Edgar Maciel Correia Pimenta

Abordagens para decomposição de problemas multi-classe: os Códigos de Correcção de Erros de Saída

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Dezembro / 2004

Edgar Maciel Correia Pimenta

Abordagens para decomposição de problemas multi-classe: os Códigos de Correcção de Erros de Saída

Tese apresentada à Faculdade de Ciências da Universidade do Porto
para obtenção do grau de Mestre em Informática

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Dezembro / 2004

Para a Marta e o Gustavo, pelo tempo que não lhes foi dedicado.

Para os meus País, por tanta coisa.

Ao Dr. João Gama um agradecimento especial pelo seu constante apoio e forte incentivo.

Resumo

Em muitos problemas de classificação o número de classes é superior a dois. Trabalhos recentes indicam algumas vantagens na decomposição desses problemas para problemas de apenas duas classes.

O métodos de decomposição mais utilizados e estudados são o todos-contra-todos (cada classe é testada face a cada uma das restantes, duas a duas), o um-contra-todos (cada classe é comparada simultaneamente com todas as restantes) e os códigos de correcção de erros de saída ou *ecocs* (cada classe é substituída por uma palavra binária).

Os *ecocs* surgiram no âmbito das telecomunicações pela sua capacidade de correcção de erros de transmissão. Assentam na capacidade de codificação com redundância de mensagens. Para permitir a sua utilização em problemas de classificação, devem ser respeitadas características específicas. Os *ecocs* são compostos por palavras binárias. Estas devem estar o mais distanciadas possível entre si. Para além disso, não podem existir colunas iguais ou complementares e nenhuma coluna pode ser formada apenas por 1 ou 0.

A obtenção de um *ecoc* é já por si um problema. Para além disso, é necessário ainda determinar o que é um bom e um mau *ecoc*. Nesta dissertação apresentam-se algumas propostas de solução para a obtenção de *ecocs* sendo, inclusive, apresentado um novo método - o algoritmo de perseguição. É também apresentada uma nova função que permite aferir da qualidade de um *ecoc*.

Mantém-se outra questão. As palavras binárias que formam os *ecocs* podem ter várias dimensões para um mesmo número de classes que se pretenda representar. O crescimento destas várias dimensões possíveis cresce de forma exponencial com o número de classes do problema multi-classe. Nesta tese são apresentados alguns métodos para definir dimensões para os *ecocs* que garantam um bom compromisso entre redundância e capacidade de correcção de erros.

Finalmente é realizado um conjunto de experiência para testar os métodos de criação de *ecocs*, para estudar o comportamento dos códigos de correcção de erros de saída para várias dimensões da palavra binária e comparados os resultados dos vários métodos de classificação referidos em vários conjuntos de dados.

Summary

Many classification problems have more than two classes. Recent work pints towards some advantages in decomposition of these problems into two classes problems.

The most used and studied decomposition methods are all-against-all (each class is tested face to each one of the remains, two by two), one-against-all (each class is compared simultaneously against all the remainings) and the error correction output codes or ecocs (each class is replaced by a binary word).

Ecocs appeared in the scope of the telecommunications thanks to the capacity to correct transmission erros. They are based in the capacity of messages codification with redundancy. To use then in classification problems, they must respect specific characteristic. Ecocs are binary words. These binary words must be further apart as much as possible. Equal or complementary columns cannot exist and no column can be formed only by 1 or 0.

Creating an ecoc is already a problem r. Moreover, it is necessary to determine what are good and bad ecocs. In this thesis are presented some proposals to the creation of ecoc and a new method is proposed - the persecution algorithm. It is also presented a new function that allows measuring the quality of an ecoc.

Other questions remains. The binary words that form the ecocs can have several dimensions for the same number of classes that if it intends to represent. The growth of these possible dimensions is exponential with the number of classes of the multi-class problem. In this thesis are presented some methods to choose the dimension of the *ecocs* that assure a good compromise redundancy/error correction.

Finally a set of experience is carried through to test the methods to create ecocs, to study the behavior of the error correction output codes for several dimensions of the binary word and compared the results of the some cited methods of classification using datasets.

Version Abrégée

Dans beaucoup de problèmes de classement le nombre de classes est supérieur à deux. Des travaux récents indiquent quelques avantages dans la décomposition de ces problèmes pour problèmes seulement de deux classes.

Les méthodes principaux de décomposition sont: tous-contre-tous (chaque classe est essayé avec tous les autres, deux à deux), et un-contre-tous (chaque classe est essayé avec toutes les autres au même temps) et les codes de rectification des erreurs de sortie ou ecocs (un mot binaire remplace chaque classe).

Les ecocs émergent avec les telecommunications par sa capacité de correction d'erreurs de transmission. Ils sont basés dans la capacité de codification avec redondance de messages. Leur utilisation dans les problèmes de classification exige prendre en attention leurs caractéristiques spécifiques. Les ecocs se composent de mots binaires. Ceux-ci doivent être éloignés le plus possible. En plus, il ne peut pas exister deux colonnes égales ou complémentaires et chaque colonne ne peut pas être composé exclusivement avec le 1 ou le 0.

Obtenir un ecoc est déjà en soi même un problème. En plus il faut déterminer qu'est ce que c'est un bon et un mauvais ecoc. Cette thèse présente quelques propositions de solution pour l'obtention d'ecocs, notamment avec la présentation d'une nouvelle méthode – l'algorithme de poursuite. Présente aussi une nouvelle fonction qui permet étalonner la qualité d'un ecoc.

En plus, pour un même numéro de classes à représenter, les mots binaires qu'intègrent les ecocs peuvent avoir plusieurs dimensions. La croissance des dimensions possibles est exponentielle avec le numéro des classes du problème multi-classe. Cette thèse présente quelques méthodes pour définir des dimensions pour ecocs qui garantissent un bon engagement entre redondance et une capacité de correction d'erreurs.

Enfin, on fait un ensemble d'expériences pour tester les méthodes de création d'ecocs; pour étudier le comportement des codes de rectification d'erreurs de sortie, pour plusieurs dimensions du mot binaire et on met en balance les résultats des plusieurs méthodes de classement rapportées dans plusieurs ensembles de données

Índice

Lista de abreviaturas e símbolos

$\lfloor x \rfloor$	Parte inteira de x arredondada para baixo
$\lceil x \rceil$	Parte inteira de x arredondada para cima
$\vec{a}$	Atributos do conjunto de dados
cw_f	Palavra binária resultante da predição dos vários modelos de predição bi-classe
a(k,e)	Função de aproximação
aval(k,e)	Função de avaliação de *ecocs*
B	Conjunto de dados bi-classe
b	Classificador bi-classe
$b_i(\vec{a})$	Classificação do modelo bi-classe ao exemplo $\vec{a}$
c	Classes do conjunto de dados original
c'	Classes do conjunto de dados bi-classe
cw	Palavra binária
D	Conjunto de dados do problema multi-classe
d(x)	Função de decomposição para transformar o problema original multi-classe em problemas bi-classe
d_{min}	Distância de Hamming mínima
e	Dimensão da palavra binária (usada em *ecocs*)
ecoc	Conjunto de palavras binárias
f	Classe resultante da predição final
F'	Conjunto de predições dos vários classificadores bi-classe
fit	Função de ajuste
k	Número de classes do problema original (multi-classe)
k'	Número de classificadores do problema bi-classe, resultante da decomposição
m	Classe principal do método um-contra-todos
me	Número máximo de erros corrigíveis
p	Probabilidade
pe(k,e)	Função que representa a percentagem máxima de predições erradas
qe(ecoc)	Função para determinar a qualidade de um ecoc
r(b)	Função de reconstrução para agregar a informação dos vários classificadores bi-classe

Índice de algoritmos

Índice de figuras

Índice de tabelas

1 Problemas de Classificação

Nos últimos anos, os sistemas de informação têm evoluído de forma exponencial. Cada vez são gerados e guardados mais dados em suporte digital e frequentemente de forma automática. A quantidade de informação residente nesses dados é de tal modo elevada que se torna cada vez mais difícil esses dados serem trabalhados manualmente e deles retirar informação útil, sobretudo em tempo igualmente útil. A solução passa por utilizar esses mesmos sistemas de informação para trabalhar os dados e deles retirar informações. Assim surge a aprendizagem automática.

O que se pretende com a aprendizagem automática é criar sistemas computacionais que sejam capazes de analisar factos e aprender descrições compactas e em compreensão desses factos. Noutras palavras, que sejam capazes de melhorar a sua performance com base nos resultados passados. Seria extraordinário se conseguíssemos, por exemplo, que um sistema de informação baseado em registos médicos fosse capaz de aprender tratamentos para novas doenças.

Uma das disciplinas da aprendizagem automática é a classificação de dados. Tendo-se diversos objectos com várias características/atributos e pertencendo cada um deles a um único grupo/classe, criar uma regra que seja capaz de prever o grupo a que pertence cada objecto, baseado nas suas características.

Para melhor se compreender os problemas de classificação, neste capítulo são apresentados os conceitos básicos associados. Posteriormente é referida em que contexto surge a decomposição de problemas de classificação e quais as motivações para a mesma. De seguida são apresentados alguns algoritmos utilizados em problemas de classificação de dados. Nas secções finais deste capítulo são apresentadas as principais contribuições desta dissertação e um resumo sobre a estrutura da mesma.

1.1 Conceitos básicos

Nos problemas de classificação de dados pretende-se, para um conjunto de exemplos baseado numa série de características (*atributos*), determinar a que grupo ou *classe* pertence esse exemplo (Tabela 1-1).

Alguns exemplos práticos:

- Dado num conjunto de exames médicos, determinar se um tumor é benigno ou maligno;
- Determinar, dado um conjunto de hábitos e características de uma pessoa se ela tem ou não alguma doença cardíaca;
- Tendo informações diversas sobre uma pessoa determinar se lhe deve ser concedido crédito;
- Conversão de texto manuscrito para formato digitalizado.

Considere um conjunto de objectos definidos por:

- $\vec{a}$ - características ou atributos
- um conjunto de grupos/classes C={c1,...,ci}

Suponhamos que existe uma função *f* desconhecida e que associa cada objecto a um grupo

$$f : \vec{a} \rightarrow c_i$$

O problema de classificação consiste em:

- dado um conjunto de treino definido por pares atributos-classes $< \vec{a}_i, c_i >^n$,
- Encontrar uma função $\hat{f} : \vec{a} \rightarrow c_i$

Tabela 1-1 – Definição do problema de classificação

Para se obter a classe a que pertence o exemplo com base nos atributos é utilizado um *modelo de decisão (*ou *classificador*). Esse modelo é criado através de um processo de aprendizagem feito com base em informação histórica (exemplos para os quais são conhecidos os atributos e a respectiva classe). Este histórico de exemplos define um *conjunto de treino.*

Esse conjunto de treino é aplicado a um algoritmo de aprendizagem que retorna um modelo de decisão. Esta é a fase de treino.

Novos exemplos não classificados são passados ao(s) modelo(s) de decisão. O conjunto de novos exemplos denominam-se *conjunto de teste.* Este conjunto terá o mesmo tipo de atributos do conjunto de treino mas a classe a que cada exemplo pertence é desconhecida, mas pertencente ao conjunto C.

A fase de teste consiste em obter predições da classe a que pertence cada exemplo do conjunto de teste.

Esquematicamente temos:

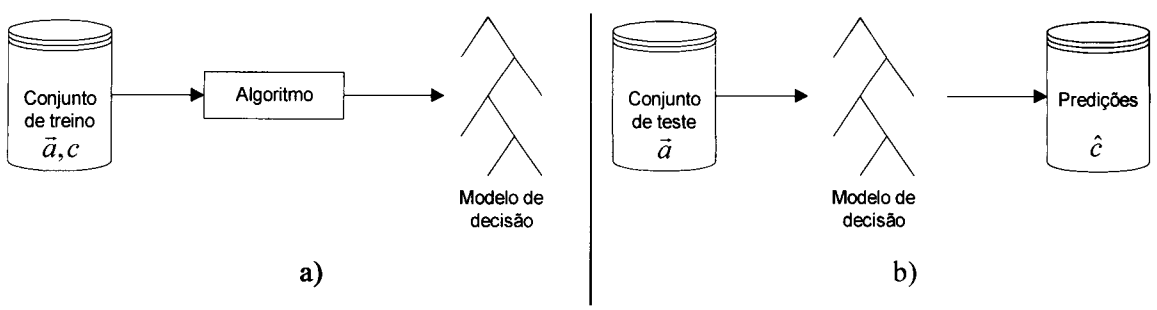


Tabela 1-2 – a) fase de treino b) fase de teste

1.2 Decomposição de problemas multi-classe

Os problemas de classificação reais apresentam normalmente um número de classes superior a dois. Diversas tentativas têm sido feitas no sentido de decompor os problemas multi-classe em problemas bi-classe, seguindo o critério "dividir e conquistar".

Esta decomposição apresenta diversas vantagens. A saber:

- **utilização de algoritmos específicos** – existem algoritmos que só conseguem tratar problemas com duas classes. Através da decomposição de problemas multi-classe em bi-classe passaria a ser possível a sua utilização. É o caso do Perceptron [19], do SVM [18] e do UFFT [4]. Existem ainda algoritmos que, embora sejam capazes de tratar problemas multi-classe, internamente usam mecanismos que implicam a criação de dois conjuntos de classes usualmente designados por super-classes (ex.: critério de selecção *twoing rule* do CART [3], agrupamento de atributos discretos do CART [3] e critério de selecção do QUEST [37]).

- **custo de erros** – em alguns tipos de problemas de classificação, diferentes erros de classificação podem ter custos distintos. No entanto, a maior parte dos métodos de classificação assume como objectivo a redução do número de erros, independentemente do seu custo. Um método possível para converter um classificador baseado na minimização do número de erros num classificador sensível ao custo dos erros é a *stratification* [38]. Este método consiste em alterar a frequência das classes no conjunto de treino de forma proporcional ao seu custo. No entanto, este método é apenas aplicável de forma eficiente a problemas bi-classe. Em problemas com mais de duas classes podem ser aplicadas heurísticas com perda de rigor [39].

- **facilidade de multiprocessamento** – uma vez que um único problema é dividido em vários problemas mais simples e independentes entre si, torna-se mais fácil a disseminação desses em diversos processadores em paralelo;

- **maior simplificação** – em termos de raciocínio humano, é de mais fácil percepção a opção entre duas hipóteses do que entre mais de duas. Para além disso, os modelos de classificação criados tenderão a ser também mais simples e consequentemente mais perceptíveis;

Finalmente refira-se que a decomposição em problemas bi-classe pode ser feita independentemente do algoritmo de classificação usado, uma vez que essa decomposição é feita com base no tratamento das classes a montante. Aliás, foi já realizado trabalho de decomposição de problemas multi-classe com algoritmos tão distintos como Ripper [2], C5.0 [2], C4.5 [3], CART [3] e UFFT [4].

1.3 Algoritmos de classificação

Conforme já foi referido anteriormente, existem diversos algoritmos de classificação. A apresentação de todos seria demasiado exaustiva, pelo que se optou por descrever de forma sucinta os algoritmos de classificação que foram utilizados no âmbito desta dissertação.

Assim, nesta secção é descrito de forma resumida o funcionamento das árvores de decisão genéricas e mais especificamente do C4.5 [3] e das máquinas de suporte vectorial (SVM) [18].

É também apresentado o funcionamento de um algoritmo genérico, embora este seja utilizado apenas de forma pontual no decorrer desta dissertação.

1.3.1 Árvores de decisão

As árvores de decisão são grafos dirigidos onde cada nó representa um teste no valor de um atributo, os arcos representam o resultado desse teste e as folhas que designam as classes. Na Figura 1-1 é apresentada uma árvore de decisão genérica.

A classificação de um exemplo é feito com este percorrendo a árvore desde a sua raiz (nó 1 na Figura 1-1) até chegar a uma folha. Essa folha corresponderá à classe predita para esse exemplo. Por exemplo: no nó 3 e consoante o resultado do teste aí executado o exemplo a classificar poderá ser da classe A, C ou D.

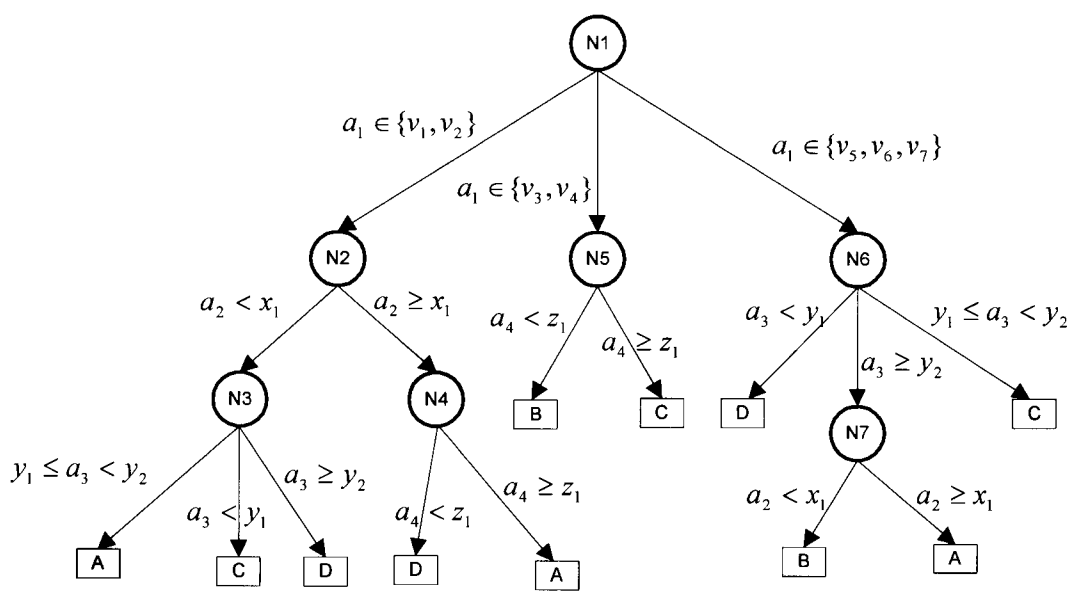


Figura 1-1 – Exemplo de árvore de decisão

Para a construção de uma árvore de decisão usando o esquema *top-down* é utilizado o algoritmo genérico do Algoritmo 1-1.

ENTRADA: Conjunto de exemplos de treino

SAÍDA: Árvore de decisão

Função GeraÀrvore(D)

Se atingido critério de paragem (D)

Cria uma folha

Senão

Selecciona o atributo $\vec{a}_i$ que maximiza função de selecção

Para cada Partição dos exemplos segundo os valores do atributo de $\vec{a}_i$

Aplica GeraÁrvore(D_k)

Algoritmo 1-1 – Pseudo código para criação de uma árvore de decisão.

Conforme foi referido, o algoritmo apresentado é genérico. No entanto, existem algumas características que distinguem umas árvores das outras, nomeadamente:

- Os testes nos nós podem dizer respeito a apenas um atributo ou vários em simultâneo;
- Cada teste pode resultar em duas hipóteses de escolha ou em mais de duas. No caso de todos os nós apresentarem apenas duas escolhas, estamos perante uma árvore de decisão binária;
- Os atributos podem ser discretos ou contínuos. Alguns algoritmos são apenas capazes de tratar atributos discretos enquanto outros tratam atributos discretos e contínuos.

A escolha do atributo a ser testado bem como o valor de teste é decisivo para a criação de boas árvores de decisão. Tipicamente, a escolha é feita considerando a impuridade do nó.

A impuridade de um nó mede a proporção de exemplos de classes diferentes nesse nó, ou seja, se um nó tem metade de exemplos da classe A e metade da classe B, a impuridade desse nó é 0,5. Se um nó (folha) tem exemplos de apenas uma classe, a sua impuridade é 0. Consideremos $p(c|t)$ como a proporção de exemplos da classe c no nó t. A impuridade é medida com base nessa proporção:

$$I(t) = phi(p(1 \mid t),..., p(c \mid t)) \qquad (1)$$

Assim, considerando $I(t)$ como a impuridade do nó t, a redução de impuridade pela aplicação do critério de divisão s (num teste binário onde E representa o filho esquerdo e D o filho direito) será

$$\Delta_{phi}(s,t) = phi(t) - p_E phi(t_E) - p_D phi(t_D) \qquad (2)$$

O critério de divisão *s* divide o nó *t* nos nós t_E com impuridade $phi(t_E)$ e t_D com impuridade $phi(t_D)$. Assim, pretende-se escolher um critério que maximize a redução da impuridade.

A função *phi* de impuridade de um nó apresentada é genérica. Dois exemplos de funções de impuridade típicas são:

- Entropia $phi(t) = -\sum p(c \mid t) \cdot \log(p(c \mid t))$
- Gini $phi(t) = -\sum p(c \mid t)^2$

Uma vez escolhido o atributo, é necessário definir o critério de divisão. Existem diversos métodos, consoante o algoritmo utilizado.

Um dos critérios de divisão utilizado no CART [3] é o *twoing rule*. Consiste em agrupar as classes semelhantes em dois conjuntos de super-classes. A decisão do atributo e do valor é aplicado a estas duas super-classes como se de apenas duas classes se tratassem.

O CART pode também agrupar valores de atributos discretos. O teorema apresentado em [3] é válido apenas para duas classes, pelo que, para ser utilizado em problemas multi-classe, implica a criação de duas super-classes, de forma semelhante ao descrito para o *twoing*.

O QUEST [37] também usa um critério de divisão baseado na análise discriminante que pressupõem a criação de duas super-classes, uma vez que esse critério é apenas aplicável a 2 classes.

O algoritmo C4.5 [26] têm como vantagem a capacidade de tratamento de atributos contínuos, implementando um teste do tipo: atributo $\leq$ valor ou atributo > valor

1.3.2 Maquinas Suporte Vectorial

As Maquinas Suporte Vectorial (SVM) [18] são actualmente um dos métodos de classificação mais em "moda". Uma das razões que conduziu a essa popularidade é uma mudança do critério de optimização. Tipicamente, os algoritmos de classificação procuram definir uma superfície de decisão que minimize o erro de classificação. Nas SVM procura-se definir uma superfície de decisão que maximize a margem, definida como a distância entre a superfície de decisão e os vectores de suporte.

Considere-se os exemplos da Figura 1-2. Um discriminante linear típico tentaria separar ambos os grupos de exemplos, não existindo qualquer preocupação extra. Assim, para esse discriminante era equivalente a divisão ser feita conforme exposto em a) ou em b). Já uma SVM faria a divisão assegurando a maior margem possível, logo b). A ideia é separar as classes mas tentando garantir que novos exemplos tinham uma margem de "segurança" que lhes garantisse uma correcta classificação.

Conforme referido, os discriminantes lineares nas SVM garante a maior margem possível entre as classes. Para isso, as SVM definem zonas convexas compostas pelos exemplos do mesmo grupo. A margem é maximizada maximizando a distância entre os pontos mais próximos de classes distintas (designados de vectores de suporte).

Considerando esta definição rapidamente se compreende a razão pela qual as SVM são apenas aplicáveis a problemas bi-classe.

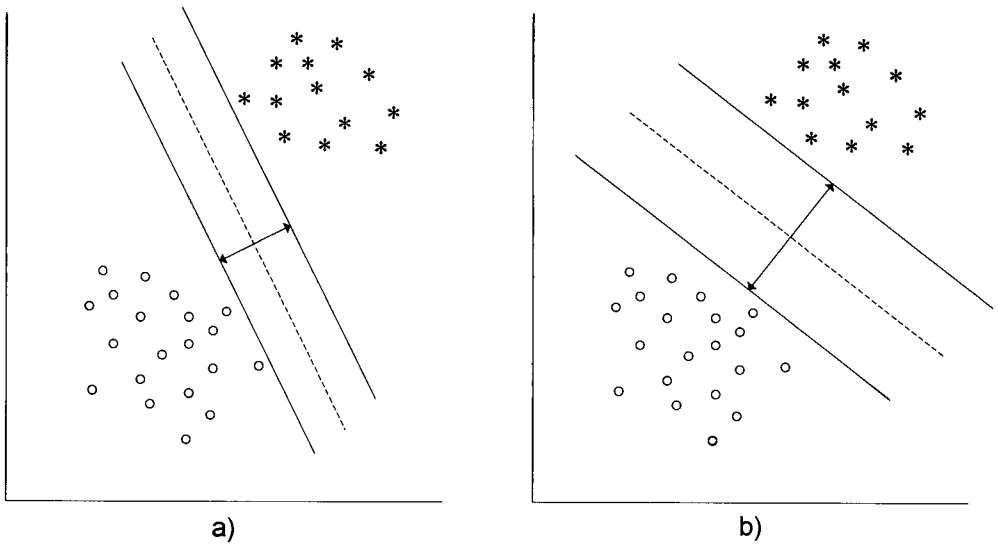


Figura 1-2 – Separação de duas classes por um hiper-plano. Dois exemplos: a) Minimizar o erro b) Maximizar a margem

As SVM podem usar classificadores lineares (conforme apresentado na Figura 1-2) ou não lineares, quando as classes não são linearmente separáveis. Nestas situações, são utilizados *kernels*. Os *kernels* são funções que permitem mapear pontos não linearmente separáveis e sitos num espaço R^N em novos pontos linearmente separáveis situados num espaço R^M (M>>N) procurando sempre maximizar a margem entre eles. A utilização de *kernels* implica que os problemas utilizados sejam bi-classe.

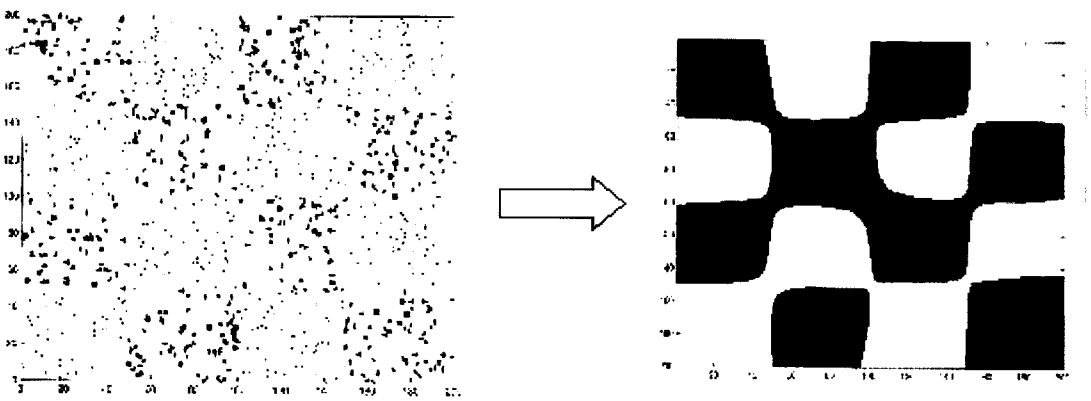


Figura 1-3 - Separação de classes com uma SVM baseada numa função não linear ($(x \cdot y + 1)^p$, com p=8). Tabuleiro de xadrez [21] .

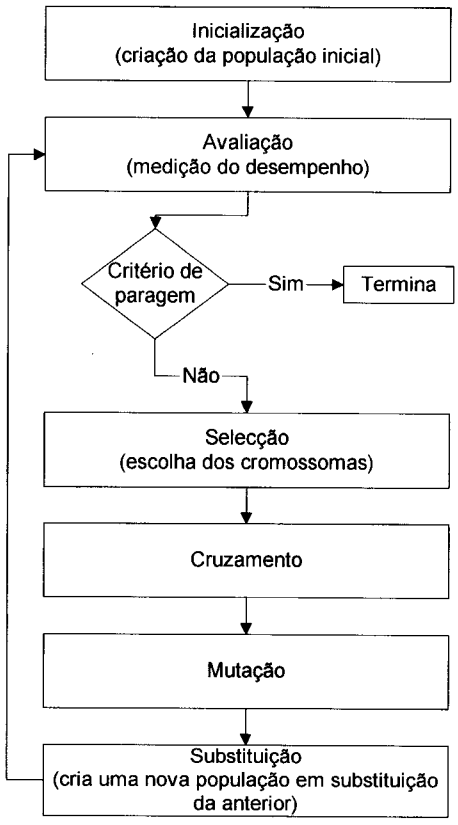
1.3.3 Algoritmos genéticos

Algoritmos genéticos são uma família de modelos computacionais inspirados na evolução natural: os mais fortes apresentam maior probabilidade de sobreviver e de se reproduzirem. Nos algoritmos genéticos pretende-se o mesmo: obter a melhor resposta para um problema. Para tal, na procura dessa mesma resposta, apenas as melhores respostas intermédias sobrevivem e se reproduzem.

Resumidamente, uma implementação de um algoritmo genético genérico começa com a definição de uma população: conjunto de soluções possíveis, tipicamente geradas de forma aleatória. Cada uma destas soluções pode ser designada como cromossoma e é constituída por genes. Os genes são, em geral, binários (bits com valor 0 ou 1), conduzindo a que os cromossomas sejam representados como palavras compostas por bits.

A essa população é aplicada uma fase de evolução que consiste no cruzamento (ou recombinação) e mutação parcial da mesma. No final de uma evolução é obtida uma nova população que apresentará um conjunto de soluções que pretende responder melhor ao problema inicial, podendo ser aplicada nova evolução a essa população. No final, obtém-se nova população geralmente de dimensão semelhante à inicial, mas, idealmente, com um melhor conjunto de cromossomas.

A escolha de uma entre as diversas soluções possíveis que vão sendo geradas é feita com base numa função de avaliação/ajuste *fit()*, função esta dependente do tipo de problema que foi codificado. Note-se que o objectivo será minimizar ou maximizar esta função de avaliação.



Algoritmo 1-2 – Algoritmo genético

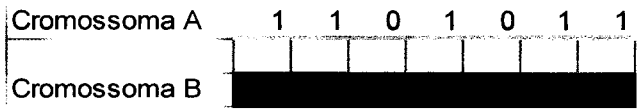
Conforme já referido, a evolução da população nos algoritmos genéricos baseia-se nos seguintes operadores:

- Cruzamento

É o principal operador e consiste na troca de informação entre dois cromossomas.

São escolhidos dois cromossomas da população, sendo que esta escolha poderá ser aleatória ou resultado de uma função (nomeadamente probabilística). De seguida são seleccionados os genes para serem trocados de um cromossoma para outro, gerando dois novos cromossomas (descendentes). Esta selecção pode ser feita de forma individual ou agrupada. Exemplos:

Antes



Após o cruzamento

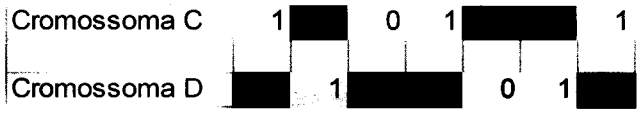


Figura 1-4- Exemplo de cruzamento num algoritmo genético

- Mutação

A mutação é um operador que tenta garantir a exploração de vários ramos de conjuntos de cromossomas distintos, evitando que o problema pare num máximo/mínimo[1] local.

Consiste em seleccionar os genes individualmente e calcular a probabilidade desse gene sofrer uma mutação (se 0, passa a 1; se 1, passa a 0).

Exemplo de mutação aplicada aos Cromossomas C e D do exemplo referido anteriormente (os genes marcados sofreram mutação):

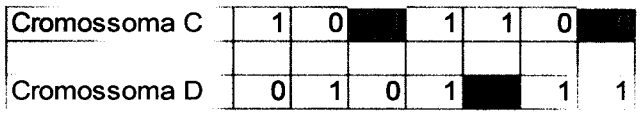


Figura 1-5- Exemplo de mutação num algoritmo genético

A probabilidade de mutação deve ser relativamente baixa. Quanto mais elevada for a probabilidade de mutação, mais nos aproximamos de uma pesquisa aleatória.

[1] Consoante se pretenda maximizar ou minimizar a função de avaliação.

1.4 Organização da dissertação

O foco desta dissertação é o estudo de códigos de correcção de erros de saída aplicados a problemas de classificação. Uma das propriedades dos códigos de correcção de erros de saída é a capacidade de corrigir erros. Esta capacidade provém do facto de os códigos de correcção de erros de saída usarem informação redundante, ou seja, usarem mais bits do que os necessários para codificar a mensagem.

Nesta tese definem-se algoritmos para a criação de códigos de correcção de erros de saída adaptados a problemas de aprendizagem automática. A ideia base da tese é encontrar códigos de correcção de erros de saída com um bom compromisso entre redundância e capacidade de correcção de erros.

1.4.1 Contribuições

As principais contribuições desta dissertação são:

Aferição da qualidade de códigos de correcção de erros de saída - um dos métodos possíveis para decomposição de problemas de classificação é a utilização de códigos de correcção de erros de saída (compostos por palavras binárias). Existem vários algoritmos para a criação destes mas, no caso específico de problemas de classificação de dados, não é referido como apurar se determinado códigos de correcção de erros de saída gerado é bom ou não. Neste dissertação apresenta-se uma função que permite aferir da qualidade de determinado código de correcção de erros de saída.

Método para a criação de códigos de correcção de erros de saída – a criação de códigos de correcção de erros de saída adaptados a problemas de classificação de dados é um problema em aberto. Nesta dissertação apresenta-se um novo método, o algoritmo de perseguição, para a criação destes códigos.

Selecção/simplificação de códigos de correcção de erros de saída – Para um determinado número de classes existem várias dimensões possíveis para os códigos de correcção de erros de saída. Simultaneamente, a quantidade de dimensões possíveis cresce de forma exponencial com o aumento do número de classes a decompor. Nesta tese são apresentadas várias funções que visam reduzir de forma considerável o número de tamanhos possíveis conduzido à escolha de uma dimensão ideal.

Análise comportamental dos códigos de correcção de erros de saída – pretende-se apresentar um estudo do comportamento dos códigos de correcção de erros de saída, tentando perceber o seu comportamento com a evolução da sua dimensão.

1.4.2 Organização

Esta dissertação é composta por 5 capítulos.

No capítulo 2 é efectuada uma introdução aos códigos de correcção de erros de saída, referindo o contexto em que surgem e as motivações para a sua criação. De seguida, são apresentados os principais métodos de decomposição de problemas multi-classe em problemas bi-classe. É feita uma análise comparativa entre os diversos métodos, apresentado as vantagens e desvantagens associadas a cada um deles.

No capítulo 3 são enumeradas e estudadas as principais características que os códigos de correcção de erros de saída devem ter para permitir a sua utilização em problemas de classificação. São também apresentados 4 métodos para a criação dos códigos de correcção de erros de saída, sendo dois adaptados de problemas de telecomunicações, um novo, criado no âmbito desta dissertação e um já utilizado anteriormente em problemas de classificação. É ainda definida uma função que permite de aferir da qualidade de um código de correcção de erros de saída. Finalmente é apresentado um método para selecção da dimensão do código de correcção de erros de saída, consoante o número de classe do problema a decompor.

No capítulo 4 são testados os métodos para criação de códigos de correcção de erros de saída. São também efectuadas experiências para estudo do comportamento dos códigos de correcção de erros de saída para diversas dimensões. Finalmente são apresentados comparativos entre os diversos métodos de decomposição de problemas multi-classe em problemas bi-classe, nomeadamente um-contra-todos, todos-contra-todos e códigos de correcção de erros de saída.

No capítulo 5 são apresentadas as conclusões finais do trabalho desenvolvido.

2 - "Estado da arte" de decomposição de problemas multi-classe

No contexto da aprendizagem automática, a decomposição de problemas multi-classe consiste em decompor um conjunto de dados com várias classes (3 ou mais) em vários conjuntos de duas classes (bi-classe). A cada um desses conjuntos de dados bi-classe é aplicado um algoritmo de classificação, obtendo-se vários modelos de decisão bi-classe b_i (um por cada conjunto). Esta é a fase de decomposição.

Posteriormente, na fase de teste, cada novo exemplo é classificado pelos vários modelos de decisão bi-classe criados anteriormente. Para cada exemplo tem-se um conjunto de predições. Esse conjunto de predições é aplicado a uma função de agregação com vista à obtenção da predição final. Esta é a fase de reconstrução.

Nesta secção é feita inicialmente uma introdução aos códigos de correcção de erros de saída, focando nomeadamente, o contexto que conduz ao seu surgimento.

Posteriormente são apresentados diversos métodos para efectuar a referida decomposição em problema bi-classe.

2.1 Breve introdução aos ecocs

A transmissão de informação sobre um canal acarreta o risco de aparecimento indesejável de ruído. Essa situação aplica-se a sistemas tão genéricos como sistemas de telecomunicações e de leitura de CDs (Figura 2-1). Com o desenvolvimento da tecnologia digital tornou-se possível ao receptor de uma mensagem detectar/corrigir erros que possam ocorrer no canal de transmissão. A ideia base é: ao invés de transmitir a informação na forma original, ela é previamente codificada. A mensagem codificada é enviada pelo canal, estando sujeita a ruído. No entanto, ao ser recepcionada é sujeita a um processo de descodificação e erros que possam ter ocorrido podem, eventualmente, ser corrigidos, obtendo-se a mensagem inicial.

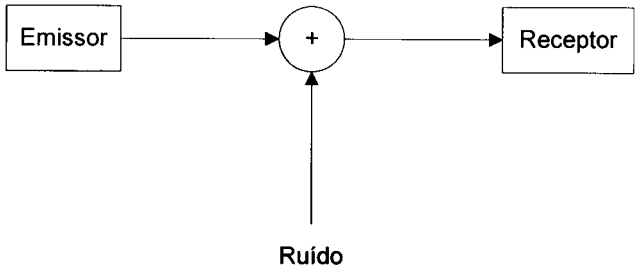


Figura 2-1 – Diagrama de um sistema de transmissão genérico

Para tentar assegurar que a mensagem inicial é recebida sem erros, o processo de codificação deve acrescentar redundância, isto é, utilizar mais informação do que aquela que seria originalmente necessária. Consequentemente, quanto maior a redundância, maior o número de erros de transmissão que podem ser corrigidos. Um caso muito simples de redundância é, dada uma mensagem binária, a repetição do envio de cada bit 3 vezes (Tabela 2-1).

Mensagem inicial:	1	0	0	1	1
Mensagem codificada enviada	111	000	000	111	111

Tabela 2-1 – Exemplo de redundância numa mensagem digital

Suponha-se que ocorre um erro na transmissão que alterava o oitavo bit da mensagem enviada, sendo a mensagem recebida a seguinte:

111	000	010	111	111

Facilmente o sistema de descodificação é capaz de reconstruir a mensagem inicial pois num segmento de três bits existem dois 0 e um 1.

Caso o erro ocorre-se no oitavo e no nono bit da mensagem transmitida, o sistema de descodificação não seria capaz de obter a mensagem inicial (uma vez que num segmento de três bits existiriam agora dois 1 e um 0). Para corrigir essa situação, seria necessária uma redundância ainda maior, de forma a obter-se igualmente maior a capacidade de correcção de erros.

O problema associado à redundância é a necessidade de mais recursos (tempo, largura de banda) para o envio da mensagem codificada do que para o envio da mensagem inicial (três vezes mais no caso exemplificado acima). Caso esses recursos fossem ilimitados, não existiria qualquer problema. Mas não o são, tornando-se necessário acrescentar redundância mas simultaneamente tentar obter rapidez.

É nesse contexto que surgem os *ecocs*. Na construção de *ecocs*, estes dois factores (redundância e rapidez) competem entre si. Em 1948, Claude Shannon [24] demostrou ser possível obter *ecocs* capazes de boas correcções de erros (boa redundância) e simultaneamente rapidez. Não nos diz, no entanto, como obter esses mesmos códigos. Pouco tempo depois da demonstração de Shannon, Richard Hamming [23] apresentaria uma solução que seria a base para o desenvolvimento desta área: a matriz de Hamming.

2.1.1 Matriz de Hamming

A matriz de Hamming (Figura 2-2) destina-se exclusivamente a codificar 4 bits de informação em palavras binárias de 7 bits, isto é, enviar 4 bits de informação utilizando 7 bits. A matriz de Hamming é capaz de detectar e corrigir um erro que ocorra na transmissão dessas palavras binárias de tamanho 7.

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figura 2-2 – Matriz de Hamming

A codificação é feita na resolução da equação

$$HC = 0 \tag{3}$$

onde C é um vector representando uma palavra binária e apresenta a seguinte estrutura:

$$C = \begin{bmatrix} x \\ y \\ d_1 \\ z \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \tag{4}$$

d_i representa cada um dos 4 bits da mensagem original. Através da equação (3) obtém-se um sistema linear de equações cuja resolução permite obter x, y e z.

Assim, a função de codificação c(d_i) será:

$$c(d_1, d_2, d_3, d_4) = (d_1 + d_2 + d_4, d_1 + d_3 + d_4, d_1, d_2 + d_3 + d_4, d_2, d_3, d_4) \tag{5}$$

A título de exemplo consideremos os 4 bits 1001 numa mensagem. A aplicação da função de codificação c(1,0,0,1) resultaria na palavra binária 0011**001**.

A função de descodificação seria simplesmente a extracção de 4 bits da palavra binária recebida (marcados a negrito no exemplo), precedida da aplicação da equação (3) para garantir que não teria ocorrido qualquer erro de transmissão.

Suponhamos que ao invés de ser recebida a palavra binária 0011001 era recebida a palavra binária 0010001.

A equação (3) não se verificaria, uma vez que

$$HC = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{6}$$

Logo, saber-se-ia que ocorreu um erro. Mas a matriz de Hamming também nos diz onde ocorreu o erro. Se "rodássemos" no sentido horário a expressão (6) obter-se-ia 100 que é a representação binária do número 4. Logo, sabíamos que tinha ocorrido um erro no quarto bit e poderíamos reconstruir a palavra binária para 0011001.

2.2 A decomposição em problemas bi-classe

Dado um problema de aprendizagem definido por um conjunto de exemplos D

$$D = < \{\vec{a}, c\}^n >, \text{c} \in \{c_1, c_2, ..., c_k\}, k>2 \tag{7}$$

D é decomposto em vários conjuntos bi-classe

$$d(D) = B_1 \wedge B_2 ... \wedge B_{k'} \tag{8}$$

onde

$$B_i = < \{\vec{a}, c'_i\}^m >, c'_i \in \{0,1\}, \text{i} \in \{1,...,k'\} \tag{9}$$

Esquematicamente:

$$\begin{matrix} & \vec{a} & & \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1j} \\ a_{21} & a_{22} & \dots & a_{2j} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nj} \end{bmatrix} \end{matrix} \begin{matrix} c \\ \begin{bmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{bmatrix} \end{matrix} \Rightarrow \begin{matrix} & \vec{a} & & \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1j} \\ a_{21} & a_{22} & \dots & a_{2j} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mi} \end{bmatrix} \end{matrix} \begin{matrix} c' \\ \begin{bmatrix} c'_1 & \dots & c'_{k'} \\ c'_2 & \dots & c'_{k'} \\ \dots & \dots & \dots \\ c'_m & \dots & c'_{k'} \end{bmatrix} \end{matrix}$$

Figura 2-3 – Decomposição em problemas bi-classe.

A cada problema B_i é aplicado um algoritmo de aprendizagem, gerando o modelo de decisão b_i.

$$b_i=\text{Algoritmo}(B_i) \tag{10}$$

Posteriormente, na fase de classificação, cada novo exemplo é aplicado a todos os b_i obtendo-se um conjunto de predições F'

$$F'=\{c'_1,c'_2,...,c'_k\},\ c' \in \{0,1\} \tag{11}$$

ao qual é aplicada a função de reconstrução para obtenção da predição final.

$$f=r(F'),\ f \in \{c_1,c_2,...,c_k\} \tag{12}$$

onde f é uma classe do conjunto de dados multi-classe.

Na literatura de estatística e aprendizagem automática, existem vários métodos para as fases de

- decomposição – $d(D)$
- reconstrução – $r(F')$

Neste capítulo apresentam-se esses vários métodos, bem como exemplos da aplicação dos mesmos. É ainda apresentada análise crítica dos métodos apresentados.

2.3 Fase de decomposição

Conforme já referido, na fase de decomposição pretende-se decompor um problema multi-classe em vários problemas bi-classe. Cada problema bi-classe terá associado um modelo de decisão construído segundo um determinado algoritmo e com base num conjunto de dados com duas classes B_i.

Nesta secção apresentam-se vários métodos referidos na literatura para essa decomposição.

2.3.1 Método Todos-contra-todos

O método de decomposição todos-contra-todos foi utilizado em [2], [5], [6], [9]. Consiste em criar conjuntos de dados com todas as combinações possíveis das classes do problema original, juntado-as duas a duas. Para cada uma das combinações é gerado um modelo de decisão. Um problema multi-classe com k classes é transformado em k' problemas bi-classe, cada um com o seu modelo de decisão b_i independente, onde

$$k'=\frac{k(k-1)}{2} \tag{13}$$

Cada problema bi-classe é definido por um conjunto de treino constituído pelos exemplos de apenas 2 classes. Por exemplo: para um problema original com 4 classes (A,B,C,D), seriam geradas 6 combinações e consequentemente 6 problemas bi-classe: A-B, A-C, A-D, B-C, B-D, C-D. O conjunto de treino de A-B consistiria apenas nos exemplos pertencentes às classes A e B. É aplicado um algoritmo de classificação a este conjunto, obtendo-se o classificador b_{AB}; o conjunto de treino de B-D teria apenas os exemplos das classes B e D. É aplicado o mesmo algoritmo de classificação a este conjunto, obtendo-se o classificador b_{DB}; etc. Cada modelo de decisão b_i é construído tendo por base um conjunto de treino de dimensão inferior ao conjunto de treino do problema multi-classe.

2.3.2 Método Um-contra-todos

Referido em [10], este método consiste em comparar cada uma das k classes do problema original contra todas as outras.

Um problema multi-classe com k classes é decomposto em k' problemas de duas classes: uma classe *principal* contra as restantes classes (todas elas denominadas como uma classe única - *outra*).

Teremos então:

$$k' = k \tag{14}$$

O método consiste em seleccionar todas as classes k do problema original uma a uma, considerando-a como sendo a classe principal (*m*). Cada conjunto de treino bi-classe será composto pelos exemplos dessa classe principal e pelos exemplos de todas as outras classes (neste caso, as diversas classes são substituídas por uma única – *outra*). Assim, os vários conjuntos de treino bi-classe tem a mesma dimensão do conjunto de treino multi-classe.

2.3.3 Método Códigos de correcção de erros de saída (*ECOC*'s)

Os códigos de correcção de erros de saída (*ecocs*) surgem no âmbito de telecomunicações como forma de tornar mais fiável a transmissão de informação digital. Foi adaptado a problemas de classificação, nomeadamente, em [3], [8], [9].

Consiste em substituir cada classe do problema original por uma palavra binária (*cw*) distinta, sendo que todas apresentam a mesma dimensão *e*. Por exemplo, para o caso de 3 classes teríamos:

Classe A ⇨ *cw*= 1 1 1

Classe B ⇨ *cw*= 0 1 0

Classe C ⇨ *cw*= 1 0 0

O conjunto de dados D inicial é agora decomposto em vários conjuntos de dados bi-classe $B_i = \{\vec{a}, c'_i\}$, onde i vai corresponder aos bits das palavras binárias. Exemplificando:

i=1 ⇨ Substituir as classes A e C por 1 e a classe B por 0;

i=2 ⇨ Substituir as classes A e B por 1 e a classe C por 0;

i=3 ⇨ Substituir a classe A por 1 e as classes B e C por 0;

É nestes "novos" exemplos que se vai aplicar o algoritmo de classificação para obtenção dos modelos de decisão.

Consequentemente, um problema multi-classe de k classes é substituído por k' problemas bi-classe onde

$$k'=e \tag{15}$$

, ou seja, o número de problemas bi-classe (e consequentemente de modelos de decisão) depende do tamanho da palavra binária. O valor mínimo de e é o número mínimo de bits necessários para a representação binária de k e é dado por:

$$e \geq \lceil \log_2 k \rceil \tag{16}$$

Uma vez que os conjuntos de treino para criação dos modelos de decisão vão usar como classes as colunas das diversas palavras binárias, dever-se-ão evitar:

i) colunas equivalentes ou complementares;

ii) colunas com apenas 1 ou 0.

Colunas equivalentes ou complementares representam o mesmo problema, dando origem ao mesmo modelo de decisão.

Colunas com apenas 1 ou 0 não representam problemas de decisão.

Com base nestas restrições, o tamanho máximo de e será:

$$e \leq \frac{2^k - 2}{2} = 2^{k-1} - 1 \tag{17}$$

2.4 Fase de reconstrução

Após a decomposição e treino obtêm-se k' modelos de decisão. Na fase de reconstrução, cada exemplo é classificado por todos os modelos, dando origem a k' predições distintas. Todas estas predições são bi-classe formam o conjunto de predições $F'=\{c'_1,c'_2,...,c'_{k'}\}$. Na fase de reconstrução agregam-se essas diversas predições para obter a predição final $f=r(F'), f \in \{c_1,c_2,...,c_k\}$.

Nesta secção apresentam-se vários métodos para a obtenção de f. Estes métodos estão intimamente ligados aos métodos apresentados na secção anterior.

2.4.1 Método Votação-directa

Esta função de reconstrução aplica-se ao método de decomposição todos-contra-todos, sendo referida em [2], [5], [6] e [9].

Após a decomposição e treino obtêm-se k' modelos de decisão. Cada exemplo é aplicado aos vários modelos b_i, resultando em k' predições.

A predição final resulta da aplicação da função de reconstrução $r(a)$

$$r(a) = \max\left[i \in \{1,...,k'\}, \sum_{j=1}^{k} (b_i(a) = j) \right] \qquad (18)$$

Basicamente, consiste na votação simples dos resultados dos diversos classificadores bi-classe, sendo que a classe mais votada seria a da predição final. No caso de empate, a selecção é feita aleatoriamente entre as classes empatadas.

2.4.1.1 Método Votação-directa com probabilidades

Na secção anterior apresentou-se um método de reconstrução que consistia, basicamente, na escolha da classe que ganhava mais comparações com as restantes classes. Uma variação seria a utilização de probabilidades apresentada em [5]. À semelhança do método anterior, também este método de reconstrução aplica-se ao método de decomposição todos-contra-todos. Requer, no entanto, que os classificadores usados retornem a probabilidade de um exemplo pertencer a cada classe.

Consideremos

$$p_{ij}=\text{Prob}(c_i|c_i \text{ ou } c_j),\ i \neq j \qquad (19)$$

como a probabilidade do exemplo ser da classe i, quando i é comparado com j. Obviamente, $p_{ij}=1-p_{ji}$. Cada modelo de decisão bi-classe retorna, para além de uma classe c'_i, a probabilidade do exemplo ser dessa classe.

Por exemplo, no caso de 3 classes teríamos 3 modelos de decisão:

- b_1 : A-B
- b_2 : A-C
- b_3 : B-C

Aplicando um exemplo a cada um dos modelos obteríamos, para cada modelo, uma de duas predições:

$b_1(a)= A, p_{A-B} \Rightarrow p_{B-A}=1-p_{A-B}$ $\vee$ $b_1(a)= B, p_{B-A} \Rightarrow p_{A-B}=1-p_{B-A}$

$b_2(a)= A, p_{A-C} \Rightarrow p_{C-A}=1-p_{A-C}$ $\vee$ $b_2(a)= C, p_{C-A} \Rightarrow p_{A-C}=1-p_{C-A}$

$b_3(a)= B, p_{B-C} \Rightarrow p_{C-B}=1-p_{B-C}$ $\vee$ $b_3(a)= C, p_{C-B} \Rightarrow p_{B-C}=1-p_{C-B}$

Com base nos diversos valores de p_{ij} obtém-se a matriz de probabilidades (exemplificado para o caso de 3 classes)

$$\begin{pmatrix} \cdot & p_{A-B} & p_{A-C} \\ p_{B-A} & \cdot & p_{B-C} \\ p_{C-A} & p_{C-B} & \cdot \end{pmatrix} \tag{20}$$

Define-se agora a probabilidade aproximada do exemplo ser da classe i como

$$\tilde{p}_i = \frac{2\sum_{i \neq j} p_{ij}}{k(k-1)} \tag{21}$$

Finalmente, a predição seria a classe com maior probabilidade aproximada

$$r(a) = \max(\tilde{p}_i) \tag{22}$$

2.4.2 Método Votação-distribuída

Este método de reconstrução aplica-se ao método de decomposição um-contra-todos e assume que cada classificador $b_i(a)$ selecciona uma classe c_i, i={1,...,k} ou a classe *"outra"*.

Para cada exemplo, e após a predição dos k' classificadores, a classe final predita é obtida através de uma votação segundo o seguinte algoritmo:

ENTRADA: $F'=\{c'_1,c'_2,...,c'_{k'}\}$
SAIDA: f **- classe predita**
para cada c'_i
  se $c'_i \neq$ ***"outra"***
    sc_i**+=1**
  senão
    sc_j**+=1/(k-1), j≠i**
máximo(sc)
sc – Pontos da classe

Algoritmo 2-1 – Algoritmo de reconstrução votação-distribuída.

Como se infere do Algoritmo 1-1, a classe predita final será a classe com mais pontos.

Resumidamente: cada exemplo é classificado k' vezes, obtendo-se uma classe c'_i por cada modelo de decisão. Para cada c'_i caso a classe predita fosse diferente da classe *"outra"*, então essa classe

acumula um ponto. Caso fosse *"outra"*, um ponto é distribuído equitativamente por todas as restantes classes. No final, a escolha recai na classe que tiver acumulado mais pontos.

Uma forma de votação ligeiramente diferente seria:

ENTRADA: $F'=\{c'_1,c'_2,...,c'_k\}$
SAIDA: f **- classe predita**
para cada c'_i
  se $c'_i \neq$ ***"outra"***
    sc_i**+=1**
  senão
    sc_i**-=-1**
máximo(sc)
sc – Pontos da classe

Algoritmo 2-2 – Alteração ao algoritmo de reconstrução de votação-distribuída

Resumidamente, se a classe predita for diferente de *"outra"*, então essa classe acumula um ponto. Caso contrário, perde um ponto.

2.4.2.1 Método Votação-distribuída com probabilidades

Com base no método de votação-directa com probabilidades (2.4.1.1) podemos aplicar as probabilidades ao método de reconstrução referido na secção anterior, logo sendo também este associado à decomposição um-contra-todos. Consideremos

$$p_{i\text{-outra}}=\text{Prob}(c_i|c_i \text{ ou outra}) \qquad (23)$$

como a probabilidade do exemplo ser da classe c_i, quando c_i é comparada com as restantes classes. Para cada uma das restantes classes a probabilidade será

$$p_{outra-i}=\frac{1-p_{i\text{-outra}}}{k'-1} \qquad (24)$$

Para o caso de 3 classes teríamos 3 modelos de decisão:

- b_1 : A-outra m=A
- b_2 : B-outra m=B
- b_3 : C-outra m=C

Cada classificador retornaria para cada exemplo a probabilidade de o exemplo ser da classe m:

$b_1(a)=A,p_{A\text{-}outra} \Rightarrow p_{outra\text{-}A}=(1-p_{A\text{-}outra})/2$

$b_2(a)=B,p_{B\text{-}outra} \Rightarrow p_{outra\text{-}B}=(1-p_{B\text{-}outra})/2$

$b_3(a) = C, p_{C\text{-}outra} \Rightarrow p_{outra\text{-}C} = (1 - p_{C\text{-}outra}) / 2$

Com base nesta informação constroi-se a matriz de probabilidades (exemplificada para k=3):

$$P = \begin{pmatrix} p_{A-outra} & p_{outra-B} & p_{outra-C} \\ p_{outra-A} & p_{B-outra} & p_{outra-C} \\ p_{outra-A} & p_{outra-B} & p_{C-outra} \end{pmatrix} \quad (25)$$

A diagonal da matriz corresponde probabilidades retornadas pelos classificadores.

A probabilidade aproximada do exemplo ser da classe c_i define-se como

$$\tilde{p}_i = \frac{\sum_{l=1}^{k'} P_{i,l}}{k'} \quad (26)$$

que não é mais do que o média das diversas probabilidades de um exemplo ser da classe i.

2.4.3 Distância de Hamming

Este método de reconstrução baseia-se na distância entre palavras binárias pelo que será o método adequado para a reconstrução de predições efectuadas com base em *ecocs*.

Após a aplicação dos k' modelos ao conjunto de teste, obtêm-se para cada exemplo uma palavra binária de comprimento e correspondente à junção da predição dos diversos modelos:

$$cw_f = b_1(a) \otimes b_2(a) \otimes \ldots \otimes b_{k'}(a) \quad (27)$$

$\otimes$ - Operador concatenação

Posteriormente calcula-se a distância desta palavra binária às palavras binárias definidas na função de decomposição, com base na distância de Hamming [3]

$$d(cw_f, cw_i) = \sum_{j=1}^{e} \left| cw_{f,j} - cw_{i,j} \right| \quad (28)$$

A classe predita seria aquela cuja palavra binária associada apresenta a menor distância a cw_f. Em caso de empate, a escolha recai aleatoriamente numa das classes empatadas.

$$r(a) = i \Rightarrow \min(d(cw_f, cw_i)) \quad (29)$$

2.4.3.1 Distância de Hamming esperada

Em [28] é apresentado um método para a utilização de probabilidades na fase de reconstrução, quando a decomposição é feita com ecocs.

Basicamente, cada modelo de decisão e para cada exemplo retornaria, para além da predição, a probabilidade associada a essa predição p. A distância de Hamming esperada será

$$d(cw_f, cw_i) = \sum_{j=1}^{e} p_j \left| cw_{f,j} - cw_{i,j} \right| \tag{30}$$

e a função de reconstrução mantém-se como na expressão (29)

$$r(a) = i \Rightarrow \min(d(cw_f, cw_i))$$

O trabalho apresentado em [27] refere um outro método para a utilização de probabilidades com ecocs. Este método baseia-se na resolução dum sistema de equações lineares, nomeadamente

$$C^T z = p \tag{31}$$

onde C corresponde à matriz dos ecocs definidos na fase de decomposição, z será o vector de probabilidades da predição final cujo valor se pretende obter e p é um vector de probabilidades associado a predição dos vários classificadores bi-classe.

A este sistema de equações estão também associadas duas restrições: $z_i \geq 0$ e $\sum z_i = 1$. Este sistema de equações é normalmente indeterminado uma vez que o número de equações é inferior ao número de variáveis. No entanto, e uma vez que p será um valor estimado, este sistema poderia ser considerado um sistema de inequações resolúvel recorrendo a métodos específicos, conforme referido em [27].

2.5 Outros métodos de decomposição e reconstrução

Miguel Moreira apresenta em [9], um método designado por "todos-contra-todos com correctores de classificação". Este método consiste em condicionar a solução dada pelo método de reconstrução votação directa (associado à decomposição todos-contra-todos) ao resultado de um conjunto de correctores de classificação. Para cada duas classes k_i e k_j combinadas no método todos-contra-todos, é criado um modelo de decisão para distinguir essas classes k_i e k_j das restantes. Na reconstrução, a votação do método todos-contra-todos é condicionada ao voto dos correctores de classificação. Suponhamos que no classificador bi-classe A vs B o exemplo é classificado como A. Se o corrector de

classificação (AB vs restantes) classificar o exemplo como AB, então o resultado do classificador binário A vs B é usado. Caso contrário, é ignorado.

Ainda em [9] é referido que os correctores de classificação aos quais se faz referência anteriormente podem, só por si, ser utilizados como método de decomposição. Este método corresponde a seleccionar todas as classes do problema original e gerar todas as combinações de classes tomadas duas a duas e compará-las com as restantes. É um método semelhante ao um-contra-todos mas ao invés de se utilizar apenas uma classe contra as restantes, utilizam-se duas classes contra as restantes. O método de reconstrução pode ser adaptado do método apresentado em 2.4.2: se a predição é efectuada numa das duas classes seleccionada, cada uma recebe um ponto, caso contrário, ambas perdem um ponto.

Em [30] é apresentado um método que consiste em criar um conjunto de árvores de decisão binárias (sistema de ninhadas de dicotomias) que divide de forma recursiva as classes do problema multi-classe em problemas de classificação bi-classe cada vez mais pequenos. Na Figura 2-4 são apresentados dois sistemas possíveis para k=4. Cada um destes sistemas representara um modelo de decisão.

Uma vez que existe um elevado número de sistemas passíveis de serem construídos e atendendo a que não existe à partida nenhuma razão para preferir uma árvore face a outra, a solução final é calculada com base num conjunto de árvores geradas aleatoriamente.

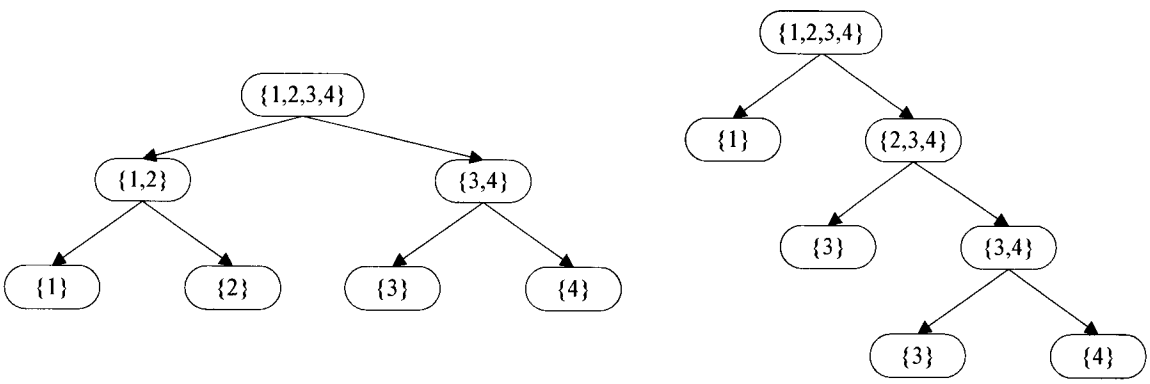


Figura 2-4 – Exemplo de sistema de ninhada de dicotomias para k=4

Em [35] é sugerida a utilização de DDAG – Decision Direct Acylic Graph. Estes grafos teriam implementariam um processo de classificação em que cada nó do grafo faria uma exclusão de uma de duas classes possíveis. Esse processo é repetido nos diversos nós do grafo até que fosse obtida uma solução, conforme exemplificado na Figura 2-5 para k=4.

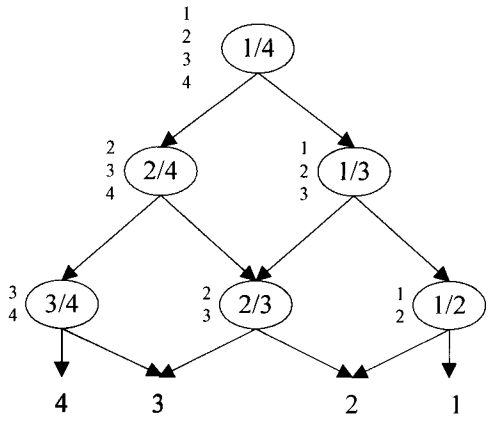


Figura 2-5 – Exemplo de DDAG para *k*=4

Este método apresenta diversas desvantagens, conforme referido em [34] Ainda em [34] é apresentada uma nova estrutura de grafos denominada ADAG - Adaptative Direct Acylic Graph. Esta estrutura é inversa à estrutura do DDAG, (Figura 2-6). Em cada nó é feita uma selecção entre duas classes. A classe seleccionada passa para a camada seguinte e é comparada com uma outra classe (que também terá vindo da classe acima). Essa selecção é feita sucessivamente até que seja atingida uma camada com apenas uma classe c_i, que será a classe predita. Embora os resultados deste método seja menos dependente da sequência de nós do que o método DDAG, ainda assim são afectados por essa sequência.

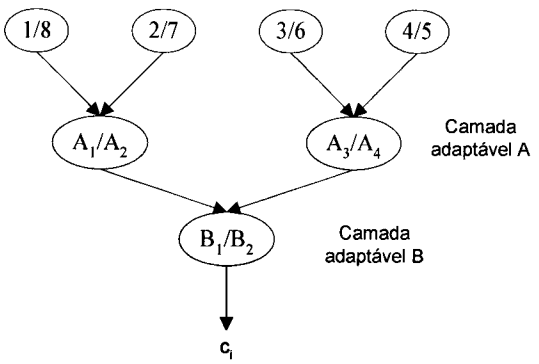


Figura 2-6 – Exemplo de ADAG para *k*=8

Para a melhoria destes métodos, é apresentada em [33] uma solução para a escolha dos melhores DDAG e ADAG com recurso a um algoritmo genético. É construída uma população de vários DDAG ou ADAG que vai sendo melhorada com base nos resultados de uma função de ajuste. A função de ajuste de cada indivíduo é dada pela percentagem média de acertos, para um determinado conjunto de dados. Ou seja, em cada iteração do algoritmo cada indivíduo é treinado e validado com um conjunto de dados. Seleccionam-se os indivíduos com maior percentagem de acertos. Este método permite obter DDAG ou ADAG adaptados ao conjunto de dados.

2.6 Exemplos

2.6.1 Método Todos-contra-todos

A aplicação do método de decomposição todos-contra-todos (2.3.1) a um problema original de quatro classes (A,B,C,D) implicaria a decomposição nas várias combinações possíveis: A-B, A-C, A-D, B-C, B-D e C-D.

Para cada uma das combinações é criado um modelo de decisão usando apenas exemplos das classes em causa como conjunto de treino, ou seja, para o problema A-B seriam usados os exemplos da classe A e os exemplos da classe B, para o problema A-C seriam usados os exemplos da classe A e os exemplos da classe C, etc.

Para a classificação de novos exemplos e posterior reconstrução utiliza-se o método exposto em 2.4.1. O conjunto de teste é constituído pelos exemplos de todas as classes, sendo este aplicado aos vários modelos de predição. Consequentemente, no caso da combinação A-B, exemplos que fossem das classes C ou D seriam classificados como sendo da classe A ou B.

Um resultado possível para um exemplo de teste em cada modelo seria:

↦ $b_i(a)$ ↴	
A-B	A
A-C	A
A-D	A
B-C	B
B-D	B
C-D	C

Votos para a classe			
A	B	C	D
3	2	1	0

$r(F')$=A

Tabela 2-2 – Exemplo de predições todos-contra-todos – 4 classes

Conforme se constata pelo exemplo, a classe predita no final é a classe A, uma vez que obtêm mais votos.

2.6.2 Método Um-contra-todos

Para um caso de 4 classes no problema original A, B, C, D, a decomposição pelo método um-contra-todos (2.3.2) conduz à criação de 4 problemas bi-classe: A vs *outras*, B vs *outras*, C vs *outras* e D vs *outras*.

Para cada um dos problemas, o conjunto de treino tem exemplos de todas as classes. No entanto, as classes que não fossem da classe *m* (associada ao modelo de decisão) seria consideradas como uma classe única - *outras*.

Por exemplo, para a classe *m*=A estaríamos no caso do problema A vs *outra*. Todos os exemplos de treino cuja classe não fosse a classe A seriam considerados como sendo da classe *outras*. Com base nesse conjunto de treino, é criado um modelo de decisão, sendo gerados um total de 4 modelos de decisão.

Cada exemplo do conjunto de teste é passado aos vários modelos de decisão. A reconstrução da predição final é feita utilizando, por exemplo, o método de reconstrução *r(F')* descrito no Algoritmo 2-1.

Um resultado possível para um exemplo de teste e em cada modelo seria:

↱ $b_i(a)$ ↴		votos			
		A	B	C	D
A vs outra	A	1	0	0	0
B vs outra	outra	0,33	0	0,33	0,33
C vs outra	outra	0,33	0,33	0	0,33
D vs outra	outra	0,33	0,33	0,33	0
Total		1,99	0,66	0,66	0,66

Predição	*r(F')*=A

Tabela 2-3 – Exemplo de predições um-contra-todos – 4 classes

Segundo esse método, a classe predita seria a classe A.

Poder-se-ia usar o método de reconstrução *r(F')* apresentado na Algoritmo 2-2, sendo o resultado apresentado na Tabela 2-4.

Também aqui o exemplo seria classificado como pertencendo à classe A.

↱ $b_{ij}(x)$ ↴		votos			
		A	B	C	D
A vs outra	A	1	0	0	0
B vs outra	outra	0	-1	0	0
C vs outra	outra	0	0	-1	0
D vs outra	outra	0	0	0	-1
Total		1	-1	-1	-1

Predição	$r(F')$ =A

Tabela 2-4 – Exemplo 2 de predições um-contra-todos – 4 classes

2.6.3 *Método ECOC's*

Utilizaremos o método de decomposição apresentado em 2.3.3 associado à função de reconstrução apresentada em 2.4.2.1. Para o caso das 4 classes do problema original substituímos cada classe por uma palavra binária distinta.

Comecemos pelo caso de e=2 bits (representação mínima). Teríamos então:

Classe original	cw_i
A	1 1
B	1 0
C	0 1
D	0 0

Tabela 2-5 – Representação mínima de *ECOC*'s para 4 classes

Dado que a representação efectuada resultou em palavras binárias de 2 bits, o conjunto de treino multi-classe é decomposto em dois conjuntos de treino bi-classe (um por cada coluna da matriz das palavras binárias). Com base nestes conjuntos e usando um mesmo algoritmo de classificação obtêm-se, no final da fase de treino, igual número de modelos de decisão.

O primeiro classificador (b_1) é obtido segundo o processo:

i)substituir no conjunto de treino as classes A e B por 1 e as classes C e D por 0;

ii) aplicação do algoritmo de classificação ao conjunto de treino para obtenção do modelo de decisão.

O segundo ($b_2(x)$) é obtido por processo semelhante:

i) substituir no conjunto de treino as classes A e C por 1 e as classes B e D por 0;

ii) aplicação do algoritmo de classificação ao conjunto de treino para obtenção do modelo de decisão.

Refira-se que estas substituições são efectuadas nos exemplos de treino aquando da fase de treino e posteriormente nos exemplos de teste (na fase de teste).

Para cada exemplo de teste, obtêm-se uma *palavra binária* correspondente à predição – cw_f. Essa *palavra binária* é obtida juntando:

- o primeiro bit correspondente à predição de $b_1(a)$ e;
- o segundo bit à correspondente de $b_2(a)$

Suponhamos: $b_1(a)=1$ e $b_2(a)=0$. Ter-se-ia como predição a palavra binária $cw_f=10$.

No exemplo e aplicando *r(x)* conclui-se que a palavra binária com menor distância à $cw_f=10$ é $cw_2=10$, que corresponde à classe B.

No enatnto, se a predição de $b_2(a)=1$, obteríamos a *palavra binária* $cw_f=11$ que corresponderia à classe A.

Vejamos agora o caso de 4 classes e palavra binária com $e=7$ [3]:

Classe original	cw_i
A	1111111
B	1110000
C	1001100
D	0101010

Tabela 2-6 – Representação *ECOC*'s para 4 classes com $e=7$

Na tabela seguinte serão apresentadas, para cada dos sete modelos de decisão a gerar, como são substituídas as classe do conjunto de treino. É também apresentada uma predição efectuada num exemplo do conjunto de treino.

		$b_1(a)$	$b_2(a)$	$b_3(a)$	$b_4(a)$	$b_5(a)$	$b_6(a)$	$b_7(a)$
A cw_1	Classe	1	1	1	1	1	1	1
B cw_2	original é	1	1	1	0	0	0	0
C cw_3	substituída	1	0	0	1	1	0	0
D cw_4	por	0	1	0	1	0	1	0
Predição cw_f		1	1	1	0	0	0	1

Tabela 2-7 – Predição de um exemplo de teste – $k=4$ e $e=7$.

Para a predição acima apresenta cw_f = 1110001 as distâncias de Hamming são:

$d(cw_f,cw_1)=3$

$d(cw_f,cw_2)=1$

$d(cw_f,cw_3)=5$

$d(cw_f,cw_4)=5$

A palavra binária com menor distância de Hamming a cw_f é a cw_2, correspondente à classe B. Logo, este exemplo seria classificado como sendo da classe B.

Suponha-se que a predição apresentada era agora cw_f = 1110011. As distâncias seriam:

$d(cw_f,cw_1)=2$

$d(cw_f,cw_2)=2$

$d(cw_f,cw_3)=6$

$d(cw_f,cw_4)=4$

Neste caso, duas palavras binárias apresentam a menor distância: cw_1 e cw_2 correspondentes respectivamente às classes A e B. O exemplo seria classificado como sendo da classe A ou B (selecção aleatória, conforme definido em 2.4.3).

2.7 Análise crítica dos diferentes métodos

Após a apresentação teórica dos diversos métodos bem como exemplos de aplicação, pretende-se nesta secção apresentar uma análise a cada um dos métodos, focando nomeadamente vantagens e desvantagens de cada um.

2.7.1 Método Todos-contra-todos

O número de problemas a classificar nesta abordagem é superior ao número de classes do problema original. Para além disso, k' cresce exponencialmente com k. No entanto, a criação dos modelos de decisão é simplificada pois os conjuntos de treino dos novos problemas baseiam-se em exemplos de apenas 2 classes. Garante-se assim que os conjuntos de treino B_i deste método apresentam uma dimensão inferior ao conjunto de treino do problema inicial multi-classe.

Considere-se um problema de 4 classes, A, B, C e D. A decomposição resultaria em 6 problemas: A-B, A-C, A-D, B-C, B-D, C-D. Caso um exemplo de teste fosse da classe A, seria sempre classificado erradamente nos seguintes casos: B-C, B-D, C-D, num total de 3. Suponha-se que, no exemplo da Tabela 2-2, $b_{A\text{-}B}(a)$=B. Nesse caso, a aplicação da função $r(F')$ resultaria na classificação do exemplo como sendo da classe B.

Para um problema de 5 classes (A, B, C, D e E) existiriam 10 combinações possíveis. Se o exemplo de teste fosse da classe A, em 6 combinações seria classificado erradamente (contra o acerto máximo de 4).

Caso se tratasse de um problema com 10 classes, o número de combinações seria de 45. Dessas, apenas 9 poderiam classificar correctamente o exemplo, sendo que as restantes 36 o classificariam erradamente.

Uma potencial desvantagem da utilização deste método reside no aumento de probabilidade de um exemplo ser mal classificado à medida que aumenta o número de classes *k* no problema original.

Número de classes *n* no problema original	Número de problemas *m* bi-classe gerados	Número de problemas que, garantidamente, farão classificação errada	Número máximo de problemas que farão classificação correcta
3	3	1	2
4	6	3	3
5	10	6	4
6	15	10	5
7	21	15	6
10	45	36	9
20	190	171	19
24	276	253	23
k	$\frac{k(k-1)}{2}$	$\frac{(k-1)(k-2)}{2}$	*k-1*

Tabela 2-8 – todos-contra-todos para *n* classes

Simultaneamente, à medida que o número de classes *k* aumenta, o número de problemas que garantidamente farão uma classificação errada aumenta igualmente e de forma exponencial. O esforço de cálculo é aplicado cada vez mais a resolver problemas que é sabido serem inúteis em termos de obtenção de uma classificação final correcta.

O método de reconstrução apresentado em 2.4.1.1 permite amenizar esta desvantagem através da utilização de probabilidades [5]. Por exemplo: se se estiver perante um exemplo da classe C e o modelo de decisão for $b_{A\text{-}B}(a)$, a predição será A ou B e obviamente errada. No entanto, espera-se que a probabilidade de ser A ou B seja relativamente semelhante, isto é, a diferença das probabilidades seja baixa. Já se o modelo de decisão for $b_{A\text{-}C}(a)$ espera-se que a probabilidade de ser da classe C seja muito superior à probabilidade de ser da classe A. Assim, para além de permitir uma diminuição no peso final de uma predição errada, todos os problemas contribuem para a solução final.

2.7.2 Método Um-contra-todos

Neste método, o número de problemas bi-classe é igual ao número de classes do problema multi-classe. Logo, o crescimento do número de problemas bi-classe é linearmente proporcional ao número de classes do problema multi-classe k. A dimensão do conjunto de treino B_i é igual para todos os problemas bi-classe. Simultaneamente, estes apresentam a mesma dimensão do conjunto de treino do problema multi-classe.

Para além disso, é possível garantir que todos os problemas contribuem para a obtenção da predição final. Esse foi o objectivo por detrás no método de votação atrás exposto (Secção 2.4.2), nomeadamente o apresentado no Algoritmo 2-1.

Não obstante, quando a distribuição das classes é muito desigual (pouco uniforme) este método apresenta algumas desvantagens. A existência de classes com baixa frequência implica que existam poucos exemplos de treino pertencentes a essas classes que permitam a construção dum modelo de predição suficientemente robusto. Mesmo nas situações de distribuições relativamente uniformes, será sempre uma classe contra k-1 classes, conduzindo a uma maior probabilidade de erro com o aumento do número de classes do problema original. Também neste caso, a utilização de probabilidades na fase de reconstrução (2.4.2.1) poderia amenizar estas desvantagens.

2.7.3 Método *ECOC's*

Neste método, a dimensão dos vários conjuntos de treino B_i é a mesma do conjunto de treino do problema multi-classe. O número de modelo de decisão bi-classe necessários é dado pelo tamanho da palavra binária e. Este valor está dependente do número de classes do problema original mas apenas para a definição dos limites mínimos e máximos (expressões (15) e (16)). Dentro deste intervalo, qualquer valor pode ser usado.

Se por um lado temos a vantagem de poder definir o número de problemas bi-classe a gerar no modelo, temos, por outro lado, um crescimento exponencialmente deste número com o número de classes k, complicando essa escolha. Sintetizando, temos:

k	*tamanho min e*	*tamanho max e*
3	2	3
4	2	7
5	3	15
6	3	31
7	3	63
10	4	511
24	5	8 388 607
k	$\lceil \log_2 k \rceil$	$2^{k-1}-1$

Tabela 2-9 – Evolução do tamanho máximo e mínimo da palavra binária

Consequentemente, uma das questões que surge associada à utilização *ecocs* é saber, dado um problema de k classes, qual o melhor tamanho para a palavra binária.

Para além disso, as palavras binárias (cw) devem respeitar algumas regras nomeadamente:

- inexistência de palavras binárias equivalentes

$$\forall_{i,j}, cw_i \neq cw_j \tag{32}$$

- inexistência de colunas iguais

$$\forall_{i,j}, cw^T{}_i \neq cw^T{}_j \tag{33}$$

- inexistência de colunas complementares

$$\forall_{i,j}, \left| cw^T{}_i - cw^T{}_j \right| \neq [1] \tag{34}$$

- nenhuma colunas apenas com 0 ou 1.

$$\begin{aligned} \forall_i, cw^T{}_i &\neq [1] \\ \forall_i, cw^T{}_i &\neq [0] \end{aligned} \tag{35}$$

Encontrar um conjunto de k palavras binárias de dimensão e que respeitem estas propriedades não é um problema trivial.

2.7.4 Discussão

A decomposição através do método um-contra-todos conduz a um menor número de problemas bi-classe do que no método todos-contra-todos, podendo-se obter aqui ganhos de eficiência tanto na criação dos modelos de decisão, bem como na aplicação desses modelos aos exemplos de teste.

A aplicação do método todos-contra-todos permite construir modelos de decisão mais eficientemente graças à utilização de conjuntos de treino reduzidos no número de exemplos. No entanto, o número de problemas bi-classe gerados é superior ao método um-contra-todos e cresce exponencialmente com k.

Em [2] demonstra-se que a complexidade do método todos-contra-todos é linear ao número de classes (e não exponencial como a expressão (13) faz crer) e, simultaneamente, é inferior ao do método um-contra-todos.

Considere-se o tempo necessário de aprendizagem de um determinado algoritmo para um conjunto de treino com n exemplos como uma função $f(n)$.

Defina-se como $g_{b|f}(k,n)$ a complexidade total resultante de usar um algoritmo b de complexidade $f(n)$ para decomposição de k classes.

Considere-se o custo como a performance de um algoritmo *a* num problema bi-classe decomposto a partir de problema de *k* classes (*k*>2) relativamente à performance do mesmo algoritmo num problema simples de duas classes e da mesma dimensão *n*:

$$\pi_{a|f}(k,n) = \frac{g_{a|f}(k,n)}{f(n)} \tag{36}$$

Para o método um-contra-todos, existem *k* problemas de aprendizagem, usando cada um deles a totalidade de exemplos *n*.

Então a complexidade é

$$g_{uct|f}(k,n) = kf(n) \tag{37}$$

E o custo será

$$\pi_{uct|f}(k,n) = k \tag{38}$$

No caso do método todos-contra-todos, cada exemplo será usado *k*-1 vezes, ou seja, uma vez em cada umas das *k*-1 que a classe a que o exemplo pertence será comparada com as restantes. Para um conjunto de treino tem dimensão *n*, o número total de exemplos utilizados será (*k*-1)*n*.

Para algoritmos de aprendizagem com complexidade linear no número de exemplos O(n)[2], temos que $f_1(n)=\lambda n$, isto é, o somatório da complexidade de diversos problemas é igual á complexidade de um problema único com a dimensão dos diversos problemas referidos:

$$\sum f_1(n_i) = f_1(\sum n_i) \tag{39}$$

A complexidade do método todos-contra-todos é:

$$g_{tct|f_1}(k,n) = f_1((k-1)n) = \lambda(k-1)n = (k-1)(\lambda n) = (k-1)f_1(n) \tag{40}$$

Logo, o custo é:

$$\pi_{tct|f_1}(k) = k-1 \tag{41}$$

Para o caso de árvores de decisão, com complexidade O(nlog(n)), $f_1(n)=nlog(n)$ e a complexidade do método todos-contra-todos é agora:

[2] Como por exemplo discriminante linear e Naive Bayes

$$g_{tct|f_1}(k,n) = f_1((k-1)n) = (k-1)n \cdot \log((k-1)n) = nf_1(k-1) + (k-1)f_1(n) \qquad (42)$$

O custo é:

$$\pi_{tct|f_1}(k) = k - 1\left(\frac{\log(k-1)}{\log(n)} + 1\right) \qquad (43)$$

Considerando que $n >> k \Rightarrow \frac{\log(k-1)}{\log(n)}$ tenderá para 0, pelo que $\pi_{tct|f_1}(k) \approx (k-1)$.

Conforme se pode concluir, o custo do método todos-contra-todos é inferior ao do método um-contra-todos e portanto mais vantajoso.

Resumidamente, o número de problemas bi-classe gerados em cada um dos 3 métodos será:

k	todos-contra-todos	um-contra-todos	*ecoc*	
			tam. min	tam. max
3	3	3	2	3
4	6	4	2	7
5	10	5	3	15
6	15	6	3	31
7	21	7	3	63
10	45	10	4	511
20	190	20	5	524 287
24	276	24	5	8 388 607
k	$\frac{k(k-1)}{2}$	k	$\lceil \log_2 k \rceil$	$2^{k-1}-1$

Tabela 2-10 – Comparativo do número de problemas bi-classe gerados nos diversos métodos

Note-se que o número de modelos de decisão bi-classe resultantes da decomposição do método um-contra-todos é inferior ao número de modelos de decisão resultantes da decomposição pelo método todos-contra-todos (excepto para k=3 onde é equivalente).

No caso dos *ecocs,* o número de modelos de decisão criados depende da dimensão das palavras binárias varia entre:

— mínimo: inferior ao da aplicação do método um-contra-todos

— máximo: superior ao do método todos-contra-todos (excepto para k=3 onde é equivalente).

Em termos de resultados na classificação, o método um-contra-todos poderá apresentar algumas vantagens caso a função de reconstrução *r(F')* utilizada seja a exposta no Algoritmo 2-1, uma vez que todos os problemas contribuem para a obtenção da predição final.

Para além disso, a utilização de probabilidades nos diversos métodos poderá melhorar o comportamento destes.

A grande vantagem dos *ecocs* está na versatilidade do número de problemas a gerar e na possibilidade de representar os outros métodos referidos (todos-contra-todos e um-contra-todos).

De facto, para o caso de 4 classes, poder-se-ia ter as seguintes palavras binárias de dimensão 6:

		$b_1(a)$	$b_2(a)$	$b_3(a)$	$b_4(a)$	$b_5(a)$	$b_6(a)$
A cw_1	Classe original é substituída por	1	1	1	0	0	0
B cw_2		-1	0	0	1	1	0
C cw_3		0	-1	0	-1	0	1
D cw_4		0	0	-1	0	-1	-1

Tabela 2-11 – Representação com palavras binárias do método todos-contra-todos

Se considerarmos que os bits a 0 não devem ser tidos em conta na criação do modelo de decisão estaremos presente um modelo tipo todos-contra-todos:

- $b_1(a)$ – Exemplo é da classe é A ou B
- $b_2(a)$ – Exemplo é da classe é A ou C
- $b_3(a)$ – Exemplo é da classe é A ou D
- $b_4(a)$ – Exemplo é da classe é B ou C
- $b_5(a)$ – Exemplo é da classe é B ou D
- $b_6(a)$ – Exemplo é da classe é C ou D

Neste caso, a função de reconstrução também deveria ser alterada, passando a ser:

$$d(cw_f, cw_i) = \sum_{j=1}^{e} \frac{\left| cw_{f,j} - w_{i,j} \right|}{2}, cw_{f,j}, cw_{i,j} \neq 0 \qquad (44)$$

Resultaria que quando os bits fossem iguais e ambos diferentes de zero, a distância de Hamming seria 0. Quando os bits fosses diferentes entre eles e simultaneamente diferentes de 0, a distância seria 1. Caso um dos bits fosse 0, não seria efectuado qualquer cálculo. É uma expressão que apresenta em termos práticos os mesmos resultados de expressão (28) ignorando as distâncias em que um dos bits seja 0.

Para o método um-contra-todos a representação por *ecocs* seria

		$b_1(a)$	$b_2(a)$	$b_3(a)$	$b_4(a)$
A cw_1	Classe	1	0	0	0
B cw_2	original é	0	1	0	0
C cw_3	substituída	0	0	1	0
D cw_4	por	0	0	0	1

Tabela 2-12 – Representação com palavras binárias do método um-contra-todos

Teríamos:

- $b_1(a)$ – Exemplo é da classe é A ou outra
- $b_2(a)$ – Exemplo é da classe é B ou outra
- $b_3(a)$ – Exemplo é da classe é C ou outra
- $b_4(a)$ – Exemplo é da classe é D ou outra

Neste caso específico, a função de reconstrução *r(F')* teria como base a distância de Hamming, e não uma votação (conforme a função de reconstrução apresentada em 2.4.2) fazendo com que o resultado de cada problema bi-classe fosse tratado de forma independente. Esta situação poderia conduzir a resultados ligeiramente diferentes do método um-contra-todos.

Os *ecocs* construídos de forma a representar o método um-contra-todos apresentam uma capacidade de correcção de erros nula, isto é, não são capazes de corrigir erros de classificação que eventualmente ocorram.

Logo, uma das vantagens dos *ecocs* provém da sua versatilidade e da capacidade de representar os outros dois métodos de decomposição apresentados.

Finalmente, em [29] é demonstrado que a decomposição por ecocs e reconstrução com base na distância de Hamming (associada a ecocs) permite reduzir tanto o desvio como a variabilidade das soluções.

3 Sobre a aplicação de ecocs a problemas de classificação

Neste capítulo são focadas as propriedades específicas que devem ser respeitadas para permitir a aplicação dos *ecocs* a problemas de classificação.

São também apresentados dois algoritmos para a criação de ecocs que surgem no âmbito de problemas de telecomunicações mas que foram adaptados nesta dissertação. É ainda apresentado um novo método para a criação de *ecocs*: o algoritmo da perseguição.

Finalmente serão apresentadas propostas de solução para as seguintes questões:

- Dado um problema de classificação com k classes, qual o melhor tamanho da palavra binária a ser utilizado na decomposição para *ecocs*?

- Como garantir que um conjunto de palavras binárias de dimensão e é o melhor que se pode obter para um problema de k classes?

- Dados dois ecocs da mesma dimensão e, como determinar em termos teóricos o melhor?

3.1 Os ecocs e a decomposição de problemas multi-classe

Um sistema de classificação de dados que utilize a decomposição de problemas multi-classe em problemas bi-classe pode ser considerado semelhante ao sistema da Figura 2-1. A codificação corresponde a uma função de decomposição genérica d, o ruído corresponde aos erros introduzidos pelo modelo de decisão e a descodificação da mensagem corresponde a uma função de reconstrução r.

A decomposição e treino usando este método corresponde a substituir as classes por uma *palavra binária* de comprimento e e criar os *vários* modelos de decisão (tantos quantos o número de bits de e). Na fase de teste utilizam-se esses mesmos modelos de decisão para se obter um conjunto de predições que formam uma *palavra binária*. A classe predita é aquela cuja *palavra binária* apresenta uma menor distância de Hamming – ver expressão (28) - à *palavra binária* obtida do conjunto das predições.

Conforme se pode inferir pelo exemplo exposto em 2.6.3, a alteração no resultado de uma das predições $b_i(a)$ corresponderia a que o conjunto das predições fosse diferente o que, sua vez, poderia conduzir a que a função de reconstrução apresentasse como resultado uma outra classe (no caso, uma classe incorrecta). Para minorar essas situações, e conforme já referido anteriormente, os *ecocs* devem apresentar duas características que competem entre si:

- redundância: a utilização de palavras binárias de tamanho superior ao mínimo necessário. Acrescentar redundância permite uma maior tolerância aos erros. Esta é a chave para correcção de erros;

- rapidez: o número de bits a utilizar deverá ser o menor possível de forma a garantir que o número de classificadores a criar/aplicar seja o menor possível.

No entanto, a redundância só por si não é garantia de uma maior tolerância a erros. Para se obter essa maior tolerância é necessário que as diversas palavras binárias sejam o mais distintas possíveis entre si. A melhor forma de efectuar essa aferição é através da distância de Hamming[3].

$$d(p, w_j) = \sum_{i=1}^{e} \left| p_i - w_{j,i} \right| \tag{45}$$

Aumentando simultaneamente a redundância e as distâncias de Hamming entre as palavras binárias obtêm-se um aumento à tolerância a erros.

Aliás, o número máximo de erros (*me*) que podem ser corrigidos para uma determinada distância de Hamming mínima entre palavras binárias *w* é:

$$me = \left\lfloor \frac{d_{\min}(p, w_j) - 1}{2} \right\rfloor \tag{46}$$

Um erro corresponde à alteração de um bit e, consequentemente, ao afastamento em uma unidade em relação à palavra binária correcta. Se o número de erros for igual ou inferior ao valor dado pela expressão (46), então a palavra binária mais próxima é a palavra correcta. Quando o número de erros for superior, então a palavra binária mais próxima já não será a correcta.

Face ao exposto, existiria a tentação de maximizar a redundância, maximizando *e* e consequentemente a tolerância a erros. No entanto, o aumento de *e* implica igualmente o aumento do número de classificadores a serem criados na fase de treino. Simultaneamente, aumenta o número de classificadores a que cada exemplo de teste deve ser submetido, diminuindo a rapidez de obtenção de uma predição final e consequentemente a eficiência do modelo.

Surge a questão de definir a melhor representação de *ecocs* que garanta um equilíbrio entre estes dois factores.

Antes de apresentar uma solução, é pertinente considerar algumas propriedades dos *ecocs* nos problemas de classificação e que são distintas da aplicação de *ecoc* em sistemas, por exemplo, de comunicações.

[3] Frequentemente e quando se esteja a fazer referência a *ecocs*, a expressão distância de Hamming será simplificada por simplesmente distância.

3.1.1 Propriedades específicas

Aplicação dos *ecocs* nos problemas de classificação implica a consideração de algumas propriedades. É importante maximizar a distância de Hamming mínima entre palavras binárias. Seria também conveniente obter um equilíbrio entre as diversas palavras binárias, de forma a garantir distâncias equivalentes, i.e., máxima distância mínima igual à média das distâncias mínimas ($\max(d_i) = \sum \frac{d_i}{n}$).

É também desejável que a distância de Hamming entre colunas seja o maior possível, levando à criação de modelos de decisão o mais distintos possível e cobrindo um maior espectro de hipóteses de solução e, simultaneamente, uma menor correlação entre os erros que possam ocorrer em cada modelo de decisão. Esta condição garante a inexistência de colunas equivalentes, uma vez que a distância de Hamming entre estas seria zero. A existência de colunas iguais é obviamente indesejável uma vez que colunas iguais corresponderiam a modelos de decisão também iguais.

Embora se pretenda maximizar a distância mínima entre colunas, deve-se garantir que a maior distância mínima obtida é inferior ao número de palavras binárias. Caso contrário estar-se-ia perante colunas complementares, situação igualmente indesejável. Colunas complementares resultam na criação de modelos de predição equivalentes. Consequentemente, um mesmo exemplo de teste classificado nesses dois modelos de decisão resultaria em predições iguais. Ou seja, o esforço de criação do modelo de decisão e o esforço de classificação não trariam qualquer vantagem.

Finalmente, nenhuma das colunas pode ser representada apenas por um bit (0 ou 1), pois, obviamente, não estaríamos presente um problema de decisão.

Um dos problemas centrais deste trabalho é o de desenvolver ecocs que satisfaçam estes critérios.

3.2 Criação de ecocs

3.2.1 Método segundo Dietterich

Conforme já referido, os ecocs surgiram no âmbito de sistemas de transmissão (nomeadamente telecomunicações). Nesse mesmo âmbito foram apresentados diversos métodos para a obtenção da solução, nomeadamente o código BCH [32] e o código Reed-Solomon [31].

No entanto, os *ecocs* aplicados a problemas de classificação apresentam características específicas (anteriormente enumeradas) inibindo a utilização directa destes métodos. É necessário utilizar métodos específicos ou adaptar métodos existentes.

Dietterich [3] apresentou a seguinte solução construção de *ecocs* para problemas de classificação (k – número de classes do problema):

- **Método exaustivo** - $3 \leq k \leq 7$

  Consiste em criar uma matriz de palavras binárias de dimensão $2^{k-1}-1$.

  A primeira linha é constituída por 1's. A segunda consiste em 2^{k-2} zeros seguidos de $2^{k-2}-1$ zeros.

  A linha i será composta alternadamente por 2^{k-i} zeros e 1's. A última linha será constituída por 0 e 1 alternados.

  Na Tabela 3-1 é apresentado um *ecoc* construído segundo este método, para k=4.

- **Subconjunto da matriz gerada pelo método exaustivo** - $8 \leq k \leq 11$

  É construída uma matriz usando o método exaustivo. Dessa matriz é seleccionado um subconjunto de colunas, obtendo-se vários *ecocs*. A dimensão do subconjunto de colunas e consequentemente das palavras binárias é definida como parâmetro do método. A selecção do conjunto de colunas é feita recorrendo ao algoritmo GSAT.

- **Método "Randomized Hill Climbing"** - $k > 11$

  De forma aleatória é construída uma matriz com k linhas e e colunas (*k palavras binárias* de dimensão *e*). Segundo Dietterich [3], os pares de palavras binárias geradas terão uma distância de Hamming com distribuição binomial de média $\frac{k}{2}$, gerando portanto códigos bastante razoáveis[4]. Para melhorar a matriz gerada de forma aleatória, o algoritmo selecciona repetidamente as palavras binárias com menor distância de Hamming entre elas e as duas colunas que apresentam distâncias mais extremas (máxima ou mínima). O cruzamento das selecções consiste numa matriz 2x2. Os bits dessa matriz 2x2 serão alterados de forma a melhorar a distância de Hamming entre linhas e colunas (Figura 3-1). Quando o algoritmo atinge um máximo local parte-se para uma selecção aleatória de pares de linhas e colunas, tentando assim obter novo máximo.

- **Método usando código BCH** - $k > 11$

  Dietterich [3] faz referência a este método (mais específico em telecomunicações) indicando no entanto três desvantagens:

  - A dimensão máxima da palavra binária é 64;
  - Nem sempre é possível obter boas distâncias de Hamming entre colunas;
  - O número de palavras binárias que é possível gerar é sempre uma potência de 2.

  Consequente, este método não será dos mais interessantes.

[4] Convém salientar que termos 2 pares de palavras binárias (ex: A e B, B e C) que apresentam uma boa distância de Hamming entre eles não garante bons *ecocs* (no limite, A e C podem ser equivalentes).

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Tabela 3-1 – *Ecoc* obtido pelo método exaustivo (*k*=4)

0 1 0 1	0 1 1 0
a) antes da aplicação do algoritmo	b) depois da aplicação do algoritmo

Figura 3-1 – Operação de melhoria com Randomized Hill Climbing

Dietterich [3] não justifica os valores de k utilizados nos intervalos para cada um dos métodos, podendo-se considerar que terá a ver com questões de complexidade, nomeadamente pelo crescimento exponencial da matriz criada pelo método exaustivo face ao crescimento de k (Tabela 3-2). Aliás, o tamanho da palavra binária nesse método é definido por $2^{k-1}-1$ e que corresponde igualmente à dimensão máxima das palavras binárias para decompor um problema de k classes (expressão (17)).

O método exaustivo apresenta, no entanto, uma característica interessante: garante a equidistância entre as diversas palavras binárias e simultaneamente a inexistência de colunas iguais ou complementares, bem como de colunas apenas com 0 ou 1. Pode ser aplicado para qualquer valor k, embora Dietterich [3] apenas o aplique de forma directa para $3 \leq k \leq 7$ e como base para $8 \leq k \leq 11$, uma vez que para valores superiores a matriz teria uma dimensão demasiado elevada.

k	e
3	3
4	7
5	15
6	31
7	63
10	511
20	524 287
24	8 388 607
k	$2^{k-1}-1$

Tabela 3-2 – Evolução do tamanho da palavra binária no método exaustivo

3.2.2 Algoritmo de Repulsa (RA[5])

Enrique Alba apresentou em [12] um método de construção de *ecocs* para problemas de telecomunicações denominado de Algoritmo de Repulsa (Algoritmo 3-1).

Este algoritmo é baseado na física e no comportamento de um conjunto de partículas carregadas com a mesma carga e o mesmo sinal sobre uma esfera. Nessa situação, as partículas afastar-se-ão uma das outras até que seja encontrado um ponto de equilíbrio. Esses movimentos são resultado das forças existentes que vão sendo exercidas entre elas, podendo ser matematicamente representados usando as leis de Coulomb.

A adaptação é feita considerando as palavras binárias de um *ecoc* como partículas sujeitas a forças, simulando o movimento destas na procura de um equilíbrio. Para tal, as palavras binárias de dimensão *e* são vistas como partículas num hipercubo de *e*-dimensões, cada uma colocada num vértice de *e* arestas. Cada movimento corresponde à deslocação numa das arestas, ou seja, a inverter um bit na *palavra binária.*

O RA calcula as forças entre as várias *palavras binárias* ($f_{ij,i\neq j}$) para posteriormente calcular a força total a que a *palavra binária i* está sujeita - F_i .

Esta força F_i vai ter duas componentes: uma normal (F_i^n) e uma tangencial (F_i^t). A componente normal não contribuí para o movimento nas arestas, pelo que não é considerada. É apenas considerada a componente tangencial. Uma vez que o movimento das palavras binárias está limitado às arestas (ao contrário das partículas que se movem livremente), é necessário verificar com que aresta a componente tangencial faz o menor ângulo (e consequentemente a maior força). Para isso, é definido um vector unitário associado a cada aresta e representado por a_i^d onde *i* representa a palavra binária e *d* a dimensão ou aresta. Através do produto interno de F_i^t e a_i^d obtêm-se

$$m_i^d = F_i^t \cdot a_i^d \tag{47}$$

Os valores de *i* e *d* para os quais m_i^d é maximizado representam a *palavra binária* e a dimensão/aresta na qual se fará o movimento (alteração do bit). Uma vez que próximo do equilíbrio m_i^d pode ainda ser superior a zero, define-se τ como o valor mínimo abaixo do qual não existirá movimento, isto é, só existe movimento se $m_i^d \geq \tau$. Um valor de τ elevado dificultará as deslocações para pontos de equilíbrio. Um valor de τ demasiado baixo implicará movimentos constantes, mesmo em situações próximas do equilíbrio, tornando-se mais difícil obter uma estabilização. O valor de τ utilizado em [12] e [13] foi de 0,001 pelo que esse será também o valor aqui utilizado.

[5] Repulsion Algorithm

Após um movimento, fica completa uma iteração do algoritmo. Os critérios de paragem do algoritmo podem ser:

- Atingido um número máximo de iterações pré-definido;
- Atingido um valor de ajuste definido;

Estes critérios podem ser usados isoladamente ou em conjunto.

while not StopCriterion **do**
 no_move = 0
 repeat
  if size(no_move) = M **then**
   exit
  end if
  repeat
   i = random(1,M)
  until not i **in** no_move

  for j=1 **to** M **do**
   if $j \neq i$ **and** $d_{ij} \neq 0$ **then**

$$f_{ij} = \frac{p_i - p_j}{d_{ij}\sqrt{d_{ij}}}$$

   end if
  end for

$$F_i = \sum_{j=1, j\neq i}^{M} f_{ij}$$

$$n_i = \frac{p_i - \frac{o_n}{2}}{\left| p_i - \frac{o_n}{2} \right|}$$

$$F_i^n = (F_i \cdot n_i) n_i$$

  max = 0
  dim[i] = 0
  for d = 1 **to** n **then**
   $m = F_i^t \cdot a_i^d$
   if $m \geq \tau$ **and** m > max **then**
    max = m
    dim[i] = d
   end if
  end for
  if dim[i] = 0 **then**
   add(no_move, i)
  end if
 until dim[i]>0
 move (p_i, dim[i])
end while

Algoritmo 3-1 – Pseudocódigo do algoritmo de repulsa (RA) conforme [13]

Nas diversas iterações, cada solução que vai sendo obtida vai sendo avaliada através de uma função de avaliação que se pretende maximizar. A função de ajuste utilizada em [12] e [13] foi

$$fit = \frac{1}{\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M}\frac{1}{d_{ij}^2}} + \left(\frac{d_{\min}}{12} - \frac{d_{\min}^2}{4} + \frac{d_{\min}^3}{6}\right) \tag{48}$$

A função apresentada tem dois argumentos:

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M}\frac{1}{d_{ij}^2}} \quad \text{e} \quad + \left(\frac{d_{\min}}{12} - \frac{d_{\min}^2}{4} + \frac{d_{\min}^3}{6}\right)$$

O primeiro argumento foi apresentado em [36] e permite medir quão bem colocadas estão as M palavras binárias num espaço de *n* dimensões. No entanto, e conforme demostrado em [12], esta função apresenta um inconveniente: pode resultar num valor mais alto da função de ajuste para um *ecoc* que tenha uma distância de Hamming mínima inferior a um outro *ecoc*, o que vai contra o pretendido. Para resolver esse problema, em [12] é proposto acrescentar o segundo argumento da função, obtendo-se finalmente a função de ajuste apresentada (expressão (48)). Esta função de ajuste aumenta com o aumento da distância mínima de Hamming dos *ecocs*.

No entanto, no caso de problemas de classificação, maximizar a função de ajuste não é suficiente para assegurar boas soluções, já que é preciso garantir outras propriedades (3.1.1). Assim, caso ocorra umas das seguintes situações:

- colunas iguais ou complementares;
- colunas com apenas 1 ou 0;

é subtraída à função de ajuste uma penalização e que terá como valor absoluto o dobro desta última, resultando na seguinte função de ajuste:

$$\begin{aligned} &fitfinal=fit-pen \\ &pen=2*fit*penalizar \\ &\quad \text{Se colunas\_iguais} \Rightarrow penalizar = 1 \\ &\quad \text{Se colunas complementares} \Rightarrow penalizar = 1 \end{aligned} \tag{49}$$

3.2.3 Algoritmo de Repulsa em Algoritmos Genéticos (GARA)

O GARA [12] e [13] é um algoritmo híbrido que combina um algoritmo genético e o algoritmo de repulsa. A combinação foi efectuada substituindo o operador mutação pelo algoritmo de repulsa.

Assim, durante a execução do algoritmo genérico, ao invés de ser efectuada uma mutação é efectuada uma iteração do RA.

Pretende-se deste modo cobrir de forma simultânea um maior espaço de possíveis soluções e evitar que o algoritmo de repulsa pare em máximos locais.

3.2.4 Algoritmo de Perseguição

Anteriormente descreveram-se três métodos para construção de *ecocs*, nomeadamente o apresentado por Dietterich e especifico para problemas de classificação e os apresentados por Enrique Alba, mais virados para problemas de telecomunicações mas adaptados no âmbito desta dissertação. Nesta secção apresenta-se um novo método para a criação de ecocs, o método da perseguição. Essa apresentação é precedida da apresentação da função de qualidade

3.2.4.1 Qualidade de um *ecoc*

Os métodos até agora apresentados permitem obter um conjunto de palavras binárias mas não se tem qualquer garantia (excepto para o código exaustivo) de que a solução obtida é a melhor. Aliás, em nenhum deles é apresentada qualquer resposta para esta questão.

Sabe-se que a melhor solução é aquela que maximiza a distância de Hamming mínima entre as palavras binárias, respeitando as restrições definidas em 3.1.1. Mas como é possível saber que esse máximo foi atingido?

Por exemplo, no caso de um problema de $k=n$ classes e para palavras binárias de dimensão $e=m$, qual é a melhor distância mínima entre palavras binárias que é possível obter?

Na procura da resposta, olhe-se com maior atenção para a matriz de *ecocs* gerada pelo método exaustivo. Esta matriz apresenta uma característica importante: corresponde à representação de todas as fronteiras de decisão possíveis para decomposição das classes do problemas original.

Na Figura 3-2 a) são apresentadas as fronteiras de decisão do problema multi-classe para o caso de $k=4$. Na(Figura 3-2 b) a h) apresentam-se todas as fronteiras de decisão de todos os problemas bi-classe possíveis. Na Tabela 3-3 é apresentada a correspondência entre as fronteiras de decisão dos problemas bi-classe, a sua representação gráfica e a respectiva coluna na matriz de *ecocs*

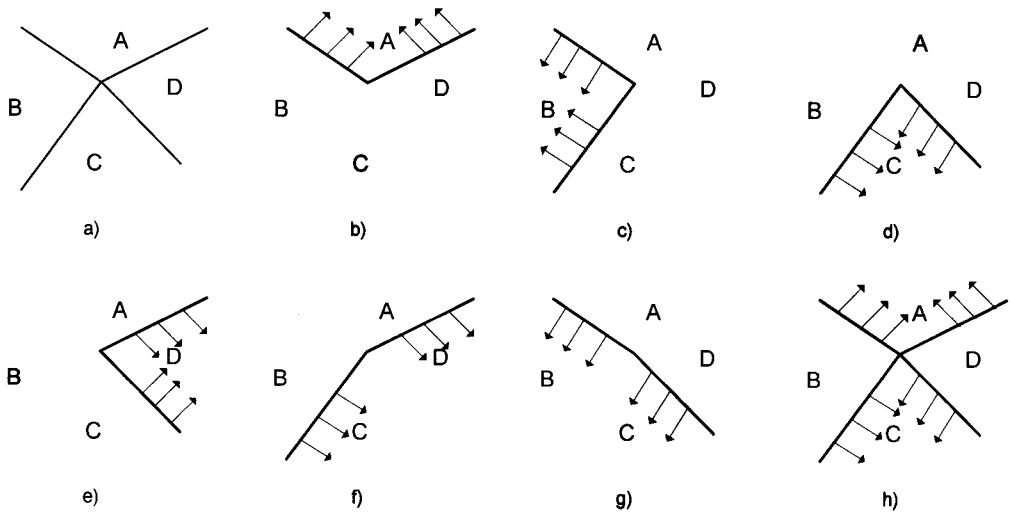


Figura 3-2 – Áreas de decisão para *k*=4

Problema bi-classe	Representação gráfica	b_i						
A vs B,C,D	Figura 3-2 b)	A	**1**	1	1	1	1	1	1
		B	**0**	0	0	0	1	1	1
		C	**0**	0	1	1	0	0	1
		D	**0**	1	0	1	0	1	0
B vs A,C,D	Figura 3-2 c)	A	1	1	1	**1**	1	1	1
		B	0	0	0	**0**	1	1	1
		C	0	0	1	**1**	0	0	1
		D	0	1	0	**1**	0	1	0
C vs A,B,D	Figura 3-2 d)	A	1	1	1	1	1	**1**	1
		B	0	0	0	0	1	**1**	1
		C	0	0	1	1	0	**0**	1
		D	0	1	0	1	0	**1**	0
D vs A,B,C	Figura 3-2 e)	A	1	1	1	1	1	1	**1**
		B	0	0	0	0	1	1	**1**
		C	0	0	1	1	0	0	**1**
		D	0	1	0	1	0	1	**0**
A,B vs CD	Figura 3-2 f)	A	1	1	1	1	**1**	1	1
		B	0	0	0	0	**1**	1	1
		C	0	0	1	1	**0**	0	1
		D	0	1	0	1	**0**	1	0
AD vs BC	Figura 3-2 g)	A	1	**1**	1	1	1	1	1
		B	0	**0**	0	0	1	1	1
		C	0	**0**	1	1	0	0	1
		D	0	**1**	0	1	0	1	0
AC vs BD	Figura 3-2 h)	A	1	1	**1**	1	1	1	1
		B	0	0	**0**	0	1	1	1
		C	0	0	**1**	1	0	0	1
		D	0	1	**0**	1	0	1	0

Tabela 3-3 – Problemas bi-classe e respectiva representação gráfica e por *ecoc* (*k*=4)

O método exaustivo garante também equidistância entre as diversas palavras binárias. Finalmente, o método exaustivo garante que a distância de Hamming entre palavras binárias é a melhor distância mínima possível de ser obtida para determinado valor de k.

Considerando o exposto, a máxima distância mínima passível de se obter é para um determinado valor de k e segundo o método exaustivo é:

$$2^{k-2} \tag{50}$$

Simultaneamente, caso os *ecocs* sejam gerados utilizando o valor mínimo de e (expressão (16)), a melhor distância mínima de Hamming terá o valor 1.

A utilização destas duas informações permite obter dois pontos com os quais se define uma recta *y(k,e)* (denominada de recta de aproximação - Figura 3-3) e que permite, dado k e e obter a indicação de qual a melhor distância de Hamming mínima que é possível obter.

$$y(k,e) = me + b \tag{51}$$

O declive da função será dado pela expressão

$$m = \frac{2^{k-2} - 1}{(2^{k-1} - 1) - \lceil \log_2 k \rceil} \tag{52}$$

e

$$b = 1 - m\lceil \log_2 k \rceil \tag{53}$$

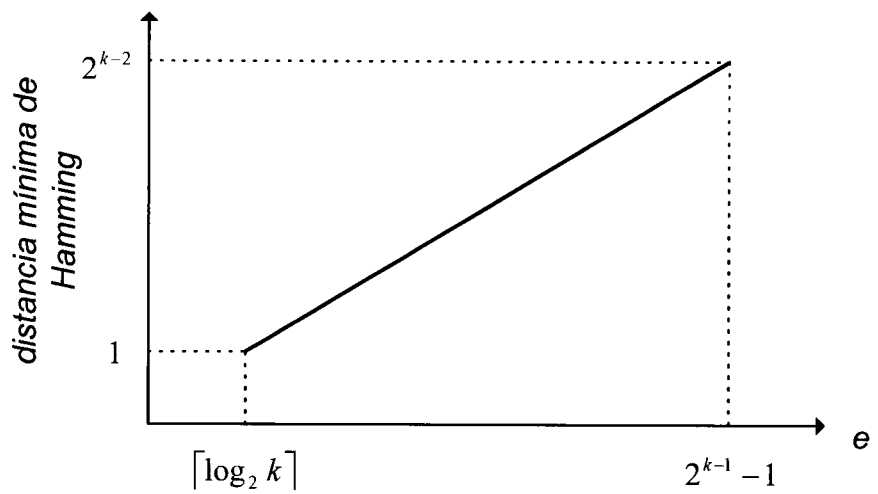


Figura 3-3 – Recta de aproximação genérica

Uma vez que as distâncias serão sempre valores inteiros, define-se a função de aproximação *a(k,e)* como a truncagem da recta de aproximação (exemplificado na Figura 3-4 para k=5):

$$a(k,e) = \lfloor y(k,e) \rfloor \tag{54}$$

,isto é,

$$a(k,e) = \left\lfloor \frac{2^{k-2}-1}{(2^{k-1}-1)-\lceil \log_2 k \rceil}(e-\lceil \log_2 k \rceil)+1 \right\rfloor \tag{55}$$

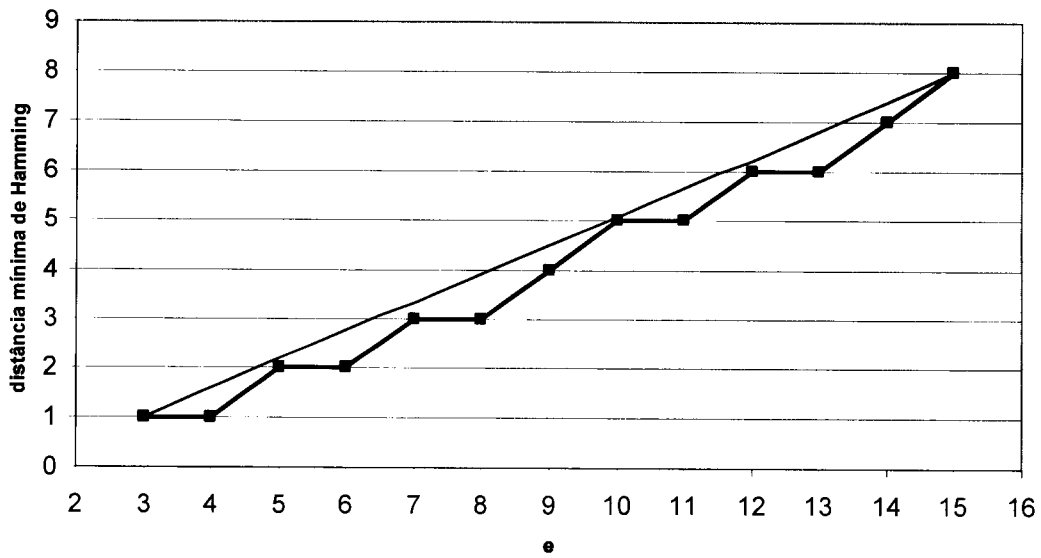


Figura 3-4 – Função de aproximação para k=5 - *a(5,e)*

Finalmente, e com base na função de aproximação *a(k,e)* define-se a função de qualidade de um *ecoc q(k,e)* e que retornará um valor qualitativo dependente do *ecoc* obtido:

$$\begin{cases} d\min(ecoc) - a(k,e) \geq 0 & q(k,e) = \text{B+} \\ 0 > d\min(ecoc) - a(k,e) \geq -1 & q(k,e) = \text{B} \\ d\min(ecoc) - a(k,e) < -1 & q(k,e) = \text{M} \end{cases} \tag{56}$$

- ***q(k,e)*=B+** quando o *ecoc* obtido se encontra, graficamente, sobre a função de aproximação *a(k,e)* ou acima desta. Considera-se que é o melhor *ecoc* possível de ser obtido (sobre recta principal da Figura 3-5 ou acima)

- ***q(k,e)*=B** quando o *ecoc* obtido se encontra abaixo da função de aproximação *a(k,e)*, a uma distância inferior ou igual a 1. O *ecoc* obtido é bom e poderá ou não ser o melhor (zona entre a recta principal e a recta a tracejado inclusive - Figura 3-5);

- ***q(k,e)*=M** quando o *ecoc* obtido se encontra abaixo da função de aproximação *a(k,e)*, a uma distância superior a 1. Neste caso, o *ecoc* obtido é fraco (abaixo da recta a tracejado - Figura 3-5) ;

Em termos comparativos considera-se que M < B < B+.

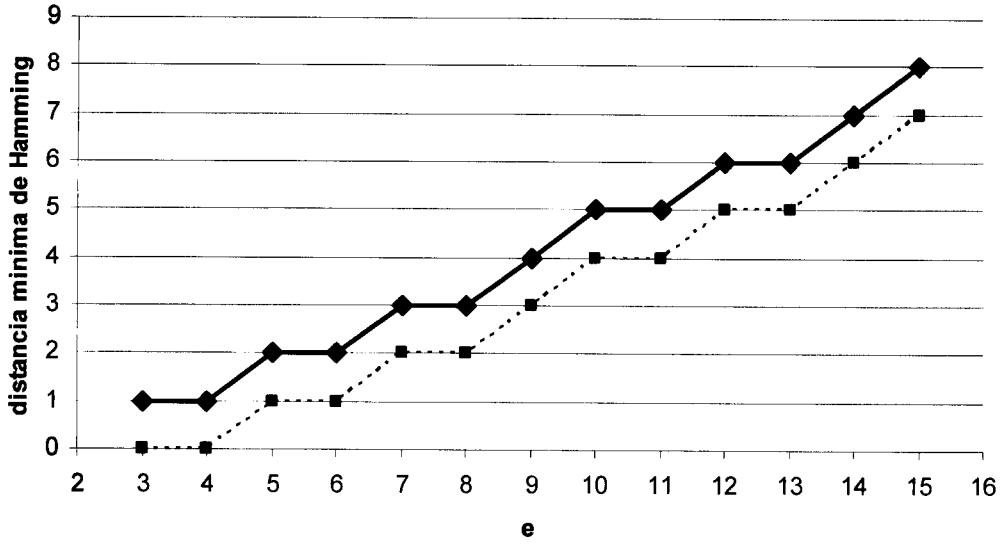


Figura 3-5 – Função de aproximação para k=5 e limites de qualidade (tracejado)

A opção de truncar a expressão (54) é um relaxamento à recta de aproximação. O arredondamento seria uma hipótese, mas muitos dos *ecocs* gerados seriam no máximo de qualidade B. No capítulo seguinte são apresentadas uma série de experiências que verificam se esta recta definida de forma teórica têm aderência à realidade.

A função de qualidade *q(k,e)* assume que os *ecocs* a avaliar não apresentam colunas equivalentes, complementares ou representadas apenas por 1 ou 0, ou seja, assume que a função de ajuste apresentada na expressão (48) é positiva.

Com base nesta função apresenta-se um novo algoritmo para criação de ecocs, o algoritmo de perseguição.

3.2.4.2 Descrição do Algoritmo de Perseguição

Tendo por base a função de qualidade definida no ponto anterior, apresenta-se agora um método para criação de ecocs: o algoritmo da perseguição. Consiste em criar um *ecoc* inicial e ir caminhado

sobre a função de qualidade até se obter a dimensão *e* desejada. Com vista a optimizar essa perseguição à função de qualidade o algoritmo foi dividido em duas partes:

— Se a dimensão pretendida for inferior a metade do valor máximo de *e* (expressão (17)), o algoritmo parte de um ecoc com dimensão mínima e vai acrescentando colunas;

— Se a dimensão pretendida for superior a metade do valor máximo de *e*, o algoritmo parte de um *ecoc* com dimensão máxima (gerado pelo método exaustivo), sendo retiradas colunas sucessivamente.

No Algoritmo 3-2 é apresentado o pseudocódigo.

```
ENTRADA: k,e,qe
SAIDA: ecoc
se e <=max(k)-min(k)/2
        cria ecoc com dimensão mínima
        enquanto dimensão(ecoc) < e
                retoma_qe()
                enquanto qe(ecoc) < qe
                        adicionacoluna(ecoc)
                        acerta_qe()
se e >max(k)-min(k)/2
        cria ecoc com dimensão máxima
        enquanto dimensão(ecoc) > e
                retoma_qe()
                enquanto qe(ecoc) < qe
                        retiracoluna(ecoc)
                        acerta _qe()
ecoc
```

k –Número de classes

e – Dimensão pretendida do *ecoc*

qe – Qualidade do *ecoc* pretendida (M, B, ou B+)

dimensão() – Função que retorna o tamanho do *ecoc*

qe() – Função que retorna a qualidade do *ecoc* (M, B, ou B+)

adicionacoluna() – Função que acrescenta ao *ecoc* uma coluna

retiracoluna() – Função que retira ao *ecoc* uma coluna

acerta _qe() – Função que ajusta o valor de *qe* pretendido

retoma_qe() – Função que atribuí a *qe* o valor original

Algoritmo 3-2 – Método de perseguição directa – algoritmo genérico.

A função *adicionacoluna()* adiciona ao ecoc uma coluna que é retirada de um ecoc previamente criado segundo o método exaustivo e com dimensão máxima. O objectivo é garantir que a coluna adicionada é diferente de qualquer coluna que o ecoc já tenha. A escolha das colunas nesta função e na função *retiracoluna()* é feita de forma aleatória.

A função *acerta_qe()* consiste num possível relaxamento na perseguição à função de qualidade. A ideia base é que para atingir um ecoc de dimensão *e* com qualidade B+ pode ser necessário criar no caminho ecocs de qualidade inferior (B e M). Assim sendo, após um determinado número de tentativas (parametrizável) podem ocorrer 3 situações:

i) O algoritmo mantém o valor de *qe*. Neste caso, nada é alterado;

ii) O algoritmo baixa gradualmente em uma unidade o valor de *qe*. Neste caso, caso que fosse B+ passaria para B e caso fosse B passaria para M;

iii) O algoritmo considera que o valor de *qe* passa para B. Esta situação só faz sentido quando o valor inicial de *qe* for B+. A diferença em relação a ii) está na não consideração da possibilidade de baixar *qe* para M.

Após o adicionar/retirar de uma coluna ao ecoc que está a ser construído, o valor de *qe* é sempre reposto para o valor original dado como parâmetro inicial da função.

O método proposto garante a inexistência de colunas iguais, complementares ou compostas apenas por 1's ou 0's.

```
ENTRADA: k,e,qe
SAIDA: ecoc
cria ecoc com dimensão mínima
    enquanto dimensão(ecoc) < e
        retoma_qe()
        enquanto qe(ecoc) < qe e fitness(ecoc) < 0
            adicionacolunaaleatoria(ecoc)
            acerto_qe()
```

k –Número de classes

e – Dimensão pretendida do *ecoc*

qe – Qualidade do *ecoc* pretendida (M, B, ou B+)

dimensão() – Função que retorna o tamanho do *ecoc*

qe() – Função que retorna a qualidade do *ecoc* (M, B, ou B+)

fitness() – Função de ajuste do *ecoc*.

adicionacolunaaleatoria() – Função que acrescenta ao *ecoc* uma coluna

acerto_qe() – Função que ajusta o valor de *qe* pretendido

retoma_qe() – Função que atribuí a *qe* o valor original

Algoritmo 3-3 – Método de perseguição aleatório – algoritmo genérico.

No entanto, este método apresenta uma desvantagem: a adição/retirada de colunas tem por base um *ecoc* do tamanho máximo possível construído segundo o método exaustivo. Para valores de k elevados as exigências de memória para criação desse *ecoc* podem inviabilizar o uso deste método.

Para obviar essa limitação é proposto o método de perseguição aleatória (Algoritmo 3-3). O princípio básico é o mesmo. No entanto, ao invés de se seleccionar uma coluna do *ecoc* gerado pelo método exaustivo, cada coluna é gerada aleatoriamente. Obviamente e ao contrário do método de perseguição directo, a coluna gerada pode ser igual ou equivalente a uma já existente, bem como composta apenas por 0's ou 1's. Logo, deve-se garantir que a função de ajuste (expressão (49)) é positiva.

3.2.4.3 Limitações ao Algoritmo de Perseguição

Conforme já referido anteriormente, o método de perseguição directa apresenta uma limitação nos valores de k passíveis de serem usados. Essa limitação resulta do método necessitar inicialmente de construir uma matriz com todas as fronteiras de decisão (usando o método exaustivo exposto em 3.2.1) e a dimensão dessa matriz crescer exponencialmente com k, ultrapassando por vezes o limite de memória disponível.

Para além disso, o método da perseguição baseia-se numa procura aleatória. Essa situação conduz a que o mesmo problema possa ter tempos de execução consideravelmente diferentes consoante a semente utilizada.

3.2.5 Distância de Hamming entre colunas

Conforme referido em 3.1.1, é recomendável maximizar a distância de Hamming entre as diversas colunas. Uma vez que a cada coluna corresponderá a um conjunto de dados bi-classe e que leva à geração de um modelo de decisão, quanto mais distintas forem as colunas, mais distintos serão esses modelos. Tenta-se assim garantir que os erros que ocorrem em diferentes problemas estão o menos possível relacionados.

No entanto, à medida que o tamanho e da palavra binária aumenta será cada vez mais difícil obter *ecocs* com distância mínima de Hamming entre colunas superior a 1, uma vez que esse critério decresce exponencialmente com e (Figura 3-6).

O gráfico foi obtido com base na função de aproximação $a(k,e)$ (expressão (54)), aplicado à distância de Hamming entre colunas.

Pela análise de evolução da melhor distância de Hamming mínima entre colunas para vários valores de k, concluímos que a partir de $\frac{2^{k-1}}{2}+1$ o melhor valor que é possível obter será 1.

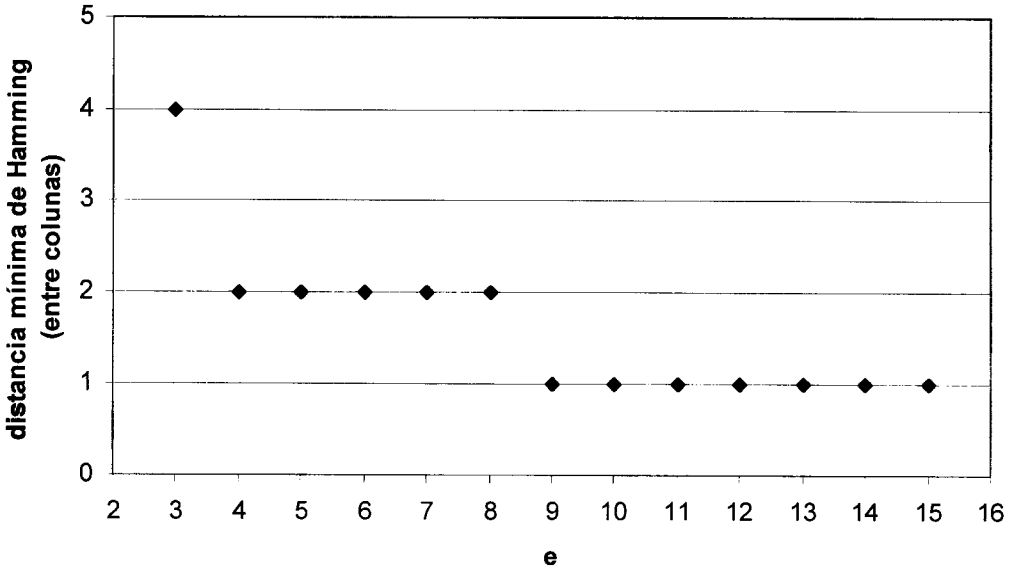


Figura 3-6 – Evolução da melhor distância de Hamming mínima entre colunas (k=5)

Dado o comportamento demostrado pela distância de Hamming entre colunas com o incremento do tamanho *e* da palavra binária este critério não será utilizado no processo de definição de um bom *ecoc*.

Mantém-se no entanto os restantes critérios enumerados em 3.1.1 uma vez que são requisitos obrigatórios:

- a inexistência de colunas iguais (este critério obriga a que a distância mínima de Hamming entre colunas seja pelo menos 1);
- a inexistência de colunas complementares;
- a inexistência de colunas constantes representadas apenas por 0 ou 1.

3.3 Selecção de ecocs

Conforme já foi amplamente referido, a decomposição para k' problemas bi-classe de um problema de classificação de k classes conduz a valores de k' que variam entre um mínimo dado por $\lceil \log_2 k \rceil$ e um máximo dado por $2^{k-1}-1$ (Tabela 3-4). Recorde-se que $k'=e$, sendo e o tamanho das palavras binárias dum *ecoc*. Tendo em conta que o valor máximo de e cresce de forma exponencial, os valores que k' pode tomar crescem na mesma proporção.

Surge naturalmente a questão de qual deverá ser o valor de k' (e consequentemente e) de entre os vários possíveis. Nesta secção apresenta-se o caminho que nos leva à apresentação de uma solução.

k	Mínimo e	Máximo e	Número de valores que e pode tomar
3	2	3	2
4	2	7	6
5	3	15	13
6	3	31	29
7	3	63	61
10	4	511	508
20	5	524 287	524 283
24	5	8 388 607	8 388 602
k	$\lceil \log_2 k \rceil$	$2^{k-1}-1$	$2^{k-1}-\lceil \log_2 k \rceil$

Tabela 3-4 – Evolução do tamanho máximo e mínimo da palavra binária

3.3.1 Selecção por distância de Hamming

Pese embora os valores que e pode tomar

$$\lceil \log_2 k \rceil \leq e \leq 2^{k-1}-1 \tag{57}$$

existem vários valores de e tal que

$$d_{\min} e_i = d_{\min} e_j, i \neq j \tag{58}$$

Nesses casos em que palavras binárias de tamanho diferente apresentam valores equivalentes para a distância mínima de Hamming, fará sentido usar o conjunto de palavras binárias de menor dimensão.

Por exemplo, para a função de aproximação *a(5,e)* representada na Figura 3-4, a distância mínima de Hamming para e=10 é igual à distância mínima de Hamming para e=11. Logo, não faria sentido usar e=11. Usando essa premissa e para o exemplo de k=5, os valores de e que deveriam ser considerados são os pontos marcados na Figura 3-7. No caso exemplificado de k=5, a selecção desses pontos permite reduzir o número de possíveis soluções para a dimensão de e de 13 para apenas 8.

De forma genérica para k classes e para cada distância de mínima de Hamming, dever-se á seleccionar o menor valor de e. Na Algoritmo 3-4 apresenta-se um algoritmo para a obtenção destas soluções.

```
ENTRADA:    k - Número de classes
SAÍDA:      listae – lista com as dimensões de e
min=⌈log2 k⌉
max=2^(k-1)-1
i=min
best_dmin=0
ENQUANTO (i <=max) {
    SE( a(k,i) > best_dmin ) {
        Acrescenta à lista listae
        best_dmin= a(k,i)
    }
    i++
}
```

a(k,i) – Função de aproximação (expressão (56))

Algoritmo 3-4 – Algoritmo para selecção de *e* com base na função de aproximação *a(k,e)*.

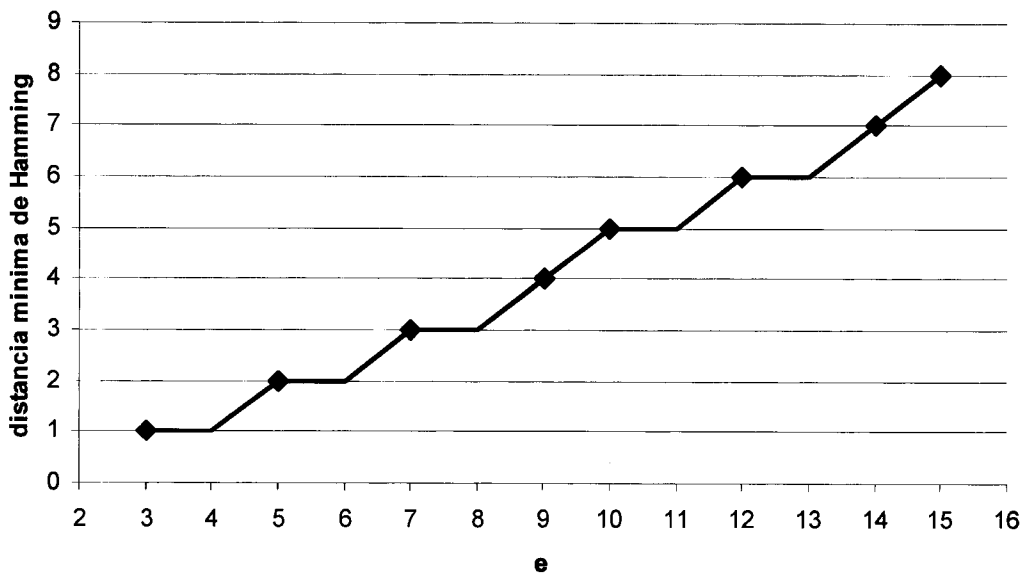


Figura 3-7 – Selecção de valores de *e* com base na função de aproximação para k=5

Este método permite diminuir substancialmente o número de valores de e possíveis (e consequentemente de *ecocs*) para a decomposição de um problema de k classe de $2^{k-1} - \lceil \log_2 k \rceil$ possibilidades para apenas 2^{k-2} (Tabela 3-5).

k	Número de valores que k' pode tomar	Número de soluções pela selecção por distância de Hamming	% redução
3	2	2	0,0000
4	6	4	33,3333
5	13	8	38,4615
6	29	16	44,8276
7	61	32	47,5410
10	508	256	49,6063
20	524 283	262 144	49,9995
24	8 388 602	4 194 304	49,9999
k	$2^{k-1}-\lceil \log_2 k \rceil$	2^{k-2}	

Tabela 3-5 – Selecção por distância mínima de Hamming

3.3.2 Selecção com base no número máximo de erros corrigíveis

Em 3.1 foi apresentado o número máximo de erros corrigíveis (*me*) para uma determinada distância mínima de Hamming:

$$me = \left\lfloor \frac{d_{\min}(p, w_j) - 1}{2} \right\rfloor \tag{59}$$

Este conceito é interessante pois, uma vez que os problemas de classificação apresentam um sistema semelhante ao apresentado na Figura 2-1, considera-se que o uso dos modelos de decisão para obter predições vai acrescentar ruído, mais propriamente erros. Com esta expressão (*me*) é possível calcular *a priori* o número de erros que podem ser corrigidos por determinado *ecoc*.

O valor dado pela expressão (59) traduz o número de bits e consequentemente de modelos de decisão que podem efectuar uma predição errada, sem que isso implique uma predição final errada.

Considere-se, a título de exemplo a matriz apresentada na Tabela 3-1. A distância mínima de Hamming tem o valor 4, o que significa que *me*=1, isto é, se apenas um modelo de decisão efectuar uma classificação incorrecta, a função de reconstrução é capaz de corrigir esse erro e obter uma classificação correcta. Suponha-se que um exemplo de treino é da classe representada pela primeira linha da Tabela 3-1 – 1111111. Se existisse um erro numa das predições dos diversos modelos de decisão, o resultado das sete predições resultaria, por exemplo, na palavra binária 1011111.

A aplicação de *r(F')* (29) retornaria como sendo a palavra binária 1111111 a mais próxima da predição, resultando numa predição correcta. No entanto se a predição apresentasse erros em dois

modelos de decisão (ex.: 1011011) a função *r(F')* retornaria como predição uma das palavras binárias mais próximas e que seriam 1111111 (correcta) ou 0011001 (incorrecta).

Concluindo, existem vários valores para distancia de Hamming mínima tal que

$$me(d_{\min} e_i) = me(d_{\min} e_j), e_i \neq e_j \tag{60}$$

Também neste caso dever-se ia escolher o *ecoc* que apresenta-se dimensão inferior.

3.3.3 Percentagem máxima predições erradas

Conforme se pode constatar pela expressão (59), a um incremento de 2 no valor da distância mínima de Hamming corresponde apenas a um incremento de 1 no número máximo de erros passíveis de serem corrigidos pela função de reconstrução. Consequentemente, o número de erros máximos passíveis de ser corrigidos não cresce na mesma proporção que a máxima distância mínima de Hamming. Cumulativamente, e conforme apresentado na Figura 3-3 e exemplificado na Figura 3-4, a máxima distância mínima Hamming cresce mais lentamente que o tamanho *e* da *palavra binária.*

Para demonstração do crescimento do número máximo de erros corrigíveis *me* face ao crescimento do tamanho *e* da palavra binária, definiu-se o rácio

$$pe = \frac{me}{e} \tag{61}$$

como a percentagem de modelos de decisão que podem efectuar predições erradas *pe(k,e)* sem que isso acarrete erros na predição final.

Considerando a expressão (59) que define *me* e a função de aproximação *a(k,e)* dada pela expressão (55) obtêm-se

$$pe(k,e) = \frac{\left\lfloor \frac{\left\lfloor \frac{2^{k-2}-1}{(2^{k-1}-1)-\lceil \log_2 k \rceil}(e-\lceil \log_2 k \rceil)+1 \right\rfloor - 1}{2} \right\rfloor}{e} \tag{62}$$

A Figura 3-8 permite visualizar de forma gráfica a evolução de *pe(k,e)*, tendo sido considerado, a título de exemplo, *k*=6.

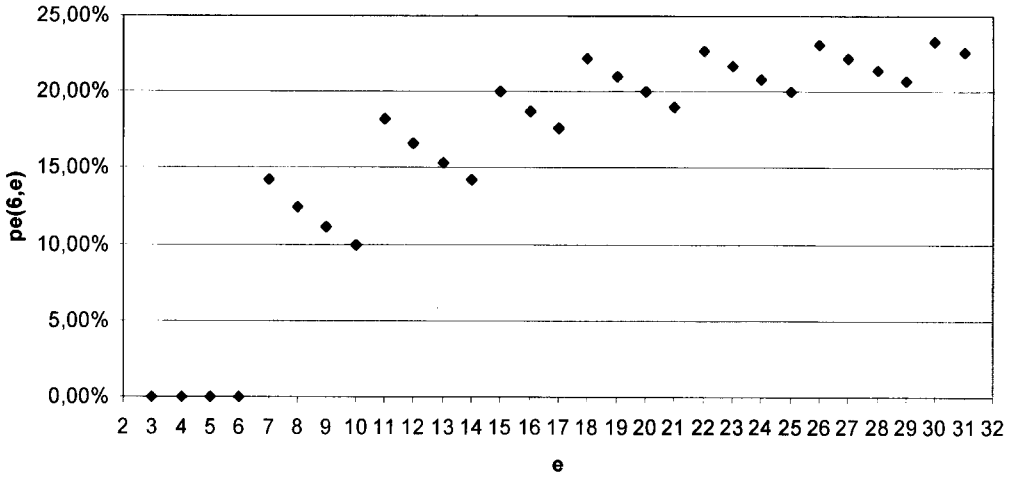


Figura 3-8 – Evolução de *pe(6,e)*

Pela análise da figura acima é possível concluir que existe um majorante à função *a(k,e)* de 25%. Para além disso, constata-se:

- Pode não existir vantagem aparente em utilizar o valor máximo de *e*;
- os máximos locais de *a(k,e)* crescem de forma logarítmica, conduzindo a que um majorante para a percentagem de predições erradas seja rapidamente obtido.

3.3.4 Função de avaliação

Foi anteriormente referido que na construção de *ecocs* procura-se um equilíbrio entre redundância (obtido com o incremento de *e*) e complexidade (implicando a redução de *e*).

Conforme já exposto, mais importante do que aumentar *e* para obter redundância é aumentar a distância mínima de Hamming e consequentemente o número máximo de erros corrigíveis (*me*). Para além disso, para cada valor de *me* existe um conjunto de soluções possíveis, representado várias dimensões *e*. Resumindo, interessa utilizar apenas a solução que apresenta o menor valor de *e* para o mesmo número de erros corrigíveis.

Com esse objectivo, definiu-se a função *bme(k,e)*

$$bme(k,e) = \frac{me^2}{e} \tag{63}$$

representada graficamente na Figura 3-9 para k=6. A função representada apresenta sete máximos locais e é semelhante à exposta na Figura 3-8, resultante da semelhança entre as expressões (61) e (63). No entanto, neste caso não existe um majorante. Esta função apresenta ainda duas características:

- cada máximo local apresenta sempre um valor superior ao anterior pelo que a função *bme(k,e)* será tendencialmente crescente;

- após o *m*-ésimo máximo local, existe sempre um mínimo local que será sempre superior ao *(m-1)*-ésimo máximo local.

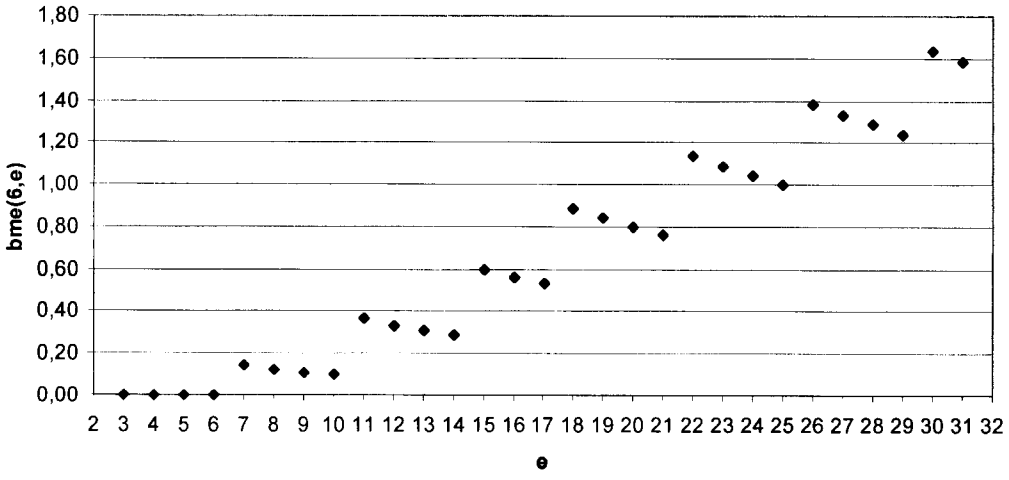


Figura 3-9 – Evolução da função *bme(k,e)* para k=6

Esta função para um mesmo valor do número máximo de erros corrigíveis, penaliza o aumento de *e*. Consequentemente, permite seleccionar de forma coerente um bom conjunto de soluções para a dimensão *e* a utilizar na construção de *ecocs*. Esse conjunto de soluções são os máximos locais da função *bme(k,e)*

No entanto, o valor dos máximos locais de *bme(k,e)* cresce sempre com o aumento de *e*. Importa também penalizar esses máximos com o aumento de *e*.

Para a aplicação dessa penalização definiu-se a expressão

$$\text{aval(k,e)} = k\,\frac{bme(k,e)}{e} \qquad (64)$$

que consiste na função de avaliação *aval(k,e)*. As Figura 3-10 e Figura 3-11 apresentam a evolução desta função para k=6 e k=7. Esta função de avaliação apresenta um comportamento tendencialmente logarítmico nos seus máximos locais. À medida que nos aproximamos do valor máximo de *e*, os máximos locais de *aval(k,e)* serão cada vez mais próximos, ou seja, A melhoria conseguida de um máximo local para o seguinte vai sendo cada vez menor.

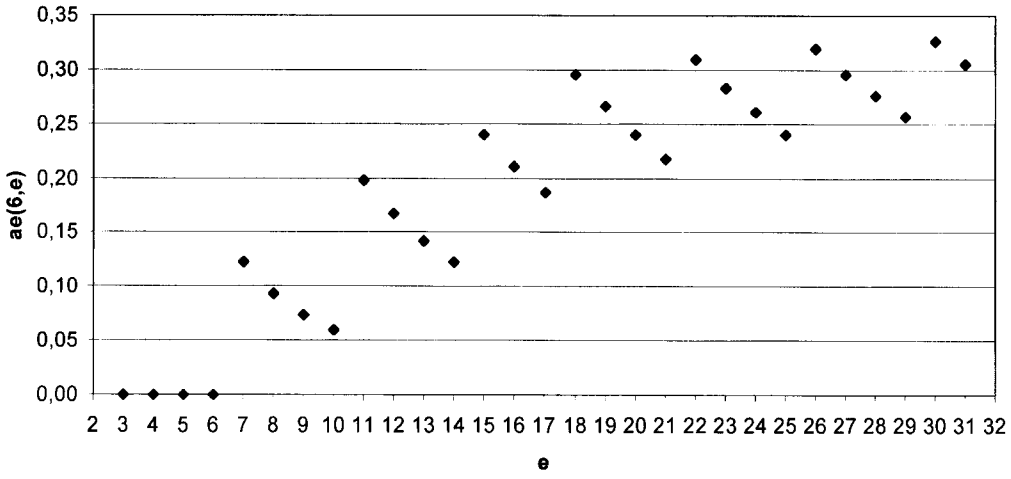


Figura 3-10 – Evolução da função *aval(6,e)*

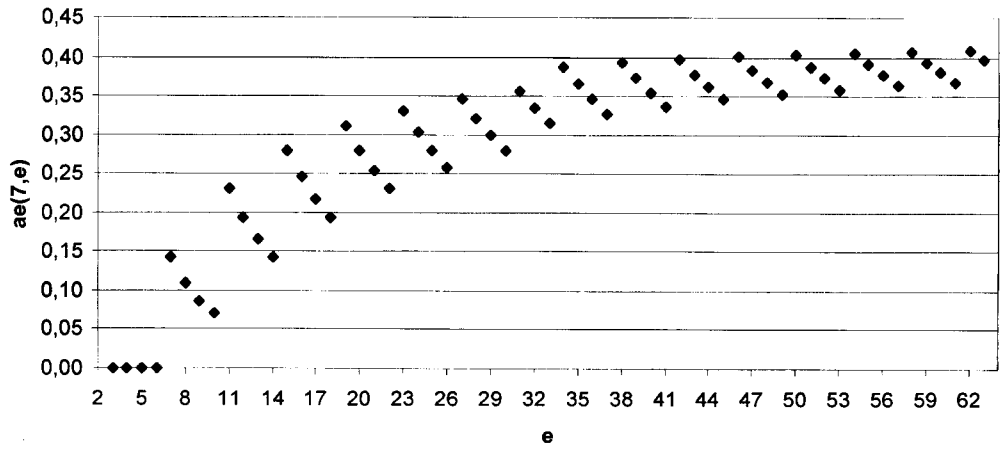


Figura 3-11 – Evolução da função *aval(7,e)*

Esta função de avaliação de um *ecoc* com dimensão *e* permite:

- de forma teórica e antes da aplicação da função de decomposição definir qual a dimensão *e* da *palavra binária* e consequentemente dos *ecocs* a criar;

- de forma prática, e dados dois *ecocs* já construídos, apresentar uma medida de comparação entre ambos, tentando antecipar qual apresentará o melhor comportamento.

As funções definidas permitem seleccionar as dimensões dos *ecocs* antes destes serem construídos, simplificado um problema de $2^{k-1} - \lceil \log_2 k \rceil$ soluções para 2^{k-2} soluções (3.3.1) ou ainda menos como é o caso da função de avaliação apresentada na expressão (64).

3.3.5 Escolha do melhor ponto da função de avaliação

A função de avaliação *aval(k,e)* apresenta diversos máximos locais. No entanto, qual o máximo local que permite obter um equilíbrio entre a dimensão da palavra binária *e* e o valor da função de avaliação *aval(k,e)*. Ou seja, procuramos um equilíbrio entre custo (*e*) e beneficio (*aval(k,e)*).

Este problema pode ser tratado do ponto de vista económico. Essencialmente admite-se que a racionalidade dos agentes conduz a um deterninado tipo de tomada de decisões: quanto ganho/perco se consumir mais uma unidade? Quanto ganho (ou me custa) se produzir mais uma unidade? Quando se tratam de valores marginais procura-se um equilíbrio com base nas derivadas. Tratando-se de valores absolutos considera-se a tangente. Neste caso, na comparação entre ganhos e perdas a tangente com 45° surge como uma boa solução [40].

Na procura do ponto de equilíbrio da função de avaliação *aval(k,e)* utilizaremos precisamente esse método. Dado que a função de avaliação *aval(k,e)* é composta por pontos discretos, a solução adoptada (Figura 3-12) consiste em seleccionar três máximos locais seguidos e criar duas rectas: uma formada pelos dois primeiros pontos (recta *r1*) e outra formada pelos dois últimos (recta *r2*). Calcula-se o ângulo β entre as duas rectas e posteriormente define-se a recta bissectriz *b1*. A recta *t1* será perpendicular à bissectriz e simultaneamente será a tangente ao ponto intermédio.

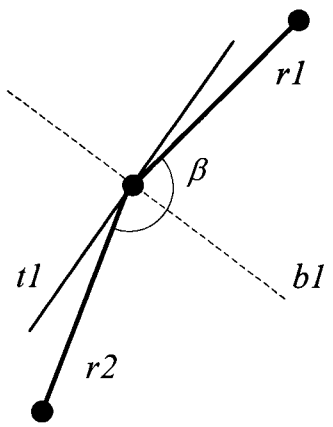


Figura 3-12 – Selecção da tangente a um ponto

A aplicação deste método à função de avaliação pressupõem a prévia normalização da mesma. Na Algoritmo 3-5 é apresentado o pseudocódigo que permite obter o ponto de equilíbrio.

Conforme se constata, o valor do ângulo é um dos argumentos de entrada do algoritmo. É assim possível encontrar outros pontos. Por exemplo, suponha-se que se pretende encontrar um equilíbrio entre *e* e *aval(k,e)*, mas privilegiando uma redução na dimensão de *e*. Poder-se-ia executar o algoritmo atrás referido mas com um ângulo de 60°. Se ao invés se preferisse privilegiar *aval(k,e)* poder-se-ia considerar um ângulo de 30°. Tem-se assim alguma versatilidade na pesquisa do melhor ponto.

ENTRADA: **k - Número de classes**

ang – ângulo desejado (por defeito – 45º)

SAÍDA: **dime – dimensão de *e***

```
avaln=normaliza(aval(k))
maxs=maximos(avaln)
i=2
enquanto i < tamanho(maxs)-2 {
    p1x,y = max[i-1]
    p2 x,y = max[i]
    p3 x,y = max[i+1]
    angp=ângulo(p1,p2,p3)
    difang=|angp-ang|
    se difang<bestdifang
        bestdifang=difang
        e=p2x
    i++
}
```

avaln – Os valores da função de avaliação normalizados. Vector de coordenadas *x* e *y*.

maxs – Os máximos locais da função de avaliação já normalizada. Vector de coordenadas *x* e *y*.

$p_{x,y}$ – Pontos com duas coordenadas: *x* e *y*.

angp – Ângulo obtido

difang- Diferença absoluta entre o ângulo desejado e o ângulo obtido

bestdifang – Melhor diferença de ângulos.

aval(k) – Função de avaliação (expressão (64))

normaliza() – Função para normalizar *aval(k)*. Retorna um vector de valores do tipo x,y.

maximos() – Função para retornar os máximos locais. Retorna um vector de valores do tipo x,y.

ângulo() – Função que dados 3 pontos retorna o ângulo formado pelas 2 rectas, conforme explicado anteriormente

Algoritmo 3-5 – Algoritmo para selecção de uma dimensão *e* com base na função de avaliação *aval(k,e)*.

3.4 Resumo

Nesta secção apresentamos as motivações para a criação de *ecocs* no âmbito das telecomunicações a as características específicas que estes devem ter quando aplicados a problemas de classificação de dados.

Foram apresentados e analisados alguns métodos para criação de *ecocs*:

- O método apresentado por Dietterich [3], específico para problemas de classificação de dados;
- Os algoritmos de repulsa e o GARA, apresentados em [13] e criados no âmbito das telecomunicações mas adaptados a problemas de classificação de dados no âmbito desta dissertação;
- O algoritmo de perseguição, novo método para a criação de ecocs apresentado no âmbito desta dissertação.

Para além disso foram apresentadas algumas propostas de solução para as seguintes questões:

i) Dado um problema de classificação com k classes, qual o melhor tamanho da palavra binária a ser utilizado na decomposição para *ecocs*?

ii) Como garantir que um conjunto de palavras binárias de dimensão e é o melhor que se pode obter para um problema de k classes?

iii) Dados dois ecocs da mesma dimensão e, como determinar em termos teóricos o melhor, com base nas suas capacidades de correcção de erros?

Essas propostas constituíram num conjunto de funções, nomeadamente:

- *qe(ecoc)* – Dado um *ecoc*, esta função permite aferir de forma qualitativa a qualidade do mesmo. Pode tomar os valor M, B e B+, consoante seja um mau *ecoc*, bom ou óptimo. Esta função permite responder às questões ii) e iii).
- *aval(k,e)* – Função que avalia, para um determinado número de classes k, as melhorias introduzidas pelo aumento da dimensão e, penalizando simultaneamente o aumento dessa mesma dimensão e. Esta função permite responder parcialmente à questão i), através dos seus máximos locais.
- melhor ponto da função *aval(k,e)* – Finalmente é apresentado um algoritmo para seleccionar o melhor máximo local da função *aval(k,e)*, respondendo à questão i).

4 Resultados experimentais

Neste capítulo sumariam-se os resultados experimentais de vários testes realizados com diversos objectivos.

Numa primeira secção pretendeu-se avaliar a qualidade dos diferentes algoritmos para a criação de *ecocs*. São testados o algoritmo de repulsa, o GARA e o algoritmo de perseguição.

Numa segunda secção são efectuados diversos testes para avaliar o comportamento prático dos *ecocs* face a variações na sua dimensão *e*. São também avaliadas as funções apresentadas anteriormente para selecção dos melhores tamanhos de *e*. Nestes testes foram utilizados conjuntos de dados de [1].

Finalmente, na última secção é efectuado um comparativo entre os vários métodos de decomposição de problemas multi-classe, nomeadamente todos-contra-todos, um-contra-todos e *ecocs*, sendo que neste último, mais uma vez, é testado o comportamento para diversos valores de *e*.

4.1 Criação de ecocs

Nesta secção é efectuado um estudo comparativo entre os diversos métodos de geração de *ecocs*, nomeadamente o algoritmo de repulsa, o GARA e o algoritmo de perseguição. Pretende-se aferir a capacidade destes algoritmos de:

- Criar *ecocs* viáveis. São considerados *ecocs* viáveis aqueles cuja expressão (65) é positiva.

$$fit = \frac{1}{\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M}\frac{1}{d_{ij}^2}} + \left(\frac{d_{\min}}{12} - \frac{d_{\min}^2}{4} + \frac{d_{\min}^3}{6}\right) \tag{65}$$

fitfinal=fit-pen

*pen=2*fit*penalizar*

Se colunas_iguais ⇨ *penalizar* = 1

Se colunas complementares ⇨ *penalizar* = 1

- Criar bons *ecocs*. Bons *ecocs* são aqueles cuja qualidade é B ou B+ (aferida pela função de qualidade *qe(ecoc)*).

Assim, e para $3 \leq k \leq 7$, foram criados *ecocs* cobrindo a totalidade de tamanhos possíveis para as palavras binárias, segundo as expressões (15) e (16). Para cada valor de *e* foram gerados 5 *ecocs* de forma aleatória.

Para a totalidade de ecocs gerados é verificada viabilidade e apresentada a percentagem dos mesmos que era viável. Obviamente, pretende-se que este valor seja o mais alto possível.

Posteriormente, e para os *ecocs* viáveis foi avaliada a qualidade dos mesmos, sendo reportada a percentagem dos mesmos de qualidade M, B e B+. Os *ecocs* não viáveis não foram considerados para cálculo destas percentagens.

4.1.1 Criação de ecocs com base no algoritmo de repulsa

Conforme referido em 3.2.2, o algoritmo de repulsa apresenta basicamente dois parâmetros: uma condição de paragem e τ - valor mínimo da força para que seja efectuado um movimento.

No nosso caso, as condições de paragem foram:

- números máximos de iterações ou
- obtenção de um ecoc de qualidade B+.

Como entrada, o algoritmo de repulsa recebe uma matriz representando um *ecoc*. Tipicamente essa matriz é gerada de forma aleatória. Os testes do comportamento do algoritmo apresentados versarão três métodos para a criação da matriz a ser introduzida:

- **Método directo**: é gerada uma matriz aleatória para um dados valor de k e e.
- **Método crescente**: excepto para o caso de e ser mínimo, no qual se usa o método directo, a matriz introduzida é a melhor matriz obtida para o mesmo k mas para $e-1$, ao qual é acrescentada uma coluna gerada aleatoriamente.
- **Método decrescente**: é gerada uma matriz utilizando o método exaustivo (3.2.1) ao qual são retiradas n colunas de forma aleatória até ser obtido o valor de e pretendido e consequentemente, a matriz de entrada.

Nas tabelas seguintes é apresentado o comportamento dos diversos métodos para diversos números de iterações para $3 \leq k \leq 6$. Não foi considerado neste método específico o valor de k=7 dado o elevado consumo de recursos e a fraca importância que os resultados teriam na análise. O valor apresentado corresponde ao número de vezes que o método foi capaz de obter *ecocs* que satisfaçam as propriedades descritas em 3.1.1 (*ecocs* viáveis), possibilitando assim a sua utilização em problemas de classificação. Ou seja, são *ecocs* cuja função de ajuste da expressão (48) é positiva.

Iterações	k=3	k=4	k=5	k=6
1 000	40 %	33,3 %	44,6 %	35,2 %
10 000	40 %	36,7 %	44,6 %	38,6 %
100 000	40 %	36,7 %	44,6 %	40,0 %

Tabela 4-1 – Criação de ecocs com o algoritmo de repulsa – método directo (% de *ecocs* viáveis).

Iterações	k=3	k=4	k=5	k=6
1 000	100 %	66,7 %	64,6 %	44,8 %
2 500	100 %	66,7 %	64,6 %	46,2 %
5 000	100 %	66,7 %	64,6 %	56,5 %
10 000	100 %	66,7 %	64,6 %	53,8 %

Tabela 4-2 – Criação de ecocs com o algoritmo de repulsa – método crescente (% de *ecocs* viáveis).

Iterações	k=3	k=4	k=5	k=6
1 000	100 %	100 %	98,5 %	97,9 %
2 500	100 %	100 %	98,5 %	97,9 %
5 000	100 %	100 %	98,5 %	97,9 %
10 000	100 %	100 %	98,5 %	97,9 %
100 000	100 %	100 %	98,5 %	97,9 %

Tabela 4-3 – Criação de ecocs com o algoritmo de repulsa – método decrescente (% de *ecocs* viáveis).

Uma vez analisada a capacidade do algoritmo de repulsa em criar *ecocs* viáveis, pretende-se agora avaliar a capacidade do algoritmo de repulsa na criação de *ecocs* que se ajustem a problemas de classificação mas que apresentem simultaneamente boa qualidade. Nas tabelas seguintes apresentam-se os resultados obtidos. Para cada valor de k e para os *ecocs* viáveis é apresentada a percentagem de *ecocs* de qualidade B+, B e M que foi possível obter. Também aqui são apresentados os resultados para diversos valores de iterações e para os diversos métodos do algoritmo de repulsa (directo, crescente e decrescente).

Iterações		k=3			k=4			k=5			k=6		
	qe	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*
1 000		100	0	0	100	0	0	100	0	0	100	0	0
10 000		100	0	0	100	0	0	100	0	0	100	0	0
100 000		100	0	0	100	0	0	100	0	0	100	0	0

Tabela 4-4 – Qualidade dos ecocs com o algoritmo de repulsa – método directo

Iterações		k=3			k=4			k=5			k=6		
	qe	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*
1 000		100	0	0	95	5	0	100	0	0	100	0	0
2 500		100	0	0	95	5	0	100	0	0	100	0	0
5 000		100	0	0	95	5	0	100	0	0	98,8	1,2	0
10 000		100	0	0	95	5	0	100	0	0	97,4	2,6	0

Tabela 4-5 – Qualidade dos ecocs com o algoritmo de repulsa – método crescente

Iterações		k=3			k=4			k=5			k=6		
	qe	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*
1 000		100	0	0	96,7	3,3	0	89,1	10,9	0	58,5	19	22,5
2 500		100	0	0	96,7	3,3	0	89,1	10,9	0	64,1	19	16,9
5 000		100	0	0	96,7	3,3	0	89,1	10,9	0	64,8	19	16,2
10 000		100	0	0	96,7	3,3	0	93,8	6,2	0	68,3	18,3	13,4
100 000		100	0	0	96,7	3,3	0	100	0	0	72,6	19,7	7,7

Tabela 4-6 – Qualidade dos ecocs com o algoritmo de repulsa – método decrescente

A capacidade do algoritmo de repulsa para a criação de *ecocs* viáveis varia desde o bastante bom para o método decrescente ao relativamente mau para o método directo. O método crescente apresenta também valores algo fraco, embora superiores ao método directo. Refira-se que o método directo tem como parâmetro um *ecoc* gerado aleatoriamente da mesma dimensão do ecoc desejado. O método crescente usa como parâmetro *ecocs* que foram melhorados sucessivamente. O método decrescente usa como base um *ecoc* criado segundo o método exaustivo (3.2.1), o que contribui a que este seja o método mais capaz de criar ecocs viáveis mas que simultaneamente limita os valores de k para os quais pode ser utilizado, por questões de memória computacional.

No que concerne à qualidade dos *ecocs* viáveis o panorama difere. O método directo apresenta os melhores resultados, seguido do método crescente. O método decrescente é o que apresenta os piores resultados.

4.1.2 Criação de ecocs com base no GARA

Nas experiências realizadas em [13] foi utilizado uma evolução do algoritmo de repulsa denominado GARA. Resumidamente consiste na aplicação de um algoritmo genético juntamente com o algoritmo de repulsa. A dimensão da população utilizada foi de 480 individuos.

Na Tabela 4-7 e na Tabela 4-8 são apresentados respectivamente a percentagem de vezes em que o algoritmo obteve *ecocs* viáveis e, desses, a percentagem de *ecocs* obtidos de qualidade B+, B e M.

Iterações	k=3	k=4	k=5	k=6	k=7
5 000	100 %	16,7 %	61,5 %	27,6 %	21,3 %
10 000	100 %	16,7 %	60,0 %	27,6 %	20,0 %
100 000	100 %	16,7 %	60,0 %	27,6 %	20,7 %

Tabela 4-7 – Criação de ecocs com o GARA (% de *ecocs* viáveis).

Iterações		k=3			k=4			k=5			k=6			k=7		
	qe	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*
5 000		100	0	0	100	0	0	50	0	50	50	0	50	61,5	0	38,5
10 000		100	0	0	100	0	0	100	0	0	100	0	0	100	0	0
100 000		100	0	0	100	0	0	100	0	0	100	0	0	100	0	0

Tabela 4-8 – Qualidade dos ecocs com o algoritmo GARA.

Conforme se constata, o número de iterações não é decisivo para obter *ecocs* viáveis. É no entanto um contributo para a obtenção de melhores *ecocs* (qualidade B+).

A capacidade do GARA na criação de *ecocs* viáveis parece diminuir rapidamente com o aumento de k. Para além disso, refira-se a fraca capacidade de criar *ecocs* viáveis quando k=4. Refira-se que para k=4 a dimensão dos *ecocs* varia entre 2 e 7. No caso de e=2 existe apenas uma solução possível não conseguindo o algoritmo convergir para essa solução única.

No que concerne à qualidade dos *ecocs* viáveis, os resultados são bastante bons pois, para 10 000 e 100 000 iterações, todos os *ecocs* viáveis são de qualidade B+.

4.1.3 Criação de ecocs com base no método de perseguição

Finalmente, testaram-se os método apresentados em 3.2.4. Os métodos apresentados recebem três parâmetros:

- Qualidade do ecoc pretendida (*qe*). Pode tomar os valores M, B ou B+;
- Numero máximo de vezes que o algoritmo tenta encontrar um *ecoc* da qualidade pretendida em cada iteração;
- Método de relaxamento. Basicamente, refere qual o comportamento do algoritmo caso seja atingido o número de vezes que o algoritmo tenta encontrar um ecoc da qualidade pretendida, sem o conseguir. Existem 3 hipóteses:

  i) O algoritmo mantém o valor de *qe*. Neste caso, nada é alterado;

  ii) O algoritmo baixa gradualmente em uma unidade o valor de *qe*. Neste caso, caso que fosse B+ passaria para B e caso fosse B passaria para M;

  iii) O algoritmo considera que o valor de *qe* passa para B. Esta situação só faz sentido quando o valor inicial de *qe* for B+. A diferença em relação a ii) está na não consideração da possibilidade de baixar *qe* para M.

Nos testes efectuados considerou-se a hipótese ii), uma vez que i) e iii) podem levar a que o algoritmo entre em ciclo, sendo incapaz de terminar. A qualidade pretendida foi sempre B+. O número de vezes – *nvezes* – que o algoritmo tenta encontrar um *ecoc* com a *qe* desejada foi:

- Perseguição directo – *nvezes* = 5

- Perseguição aleatória - $nvezes = (k * \lceil \log_2 k \rceil) * 2$

No caso da perseguição aleatória a expressão consiste no dobro da multiplicação do número de classes pelo número mínimo de bits para a sua representação. Uma vez que este método implica a geração de colunas aleatórias, considerou-se ser necessário um número de tentativas superior ao do método directo. Para além disso, considerou-se que quanto maior *k*, maior a dificuldade em encontrar uma coluna adequada. Logo, são necessárias mais tentativas.

Para cada valor de *k* e *e* foram executados 5 testes.

Os resultados apresentados na Tabela 4-9 são bastante bons uma vez que ambos os algoritmo foram capazes de obter sempre *ecocs* que garantissem que a função de ajuste (expressão (65)) é sempre positiva. Tal resultado não é surpreendente para o método de perseguição directo uma vez que os *ecocs* finais são um subconjunto de um *ecoc* criado segundo o método exaustivo.

Método	*k*=3	*k*=4	*k*=5	*k*=6	*k*=7
Perseguição directo	100	83,3	100	94,4	96,1
Perseguição aleatório	100	100	100	100	97,4

Tabela 4-9 – Criação de *ecocs* com o método de perseguição (% de *ecocs* viáveis)

Finalmente, verificou-se para situações em que a função de avaliação foi positiva, qual a capacidade dos algoritmos para a criação de bons *ecocs* (qualidade B e B+). Os resultados encontram-se espelhados na Tabela 4-10

Método		*k*=3			*k*=4			*k*=5			*k*=6			*k*=7		
	qe	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*	*B+*	*B*	*M*
directo		100	0	0	100	0	0	93,9	6,1	0	69,3	22,6	8,0	53,6	23,9	22,5
aleatório		100	0	0	100	0	0	98,5	1,5	0	63,4	36,6	0	17,8	49,2	33

Tabela 4-10 – Criação de ecocs com o método de perseguição (% de *ecocs* obtidos por qualidade)

Ambos os métodos apresentaram bons resultados na capacidade de criação de *ecocs* viáveis, com ligeira vantagem para o método de perseguição aleatório. Os resultados na obtenção de bons *ecocs* viáveis (B ou B+) foi também bastante bom, com alguma vantagem pontual para o método de perseguição directo.

4.1.4 Análise crítica dos diferentes métodos

Nesta secção foram apresentados os resultados de diversos testes de 3 algoritmos para construir *ecocs*. Os dois primeiros algoritmos (algoritmo de repulsa e GARA) são adaptações de métodos para

criação de *ecocs* em problemas de telecomunicações [13]. O algoritmo de perseguição é um novo algoritmo apresentado no âmbito desta dissertação.

Os testes apresentaram duas vertentes:

- capacidade para obter *ecocs* viáveis, isto é, cuja função de ajuste (expressão (65)) é positiva;
- capacidade para obter *ecocs* de qualidade B ou B+ (bons ecocs).

Relembre-se que apenas é verificada a qualidade dos *ecocs* viáveis. Logo, é importante que cada método seja capaz de obter *ecocs* viáveis e, simultaneamente, bons *ecocs*

De forma sumária, apresentam-se os resultados para cada algoritmo.

- O algoritmo de repulsa é algo dependente do *ecoc* inicial introduzido como parâmetro. Daí resulta que os métodos crescentes e decrescentes tenham apresentado maior capacidade para obter *ecoc* viáveis, uma vez que os *ecocs* dados como parâmetro serão já bons ecocs mas de dimensão inferior/superior. Na procura de *ecocs* de boa qualidade, os melhores resultados globais foram obtidos pelos métodos directo e crescente.
- O GARA apresenta uma melhor capacidade que o algoritmo de repulsa – método directo no que concerne à capacidade para criar *ecocs* viáveis. Já em relação ao outros métodos (crescente e decrescente) apresenta resultados ligeiramente inferiores na procura de *ecocs* viáveis mas melhores resultados em termos de qualidade dos *ecocs* obtidos.
- O algoritmo de perseguição é o melhor algoritmo na capacidade de encontrar ecocs viáveis, tendo também obtidos bons resultados na geração de bons *ecocs*. Os métodos de perseguição directa e aleatório apresentaram resultados equivalentes.

O tempo que cada um dos métodos demora até obter um bom *ecoc* viável não é importante uma vez que este processo ocorre uma só vez e durante a fase de treino. Inclusive, para problemas distintos mas com o mesmo número de classes k e para a mesma dimensão e dos *ecocs*, podem ser utilizadas as mesmas palavras binárias.

4.2 Comportamento dos ecoc em problemas de classificação

Após os testes à criação de *ecocs*, pretende-se nesta secção avaliar:

- se os máximos locais da função de avaliação *aval(k,e)* correspondem a pontos com bons resultados em termos de percentagem de acerto;
- o comportamento dos *ecocs* em problemas de classificação. Pretende-se verificar a variabilidade na percentagem de acertos para vários *ecocs* da mesma dimensão e a variabilidade na percentagem de acertos face ao aumento da dimensão e dos *ecocs*.

Assim, para vários problemas multi-classe com k classes foram efectuados testes com os *ecocs* de todas as dimensões e possíveis. Para cada dimensão e, foram efectuados testes com dois *ecocs* distintos.

Os testes efectuados utilizaram como algoritmo de classificação o C4.5 e os ecocs foram obtidos utilizando o algoritmo da perseguição.

As percentagens de acertos resultam de uma média de uma validação cruzada 10.

No Anexo 1 são apresentados os conjuntos de dados utilizados de forma mais detalhada.

4.2.1 *Experiências para K=5*

O conjunto de dados utilizados para testar o comportamento dos ecocs para k=5 denomina-se *heart-disease-cleveland.*

Nas Figura 4-1 e Figura 4-2 são apresentados dois testes onde é possível verificar o comportamento dos ecocs e compará-lo simultaneamente com a função de avaliação definida em 3.3.4. Os pontos marcados a ■ na predição corresponde aos pontos onde a função *aval(5,e)* tem máximos locais. No caso de k=5 a função de avaliação apresenta 3 máximos locais. Em ambos os testes, os máximos locais para e=10 e e=14 resultaram em predições relativamente boas face aos pontos vizinhos. Para o ponto e=7 apenas no primeiro teste o resultado foi satisfatório.

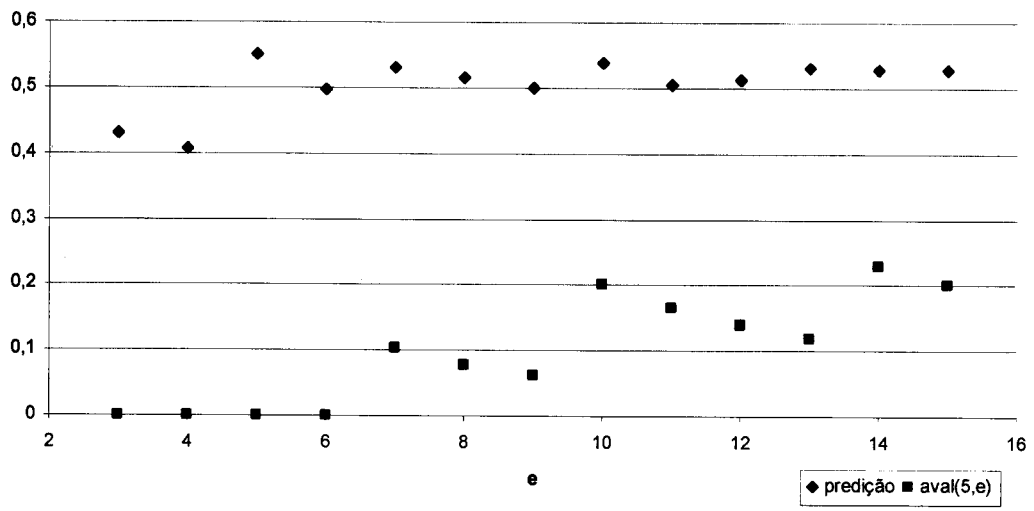


Figura 4-1 – Classificação com c4.5 com *aval(5,e)* – dados: *heart-disease-cleveland – teste1*

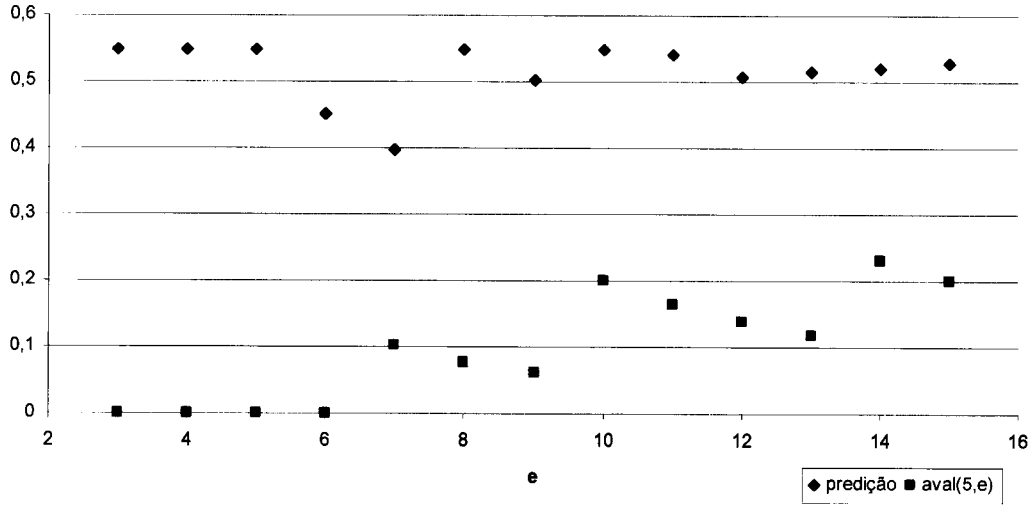


Figura 4-2 – Classificação com c4.5 com *aval(5,e)* - dados *heart-disease-cleveland – teste2*

Simultaneamente comparou-se o comportamento da classificação dos ecocs de ambos os testes (Figura 4-3) e constata-se uma elevada variabilidade nos resultados obtidos sobretudo para baixos valores de *e*. À medida que *e* aumenta, a variabilidade diminui. Tal facto resulta de existirem cada vez menos combinações possíveis. De facto, para *e*=4 existe um total de 1365 ecocs possíveis, sendo que destes, 1015 são de qualidade B+. No caso de *e*=14 o número total de ecocs possíveis é de 15, sendo todos de qualidade B+.[6] No caso de *k*=5 existem 15 fronteiras de decisão distintas. Para *e*=4 podem apenas ser escolhidas 4 dessas 15, sendo que podem ser escolhidas 4 boas fronteiras de decisão ou 4 más. Daí a maior variabilidade encontrada

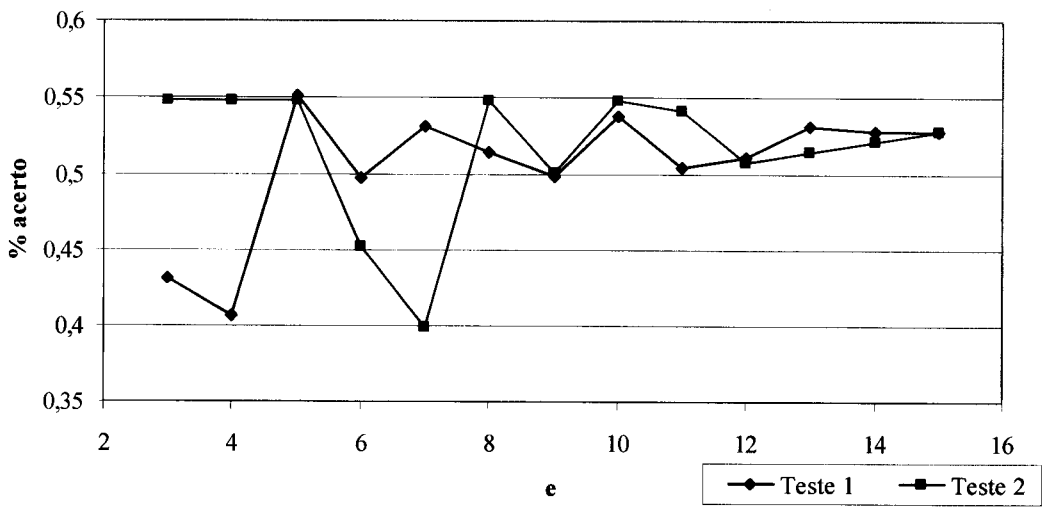


Figura 4-3 – Classificação com c4.5–*teste1 vs teste2* (dados: *heart-disease-cleveland*)[7]

[6] Com base na matriz de ecocs criada usando o método exaustivo calculou-se o número de combinações possiveis e para cada uma das combinações verificou-se a qualidade.

[7] Pese embora a função ser discreta, foi feita a ligação entre os vários pontos de um mesmo teste para uma leitura mais fácil.

Para o caso e=4 (ponto com maior variabilidade) comparou-se os *ecocs* gerados em ambos os testes, conforme se constata na Figura 4-4. Existe uma coluna comum (marcada), correspondente a uma mesma fronteira de decisão. No entanto, as restantes 3 são distintas. Com base nos resultados apresentados, conclui-se que as fronteiras de decisão do teste 2 são mais adequadas que a do teste 1 para este conjunto de dados.

A	1	**1**	1	1
B	0	**0**	1	1
C	0	**1**	0	1
D	0	**1**	1	0
E	1	**0**	1	1

teste1

A	1	1	**1**	1
B	0	0	**0**	1
C	0	1	**1**	0
D	1	0	**1**	0
E	1	1	**0**	0

teste2

Figura 4-4 – ecocs criados k=4 e e=4 – *teste1 vs teste2*

Finalmente refira-se que a melhor percentagem de acertos não ocorre para o máximo valor de e.

4.2.2 *Experiências para K*=6

Para testar os *ecocs* para valores de k=6 foi utilizado o conjunto de dados *glass*. O comportamento obtido, para duas classificações utilizando o C4.5 está exemplificado na Figura 4-5 e na Figura 4-6. À semelhança do que foi feito na secção anterior, nestas mesmas figuras é apresentada a comparação com a função *aval(6,e)*. Constata-se que nem sempre correspondem a máximos locais na curva de percentagem de acertos dos *ecocs* (pontos marcados a ■). No caso de e=15 verifica-se inclusive um mínimo local. No entanto, um dos máximos locais da função *aval(6,e)* verificada corresponde ao valor de e que apresentou melhor percentagem de acertos (68,31% para e=27). Para além disso, das 3 melhores percentagens de acertos, 2 correspondem a máximos locais da função *aval(6,e)* verificada (e=25, e=27, e=30). É possível ainda constatar que o aumento de e, conduz, tendencialmente a um aumento da percentagem de acertos. Este comportamento era esperado. Finalmente, a função *aval(6,e)* teórica apontava para que um bom compromisso em termos de dimensão de e seria e=18, uma vez que para e>18 os valores de *aval(6,e)* nos máximos locais apresentavam valores muito semelhantes. Neste ponto, a percentagem de acerto é de 62,12%

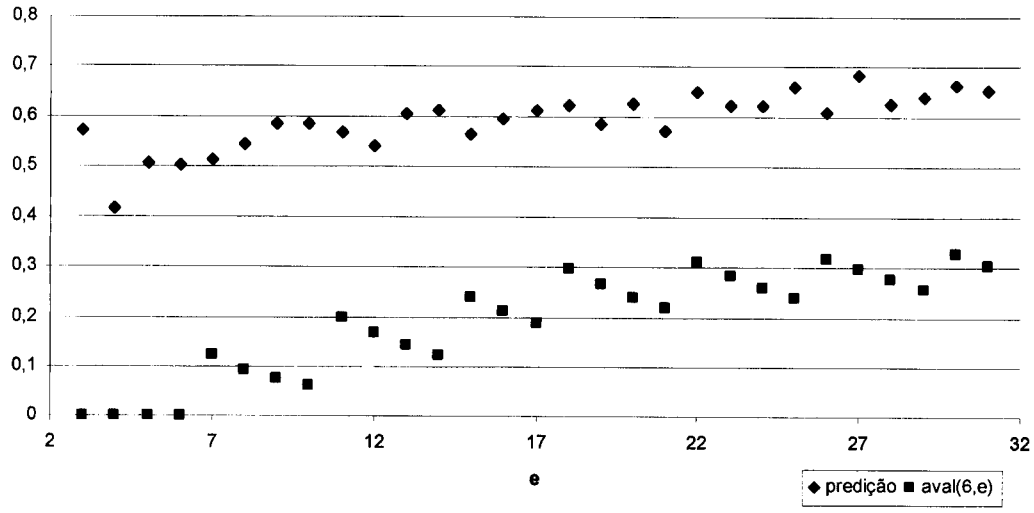


Figura 4-5 – Classificação com c4.5 com *aval(6,e)* – dados: *glass* - *teste1*

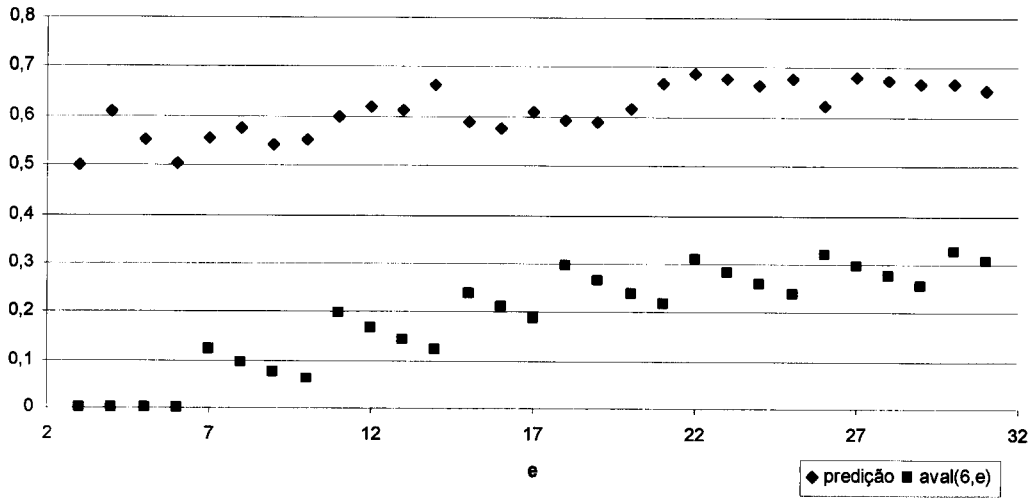


Figura 4-6 – Classificação com c4.5 com *aval(6,e)* – dados: *glass* - *teste2*

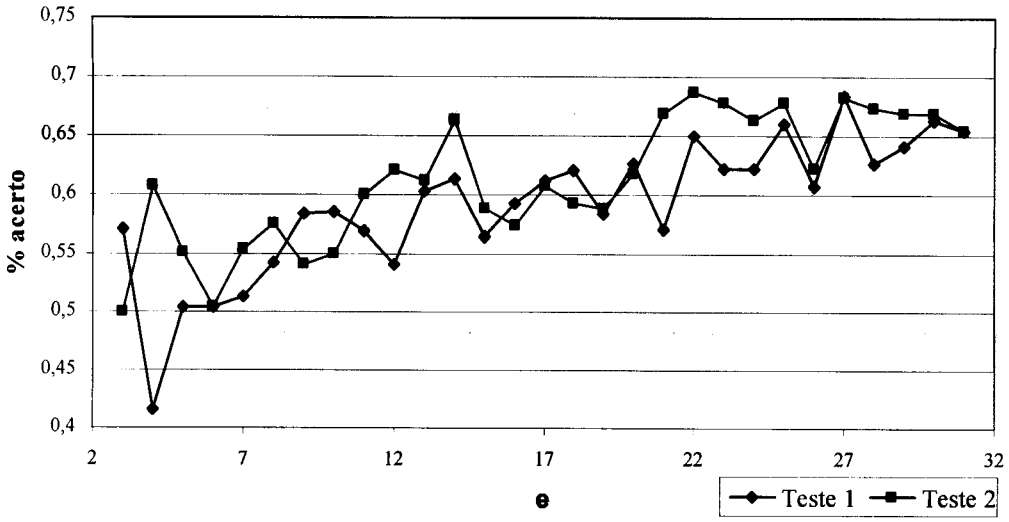


Figura 4-7 – Classificação com c4.5–*teste1 vs teste2 (dados: glass)* [8]

A comparação dos dois testes efectuados permite confirmar que os resultados tendem a aumentar com o aumento de *e*. No entanto, em ambos os testes, as melhores percentagens de acerto não ocorrem para o valor máximo de *e*.

Para além disso constata-se que com o aumento de *e* a variabilidade tende a diminuir.

4.3 Comparativo Todos-contra-todos vs Um-contra-todos vs Ecocs

Nesta secção pretende-se comparar o comportamento dos vários métodos de decomposição de problemas multi-classe, nomeadamente todos-contra-todos, um-contra-todos e *ecocs*.

Numa primeira fase são comparados os resultados obtidos com ecocs de várias dimensões face ao problema multi-classe, ao método um-contra-todos e ao métodos todos-contra-todos. Nesta fase foram apenas usados alguns conjuntos de teste.

Numa segunda fase é apresentado um comparativo semelhante mas apenas para algumas dimensões dos *ecocs* e para um mais conjuntos de dados. Neste fase são apresentados os resultados usando o algoritmo C4.5 e o SVM. Os *ecocs* utilizados foram obtidos com recurso ao algoritmo de perseguição.

[8] Pese embora a função ser discreta, foi feita a ligação entre os vários pontos de um mesmo teste para uma leitura mais fácil.

4.3.1 *Experiências para K=5*

Para k=5 constata-se que os *ecocs* da mesma dimensão do método um-contra-todos foram em ambos os testes ligeiramente superiores a este. Simultaneamente, ambos os métodos obtiveram resultados superiores ao problema original multi-classe. Também no caso dos ecocs de dimensão equivalente ao método todos-contra-todos se verifica um melhor comportamento dos ecocs. Neste caso, os ecocs são superiores ao problema multi-classe, o que não sucede ao todos-contra-todos.

Finalmente, saliente-se neste gráfico que no teste 2, um *ecoc* com a dimensão mínima (e=3) apresenta melhores resultados que o problema multi-classe original e mesmo que os restantes métodos.

Os testes 1 e 2 utilizados nesta secção são os mesmos da secção 4.2.1.

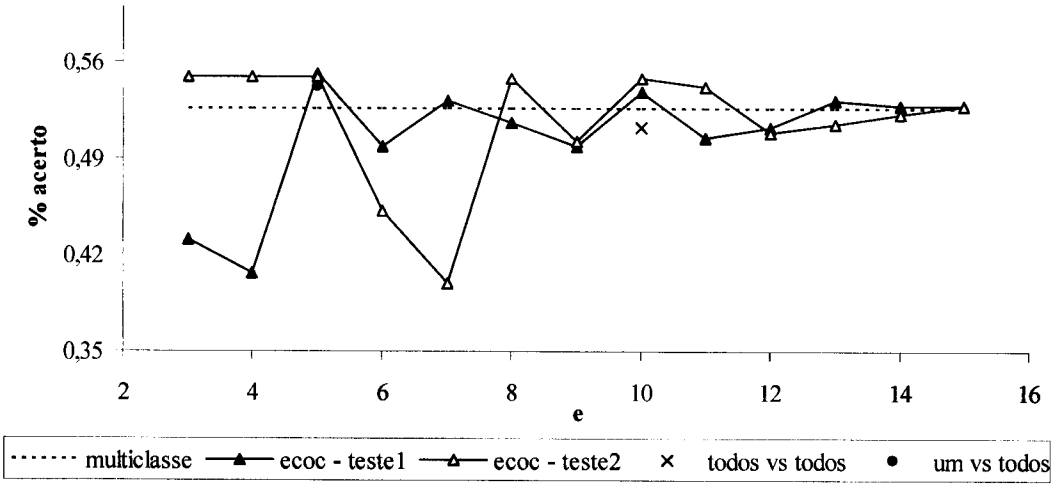


Figura 4-8 – Classificação com c4.5- *ecocs vs um-contra-todos vs todos-contra-todos* (k=5, dados: *heart-disease-cleveland*) [9]

4.3.2 *Experiências para K=6*

Neste conjunto de dados (*glass)* é possível constatar que grande parte dos *ecocs* apresentam melhores resultados que o problema multi-classe. Existe inclusive um *ecoc* com a dimensão mínima com melhor resultado que o método multi-classe.

O método um-contra-todos é inferior ao problema multi-classe, o mesmo sucedendo para os dois *ecocs* gerados com a mesma dimensão.

O método todos-contra-todos apresentou melhores resultados face ao problema multi-classe e face aos dois *ecocs* com a mesma dimensão.

Os testes 1 e 2 utilizados nesta secção são os mesmos da secção 4.2.2.

[9] Pese embora as funções sejam discretas, foi feita a ligação entre os vários pontos de cada teste para uma leitura mais fácil.

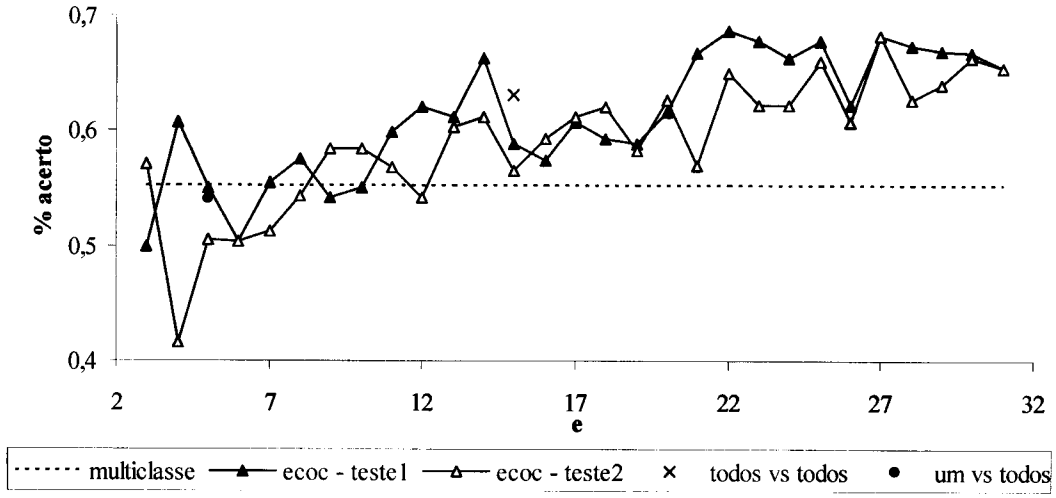


Figura 4-9 – Classificação com c4.5- *ecocs vs um-contra-todos vs todos-contra-todos* (*k*=6, dados: *glass*) [10]

4.3.3 *Comparativo para vários conjuntos de dados*

Nesta secção apresentam-se os resultados comparativos de vários conjuntos de dados para os vários métodos: multi-classe, decomposição pelo método todos-contra-todos, um-contra-todos e ecocs. No caso dos ecocs, são apresentados resultados para vários valores de *e*, nomeadamente:

- valor mínimo de *e* ($ecoc_{min}$);
- *e* de dimensão equivalente a *k'* quando decomposto pelo método um-contra-todos ($ecoc_{ut}$)
- *e* de dimensão equivalente a *k'* quando decomposto pelo método todos-contra-todos ($ecoc_{tt}$)
- valor de *e* retirado da função de avaliação *aval(k,e)* e seguindo o método das rectas exposto em 3.3.5($ecoc_{aval}$). Para os conjuntos de dados usados, a dimensão do *ecoc* é a exposta na Tabela 4-11.

k	Dimensão do $ecoc_{aval}$
5	10
6	18
10	64

Tabela 4-11 – Dimensão do *ecoc* obtido segundo o método exposto em 3.3.5

Os ecocs utilizados apresentavam qualidade B+, excepto quando indicado o contrário. Os testes foram efectuados usando dois algoritmos: C4.5 e SVM.

[10] Pese embora as funções sejam discretas, foi feita a ligação entre os vários pontos de cada teste para uma leitura mais fácil.

À semelhança do realizado nos testes anteriores, os valores apresentados resultam da média da validação cruzada 10. Para verificar se de facto os valores apresentados apresentam significância estatística foi utilizado o teste das médias de Wilcoxon [42].

Na secção 4.3.3.1, são sempre comparadas as médias do problema bi-classe face ao problema multi-classe.

Quando existem vários resultados para uma mesma dimensão de um *ecoc* é utilizado o resultado com maior significância estatística.

4.3.3.1 Algoritmo C4.5

A implementação do C4.5 utilizada foi a R8 fornecida por Ross Quinlan.

Os resultados obtidos encontram-se sintetizados na Tabela 4-12. Para cada conjunto de dados são apresentados a taxa de acertos e a dimensão do problema. Os testes de significância estatística são sempre feitos em relação ao problema multi-classe. As indicações $\perp$ e $\top$ significam que a média apresentada difere da média do problema multi-classe com um nível de significância de pelo menos 95%. A indicação $\perp$ indica uma melhoria e a indicação $\top$ uma degradação do resultado de referência (problema multi-classe).

Dados	multi-classe	um vs todos	todos vs todos	$ecoc_{min}$	$ecoc_{ul}$	$ecoc_{tl}$	$ecoc_{aval}$
Car	93,1%	$\top$ 20,5 %	$\perp$ **94,3 %**	$\top$ 90,6 %	$\top$ 90,4 %	$\top$ 92,1 %	NA[2]
	4	4	6	2	4	6	
heart disease Cleveland	52,5%	54,1 %	51,1 %	54,8 %	**55,1 %**	54,8 %	54,8 %
	5	5	10	3	5	10	10
Glass	55,3%	49,5 %	$\perp$ 63,1 %	57,1 %	50,4 %	58,8 %	$\perp$ **70,6 %**
	6	6	15	3	6	15	18
Sat	86,7%	$\top$ 83,0 %	87,4 %	$\top$ 82,7 %	85,5 %	$\perp$ 89,7 %	$\perp$ **90,2 %**
	6	6	15	3	6	15	18
Pendigits	96,3%	$\top$ 94,2 %	$\perp$ 96,5 %	$\top$ 93,3 %	$\perp$ 97,4 %	$\perp$ 99,1 %	$\perp$ **99,2 %**[1]
	10	10	45	4	10	45	64
Optidigits	90,1%	$\top$ 88,6 %	$\perp$ 94,8 %	$\top$ 85,5 %	$\perp$ 92,3 %	$\perp$ 97,7 %	$\perp$ **98,2 %**[1]
	10	10	45	4	10	45	64

$\perp$ - Melhoria com significância estatística

$\top$ - Degradação com significância estatística

1 - *ecocs* de qualidade B

1 – Uma vez que para $k=4$ a função de avaliação origina apenas um máximo local para e=6, não é aplicável o método exemplificado em 3.3.5.

Tabela 4-12 – Comparativo entre multi-classe, um-contra-todos, todos-contra-todos e *ecocs* (várias dimensões). % de acerto e dimensão do problema. Algoritmo: C4.5. Coluna de referência para os testes de significância: multi-classe

Na tabela estão salientados os melhores resultados para cada conjunto de dados. Da análise dos resultados extraem-se as seguintes conclusões:

- Os *ecocs* apresentaram os melhores resultados em 5 de 6 conjuntos de dados. Em 4 deles, as melhorias obtidas apresentam significância estatística e foram obtidos por *ecoc* cuja dimensão foi definida segundo o método apresentado em 3.3.5 (coluna $ecoc_{aval}$);
- a decomposição pelo método um-contra-todos apenas uma vez foi capaz de apresentar melhores resultados que o problema multi-classe original (mas sem significância estatística). Nos casos em que existe significância estatística, o método um-contra-todos foi sempre inferior ao método multi-classe;
- a decomposição pelo método todos-contra-todos apenas uma vez não foi capaz de melhores resultados que o problema multi-classe original. No entanto, essa única vez não apresenta significância estatística, apresentado para as restantes;
- Nos 4 casos com significância estatística, os *ecoc* de dimensão mínima ($ecoc_{min}$) não degradam substancialmente o problema multi-classe (degradação máxima de 5%);
- os *ecocs* da mesma dimensão da resultante da decomposição pelo método um-contra-todos apresentam melhores resultados que o referido método (coluna $ecoc_{ut}$);
- os *ecocs* da mesma dimensão da resultante da decomposição pelo método todos-contra-todos ($ecoc_{tt}$) apresentam resultados semelhantes a este método, apresentado nalguns casos melhores resultados e noutros casos piores.

Os melhores resultados são obtidos com os *ecocs* cuja dimensão foi definida segundo o método apresentado em 3.3.5 e que tenta obter um equilíbrio entre custo e benefício (coluna $ecoc_{aval}$). Os resultados dos *ecocs* de $ecoc_{aval}$ foram inclusive superiores ao método todos-contra-todos. Não obstante, os resultados obtidos pelo método todos-contra-todos estão consistentes com os referidos em [2]. Estes consistiam em melhorias face ao problema multi-classe.

4.3.3.2 Algoritmo SVM

A implementação das SVM utilizadas foi a fornecida na biblioteca e1071 do R [41].

Os resultados obtidos encontram-se sintetizados na Tabela 1-1. Também aqui, para cada conjunto de dados são apresentados a taxa de acertos e a dimensão do problema. Uma vez que as SVM são apenas capazes de tratar problemas bi-classe, não são apresentados os resultados para o problema multi-classe.

A implementação das SVM usada utiliza a decomposição todos-contra-todos para tratar problemas de $k>2$. Assim, os teste de significância estatística são sempre feitos em relação à decomposição todos-contra-todos. As indicações $\perp$ e $\top$ significam que a média apresentada difere da média do problema multi-classe com um nível de significância de pelo menos 95%. A indicação $\perp$ indica uma melhoria e a indicação $\top$ uma degradação do resultado de referência (problema multi-classe).

Dados	um vs todos	todos vs todos	$ecoc_{min}$	$ecoc_{ut}$	$ecoc_{it}$	$ecoc_{aval}$
Car	⊤ 21,8%	83,3%	**86,1%**	⊤ 21,2%	84,4%	NA
	4	6	2	4	6	
heart disease Cleveland	54,1%	**59,4%**	56,8%	58,4%	57,8%	57,8%
	5	10	3	5	10	10
Glass	48,2%	51,4%	**55,2%**	48,6%	50,4%	54,2%
	6	15	3	6	15	18
Sat	⊤ 86,5%	**90,9%**	⊤ 87,5%	⊤ 88,8%	⊤ 88,1%	⊤ 89,0%
	6	15	3	6	15	18
Pendigits	⊤ 99,1%	**99,6%**	⊤ 98,9%	⊤ 99,3%	⊤ 99,3%	⊤ 99,4%
	10	45	4	10	45	64
Optidigits	⊤ 97,0%	98,5%	⊤ 97,4%	⊤ 98,0%	⊥ **99,3%**	⊥ 98,5%
	10	45	4	10	45	64

⊥ - Melhoria com significância estatística

⊤ - Degradação com significância estatística

1 - *ecocs* de qualidade B

Tabela 4-13 – Comparativo entre um-contra-todos, todos-contra-todos e *ecocs* (várias dimensões). % de acerto e dimensão do problema. Algoritmo: SVM. Coluna de referência para os testes de significância: todos-contra-todos.

Na tabela estão salientados os melhores resultados para cada conjunto de dados. Da análise dos resultados extraem-se as seguintes conclusões:

- Os *ecocs* de uma forma geral apresentam melhores resultados que o método todos-contra-todos utilizado internamente pelas SVM em 3 de seis conjuntos de dados. No entanto, apenas num caso essa melhoria é estatisticamente significativa;
- O método todos-contra-todos é melhor que os *ecocs* em 3 casos, sendo esse resultado estatisticamente significativo em 2 casos. Nesses dois casos, o melhor dos *ecocs* representou uma degradação máxima de 2,1%;
- os *ecocs* da mesma dimensão da resultante da decomposição pelo método um-contra-todos apresentam melhores resultados que o referido método em 5 de 6 conjuntos de dados.

Os melhores resultados foram obtidos pelos métodos de decomposição todos-contra-todos e pelos *ecocs*, existindo um grande equilíbrio entre estes métodos. Os resultados do método um-contra-todos foram sempre inferiores ao método todos-contra-todos, contrariando o exposto em [10].

4.4 Resumo

As experiências deste capítulo dividiram-se em 3 secções:

- Análise de vários métodos para criar *ecocs* (Algoritmo de repulsa, GARA e o método de perseguição);
- Análise do comportamento dos ecocs face à evolução da dimensão do *ecoc*;
- Comparação dos diversos métodos de decomposição de problemas multi-classe, com especial foco nos *ecoc*;

Dos 3 métodos avaliados para a criação de *ecocs*, os melhores resultados foram obtidos pelo método de perseguição apresentado em 3.2.4. Este método apresenta uma elevada capacidade de obtenção de *ecocs* viáveis (aqueles cuja expressão (65) é positiva) associada a uma elevada capacidade de criação de bons *ecocs*. O método GARA é também capaz de criar bons *ecocs* mas é inferior na criação de *ecocs* viáveis.

A avaliação da evolução do comportamento dos *ecocs* para várias dimensões *e* num mesmo problema permitiu constatar alguma variabilidade nos valores obtidos. Essa variabilidade tende a diminuir com o aumento da dimensão do *ecoc*. À medida que se aumenta a dimensão, aumentam-se também as fronteiras de decisão, permitindo que más fronteiras sejam diluídas nas boas. Quando a dimensão é mais reduzida, a escolha das fronteiras pode recair em fronteiras mais ou menos adequadas. Constata-se também que nem sempre os melhores resultados são obtidos pelos *ecocs* de maior dimensão.

Finalmente foram comparados os métodos de decomposição usando conjuntos de dados. No C4.5 os resultados apontaram para melhorias do método todos-contra-todos e dos *ecocs* face ao problema multi-classe. Apenas o método um-contra-todos foi inferior ao problema multi-classe. Globalmente, os *ecocs* cuja dimensão foi definida pelo método apresentado em 3.3.5 apresentaram os melhores resultados. Nas SVM, o método todos-contra-todos e os *ecocs* apresentaram mais uma vez os melhores resultados. Globalmente, no entanto, existe um maior equilíbrio entre estes dois métodos.

5 Conclusões

Nesta secção apresentam-se de forma resumida as principais conclusões do trabalho realizado.São também apresentadas algumas notas e caminhos a considerar para trabalho futuro.

5.1 Conclusões

Uma das questões que se coloca na criação de *ecocs* é determinar o que é um bom e um mau *ecoc*. Nesta dissertação foi apresentada uma função que permite aferir de forma qualitativa a qualidade de um *ecoc*. Com base nessa mesma função foi também definido e testado um novo algoritmo para a criação de *ecocs*: o método da perseguição. Os testes realizados sobre este método foram bastante positivos, comprovando ser uma boa solução para criar *ecocs*.

Outra questão que se coloca aquando da utilização de *ecocs* para a decomposição de problemas de classificação multi-classe é qual deverá ser o tamanho dos *ecocs*, uma vez que o número de dimensões possíveis cresce de forma exponencial com o número de classes do problema original. Nesta dissertação são apresentadas vários métodos que permitem reduzir esse número de dimensões, tendo especial importância a função de avaliação. Para esta função, foi utilizado um método que permite encontrar um ponto de equilíbrio entre dimensão e desempenho. Esse ponto determina a dimensão do *ecoc* para um problema de k classes que garante um equilíbrio entre redundância e correcção de erros. Os testes efectuados com conjuntos de dados recorrendo à dimensão dada por esse ponto foram positivos, sobretudo usando o algoritmo C4.5.

Finalmente foi estudado o comportamento dos *ecocs* com várias dimensões para um mesmo conjunto de dados. Para valores próximos da dimensão mínima dos *ecocs* constata-se alguma variabilidade dos resultados para *ecocs* distintos mas com a mesma dimensão. Concluí-se também que o aumento da dimensão do *ecoc* permite reduzir a variabilidade dos resultados. Finalmente, verifica-se que nem sempre os *ecocs* com a maior dimensão apresentam os melhores resultados.

5.2 Trabalho futuro

Esta tese faz uma reflexão sobre a utilização de *ecocs* em problemas de classificação. Não sendo um trabalho acabado, abre caminho para futuras investigações, nomeadamente:

- Adaptação dos ecocs ao problemas.

Nesta tese é apresentado um conceito de bons e maus *ecocs*. No entanto, essa definição é feita de forma independente do problema de classificação. Dois problemas multi-classe com o mesmo número de classes terão os mesmo bons e maus *ecocs*. No entanto, é legitimo considerar que o mesmo *ecoc* poderá conduzir a mais melhorias num problema do que no outro. Relembre-se que os ecocs definem fronteiras de decisão e uma boa fronteira num problema poderá ser uma má fronteira noutro. Assim, a

definição de critérios que permitam adaptar os melhores *ecocs* aos problemas de forma independente, poderá levar o uso dos *ecocs* ainda mais longe.

- A utilização de probabilidades

Na literatura ainda são poucas as referências à utilização de probabilidades nos *ecocs*. No entanto, poderá ser um caminho que abra novas luzes sobre os *ecocs*. Refira-se adicionalmente que o uso de probabilidades poderia ser articulados com o ponto anterior, para a escolha de *ecocs* adaptados a problemas de decisão.

- Extender a aplicação dos métodos propostos

A fim de efectuar uma mais profunda avaliação dos métodos propostos nesta tese, deveria ser extendida a sua aplicação a um maior número de conjuntos de dados e algoritmos. Dois dos algoritmos que se pretende utilizar num futuro próximo são o QUEST [37] e o UFFT [4].

- Afinação do ponto de equilíbrio

A utilização de técnicas baseadas no MDL para determinar a melhor dimensão do tamanho do *ecoc* para determinado problema.

6 Bibliografia

[1] C: Blake, E. Keogh, and C.J.Merz. *UCI repository of Machine Learning databases*, 1999.

[2] J. Furnkranz. *Round Robin Classification*. Journal of Machine Learning Research 2:721-747, MIT Press, 2002.

[3] T. G. Dietterich, G. Bakiri. *Solving Multiclass Learning Problems via Error-Correcting Output Codes*. Journal of Artificial Intelligence Research, 2:263-286, 1995.

[4] J. Gama, P. Medas, R. Rocha. *Forest Trees for On-line Data*. ACM (ed.), Proceedings of ACM SAC'04, Nicosia, Cyprus, Vol 2, pp. 632 - 636, 2004

[5] T. Hastie, R. Tibshirani. *Classification by Pairwise Coupling*. Advances in Neural Information Processing Systems 10, 1998.

[6] J. Furnkranz. *Round Robin Ensembles*, Intelligent Data Analysis, 7(5):385-404, 2003.

[7] Johannes Furnkranz, E. Hullermeier. *Pairwise Preference Learning and Ranking*, In Proceedings of the 14th European Conference on Machine Learning (ECML-03), pp.145-156, Springer Verlag 2003.

[8] J. Carlson. *Error Correcting Codes: An Introduction Through Problems*, Thecnical report http://www.math.utah.edu/ carlson/ugc/ecc/ecc.pdf , 1999.

[9] Miguel Moreira. *The use of boolean concepts in general classification contexts*, Tese de doutouramento da Escola Politécnica Federal de Lausanne, Suiça, 2000.

[10] R. Rifkin, A. Klautau, *In Defense of One-vs-All Classification*. Journal of Machine Learning Research, 5:101-141, MIT Press, 2004.

[11] E. L. Allwein, R. E. Shapire, Y. Singer. *Reducing multiclass to binary: a unifying approach for margin classifiers*. Journal of Machine Learning Research, 1:113-141, MIT Press, 2000.

[12] E. Alba, J. F. Chicano, B. Dorronsoro, G. Luque, *Diseño de Códigos Correctores de Errores con Algoritmos Genéticos*. Actas del Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (*MAEB'04)*, Córdoba, pp. 51-58, 2004.

[13] E. Alba, J. F. Chicano, *Solving the Error Correcting Code Problem with Parallel Hybrid Heuristics*. ACM (ed.), Proceedings of ACM SAC'04, Nicosia, Cyprus, Vol 2, pp. 985-989, 2004.

[14] T. G. Dietterich, *Aproximate statistical testes for comparing supervised classification learning algorithms*. Neural Computation, 10(7):1895-1924, 1998.

[15] F. Ricci, D. W. Aha. *Extending Local Learners with Error-Correcting Output Codes*. Technical report, 9701-08, 1997.

[16] E. V. York, *Algebraic description and construction of error correcting codes: a linear system point of view*. Tese de doutoramento da Universidade de Notre Dame, EUA, 1997.

[17] W. C. Huffman, V Pless, *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003

[18] B. E. Boser, I. Guyon, V. N. Vapnik, *A Training Algorithm for Optimal Margin Classifiers*, in Proceedings of the Fifth Annual Workshop on Computational Learning Theory 5:144-152, 1992.

[19] F. Rosenblatt, *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386-408, 1958

[20] M. L. Minsky, S. A. Papert, *Perceptrons*. Cambridge, MA: MIT Press, 1969

[21] S. Czanner, A. Wetzel *A Tutorial Overview of Machine Learning, Feature Selection and Classification With Application to Cancer Detection*, Pittsburgh Supercomputing Center, Mellon Institute, Carnegie Mellon University, Pittsburgh, 2002.

[22] R. Hamming, *Coding and Information Theory*. Prentice-Hall, Englewood Cliffs, 1980.

[23] R. Hamming, *Error-detecting and error-correcting codes*. Bell System Technical Journal 29, pp 147-160,1950.

[24] C. E. Shannon, *A mathematical theory of communication*. Bell System Technical Journal 27, pp. 379-423, 623-656, 1948.

[25] J. R. Quinlan. *Discovering rules by induction from large collections of examples.* Expert Systems in the Micro-electronic Age, Michie, D., Editor, Edinburgh University Press, Edinburgh, Scotland pp 168-201, 1979.

[26] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kauffmann, Los Altos, CA, 1993.

[27] E. Kong, T Dietterich. *Probability estimation using error-correcting output coding*. Procedings of the IASTED International Conference: Artificial Intelligence and Soft Computing. ACTA Press, 1997.

[28] O. Dekel, Y. Singer. *Multiclass Learning by Probalistic Embeddings*. Advances in Neural Information Processing Systems 15 (pp. 945-952). MIT Press, 2002.

[29] E. Kong, T Dietterich. *Error-correcting output coding corrects bias and variance*. In the 12th International Conference on Machine Learning, pp313-321. Morgan Kaufmann, 1985.

[30] E. Frank, S. Kramer. *Ensembles of Nested Dichotomies for Multi-class Problems*. Machine Learning, Proceedings of the 21th International Conference. OmniPress, 2004.

[31] I. S. Reed, G. Solomon, *Polynomial Codes Over Certain Finite Fields*, SIAM Journal of Applied Math., vol. 8, , pp. 300-304, 1960.

[32] R. Bose, D. Ray-Chaudhuri, *On a class of error correcting binary group codes,* Information and Control, vol. 3, pp. 68–79, 1960.

[33] A. C. Lorena, A. Carvalho, *An Hybrid GA/SVM Approach for Multiclass Classification with Directed Acyclic Graphs*. XVII SBIA04, LNAI, Springer Verlag, 2004.

[34] B. Kijsirikul, N. Ussivakul, *Multiclass Support Vector Machines using Adaptive Directed Acyclic Graph*, Proceedings of International Joint Conference on Neural Networks, pp 185-208, 2002.

[35] J. C. Platt, N. Cristiani, J. Shawe-Taylor, *Large Margin DAGs for Multiclass Classification*, Advances in Neural Information Processing Systems, vol 12, pp 547-553, MIT Press, 2000.

[36] K. Dontas, K. De Jong, *Discovery of maximal distance codes using genetic algorithms*, Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence, pp 905--811, IEEE Press, 1990.

[37] W.-Y. Loh, Y.-S. Shih, *Split selection methods for classification trees*, Statistica Sinica, vol. 7, pp. 815-840, 1997.

[38] L. Breinman, J. Friedman, R. Olsen, C. Stone, *Classification and Regression Trees*, Wadsworth International Group, 1984.

[39] J. Gama, *Iterative Bayes*, Intelligent Data Analysis 4, pp 475-488, IOS Press, 2000.

[40] G. S. Maddala, *Microeconomics - Theory and Applications Nova Iorque*, McGraw-Hill Book Company, 1989.

[41] R Development Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing (www.R-project.org), Vienna, Austria, 2004.

[42] G. Bhattacharyya e R. Johnson, *Statistical Concepts and Methods*, New York, John Willey \& Sons, 1977.

Anexo A – Resumo dos conjuntos de dados utilizados

- **Car**

Número de classes	Número de atributos	Número de exemplos
4	6	1 728

Com base em algumas características técnicas, de preço e de conforto de um carro obter a aceitação do mesmo por parte dos Clientes.

- **Heart disease Cleveland**

Número de classes	Número de atributos	Número de exemplos
5	13	303

O objectivo deste conjunto de dados é determinar se um paciente apresenta doenças cardíacas, baseado em informação diversa sobre o mesmo (tal como idade, sexo, fumador, etc.)

Os dados fazem referência ao Cleveland Clinic Foundation e foi responsável pela sua extracção Robert Detrano, M.D., Ph.D.

- **Glass**

Número de classes	Número de atributos	Número de exemplos
6	9	214

Determinação do tipo de vidro, com base nas suas características químicas. O estudo deste tipo de informação surgiu na investigação criminal, onde bocados de vidro podem ser deixados numa cena de crime.

- **Sat**

Número de classes	Número de atributos	Número de exemplos
6	36	6 435

Baseado na informação obtida por valores espectrais de uma satélite, determinar qual o tipo de ocupação de solo que está a ser visualizado.

- **Pendigits**

Número de classes	Número de atributos	Número de exemplos
10	16	7 494

Determinar o algarismo escrito manualmente por uma pessoa num painel LCD específico com uma caneta e passado para um bitmap.

- **Optidigits**

Número de classes	Número de atributos	Número de exemplos
10	64	5 620

Baseado em bitmaps normalizados com informação de algarismos escritos manualmente, determinar qual o algarismo.