erro de teste. A estimativa final para o erro de teste é a média dos 10 erros conseguidos repetindo este procedimento para os 10 conjuntos.

Tal como foi apresentado nos capítulos anteriores, as máquinas de suporte vectorial usam diferentes tipos de função *kernel* o que permite ter diferentes máquinas dependendo da função *kernel* escolhida e dependendo também dos parâmetros escolhidos para essa função.
Apesar de a maioria dos autores argumentarem que a escolha da função *kernel* é decisiva quanto à "qualidade" final da máquina de aprendizagem, alguns contrapõem dizendo que a escolha da função *kernel* é menos crucial do que parece à primeira vista [13]. De facto, pelos resultados obtidos com os dados usados neste trabalho verificou-se que a escolha da função *kernel*, assim como os seus parâmetros, é determinante no que diz respeito à estimativa do erro.
Os valores dos seus parâmetros variam com o tipo de dados a que se aplicam. Infelizmente, o problema de determinar a melhor função *kernel* a usar, assim como os respectivos parâmetros, é um problema ainda difícil de resolver, isto porque não existe nenhum método eficiente de resolver esta situação. Das funções *kernel* usadas nas máquinas de suporte vectorial (Eqs.(1.9), (1.10) e (1.11)) a mais comummemente usada é a *kernel* gaussiana (RBF-Radial Basis Function) por ser flexível, não muito complexa e por ter associado apenas um parâmetro e portanto ser aquela que melhor resultados tem dado em aplicações de máquinas de suporte vectorial.
Para além da função *kernel* e respectivos parâmetros, é necessário ainda escolher o parâmetro C (Eq.(2.22)) que controla o compromisso entre o erro e a capacidade da máquina.
Actualmente, o método mais usado para escolher estes parâmetros continua a ser a validação cruzada. A validação cruzada pode ser bastante dispendiosa [1]. Contudo, alguns investigadores têm vindo a explorar as formulações matemáticas das máquinas de suporte vectorial e a teoria de aprendizagem estatística associada para desenvolver modelos eficientes de critérios de selecção de parâmetros [5]; estes modelos têm porém como base o método de validação cruzada.

Para testar as máquinas de suporte vectorial, seguiu-se as sugestões de [15]: tentar primeiro pequenos e grandes valores para C - por exemplo de 1 até 1000 - escolhendo o melhor com validação cruzada, e finalmente, testar vários γ's (no caso de *kernel* RBF) para o melhor C. Para outras funções *kernel* procede-se de forma semelhante.

Existem outros métodos de busca dos parâmetros óptimos de uma determinada função *kernel*, por exemplo [10] refere que uma boa regra prática é escolher um valor de C ligeiramente mais pequeno do que o maior coeficiente ou valor de α obtido no treino com $C = \infty$.

Dado que as variáveis consideradas têm um intervalo de variação consideravelmente diferente, para além de se testar as máquinas de suporte vectorial nos dados *"brutos"*, também se testou nos dados normalizados. Nos dados *"brutos"* os parâmetros óptimos usando um função *kernel* RBF foram $C = 0.51$ e $\gamma = 0.008$ o que correspondeu a um erro de 18.21%. Usando uma função *kernel* polinomial, procurou-se primeiro o melhor grau mantendo os valores por defeito dos restantes parâmetros (no R por defeito, $C = 1$ e $\gamma = 1/dim(dados)$) e aqui surgiu uma dificuldade que faz da função RBF a mais aconselhada neste tipo de probemas: para polinómios de grau superior a 6 torna-se impossível determinar SVMs em tempo útil. Até este grau, o erro encontrado (tabela 3.5) é muito superior ao conseguido com uma função *kernel* RBF, ele vai

Grau:	1	2	3	4	5	6	7
Erro:	44.50%	41.41%	38.49%	38.32%	37.28%	32.65%	30.93%

Tabela 3.5: Erro obtido com máquinas de suporte vectorial com função *kernel* polinomial.

diminuindo à medida que se aumenta o grau da função *kernel* polinomial. No entanto essa diminuição não é muito significativa, o que leva a crer que o esforço computacional requerido para a determinação de máquinas com *kernel* polinomial não compensa o erro obtido.
Normalizando os dados, isto é, subtraindo a média e dividindo pelo desvio padrão, os resultados não melhoram. Aliás, a estimativa do erro chega até a aumentar! Conseguiu-se uma estimativa de 24.57% para o erro com uma máquina com *kernel* RBF de parâmetro $\gamma = 0.05$ e com $C = 100$. Usando funções *kernel* polinomiais o resultado é absurdo. Neste caso a máquina classifica todos os objectos numa só classe!

Quer no caso dos dados *"brutos"*, quer com os dados normalizados, a tarefa de determinar a melhor máquina é bastante dispendiosa. Relembre-se que para cada máquina o erro é estimado usando validação cruzada e são determinados várias máquinas usando diferentes parâmetros de forma a se encontrarem os óptimos. Assim, para cada parâmetro diferente são determinadas 10 máquinas de aprendizagem em 10 conjuntos de treino diferentes (cada um com cerca de 524 elementos) e testadas em conjuntos de 58 elementos. Esta é uma tarefa bastante morosa e que se agrava com o aumento do grau das funções *kernel* polinomial.

3.4 Técnicas de Redução da Dimensão dos Dados

O esforço computacional dispendido na procura da melhor máquina de suporte vectorial levou a que se recorresse a técnicas de redução da dimensão dos dados. Existem

diferentes métodos dependendo da estrutura do problema. A análise em componentes principais inspecciona a estrutura entre todas as variáveis. Se os indivíduos estão classificados em grupos então a análise de variáveis canónicas examina a estrutura entre-grupos. Estes dois métodos têm como base uma decomposição em valores próprios ou uma decomposição em valores singulares de uma matriz apropriada. As duas técnicas foram usadas e serão brevemente descritas na secção seguinte.

3.4.1 Análise em Componentes Principais

A análise em componentes principais (PCA Principal Component Analysis) é uma poderosa técnica de extracção de estrutura de dados de dimensão elevada. Um problema de componentes principais resolve-se recorrendo a um problema de valores próprios, ou usando algoritmos iterativos que estimam as componentes principais. A PCA é uma transformação ortogonal do sistema de coordenadas onde estão descritos os dados. Procura-se assim uma combinação linear das l observações iniciais com variância máxima. O resultado desta transformação são as componentes principais. É frequente que um pequeno número de componentes principais seja suficiente para descrever a estrutura dos dados.

Dado um conjunto de observações centradas $\mathbf{x}_k \in \mathbf{R}^d$, $k = 1, \ldots, l$, $\sum_{k=1}^{l} \mathbf{x}_k = 0$, a análise em componentes principais procura os vectores $\mathbf{v}$ que maximizam a variância de :

$$\mathbf{v}^T\mathbf{x}_k = v_1 x_{k1} + \ldots + v_d x_{kd}, \; \forall k = 1, \ldots, l$$

Seja

$$C = \frac{1}{l}\sum_{j=1}^{l} \mathbf{x}_j \mathbf{x}_j^T \tag{3.1}$$

a matriz de covariância de $\mathbf{x}$. Consequentemente, a variância de $\mathbf{v}^T\mathbf{x}$ é $\mathbf{v}^T C\mathbf{v}$. Assim, pretende-se maximizar essa variância impondo que $\mathbf{v}^T\mathbf{v} = 1$ para que o máximo possa ser atingido para um $\mathbf{v}$ finito.
A forma usual de resolver este problema de maximização é recorrendo à técnica dos multiplicadores de Lagrange. O objectivo torna-se então em maximizar

$$\mathbf{v}^T C\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{v} - 1)$$

onde λ é o multiplicador de Lagrange. Derivando esta função em ordem a $\mathbf{v}$ tem-se

$$C\mathbf{v} - \lambda\mathbf{v} = 0 \Leftrightarrow \lambda\mathbf{v} = C\mathbf{v}$$

Isto não é mais do que determinar os valores próprios λ de C e respectivos vectores próprios $\mathbf{v}$. Assim pode-se dizer que a PCA diagonaliza a matriz de covariância C. Tem-se então d candidatos a máximos: cada um dos vectores próprios. Para decidir qual deles dá $\mathbf{v}^T\mathbf{x}$ com variância máxima, relembre-se que se pretende maximizar

$$\mathbf{v}^T C\mathbf{v} = \mathbf{v}^T\lambda\mathbf{v} = \lambda\mathbf{v}^T\mathbf{v} = \lambda,$$

portanto λ deve ser o maior possível. Assim, a primeira componente principal é a combinação linear

$$\mathbf{v}_1^T\mathbf{x} = v_{11}x_1 + \ldots + v_{1d}x_d$$

onde $\mathbf{v}_1$ é o vector próprio correspondente ao maior valor próprio de C.
Genericamente, a i-ésima componente principal é a variável

$$\mathbf{v}_i^T\mathbf{x} = v_{i1}x_1 + \ldots + v_{id}x_d, \forall i = 1, \ldots, d$$

onde $\mathbf{v}_i$ é o vector próprio correspondente ao i-ésimo maior valor próprio de C, λ_i. Para além disso, $var(\mathbf{v}_i^T\mathbf{x}) = \lambda_i$. Tem-se ainda que a i-ésima componente principal explica $\frac{\lambda_i}{\sum_{i=1}^{d}\lambda_i}$ da variância.

3.4.2 Variáveis Canónicas

Se os indíviduos forem classificados em uma de K classes então, a dispersão total pode ser vista como uma combinação da dispersão entre grupos e a dispersão dentro dos grupos. A melhor discriminação entre os grupos é obtida maximizando a razão entre a variação entre-grupos e a variação dentro dos grupos. A análise de variáveis canónicas procura a combinação linear entre as d variáveis que maximiza esta razão. Estas novas variáveis são conhecidas por variáveis canónicas (ou variáveis discriminantes).

Para o caso de duas classes ($K = 2$), Fisher em 1936 formula o problema da seguinte forma:
Encontrar a combinação linear $Y = V^TX$ tal que a variância entre as classes é maximizada relativamente à variância dentro das classes.

Define-se assim uma medida de separação entre as observações projectadas como sendo a diferença entre as médias das amostras. Se $\mathbf{m}_i$ é a média dos elementos da classe C_i dada por

$$\mathbf{m}_i = \frac{1}{l_i}\sum_{\mathbf{x}\in C_i}\mathbf{x}$$

então, a média dos pontos projectados em $\mathbf{v}$ é dada por

$$\tilde{m}_i = \frac{1}{l_i}\sum_{\mathbf{x}\in C_i}\mathbf{v}^T\mathbf{x} = \mathbf{v}^T\mathbf{m}_i$$

Segue-se que $|\tilde{m}_1 - \tilde{m}_2| = |\mathbf{v}^T(\mathbf{m}_1 - \mathbf{m}_2)|$, e que se pode fazer esta diferença tão grande quanto se queira fazendo variar $\mathbf{v}$. Para ter uma boa separação das observações projectadas é necessário que a separação entre a média das classes seja grande relativamente a uma medida de dispersão de cada classe. Assim, defina-se a dispersão dos objectos da classe C_i projectados como

$$\tilde{s}_i^2 = \sum_{y\in C_i}(y - \tilde{m}_i)^2.$$

Então $\tilde{s}_1^2 + \tilde{s}_2^2$ é a dispersão total dentro das classes. Assim, o discriminante linear definido por Fisher é a função linear $\mathbf{v}^T\mathbf{x}$ para a qual o critério

$$J(\mathbf{v}) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

é máximo.
Para se ter J como função explícita de $\mathbf{v}$, definem-se matrizes de dispersão S_i e S_W como

$$S_i = \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T$$

e

$$S_W = S_1 + S_2.$$

Então

$$\tilde{s}_i^2 = \mathbf{v}^T S_i \mathbf{v},$$

assim,

$$\tilde{s}_1^2 + \tilde{s}_2^2 = \mathbf{v}^T S_W \mathbf{v}.$$

Analogamente,

$$(\tilde{m}_1 - \tilde{m}_2)^2 = \mathbf{v}^T S_B \mathbf{v},$$

onde

$$S_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T.$$

Pode-se então reescrever o critério de Fisher em termos de S_B e S_W como

$$J(\mathbf{v}) = \frac{\mathbf{v}^T S_B \mathbf{v}}{\mathbf{v}^T S_W \mathbf{v}}$$

Prova-se que a solução satisfaz:

$$S_B \mathbf{v} = \lambda S_W \mathbf{v}$$

que é um problema de valores próprios generalizado. Se S_W é não singular tem-se

$$S_W^{-1} S_B \mathbf{v} = \lambda \mathbf{v}.$$

Neste caso particular, não é necessário resolver para os valores e vectores próprios de $S_W^{-1} S_B$ uma vez que a direcção de $S_B\mathbf{v}$ é a mesma de $\mathbf{m}_1 - \mathbf{m}_2$. Assim, pode-se imediatamente escrever a solução:

$$\mathbf{v} = s_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2).$$

Tem-se assim o discriminante linear de Fisher, a função linear que maximiza a razão entre a dispersão entre as classes e a dispersão dentro das classes. O problema passou de uma dimensão d para dimensão 1.

No caso de K classes, a generalização do problema de Fisher envolve projectar os pontos num espaço de dimensão $K-1$.
Neste caso a matriz de dispersão entre as classes é definida por

$$S_W = \sum_{i=1}^{K} S_i$$

onde S_i é definido como no caso de $K=2$. A generalização de S_B não é óbvia [8]. Para isso é necessário a definição de uma matriz de dispersão total:

$$S_T = \sum_{\forall \mathbf{x}} (\mathbf{x}-\mathbf{m})(\mathbf{x}-\mathbf{m})^T$$

onde

$$\mathbf{m} = \frac{1}{l} \sum_{\forall \mathbf{x}} \mathbf{x}.$$

Expandindo esta matriz, tem-se que

$$S_B = \sum_{i=1}^{K} l_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T$$

onde l_i é o númeo de elementos da classe C_i, e

$$S_T = S_W + S_B$$

A projecção dos dados de um espaço de dimensão d para um novo espaço de dimensão $K-1$ é obtida por $K-1$ funções discriminantes $y_i = \mathbf{v}_i^T \mathbf{x}$, $i = 1, \ldots, K-1$. Se y_i forem vistos como componentes de um vector $\mathbf{y}$ e os vectores de peso $\mathbf{w}_i$ como colunas de uma matriz V de dimensão $d \times K-1$, então a projecção pode ser escrita como

$$\mathbf{y} = V^T \mathbf{x}.$$

Mostra-se que as matrizes de dispersão entre ($\tilde{S}_B$) e dentro ($\tilde{S}_W$) das classes no novo espaço, são dadas por

$$\tilde{S}_B = V^T S_B V \ \ e \ \ \tilde{S}_W = V^T S_W V$$

Desta feita, o critério a maximizar é

$$J(V) = \frac{|\tilde{S}_B|}{|\tilde{S}_W|} = \frac{|V^T S_B V|}{|V^T S_W V|}$$

O problema de encontrar uma matriz V que maximize J não é fácil. Felizmente, a solução acaba por se tornar simples [8]. As colunas de V são os vectores próprios generalizados correspondentes aos maiores valores próprios de

$$S_B \mathbf{v}_i = \lambda_i S_W \mathbf{v}_i.$$

Esta técnica projecta os dados num novo espaço de dimensão $K-1$. Se K for muito menor que d ter-se-á uma diminuição significativa da dimensão dos dados.

3.5 Discussão dos Resultados

O uso de novas variáveis de dimensão menor do que os dados originais tem a vantagem de tornar os processos de aprendizagem e classificação mais rápidos, no entanto é de ter em conta o custo de ter de determinar as novas variáveis. Contudo, com a experiência tida nestes dados, conclui-se ser vantajoso o uso de métodos de redução de variáveis, principalmente no caso da análise de variáveis canónicas.

No caso de funções *kernel* RBF, a melhor estimativa de erro foi conseguida usando as variáveis canónicas com $C = 4$ e $\gamma = 0.17$. O erro neste caso foi de 14.6%, uma melhoria significativa comparando com os 18.21% obtidos sem o uso desta técnica. A matriz de confusão desta classificação (tabela 3.6) revela a dificuldade em distinguir os diferentes tipos de vegetação. As distâncias de JM (tabela 3.2) davam já algumas indicações de que o processo de classificação poderia ter problemas na distinção de algumas classes. Por exemplo, repare-se na quantidade de objectos da classe FLmista que foram classificados como FLEcEc (17). De facto, a distância de JM entre estas duas classes era muito baixa (0.42). Observe-se ainda que a proporção de objectos correctamente classificados nas classes de floresta foi baixa relativamente às outras classes. O caso onde este aspecto se evidencia mais é na classe FLmista: apenas 57% dos objectos desta classe foram correctamente classificados. No entanto, este não foi o único caso em que a máquina de aprendizagem teve alguma dificuldade em distinguir diferentes classes. O caso das classes AG e SC também foi problemático. Repare-se no

	SC	HH	FG	FLEcEc	FLmista	FLFdFd	FLPbPb	IC	AG
SC	200	1	5	0	4	0	1	1	10
HH	0	13	0	0	0	0	0	0	0
FG	6	0	58	0	0	0	0	1	0
FLEcEc	0	0	0	29	4	0	0	2	0
FLmista	2	0	0	17	40	4	2	3	2
FLFdFd	1	0	0	0	0	8	0	2	1
FLPbPb	1	0	0	0	2	0	11	2	0
IC	1	0	0	0	0	0	0	56	1
AG	8	0	0	0	2	3	0	1	77

Tabela 3.6: Matriz de confusão da classificação com SVMs com $C = 4$ e função *kernel* RBF com $\gamma = 0.17$, com variáveis canónicas

número de elementos de SC que foram classificados como AG, e vice-versa. No entanto, se se tiver em conta o número de objectos destas classes, o número de objectos mal classificados acaba por não ser significativo. Em contrapartida, todos os elementos da classe HH foram correctamente identificados, como já era previsível. Esperava-se que esta classe (água) fosse fácil de identificar uma vez que, para além de ter uma resposta espectral significativamente diferente das restantes, foi a única classe para a qual as distâncias de JM relativamente a todas as outras classes atingiram o máximo. Um facto curioso tem a ver com a quantidade de objectos "absorvidos" pela classe IC. No

resultado desta classificação a classe IC contém elementos de todas as outras classes, com a excepção da classe HH. Sendo que as classes de floresta são as mais frequentes, evidenciando mais uma vez o facto de existir muita mistura entre as classes de floresta e a classe IC tal como tinha sido visto nos resultados apresentados na tabela 3.3. Usando componentes principais e funções *kernel* RBF, o melhor erro conseguido foi de 21% para $C = 1$ e $\gamma = 0.07$.

No caso de funções *kernel* polinomiais os resultados já não foram tão bons. Verificou-se o facto referido várias vezes na literatura, de que as máquinas de suporte vectorial que usam funções *kernel* RBFs têm uma melhor performance do que as restantes. De facto, para além de as estimativas de erro serem melhores com funções *kernel* RBF, é muito mais rápido determinar máquinas com este tipo de funções, e como consequência o melhor parâmetro para esse caso, acrescentando ainda a vantagem de o número de parâmetros a usar ser menor.
Usando variáveis canónicas, a máquina com função *kernel* polinomial que devolveu uma melhor estimativa de erro tem grau 17, o erro foi de 23.54% para este resultado o valor de C foi de 2.1. A alteração de outros parâmetros da função *kernel* polinomial não reflecte grandes melhorias na estimativa do erro. Usando componentes principais

	SC	HH	FG	FLEcEc	FLmista	FLFdFd	FLPbPb	IC	AG
SC	201	0	7	0	0	0	3	0	11
HH	3	10	0	0	0	0	0	0	0
FG	8	0	53	0	1	0	2	0	1
FLEcEc	1	0	0	0	33	0	0	1	0
FLmista	4	0	0	0	47	4	4	1	10
FLFdFd	1	0	0	0	1	3	0	0	7
FLPbPb	1	0	0	0	3	0	9	2	1
IC	1	0	0	0	2	3	5	42	5
AG	15	0	0	0	2	0	1	1	72

Tabela 3.7: Matriz de confusão da classificação com SVM com $C = 2.1$, com função *kernel* polinomial de grau 17, com variáveis canónicas

reduziu-se a complexidade uma vez que o grau óptimo é 3, no entanto esta redução da complexidade da máquina foi prejudicial uma vez que aumentou bastante a estimativa do erro. Mesmo aumentando o factor C de forma a penalizar as observações mal classificadas, a estimativa do erro mantém-se elevada: 31.44% para $C = 200$.
Tentou-se ainda usar outras funções *kernel*, no entanto, por serem demasiado complexas não foi possível determinar máquinas nessas condições para estes dados.

Os resultados obtidos com as variáveis canónicas foram francamente melhores. Se se tiver em conta que a análise de variáveis canónicas incorpora, no processo de redução da dimensão, a informação relativa à classe a que pertencem indivíduos, contrariamente ao que acontece na análise de componentes principais, este resultado faz todo o sentido. Outra nota positiva na análise de variáveis canónicas tem a ver com o tempo de

processamento: bem mais rápido do que as componentes principais.

Capítulo 4

Outros métodos de classificação supervisionada

Neste capítulo são brevemente apresentados dois métodos de classificação supervisionada: os K vizinhos mais próximos e discriminação logística.
O objectivo da introdução destes métodos neste trabalho prende-se com a necessidade de comparar os resultados obtidos no capítulo anterior com outros métodos mais conhecidos. O facto de serem métodos mais conhecidos, permite-nos ter uma certa confiança no que diz respeito a resultados uma vez que são métodos "validados" e cuja performance é conhecida.

4.1 K vizinhos mais próximos

Apesar de simples, o método dos K vizinhos mais próximos é bastante poderoso. Neste método assumem-se muito poucas restrições estruturais, sendo as suas previsões frequentemente correctas, apesar de um pouco instáveis.
A regra de classificação dos K vizinhos mais próximos é um método de classificação não paramétrico. Surge como resultado da aplicação do teorema de Bayes a estimativas de funções de densidade de probabilidade por métodos não paramétricos. Pretende-se então maximizar a probabilidade à posteriori:

$$P(G = i|X = \mathbf{x}) = \frac{\pi_i p_i(\mathbf{x})}{p(\mathbf{x})}$$

Uma vez que não se conhecem as funções de densidade de probabilidade, é necessário estimar $p(\mathbf{x})$. Nos métodos não paramétricos nada se assume sobre as f.d.p.. Dado um ponto $\mathbf{x}$, pretende-se estimar $p(\mathbf{x})$. Seja R uma região que contém $\mathbf{x}$ e V o volume dessa região. Tem-se que a probabilidade dessa região é dada por

$$P(R) = \int_R p(t)dt \approx Vp(\mathbf{x}) \Leftrightarrow p(\mathbf{x}) \approx \frac{P(R)}{V}$$

Se se considerar uma amostra $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l$, e K o número de elementos da amostra que estão em R, tem-se que $\hat{P}(R) = \frac{K}{l}$, logo uma estimativa para $p(\mathbf{x})$ é

$$\hat{p}(\mathbf{x}) = \frac{K/l}{V} = \frac{K}{lV}$$

Prova-se que $\frac{K}{l}$ é estimador de máxima verosimilhança de $P(R)$.
Uma vez que R é fixo, $\hat{p}(\mathbf{x})$ não converge para $p(\mathbf{x})$ quando $l \to \infty$. Para que isso aconteça, deve-se diminuir o volume V quando $l \to \infty$. Se se fizer $V \to 0$ quando $l \to \infty$, eventualmente $\frac{K/l}{V}$ converge para $p(\mathbf{x})$. O problema é que se se fizer o volume muito pequeno, como a amostra é finita, acaba por não se ter nenhuma observação em R, e portanto $\hat{p}(\mathbf{x}) = 0$.
Existem dois tipos de estimação não paramétrica: as janelas de Parzen e os k-vizinhos mais próximos. Aqui vai-se ter em conta apenas o método dos k-vizinhos mais próximos. Neste método, especifica-se o número de elementos K que estão em R, em função de l - K_l. Depois, o volume é ajustado de forma a conter K_l pontos.
Relembre-se que se pretende estimar

$$\hat{P}(G = i|X = \mathbf{x}) = \frac{\hat{p}(\mathbf{x}, G_i)}{\sum_{j=1}^{K} \hat{p}_j(\mathbf{x}, G_j)}$$

Como $\hat{p}(\mathbf{x}) = \frac{K}{lV}$ e $\hat{p}(\mathbf{x}, G_i) = \frac{K_i}{lV}$, tem-se que

$$\hat{P}(G = i|X = \mathbf{x}) = \frac{K_i/lV}{\sum_{j=1}^{K} K_j/lV} = \frac{K_i}{K}.$$

Ora, como o objectivo é maximizar $\hat{P}(G = i|\mathbf{x})$, portanto o máximo é o maior K_i, assim, escolhe-se a classe mais frequente no volume.
O método de classificação dos k-vizinhos mais próximos torna-se muito simples: dado um ponto $\mathbf{x}_0$ a classificar, identificam-se os objectos mais próximos. A noção de proximidade implica a existência de uma métrica. É comum usar-se a distância euclidiana para identificar os objectos mais próximos. Assim, é calculada a distância a todos os objectos do treino e considera-se a classe dos K objectos mais próximos (os K vizinhos mais próximos). A classe de $\mathbf{x}_0$ é determinada pela classe mais frequente no conjunto dos K vizinhos mais próximos. No caso de empates, o desempate é feito de forma aleatória.
Uma das vantagens deste método é o facto de ser não paramétrico, isto é, não supor nenhuma distribuição particular dos dados, e assim pode ser aplicado em vários casos. Tem ainda a vantagem de ter apenas um parâmetro a estimar - K. Uma desvantagem é a necessidade de ter disponível muitos exemplos classificados para classificar um novo exemplo, o que exige recursos de memória que nem sempre estão disponíveis. Outra desvantagem tem a ver com a maldição da dimensão - um aspecto ultrapassado nas máquinas de suporte vectorial (sec.2.2.3).
Um caso particular ocorre quando K, o número de vizinhos, é um. Neste caso pode demonstrar-se que o erro máximo cometido pelo classificador nunca é maior do que o dobro do menor erro possível, de acordo com a regra de Bayes, se o número de

exemplos classificados disponíveis for infinito [8]. Para K maior o erro aproxima-se do menor erro possível, mas o facto de o número de exemplos disponíveis ser finito contraria esta tendência.

4.2 Discriminação Logística

A discriminação logística pertence ao grupo de métodos de regressão que descrevem a relação entre variáveis explicativas e uma variável de resposta discreta. A diferença entre a modelação da regressão linear usual e a regressão logística está no facto de que para o caso da regressão linear, a variável resposta é contínua. Na regressão logística, a variável resposta é discreta. Esta diferença manifesta-se na escolha dos parâmetros e nas hipóteses formuladas.
O objectivo do método de regressão logística é modelar as probabilidades *à posteriori* de K classes recorrendo a funções lineares em $\mathbf{x}$. O modelo é da forma [12]:

$$\begin{aligned} log\frac{P(G=1|X=\mathbf{x})}{P(G=K|X=\mathbf{x})} &= \beta_{10} + \beta_1^T\mathbf{x} \\ log\frac{P(G=2|X=\mathbf{x})}{P(G=K|X=\mathbf{x})} &= \beta_{20} + \beta_2^T\mathbf{x} \\ &\vdots \\ log\frac{P(G=K-1|X=\mathbf{x})}{P(G=K|X=\mathbf{x})} &= \beta_{(K-1)0} + \beta_{K-1}^T\mathbf{x} \end{aligned} \tag{4.1}$$

Com cálculos simples mostra-se que

$$\begin{aligned} P(G=k|X=\mathbf{x}) &= \frac{exp(\beta_{k0}+\beta_k^T\mathbf{x})}{1+\sum_{l=1}^{K-1}exp(\beta_{l0}+\beta_l^T\mathbf{x})},\ k=1,\ldots,K-1 \\ P(G=K|X=\mathbf{x}) &= \frac{1}{1+\sum_{l=1}^{K-1}exp(\beta_{l0}+\beta_l^T\mathbf{x})} \end{aligned} \tag{4.2}$$

A classificação é feita usando a regra de Bayes, isto é, atribuindo a $\mathbf{x}$ a classe k que maximiza $P(G = k \mid X = \mathbf{x})$. Este procedimento é conhecido por discriminação logística [20]. Os termos $\beta_0, \ldots, \beta_{K-1}$ são parâmetros desconhecidos que devem ser estimados com base nos objectos disponíveis. É normal usar-se o método de máxima verosimilhança para a estimação dos parâmetros.

Denotem-se as probabilidades $P(G = k|X = \mathbf{x})$ por $p_k(\mathbf{x};\theta)$ de forma a evidenciar a dependência no conjunto de parâmetros $\theta = \{\beta_{10}, \beta_1, \ldots, \beta_{(K-1)0}, \beta_{K-1}\}$.
Quando $K = 2$, o modelo é relativamente simples, uma vez que existe uma única função linear. É regularmente usado em aplicações bioestatísticas onde são frequentes respostas binárias.
O logaritmo da função de verosimilhança para l observações é

$$l(\theta) = \sum_{i=1}^{l} \log p_{g_i}(\mathbf{x}_i;\theta) \tag{4.3}$$

Para ilustrar a maximização de $l(\theta)$, considere-se o caso de $K = 2$.
No caso de 2 classes, é conveniente codificar as duas classes g_i a resposta y_i por 0/1, onde $y_i = 1$ quando $g_i = 1$, e $y_i = 0$ quando $g_i = 2$.
Seja $p_1(\mathbf{x};\theta) = p(\mathbf{x};\theta)$, e $p_2(\mathbf{x};\theta) = 1 - p(\mathbf{x};\theta)$. Para a estimação dos parâmetros considere-se que os valores de $\mathbf{x}_i$ são fixos e que a única variável aleatória é o valor da classe y_i. Assumindo que esta variável segue uma distribuição de Bernoulli, tem-se

$$p_{g_i}(\mathbf{x}_i;\theta) = p(\mathbf{x}_i;\beta)^{y_i}(1 - p(\mathbf{x}_i;\beta))^{1-y_i},$$

onde $\beta^T = (\beta_{10}, \beta_1^T)$. Vem então que

$$\begin{aligned} l(\beta) &= \sum_{i=1}^{l}\{y_i \log p(\mathbf{x}_i;\beta) + (1 - y_i)\log(1 - p(\mathbf{x}_i;\beta))\} \\ &= \sum_{i=1}^{l}\{y_i \log \tfrac{p(\mathbf{x}_i;\beta)}{1-p(\mathbf{x}_i;\beta)} + log(1 - p(\mathbf{x}_i;\beta))\} \end{aligned} \tag{4.4}$$

Pelas Eqs.(4.1) tem-se que

$$\log(\frac{p(\mathbf{x}_i;\beta)}{1 - p(\mathbf{x}_i;\beta)}) = \beta^T\mathbf{x}_i,$$

e pelas Eqs.(4.2) tem-se que

$$p(\mathbf{x}_i;\beta) = P(G = 1|X = \mathbf{x}_i) = \frac{e^{\beta^T\mathbf{x}_i}}{1 + e^{\beta^T\mathbf{x}_i}}$$

assim,

$$1 - p(\mathbf{x}_i;\beta) = \frac{1}{1 + e^{\beta^T\mathbf{x}_i}}$$

e portanto,

$$\log(1 - p(\mathbf{x}_i;\beta)) = -\log(1 + e^{\beta^T\mathbf{x}_i}).$$

Substituindo estas expressões na Eq.(4.4) tem-se

$$l(\theta) = \sum_{i=1}^{l}\left\{y_i\beta^T\mathbf{x}_i - \log(1 + e^{\beta^T\mathbf{x}_i})\right\}$$

Aqui assume-se que o vector de entrada $\mathbf{x}_i$ inclui o termo 1 de forma a considerar a constante de intercepção.
Para maximizar a função verosimilhança $l(\beta)$, iguala-se a sua derivada a zero:

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{l}\mathbf{x}_i(y_i - p(\mathbf{x}_i;\beta)) = 0 \tag{4.5}$$

obtendo assim $d + 1$ equações (d é a dimensão das variáveis) não lineares em β.
Para encontrar as raízes desta equação, recorre-se ao método de Newton-Raphson[1].

[1]Para calcular a raíz de uma função $f(x_n)$ calcula-se iterativamente: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

Então, para determinar as raízes de (4.5) resolve-se iterativamente:

$$\beta^{n+1} = \beta^n - \left(\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T}\right)^{-1} \frac{\partial l(\beta)}{\partial\beta} \tag{4.6}$$

Se se considerar $\mathbf{y}$ como o vector dos y_i's; $X_{l\times(d+1)}$ a matriz dos vectores $\mathbf{x}_i$ e com a primeira coluna de 1's; $\mathbf{p}$ o vector das probabilidades estimadas, cujo i-ésimo elemento é $p(\mathbf{x}_i;\beta^n)$; e $\mathbf{W}_{l\times l}$ a matriz diagonal cujo i-ésimo elemento da diagonal é $p(\mathbf{x}_i;\beta^n)(1-p(\mathbf{x}_i;\beta^n))$, pode escrever-se:

$$\frac{\partial l(\beta)}{\partial\beta} = X^T(\mathbf{y}-\mathbf{p}) \quad \text{e} \quad \frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T} = -X^T W X \tag{4.7}$$

Então o passo de Newton-Raphson é:

$$\begin{aligned}\beta^{n+1} &= \beta^n + (X^T W X)^{-1} X^T(\mathbf{y}-\mathbf{p}) \\ &= (X^T W X)^{-1} X^T W (X\beta^n + W^{-1}(\mathbf{y}-\mathbf{p})) \\ &= (X^T W X)^{-1} X^T W \mathbf{z}\end{aligned} \tag{4.8}$$

Na segunda e terceira linha, o passo de Newton-Raphson é equivalente a um passo do Método dos Mínimos Quadrados Pesados com resposta:

$$\mathbf{z} = X\beta^n + W^{-1}(\mathbf{y}-\mathbf{p})$$

Assim, o β óptimo é dado por:

$$\beta^{n+1} = argmin_\beta (\mathbf{z}-X\beta)^T W (\mathbf{z}-X\beta) \tag{4.9}$$

Neste procedimento iterativo $\beta = 0$ é um bom ponto de partida, embora a convergência nunca seja garantida. Normalmente o algoritmo converge, uma vez que a função de log-verosimilhança é côncava, mas pode acontecer *overshooting* [12].

No caso multiclasse ($K > 3$) o algoritmo de Newton pode também ser expresso como um algoritmo iterativo de mínimos quadrados pesados, mas com um vector de $K-1$ respostas e uma matriz de pesos não diagonal por observação.

Tendo estimados os valores de β, dado um novo objecto $\mathbf{x}$, avalia-se as funções logísticas estimadas e calcula-se as estimativas das probabilidades de esse novo objecto pertencer a uma determinada classe.

4.3 Discussão de resultados

Os métodos dos K vizinhos mais próximos e discriminação logística foram usados no conjunto de dados apresentado no capítulo 3. A aplicação destes métodos nestes dados permite comparar com os resultados obtidos pelas máquinas de suporte vectorial.

Antes de mais, repare-se que as máquinas de suporte vectorial têm vários parâmetros a serem definidos pelo utilizador. Este número é mínimo no caso de o utilizador escolher uma máquina que use uma função *kernel* RBF. Neste caso, são três os parâmetros a ser definidos pelo utilizador. No que diz repeito ao método dos K vizinhos mais próximos o utilizador só necessita de indicar o número de vizinhos mais próximos que deve utilizar na tarefa de classificação. Quanto à discriminação logística, o utilizador não define directamente nenhum parâmetro. No entanto, neste método é estimado um razoável número de parâmetros o que pode introduzir um erro significativo no processo de classificação.

À semelhança do que se fez no caso das máquinas de suporte vectorial, usou-se o método de validação cruzada para estimar o erro. Esta estimativa do erro dá indicação da qualidade da classificação. Tal como no trabalho apresentado no capítulo anterior, a validação cruzada envolveu a divisão do conjunto de treino em 10 subconjuntos aleatórios. Para cada um desses 10 subconjuntos, testou-se um classificador construído com os restantes 9 subconjuntos. Sendo a estimativa do erro final dada pela média dos 10 erros obtidos em cada um dos conjuntos.

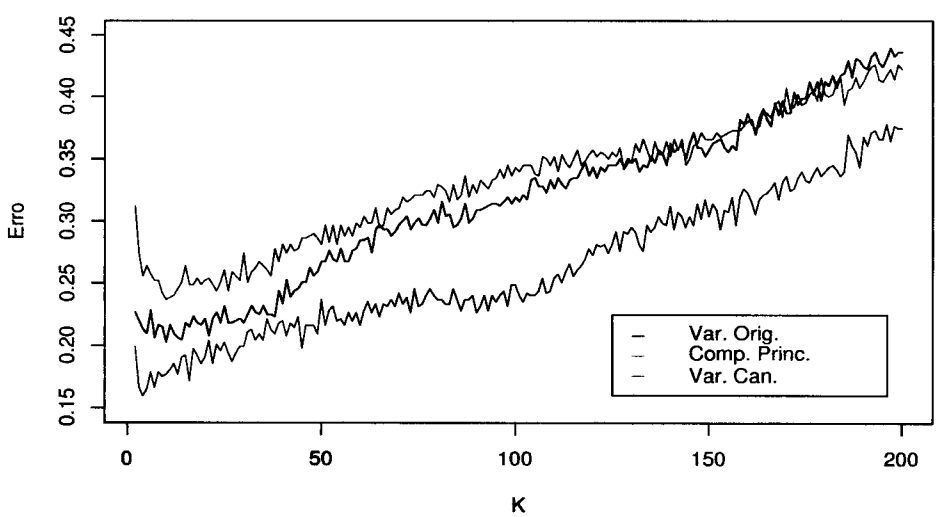

Figura 4.1: A variação da estimativa do erro na classificação com a variação de K no método dos K vizinhos mais próximos. A preto estão os resultados obtidos nos dados originais. A vermelho os resultados obtidos com as componentes principais que explicam 95% da variância dos dados. A azul com as 8 variáveis canónicas.

Analogamente ao que foi feito no capítulo anterior, estes dois métodos de classificação foram usados tanto no conjunto original de dados, como nas suas componentes principais e variáveis canónicas.

No caso dos K vizinhos mais próximos, a estimativa do erro via validação cruzada permite ter indicação do melhor K, isto é, do número de vizinhos a considerar. Assim, determinou-se estimativas do erro para K a variar entre 2 e 200. Esses resultados

são apresentados na figura 4.1 para os dados originais, as componentes principais e variáveis canónicas.

Ao analisar a figura 4.1 é evidente a diferença entre os erros obtidos com as variáveis canónicas e com as restantes variáveis. Com a regra dos K vizinhos mais próximos repete-se o verificado com as máquinas de suporte vectorial: as variáveis canónicas conseguem estimativas de erro mais baixas. Quer para os dados originais quer para as componentes principais o valor de K óptimo foi 10. Para o primeiro caso o erro foi de 20.27%, enquanto que para o segundo foi de 23.71%. Quanto às variáveis canónicas, para além de reflectirem um erro menor, o número de vizinhos a considerar também é menor: um erro de 15.98% usando apenas os 4 vizinhos mais próximos.

A tabela 4.1 mostra a matriz de confusão obtida com o método dos 4 vizinhos mais próximos nas variáveis canónicas. De facto é complicada a distinção entre as classes de Floresta. Este problema já tinha sido referido aquando da caracterização das classes na secção 3.2.2.

	SC	HH	FG	FLEcEc	FLmista	FLFdFd	FLPbPb	IC	AG
SC	199	1	2	0	1	0	0	3	14
HH	1	12	0	0	0	0	0	0	0
FG	2	0	59	0	0	0	0	0	0
FLEcEc	0	0	0	16	9	0	0	1	0
FLmista	1	0	1	16	51	7	8	2	3
FLFdFd	0	0	0	0	0	0	0	0	0
FLPbPb	0	0	0	0	1	0	3	0	0
IC	7	0	3	3	3	2	4	52	2
AG	12	0	0	0	5	3	1	0	72

Tabela 4.1: Matriz de confusão da classificação com método 4 vizinhos mais próximos, com variáveis canónicas.

Repare-se na linha correspondente à classe FLFdFd. Nenhum objecto foi atribuído a esta classe. No entanto à classe FLmista foram atribuídos objectos de quase todas as classes. A única excepção foram objectos pertencentes à classe HH (água). Relativamente às classes de vegetação, apenas a classe FLmista tem a maior parte dos seus objectos classificados correctamente. Os objectos pertencentes às restantes estão incorrectamente distribuídos por outras classes de vegetação. É de notar ainda alguma mistura na classificação entre as classes SC e AG. Este facto não será de estranhar uma vez que certamente muitos objectos definidos como pertencentes a uma destas classes contêm pixels relativos à outra.

Quanto à discriminação logística, também aqui os melhores resultados foram obtidos com as variáveis canónicas. A tabela 4.2 apresenta a matriz de confusão para esta classificação. Também aqui o erro conseguido foi de 15.98%. Com as componentes principais foi de 20.62% e com os dados originais 17.18%.

	SC	HH	FG	FLEcEc	FLmista	FLFdFd	FLPbPb	IC	AG
SC	196	4	6	0	2	0	1	1	12
HH	0	11	1	0	0	0	1	0	0
FG	6	0	57	0	0	0	0	2	0
FLEcEc	0	0	0	25	9	0	0	1	0
FLmista	2	0	0	8	52	3	1	1	3
FLFdFd	0	0	0	0	4	5	0	3	0
FLPbPb	2	0	1	0	1	0	11	1	0
IC	1	0	0	1	2	0	0	53	1
AG	7	0	1	0	2	1	0	1	79

Tabela 4.2: Matriz de confusão da classificação obtida com discriminação logística, com variáveis canónicas

A clasificação com a discriminação logística parece distinguir mais as classes de floresta. Ao passo que o método dos K vizinhos atribui a classe dos objectos que estão mais próximos, a regressão logística tem em conta outros factores que permitem um pouco mais de distinção entre as classes de floresta. A proximidade espectral destas classes leva a que o método dos k-vizinhos tenha maior dificuldade em separá-las correctamente.

Os erros obtidos com as máquinas de suporte vectorial foram mais baixos. No entanto é de ter em conta o tempo dispendido na escolha da melhor *kernel*, assim como o parâmetro C óptimo. Tanto os K-vizinhos como a discriminação logística permitem atingir o valor óptimo da sua classificação de uma forma mais rápida. Em futuros trabalhos é necessário ter em conta estes aspectos. Dependendo do objectivo da classificação, deverá optar-se por um método mais rápido ou por aquele que devolve erros menores. É ainda de referir que após a determinação dos parâmetros óptimos, as máquinas de suporte têm uma performance relativamente rápida. A dificuldade em determinar a melhor máquina de suporte vectorial não deve ser um entrave ao uso deste método. De facto, ele tem melhores resultados do que os outros dois usados. Para além disso, encontram-se em desenvolvimento técnicas matemáticas mais eficientes de procura dos parâmetros óptimos [5].

Capítulo 5

Análise de componentes principais com funções *kernel*

No capítulo 2 mostrou-se que a ideia de transformar implicitamente os dados para um espaço de dimensão superior tem sido bastante produtivo no contexto das máquinas de suporte vectorial. Este é um factor que permite a aplicação deste método a problemas complexos do mundo real, onde as observações não são linearmente separáveis. Assim, é natural que se coloque a questão de saber se a mesma ideia pode ser aplicada em outros domínios da aprendizagem.
Este capítulo apresenta um método proposto por Schölkopf [23] que determina uma forma não linear das componentes principais. Usando funções *kernel* de Mercer, pode-se efectivamente calcular as componentes principais em espaços de dimensão superior, ligados ao espaço inicial por uma função não linear.

5.1 Introdução

A análise de componentes principais é um método vulgarmente usado nas mais variadas áreas. Este método é descrito na secção 3.4.1.
A abordagem feita neste capítulo não tem em vista as componentes principais no espaço inicial. O interesse é nas componentes principais de variáveis que estão relacionadas com as originais por uma função não linear. Para isto, são determinados produtos internos no espaço de dimensão superior por meio de funções *kernel* no espaço inicial. Dado qualquer algoritmo, que esteja expresso apenas em função de produtos internos, o método *kernel* permite construir diferentes versões não lineares desse algoritmo.

Como já foi referido o método de componentes principais foi apresentado na secção 3.3.1. Intencionalmente, este método foi apresentado usando uma formulação que apenas usa produtos internos. Usando a representação *kernel* de produtos internos (Eq.(1.7)), é possível ter-se um algoritmo baseado em funções *kernel* para componentes

principais não lineares.

5.2 Componentes Principais no espaço final

Antes de avançar, é necessário esclarecer o que se entende por espaço final[1]. Espaço final é aquele onde estão representadas as variáveis transformadas pela função não linear implícita na transformação *kernel*.

Tendo em conta a teoria apresentada na secção 3.3.1, considere-se a partir de agora os mesmo cálculos mas num outro espaço Y com o produto vectorial definido, que se encontra relacionado com o espaço inicial pela função não linear:

$$\begin{aligned} \Phi : R^d &\to Y \\ \mathbf{x} &\mapsto \mathbf{X} \end{aligned} \tag{5.1}$$

Note-se que a dimensão do espaço Y pode ser infinita. A partir de agora, os caracteres minúsculos são usados para elementos de R^d, enquanto que os maiúsculos para elementos de Y.
Assume-se novamente que os dados são centrados, $\sum_{k=1}^{l} \Phi(\mathbf{x}_k) = 0$. Em Y a matriz de covariância é da forma:

$$\overline{C} = \frac{1}{l} \sum_{j=1}^{l} \Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^T \tag{5.2}$$

Como se viu no caso tradicional, a PCA diagonaliza esta matriz de covariâncias. Assim, tem de se determinar os valores próprios $\lambda \geq 0$ (porque C é simétrica semi-definida positiva) e vectores próprios $V \in Y \backslash \{0\}$ que satisfazem

$$\lambda \mathbf{V} = \overline{C}\mathbf{V} \tag{5.3}$$

Novamente, todas as soluções $\mathbf{V}$ com $\lambda \neq 0$ encontram-se no espaço $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_l)$. Isto tem duas consequências: primeiro, pode-se considerar o conjunto de funções

$$\lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot \overline{C}\mathbf{V}), \ \forall k = 1, \ldots, l. \tag{5.4}$$

e segundo, existem coeficientes α_i $(i = 1, \ldots, l)$ tais que

$$\mathbf{V} = \sum_{i=1}^{l} \alpha_i \Phi(\mathbf{x}_i). \tag{5.5}$$

Combinando Eq.(5.4) e Eq.(5.5), tem-se

$$\lambda \sum_{i=1}^{l} \alpha_i (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_i)) = \frac{1}{l} \sum_{i=1}^{l} \alpha_i \left(\Phi(\mathbf{x}_k) \cdot \sum_{j=1}^{l} \Phi(\mathbf{x}_j)(\Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i)) \right), \ \forall\, k = 1, \ldots, l. \tag{5.6}$$

[1]Em inglês normalmente é denominado por *feature space*

Definindo uma matriz K de dimensão $l \times l$ como $K_{ij} := (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$, fica

$$l\lambda K\alpha = K^2\alpha \tag{5.7}$$

onde α é um vector coluna com entradas $\alpha_1, \ldots, \alpha_l$. Para encontrar as soluções de Eq.(5.7), resolve-se o poblema de valores próprios:

$$l\lambda\alpha = K\alpha \tag{5.8}$$

para valores próprios diferentes de zero.
Sejam $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_l$ os valores próprios de K (i.e, as soluções $l\lambda$ de Eq.(5.8)), e $\alpha^1, \ldots, \alpha^l$ o correspondente conjunto de vectores próprios, com λ_p sendo o primeiro valor próprio diferente de zero. Normaliza-se $\alpha^p, \ldots, \alpha^l$ exigindo que os correspondentes vectores em Y estejam normalizados, i.e,

$$(\mathbf{V}^k \cdot \mathbf{V}^k) = 1 \ \forall k = p, \ldots, l$$

Para a extracção de componentes principais, é necessário calcular as projecções nos vectores próprios $\mathbf{V}^k$ em Y $(k = p, \ldots, l)$. Seja $\mathbf{x}$ um ponto de teste, com imagem $\Phi(\mathbf{x})$ em Y, então

$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{l} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{l} \alpha_i^k K(\mathbf{x}_i, \mathbf{x}) \tag{5.9}$$

pode ser chamado como as suas componentes principais não lineares relativamente a $\mathbf{\Phi}$.

Resumindo, para determinar as componentes principais não lineares os passos a seguir são: primeiro, calcular a matriz K; segundo, calcular os seus vectores próprios e normalizá-los em Y; terceiro, calcular as projecções dos pontos de teste nos vectores próprios.
Inicialmente assumiu-se que as observações são centradas. Isto é fácil de se atingir no espaço inicial, mas mais difícil em Y, uma vez que não se consegue calcular explicitamente a média das novas variáveis em Y. Existe contudo uma forma de o fazer; no entanto, leva a ligeiras modificações nas equações apresentadas [22].
Para a transformação das variáveis originais no espaço inicial no novo espaço, podem ser usadas as funções *kernel* apresentadas na secção 1.3.

5.3 Propriedades

Para funções *kernel* de Mercer, sabe-se que de facto se determinam as componentes principais usuais em Y. Consequentemente, as propriedades matemáticas e estatísticas das componentes principais mantêm-se, com as modificações adequadas para afirmações sobre $\Phi(\mathbf{x}_i)$ em Y, no lugar de R^d. Assim, em Y pode afirmar-se que as componentes principais são uma transformação ortogonal com as seguintes propriedades:

- as primeiras $q(p \in \{1, \ldots, l\})$ componentes principais, i.e, projecções nos vectores próprios, contêm mais informação do que outras quaisquer q direcções ortogonais;
- o erro em média quadrática da representação das variáveis pelas q componentes principais é mínimo;
- as componentes principais são não correlacionadas;
- as primeiras q componentes principais têm informação máxima no que respeita aos inputs.

No que diz respeito à complexidade computacional, este método é mais dispendioso do que o equivalente linear. Contudo, este "investimento" pode ter frutos mais tarde. Por exemplo, Schölkopf refere que, em experiências de classificação usando os resultados deste método, foi suficiente uma máquina de suporte vectorial linear para construir a fonteira de decisão. Máquinas de suporte vectorial lineares são muito mais rápidas do que as não lineares. Assim a fase final de classificação pode tornar-se bastante rápida.

Comparando com outras técnicas não lineares de extracção de variáveis, as componentes principais com funções *kernel* têm a vantagem de não necessitar de optimização não linear. Precisam apenas da solução de um problema de valores próprios. Têm ainda a vantagem de, usando diferentes funções *kernel*, se ter um maior número de classes de funções não lineares a usar. O problema de determinar qual a melhor função *kernel* a usar continua a ser um problema em aberto.
Este método de componentes principais com funções *kernel* pode ser usado em todos os campos onde se tem usado a análise de componentes principais tradicional para extracção de variáveis, e onde uma extensão não linear faria sentido.

Capítulo 6

Conclusão

Neste trabalho foi apresentada uma técnica relativamente recente de classificação baseada na Teoria da Aprendizagem, desenvolvida por Vapnik nos finais dos anos 70[24]: máquinas de suporte vectorial. As máquinas de suporte vectorial diferem de métodos semelhantes (tais como redes neuronais) no sentido de encontrarem sempre um mínimo global. A simplicidade da sua interpretação geométrica proporciona bases para mais investigação. Estas máquinas são amplamente caracterizadas pela escolha da função *kernel*.

Este trabalho visou essencialmente explorar as capacidades de máquinas de suporte vectorial na classificação de imagens multi-espectrais. Evidencie-se o facto de a classificação feita neste trabalho ser baseada em objectos e não na informação de pixels como habitualmente. Esta abordagem tem duas grandes vantagens. A primeira é que uma classificação direccionada a objectos é mais realista do que a classificação tradicional de pixel. No problema considerado este é um facto por demais evidente, uma vez que se pretende classificar porções de solo consideravelmente superiores à dimensão representada pelo pixel. Outra vantagem de usar uma imagem segmentada em objectos, em vez da imagem com a informação de pixels, é a drástica redução do número de objectos a classificar. No entanto, surgiu uma grande dificuldade na classificação deste tipo de objectos. Essa dificuldade teve a ver com as áreas de treino definidas. O facto de um objecto ser formado por um conjunto de pixels introduz ruído em cada novo objecto. Ora, se é quase certa a existência de ruído no pixel, num conjunto de pixels esse ruído é ainda maior. Para além disso, verificou-se que as áreas de treino de uma determinada classe frequentemente incluem pixels de outras classes. Uma outra dificuldade com que se deparou foi com o facto de muitas das classes definidas terem características espectrais muito semelhantes.

Comparando os resultados das máquinas de suporte vectorial com os obtidos com o método dos K vizinhos e discriminação logística, os primeiros foram melhores. No entanto, é de salientar o esforço computacional na determinação da melhor máquina de suporte vectorial. Ultrapassado este problema, o método desenvolvido por Vapnik tem resultados bastante satisfatórios de uma forma bastante rápida.

Actualmente, a determinação dos parâmetros que conduzem à melhor máquina de suporte vectorial continua a ser uma tarefa custosa. Isto porque o comum é esses parâmetros serem determinados por experimentação. Consideram-se os parâmetros que minimizam o erro estimado por métodos como validação cruzada. Este é portanto um campo a explorar. Alguns investigadores têm procurado obter uma solução eficiente para este problema.

O uso de variáveis canónicas revelou-se mais proveitoso do que a aplicação das componentes principais. Estes métodos permitiram reduzir a dimensão dos dados. No entanto, a qualidade da classificação não melhorou com a introdução das componentes principais. O mesmo não aconteceu com as variáveis canónicas. Neste caso, a estimativa do erro diminui consideravelmente. Para além da vantagem de diminuir a dimensão dos dados, revelou-se um processo bastante rápido.

Tanto o método de máquinas de suporte vectorial como a análise de componentes principais usando funções *kernel* de Mercer, são dois exemplos do potencial de aplicações baseadas em funções *kernel* de Mercer. Qualquer algoritmo que possa ser formulado apenas em termos de produtos internos pode ser feito não linear transportando-o para espaços de dimensão superior através destas funções. Este é o grande potencial deste tipo de métodos. A análise de componentes principais usando funções *kernel* de Mercer parece bastante promissora. Tal como no caso das máquinas de suporte vectorial, também aqui a escolha das funções *kernel* é de grande importância. Infelizmente, por falta de tempo não foi possível usar este método nos dados analisados neste trabalho. No entanto é uma tarefa a ter em conta em trabalhos futuros.

Apêndice A

Imagens Multi-espectrais

A.1 O espectro electromagnético

O espectro electromagnético é composto por uma vasta gama de comprimentos de onda electromagnética, desde os raios gama (10^{-14}m) até às ondas de rádio (10^{4}m). Tende-se a pensar na radiação óptica como "luz", mas o arco-íris de cores que compõe a luz "visível" é apenas uma pequena parte do espectro electromagnético. (Algumas partes do espectro são total ou parcialmente bloqueadas pela atmosfera terrestre, sendo por isso pouco utilizadas por satélites de observação da Terra). Não existe nenhum limite ou fronteira natural nesta distribuição de energia, embora por conveniência se divida esta escala em várias secções (faixas ou bandas) atribuindo-lhes um nome tal como a figura A.1 indica.

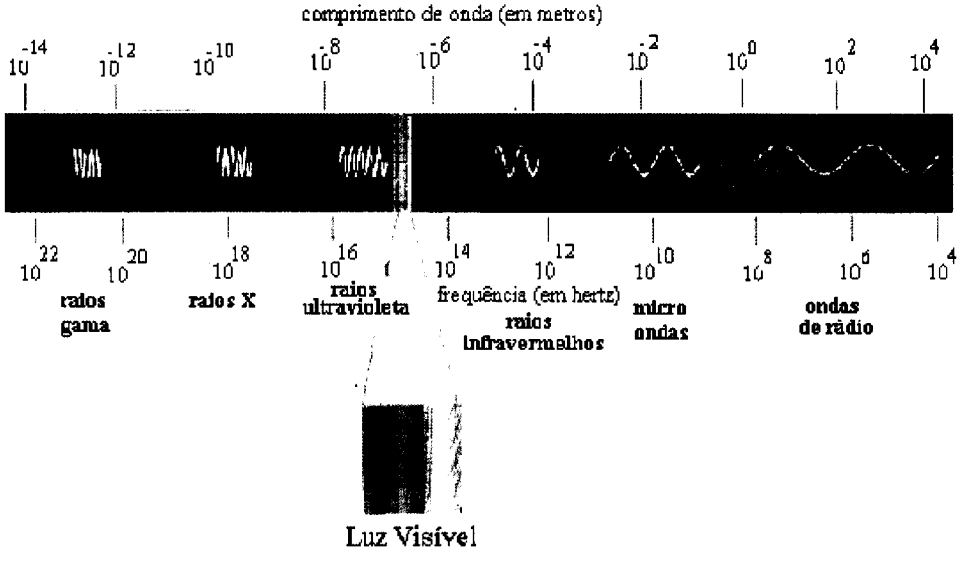

Figura A.1: O espectro electromagnético (fonte: *http://nautilus.fis.uc.pt/softc/Read_c/a_experimentais/espectro.htm*)

A.2 As imagens

Diferentes materiais reflectem e emitem radiação electromagnética de forma diferente. Por exemplo no visível a erva reflecte predominantemente verde e o asfalto tem baixa reflectância em todo o visível, parecendo-nos por isso preto (ou cinzento escuro). Os sensores colocados a bordo de satélites de observação da Terra tem normalmente um conjunto de bandas de comprimentos de onda a que são sensíveis. Um sensor com uma banda na zona do azul, uma no verde e outra no vermelho poderia produzir imagens coloridas semelhantes ao que os nossos olhos observam. No entanto a utilização da zona do azul é pouco comum devido à dispersão da atmosfera ser muito elevada nesta zona do espectro electromagnético. Por outro lado, a inclusão de bandas no infravermelho permite obter informação adicional sobre a superfície da Terra. Se por exemplo se pretendesse distinguir os três tipos de cobertura diferentes, A, B e C indicados na figura A.2, observando a imagem na banda 4 seria possível diferenciar o tipo de cobertura A das outras duas (teria um tom mais claro). Analogamente, observando a imagem na banda 7 seria possível identificar com relativa facilidade a classe B. Esta distinção de coberturas seria concerteza mais difícil na banda 2, por exemplo, uma vez que nesta banda o grau de reflectância de cada umas das coberturas é muito idêntico.

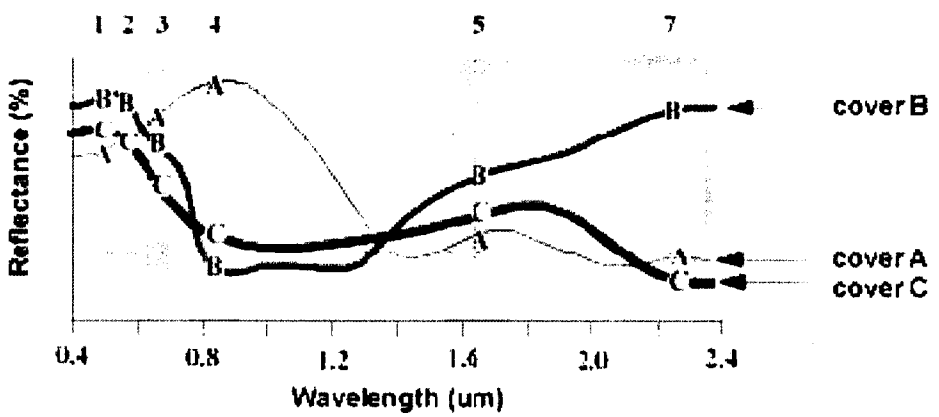

Figura A.2: Exemplos de assinaturas espectrais (*fonte: http://www.sc.chula.ac.th/ courseware/2309507/Lecture/remote18.htm*)

A natureza multi-espectral das imagens é importante para detectar diferenças que de outra forma seriam imperceptíveis. Por exemplo, numa fotografia aérea, um parque de estacionamento pintado de verde teria o mesmo aspecto que um campo com erva verde. Quando combinada com uma imagem infra-vermelho próxima do mesmo parque de estacionamento e campo, a erva iria aparecer muito mais brilhante do que o parque de estacionamento devido à maior reflectância do infra-vermelho próximo na erva. Estendendo este exemplo a outra banda, o parque de estacionamento pode emitir mais radiação infravermelha térmica (ser mais quente) do que a erva. Observando a imagem térmica a diferença entre a erva e o parque de estacionamento seria óbvia.

Referências

[1] Kristin P. Bennet and Colin Campbell. Support Vector Machines: Hype or Hallelujah? *SIGKDD Explorations*, 2:1–9, 2000.

[2] T. Blaschke and J. Strobl. What´s wrong with pixels? Some recent developments interfacing remote sensing and GIS. *Geographic Information Systems*, 6:12–17, 2001.

[3] Christopher J. C. Burges. A tutorial on Support Vector Machines for Pattern Recognition. *Data Mining*, 2:121–167, 1998.

[4] Campbell and B. James. *Introduction to Remote Sensing.* The Guildford Press, 1987.

[5] O. Chapelle and V Vapnik. Model selection for support vector machines. *Advances in Neural Information Processing Systems*, 12:230–237, 2000.

[6] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press, 2000.

[7] Grupo de Inventariação e Modelação de Recursos Florestais. *Normas de fotointerpretação para as áreas de demonstração da Associação Florestal do Vale do Sousa.* Universidade Técnica de Lisboa - Instituto Superior de Agronomia - Dep. de Engenharia Florestal, 2002.

[8] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[9] eCognition Users Guide. *eCognition Object Oriented Image Analysis.* Definiens Imaging, http://www.definiens-imaging.com, 2001.

[10] C. Saunders et al. Support Vector Machine Reference Manual. 1998.

[11] T. Evgeniou, T. Poggio, M. Pontil, and A. Verri. Regularization and statistical learning theory for data analysis. *Computational Statistics and Data Analysis*, 2002.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: data mining, inference, and prediction.* Springer, 2001.

[13] Marti A. Hearst. SVMs - a pratical consequence of learning theory. *IEEE Inteligent Systems*, pages 18–21, 1998.

[14] U. KreBel. Pairwaise classification and support vector machines. *Advances in Kernel Methods - Support Vector Learning, Cambridge, MIT Press*, pages 255–268, 1999.

[15] David Meyer. Support Vector Machines - The Interface to libsvm in package e1071. 2001.

[16] PCI Reference Manual. *PCI Geomatics.* http://www.pcigeomatics.com, 2000.

[17] J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. *Advances in Neural Network Information Processing Systems*, 12:547–553, 2000.

[18] R-project Reference Manual. *An Introduction to R.* http://www.r-project.org/, 2002.

[19] J. A. Richards. *Remote Sensing Digital Image Analysis - an Introduction.* Springer-Verlag, third edition, 1999.

[20] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[21] J. Schiewe, L. Tufte, and M. Ehlers. Potencial and problems of multi-scale segmentation methods in remote sensing. *Geographic Information Systems*, 6:34–39, 2001.

[22] B. Schölkopf. *Support Vector Learning.* Dissertation, Informatik der Technischen Universität Berlin, 1997.

[23] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel Eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[24] V. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

Anexos

- Esta rotina foi criada em Borland C++ e permite identificar, numa lista com os centróides dos objectos, as áreas de treino a partir de uma imagem segmentada com as áreas de treino aí assinaladas.

```
/*********************************************************************/
/*   id_treino.c                                                     */
/*                                                                   */
/*  Dada a imagem *.raw com áreas de treino e uma lista com as       */
/* do centróide de cada objecto, identificar os objectos à área      */
/* de treino.                                                        */
/*                                                                   */
/* JANETE BORGES                                    2 Set 2002       */
/*                                                                   */
/*********************************************************************/

#include "stdio.h"
#include "math.h"
#define nl 3340
#define np 206
int main()
{
FILE  *pxymilitar, *ptreino, *pout;
unsigned char treino[nl][np];
char fxout[100]="treino.txt";
char linha[200];
int i, x[51186], y[51186];

/****** Abrir ficheiros ******/
ptreino=fopen("TTAMask.raw","rb");
if (ptreino==NULL)
   {printf ("\n\nErro na abertura do ficheiro TTAMask_P4.raw ");}
pxymilitar=fopen("centros_xy.txt","r");
if (pxymilitar==NULL)
   {printf ("\n\nErro na abertura do ficheiro xy mlitares ");}
pout=fopen(fxout,"w");
if (pout==NULL)
   {printf ("\n\nErro na abertura do ficheiro saída ");}

/****** Ler ficheiros ******/
// Ler .raw
for (i=0;i<nl;i++)
   {
   fread(treino[i],10,np,ptreino);
   }
// Ler .txt
for (i=0;i<51186;i++)
   {
   fgets(linha,200,pxymilitar);
   sscanf(linha, "%d\t%d",&x[i],&y[i]);
   }

/****** Transformar as coordenadas militares xymlt em coordenadas xy ******/
for(i=0;i<51186;i++)
```

```
   {
   x[i]=floor((x[i]-171000+1)/15);
   y[i]=3340-floor((y[i]-443800+1)/15);
   }

/****** Identificar os objectos que são área de treino ******/
for (i=0;i<51186;i++)
   {
   fprintf(pout,"%d\t%d\n",i,treino[x[i]][y[i]]);
   }

/****** Fechar ficheiros ******/
fclose(ptreino);
fclose(pxymilitar);
fclose(pout);
return(0);
}
```

As rotinas que se apresentam de seguida foram implementadas em R1.6.1.

- Determinar as Componentes Principais do conjunto de treino e do conjunto de teste.

```
> Comp.Princ
function (treino,teste)
{
library(mva)
cp.treino1<-princomp(treino, cor=TRUE, scale=TRUE)
var<-cumsum((cp.treino1$sdev)^2/sum((cp.treino1$sdev)^2))
vectorespp<-cp.treino1$loadings[,var<0.95]
n.teste<-dim(teste)[1]
n.centros<-length(cp.treino1$center)
teste.normalizado<-(as.matrix(teste)-t(matrix(rep(cp.treino1$center,n.teste),n.centros,n.teste)))
                   /t(matrix(rep(cp.treino1$scale,n.teste),n.centros,n.teste))
cp.teste<-teste.normalizado%*%vectorespp
cp.treino<-cp.treino1$scores[,var<0.95]
return(cp.treino,cp.teste)
}
```

- Determinar as Variáveis Canónicas do conjunto de treino e do conjunto de teste.

```
> Var.Canonicas
function (treino,classe.treino,teste)
{
library(MASS)
var.can<-lda(treino,classe.treino)
vc.treino<-as.matrix(treino)%*%(normalizar(var.can[[4]]))
vc.teste<-as.matrix(teste)%*%(normalizar(var.can[[4]]))
return(vc.treino,vc.teste,var.can)
}
```

- Normalizar as colunas de uma matriz

```
> normalizar
function (matriz)
{
n<-dim(matriz)[1]
m<-dim(matriz)[2]
normas<-sqrt(rowSums(t(matriz^2)))
matriz.normalizada<-matriz/t(matrix(rep(normas,n),m,n))
return(matriz.normalizada)
}
```

- Gerar uma amostra aleatória

```
> amostra.aleatoria
function (n.conjt,X)
{
aux<-floor(length(X)/n.conjt)
a<-c(1:length(X))
ind<-rep(0,length(X))
for (i in c(1:(n.conjt-1)))
 {
 ind[sample(a[a!=0],aux)]<-i
 a[ind!=0]<-0
 }
return(ind)
}
```

- Normalizar uma matriz

```
> normalizacao
function (matriz)
{
n<-dim(matriz)[1]
m<-dim(matriz)[2]
medias.col<-colMeans(matriz)
matriz.centrada<-matriz-t(matrix(rep(medias.col,n),m,n))
normas<-sqrt(colSums(matriz.centrada^2))
matriz.normalizada<-t(t(matriz)-medias.col)/t(matrix(rep(normas,n),m,n))
return(matriz.normalizada,medias.col,normas)
}
```

- Normalizar o conjunto de teste com os parâmetros obtidos no conjunto de treino.

```
> normalizacao.teste
function (matriz, medias.col,normas)
{
n<-dim(matriz)[1]
m<-dim(matriz)[2]
matriz.normalizada<-t(t(matriz)-medias.col)/t(matrix(rep(normas,n),m,n))
return(matriz.normalizada)
}
```

- Estimar o erro de classificação obtido com o método dos K vizinhos mais próximos (para diferentes valores de K) por validação cruzada.

```
> variar.K
function (objs,classes,K,tipo.dados)
{
##
## tipo.dados:
## original
## comprin
## varcan
##
n<-length(K)
erros<-rep(1,n)
melhor.erro<-1
melhor.k<-0
classif<-rep(0,length(classes))
for (i in 1:n)
        {
        if (tipo.dados=="original")
                {
                kviz<-cross.validation.knn(objs,classes,K[i])
                print("original")
                }
        if (tipo.dados=="comprin")
                {
```

```
                kviz<-cross.valid.knn.cp(objs,classes,K[i])
                print("cp")
                }
        if (tipo.dados=="varcan")
                {
                kviz<-cross.valid.knn.vc(objs,classes,K[i])
                print("vc")
                }
        erros[i]<-kviz[[2]]
        if (kviz[[2]]<=melhor.erro)
                {
                melhor.erro<-kviz[[2]]
                melhor.classif<-kviz[[1]]
                melhor.k<-K[i]
                }
        print(i)
        }
return(melhor.k,melhor.erro,melhor.classif,K,erros)
}
```

- Validação Cruzada, com os dados originais, com o método dos k vizinhos mais próximos

```
> cross.validation.knn
function (objs,classes,k)
{
library(class)
n<-length(classes)
ind<-amostra.aleatoria(10,c(1:n))
classif<-factor(rep((1:9),c(574,rep(1,8))))
for(i in 0:9)
        {
        treino<-objs[ind!=i,]
        teste<-objs[ind==i,]
        treino.normalizado<-normalizacao(treino)
        treino<-treino.normalizado[[1]]
        teste<-normalizacao.teste(teste,treino.normalizado$medias.col,treino.normalizado$normas)
        classif[ind==i]<-knn(treino,teste,classes[ind!=i],k)
        }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Validação cruzada, nas componentes principais, com o método dos k vizinhos mais próximos

```
> cross.valid.knn.cp
function (objs,classes,k)
{
library(class)
n<-length(classes)
ind<-amostra.aleatoria(10,c(1:n))
classif<-factor(rep((1:9),c(574,rep(1,8))))
for(i in 0:9)
        {
        treino<-objs[ind!=i,]
        teste<-objs[ind==i,]
        cp<-Comp.Princ(treino,teste)
        classif[ind==i]<-knn(cp$cp.treino,cp$cp.teste,classes[ind!=i],k)
        }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Validação cruzada, nas variáveis canónicas, com o método dos k vizinhos mais próximos

```
> cross.valid.knn.vc
function (objs,classes,k)
{
library(class)
n<-length(classes)
ind<-amostra.aleatoria(10,c(1:n))
classif<-factor(rep((1:9),c(574,rep(1,8))))
for(i in 0:9)
        {
        treino<-objs[ind!=i,]
        teste<-objs[ind==i,]
        classe.treino<-classes[ind!=i]
        vc<-Var.Canonicas(treino,classe.treino,teste)
        classif[ind==i]<-knn(vc$vc.treino,vc$vc.teste,classe.treino,k)
        }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Estimar o erro de classificação obtido com Discriminação Logística por validação cruzada.

```
> cross.valid.RegLog
function (objs,classes,tipo.dados)
{
library(MASS)
library(nnet)
##
## tipo.dados:
## "original"
## "comprin"
## "varcan"
##
n<-length(classes)
ind<-amostra.aleatoria(10,c(1:n))
classif<-factor(rep((1:9),c(574,rep(1,8))))
for(i in 0:9)
        {
        treino<-objs[ind!=i,]
        teste<-objs[ind==i,]
        classe.treino<-classes[ind!=i]
        if(tipo.dados=="originais")
                {
                treino.normalizado<-normalizacao(treino)
                treino<-treino.normalizado[[1]]
                teste<-normalizacao.teste(teste,treino.normalizado$medias.col,
                        treino.normalizado$normas)
                }
        if(tipo.dados=="comprin")
                {
                cp<-Comp.Princ(treino,teste)
                treino<-cp$cp.treino
                teste<-cp$cp.teste
                }
        if(tipo.dados=="varcan")
                {
                vc<-Var.Canonicas(treino,classe.treino,teste)
                treino<-vc$vc.treino
                teste<-vc$vc.teste
                }
        {classif[ind==i]<-RegLog(treino,classe.treino,teste)}
        }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Classificação por Discriminação Logística

```
> RegLog
function (treino,classe.treino,teste)
{
classe.prob<-factor(c(1:9))
classes.exist<-classe.prob[summary(classe.treino)!=0]
nclasses.exist<-length(classes.exist)
classe.mais.freq<-which.max(summary(classe.treino))
classe.prob<-classes.exist[classes.exist!=classe.mais.freq]
nteste<-dim(teste)[1]
classif<-rep(0,nteste)
dados.treino<-data.frame(treino,classe.treino)
result.multinom<-multinom(classe.treino~.,dados.treino)
vcoef<-result.multinom$wts[result.multinom$wts!=0]
mcoef<-t(matrix(vcoef,(dim(treino)[2]+1),(nclasses.exist-1)))
prob<-mcoef[,2:(dim(treino)[2]+1)]%*%t(teste)
prob<-prob+matrix(rep(mcoef[,1],nteste),(nclasses.exist-1),nteste)
pmax<-max(prob[,1],0)
if(pmax==0){classif[1]<-classe.mais.freq}
if(pmax!=0){classif[1]<-classe.prob[which.max(prob[,1])]}
for (i in c(2:nteste))
        {
        pmax<-max(prob[,i],0)
        if(pmax==0){classif[i]<-classe.mais.freq}
        if(pmax!=0){classif[i]<-classe.prob[which.max(prob[,i])]}
        }
return(classif)
}
```

- Testar diferentes valores de C em SVMs com kernel RBF:

```
> variar.C
function (objs,classes,tipo.dados,C)
{
##
## tipo.dados:
## originais
## varcan
## comprin
##
melhorC<-10000
erro1<-1
m<-length(C)
plot(C,type="n",xlim=c(1,m),ylim=c(0,1))
result.anterior<-0
erroC<-rep(10,m)
for (j in 1:m)
   {
   result.actual<-cross.valid.C(objs,classes,tipo.dados,C[j])
   lines(c(j-1,j),c(result.anterior,result.actual[[2]]))
   result.anterior<-result.actual[[2]]
   erroC[j]<-result.actual[[2]]
   if(result.actual[[2]]<erro1)
      {
      erro1<-result.actual[[2]]
      melhorC<-C[j]}
      }
return(C,melhorC,erroC)
}
```

- Validação cruzada em SVMs com *kernel* RBF

```
> cross.valid.C
function (objs,classes,tipo.dados,custo)
{
library(mva)
library(e1071)
```

```
## tipo.dados:
## originais
## varcan
## comprin
##
n<-dim(objs)[1]
ind<-amostra.aleatoria(10,c(1:n))
classif<-factor(rep((1:9),c(574,rep(1,8))))
for (i in c(0:9))
  {
  teste<-objs[ind==i,]
  treino<-objs[ind!=i,]
  if(tipo.dados=="originais")
   {
   treino.normalizado<-normalizacao(treino)
   treino<-treino.normalizado[[1]]
   teste<-normalizacao.teste(teste,treino.normalizado$medias.col,treino.normalizado$normas)
   }
  classe.treino<-classes[ind!=i]
  if(tipo.dados=="comprin")
   {
   cp<-Comp.Princ(treino,teste)
   treino<-cp$cp.treino
   teste<-cp$cp.teste
   }
  if(tipo.dados=="varcan")
   {
   vc<-Var.Canonicas(treino,teste)
   treino<-vc$vc.treino
   teste<-vc$vc.teste
   }
  maq.svm<-svm(treino,classe.treino,kernel="radial",cost=custo)
  classif[ind==i]<-predict.svm(maq.svm,teste)
  }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Testar diferentes valores de γ em SVMs com *kernel* RBF:

```
> variar.gama
function (objs,classes,tipo.dados,c.optm,GAMA)
{
melhorGAMA<-10000
erro1<-1
m<-length(GAMA)
erroGAMA<-rep(10,m)
for (j in 1:m)
   {
   result<-cross.valid.RBFgama(objs,classes,tipo.dados,c.optm,GAMA[j])
   erroGAMA[j]<-result[[2]]
   if(result[[2]]<erro1){
   erro1<-result[[2]]
   melhorGAMA<-GAMA[j]}
   }
return(c.optm,GAMA,melhorGAMA,erroGAMA)
}
```

- Validação cruzada em SVMs com *kernel* RBF, com C=c.optm

```
> cross.valid.RBFgama
function (objs,classes,tipo.dados,c.optm,gama)
{
library(mva)
library(e1071)
## tipo.dados:
```

```
## originais
## varcan
## comprin
##
n<-dim(objs)[1]
ind<-amostra.aleatoria(10,c(1:n))
classif<-factor(rep((1:9),c(574,rep(1,8))))
for (i in c(0:9))
  {
  teste<-objs[ind==i,]
  treino<-objs[ind!=i,]
  if(tipo.dados=="originais")
   {
   treino.normalizado<-normalizacao(treino)
   treino<-treino.normalizado[[1]]
   teste<-normalizacao.teste(teste,treino.normalizado$medias.col,treino.normalizado$normas)
   }
  classe.treino<-classes[ind!=i]
  if(tipo.dados=="comprin")
   {
   cp<-Comp.Princ(treino,teste)
   treino<-cp$cp.treino
   teste<-cp$cp.teste
   }
  if(tipo.dados=="varcan")
   {
   vc<-Var.Canonicas(treino,teste)
   treino<-vc$vc.treino
   teste<-vc$vc.teste
   }
  maq.svm<-svm(treino,classe.treino,kernel="radial",cost=c.optm,gamma=gama)
  classif[ind==i]<-predict.svm(maq.svm,teste)
  }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Testar diferentes graus em SVMs com *kernel* polinomial

```
> variar.grau
function (objs,classes,tipo.dados,GRAU)
{
n<-dim(objs)[2]
melhor.grau<-10000
erro1<-1
m<-length(GRAU)
plot(GRAU,type="n",xlim=c(1,m),ylim=c(0,1))
result.anterior<-0
erro.grau<-rep(10,m)
for (j in c(1:m))
        {
        result.actual<-cross.valid.pol.grau(objs,classes,tipo.dados,grau=GRAU[j])
        lines(c(j-1,j),c(result.anterior,result.actual[[2]]))
        result.anterior<-result.actual[[2]]
        erro.grau[j]<-result.actual[[2]]
        if(result.actual[[2]]<erro1){
                erro1<-result.actual[[2]]
                melhor.grau<-GRAU[j]}
        }
return(GRAU,melhor.grau,erro.grau)
}
```

- Validação cruzada com SVMs com *kernel* polinomial

```
> cross.valid.pol.grau
function (objs,classes,tipo.dados,grau)
```

```
{
library(mva)
library(e1071)
## tipo.dados:
## originais
## varcan
## comprin
##
n<-dim(objs)[1]
ind<-amostra.aleatoria(10,c(1:n))
classif<-factor(rep((1:9),c(574,rep(1,8))))
for (i in c(0:9))
  {
  teste<-objs[ind==i,]
  treino<-objs[ind!=i,]
  if(tipo.dados=="originais")
   {
   treino.normalizado<-normalizacao(treino)
   treino<-treino.normalizado[[1]]
   teste<-normalizacao.teste(teste,treino.normalizado$medias.col,treino.normalizado$normas)
   }
  classe.treino<-classes[ind!=i]
  if(tipo.dados=="comprin")
   {
   cp<-Comp.Princ(treino,teste)
   treino<-cp$cp.treino
   teste<-cp$cp.teste
   }
  if(tipo.dados=="varcan")
   {
   vc<-Var.Canonicas(treino,teste)
   treino<-vc$vc.treino
   teste<-vc$vc.teste
   }
  maq.svm<-svm(treino,classe.treino,kernel="polynomial",degree=grau)
  classif[ind==i]<-predict.svm(maq.svm,teste)
  }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Testar diferentes valores de C em SVMs com *kernel* polinomial

```
> variar.C.pol
function (objs,classes,tipo.dados,C,grau.optm)
{
n<-dim(objs)[2]
melhor.C<-10000
erro1<-1
m<-length(C)
plot(C,type="n",xlim=c(1,m),ylim=c(0,1))
result.anterior<-0
erro.C<-rep(10,m)
for (j in 1:m)
{
        result.actual<-cross.valid.pol.C(objs,classes,tipo.dados,custo=C[j],grau.optm=grau.optm)
        lines(c(j-1,j),c(result.anterior,result.actual[[2]]))
        result.anterior<-result.actual[[2]]
        erro.C[j]<-result.actual[[2]]
        if(result.actual[[2]]<erro1)
         {
                erro1<-result.actual[[2]]
                melhor.C<-C[j]
                }
        }
return(C,grau.optm,melhor.C,erro.C)
}
```

- Validação cruzada em SVMs com *kernel* polinomial

```
> cross.valid.pol.C
function (objs,classes,tipo.dados,grau.optm,custo)
{
library(mva)
library(e1071)
## tipo.dados:
## originais
## varcan
## comprin
##
n<-dim(objs)[1]
ind<-amostra.aleatoria(10,c(1:n))
for (i in c(0:9))
{
   teste<-X[ind==i,]
   treino<-X[ind!=i,]
   if(tipo.dados=="originais")
   {
   treino.normalizado<-normalizacao(treino)
   treino<-treino.normalizado[[1]]
     teste<-normalizacao.teste(teste,treino.normalizado$medias.col,treino.normalizado$normas)
   }
   classe.treino<-classes[ind!=i]
   if(tipo.dados=="comprin")
   {
   cp<-Comp.Princ(treino,teste)
   treino<-cp$cp.treino
   teste<-cp$cp.teste
     }
  if(tipo.dados=="varcan")
   {
   vc<-Var.Canonicas(treino,teste)
   treino<-vc$vc.treino
   teste<-vc$vc.teste
  }
  maq.svm<-svm(treino,classe.treino,kernel="polynomial", cost=custo, degree=grau.optm)
  classif[ind==i]<-predict.svm(maq.svm,teste)
  }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```

- Testar diferentes valores de γ em SVMs com *kernel* polinomial

```
> variar.gama.pol
function (objs,classes,tipo.dados,custo.optm,grau.optm,GAMA)
{
n<-dim(objs)[2]
melhor.GAMA<-10000
erro1<-1
m<-length(GAMA)
plot(C,type="n",xlim=c(1,m),ylim=c(0,1))
result.anterior<-0
erro.GAMA<-rep(10,m)
for (j in 1:m){
        result.actual<-cross.valid.pol.gama(objs,classes,tipo.dados,custo.optm=custo.optm,
                        grau.optm=grau.optm,GAMA[j])
        lines(c(j-1,j),c(result.anterior,result.actual[[2]]))
        result.anterior<-result.actual[[2]]
        erro.GAMA[j]<-result.actual[[2]]
        if(result.actual[[2]]<erro1)
         {
          erro1<-result.actual[[2]]
          melhor.GAMA<-GAMA[j]
          }
```

```
        }
return(GAMA,custo.optm,grau.optm,melhor.GAMA,erro.GAMA)
}
```

- Validação cruzada em SVMs *com kernel* polinomial, com C=custo.optm

```
> cross.valid.pol.gama
function (objs,classes,tipo.dados,grau.optm,custo.optm,gama)
{
library(mva)
library(e1071)
## tipo.dados:
## originais
## varcan
## comprin
##
n<-dim(objs)[1]
ind<-amostra.aleatoria(10,c(1:n))
for (i in c(0:9))
  {
  teste<-X[ind==i,]
  treino<-X[ind!=i,]
  if(tipo.dados=="originais")
  {
   treino.normalizado<-normalizacao(treino)
   treino<-treino.normalizado[[1]]
   teste<-normalizacao.teste(teste,treino.normalizado$medias.col,treino.normalizado$normas)
   }
classe.treino<-classes[ind!=i]
  if(tipo.dados=="comprin")
   {
   cp<-Comp.Princ(treino,teste)
   treino<-cp$cp.treino
   teste<-cp$cp.teste
   }
  if(tipo.dados=="varcan")
   {
   vc<-Var.Canonicas(treino,teste)
   treino<-vc$vc.treino
   teste<-vc$vc.teste
   }
  maq.svm<-svm(treino,classe.treino,kernel="polynomial", cost=custo.optm, degree=grau.optm,
           gamma=gama)
  classif[ind==i]<-predict.svm(maq.svm,teste)
  }
erro<-sum(classes!=classif)/n
return(classif,erro)
}
```