FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Árvores de Decisão Desequilibradas para deteção de erros em transações de comércio externo usando técnicas de data mining

Johny Gueirez

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Carlos Manuel Milheiro de Oliveira Pinto Soares

Co-orientador: Rita Paula Almeida Ribeiro

1 de Julho de 2014

Árvores de Decisão Desequilibradas para deteção de erros em transações de comércio externo usando técnicas de data mining

Johny Gueirez

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Eugénio da Costa Oliveira

Arguente: Paulo Cortez

---

1 de Julho de 2014

Resumo

Hoje em dia é frequente existirem grandes quantidades de dados de onde são extraídos diferentes tipos de conhecimento, para posteriormente se tomarem diversas decisões. Um dos grandes desafios na análise de dados é o tratamento de casos raros que, face a um modelo (ou comportamento normal esperado), representam anomalias. A deteção dessas anomalias pode ser utilizada numa grande variedade de aplicações tais como, deteção de fraudes de cartões de crédito, deteção de intrusos em termos de segurança na internet, deteção de falhas em sistemas críticos, entre outras.

Um exemplo de um desses problemas são as estatísticas do comércio externo publicadas pelo Instituto Nacional de Estatística (INE). Assim, é importante que as anomalias criadas por erros nas declarações feitas pelas organizações ao INE, sejam detetadas para garantir que essas estatísticas são fiáveis. Para a deteção dessas anomalias serão estudadas e utilizadas variantes de árvores de decisão, que são algoritmos de *data mining* e *machine learning*. Embora as árvores de decisão não tenham sido originalmente desenvolvidas para a detecção de anomalias, foram propostas algumas adaptações com esse fim.

Este projeto tem dois objetivos: realizar um estudo empírico sobre o desempenho de variantes de árvores de decisão no problema de deteção de anomalias; e, com base nos resultados desse estudo, propor uma nova variante, implementá-la e testá-la.

Abstract

Nowadays there are frequently large amounts of data, which are used to extract different types of knowledge. A major problem existing in these data are the outliers. The detection of these outliers is important for a variety of applications, such as detecting credit card fraud, network intrusion, fault detection in critical systems, among others.

An example of such problems are the foreign trade statistics published by the Instituto Nacional de Estatística (INE). Thus, it is important that the anomalies created by erroneous statements made by organizations to the INE are detected to ensure that these statistics are reliable. For the detection of these anomalies we will study and use variants of decision trees, which are algorithms for data mining and machine learning. Although decision trees have not been originally developed for the detection of anomalies, some adaptations were proposed for this purpose.

In this project we intend to do a study about the detection of outliers and adopt a way to detect these values through the use of decision tree algorithms.

Agradecimentos

Esta dissertação só foi possível graças à contribuição de várias pessoas, que merecem ser reconhecidas.

Em primeiro lugar queria agradecer à FEUP pela qualidade de ensino que me proporcionou.

Em seguida, queria agradecer ao meu orientador Carlos Soares e à co-orientadora Rita Ribeiro por todo o empenho neste projeto. Através das suas ideias e supervisionamento foi possível o alcance dos objetivos da dissertação.

Queria também agradecer a todas as amizades feitas no percurso académico, pois sem eles o caminho era possível, mas com um maior número de obstáculos.

Por fim, queria agradecer aos meus pais por a ajuda e acompanhamento presentes em todos os momentos, mas acima de tudo pela confiança que depositaram em mim durante este meu percurso.

Johny Emanuel de Jesus Gueirez

O único lugar onde sucesso vem antes do trabalho é no dicionário.

Albert Einstein

Conteúdo

1 Introdução **1**
- 1.1 Objetivos . . . 1
- 1.2 Estrutura da Dissertação . . . 2

2 Trabalho Relacionado **3**
- 2.1 *Data Mining* e *Machine Learning* . . . 3
- 2.2 Deteção de Anomalias . . . 4
- 2.3 Árvores de Decisão . . . 5
  - 2.3.1 Indução de Árvores de Decisão . . . 6
  - 2.3.2 Critérios de Divisão . . . 7
  - 2.3.3 Poda . . . 9
  - 2.3.4 Medidas de Avaliação . . . 10

3 Árvores de decisão para deteção de casos raros **15**
- 3.1 Medidas de avaliação para problemas com classes desbalanceadas . . . 15
- 3.2 Métodos Existentes . . . 17
  - 3.2.1 Class Confidence Proportion . . . 17
  - 3.2.2 One-sided Purity e One-sided Extremes . . . 17
- 3.3 One-sided F-measure Maximization Classification . . . 18

4 Resultados **21**
- 4.1 Descrição do Problema . . . 21
- 4.2 Descrição dos Dados . . . 22
- 4.3 Metodologia Experimental . . . 24
- 4.4 Resultados . . . 25

5 Conclusões e Trabalho Futuro **29**
- 5.1 Trabalho Futuro . . . 29

Referências **33**

CONTEÚDO

Lista de Figuras

2.1 Exemplo de dois conjuntos de dados com anomalias 4
2.2 Uma possível árvore de decisão para o problema de diagnosticar pacientes [1] . . 6
2.3 Indução de um classificador e dedução das classes para novas amostras [1] 7
2.4 Comparação das medidas de impureza para um problema de classificação binária [16] 9
2.5 Taxa de erro em relação ao tamanho da árvore de decisão 9
2.6 Exemplo de uma poda numa árvore de decisão [7] 10
2.7 Performance de 5 classificadores no espaço ROC [6] 13

3.1 Comparação da nossa proposta de árvore de decisão e uma árvore tradicional . . 20

4.1 Exemplo de conjunto de dados do sistema INTRASTAT [14] 22
4.2 Análise exploratória do conjunto de dados 23
4.3 Número de transações em cada mês 24
4.4 Número de casos raros em relação à família de produtos 24
4.5 Gráfico ROC dos três algoritmos de árvores de decisão 27

Lista de Tabelas

2.1 Conjunto de dados para o diagnóstico da saúde de pacientes 5
2.2 Matriz de confusão 11

4.1 Desempenho médio das diferentes árvores 25
4.2 Desvio padrão das diferentes medidas nas árvores 26

Capítulo 1

Introdução

Este projeto constitui um trabalho na área de *data mining* e *machine learning*. O problema em questão é a deteção de casos raros (*outliers*) num conjunto de dados.

A existência de casos raros coloca alguns desafios a nível da análise de dados o que faz com que a sua deteção assuma um papel relevante. Assim, a deteção de casos raros pode assumir duas vertentes distintas. Numa primeira vertente estes casos raros são considerados erros, que podem, por exemplo, levar a conclusões estatísticas erradas sobre o conjunto de dados. Dessa forma, a sua deteção e eliminação conduz à obtenção de estatísticas mais robustas. Numa segunda vertente, que vai ser o foco deste estudo, estes casos raros são considerados como informação relevante no sentido de descreverem uma anomalia. Nesta vertente, ao contrário do que é tradicional em *data mining*, pretende-se encontrar padrões que se afastem daquilo que é mais comum. Por exemplo, um caso raro numa transação de um cartão de crédito pode significar uma fraude, assim como um caso raro no tráfico de rede de um computador pode significar que o computador está a enviar dados para um destino não autorizado. Nestes dois exemplos anteriores, os casos raros assumem o papel de uma anomalia no domínio do problema, tornando-se importante a sua deteção.

Uma técnica muito comum em data *mining* e *machine learning* são as árvores de decisão. As árvores de decisão representam um modelo preditivo, e assim torna-se uma boa abordagem para o nosso problema. Os algoritmos de indução de árvores de decisão tentam encontrar padrões frequentes num conjunto de dados. Apesar disso, o algoritmo tem sido usado [14] e adaptado [19] para a identificação de casos raros.

1.1 Objetivos

Pretende-se, então, desenvolver uma nova adaptação do algoritmo de árvores de decisão que seja capaz de detetar casos raros. Para isso foi realizado um estudo empírico do desempenho dos algoritmos baseado em árvores de decisão existentes para a sua deteção. Esse estudo será feito

com base nos dados de transações de comércio externo disponíveis [14]. Com base no conhecimento adquirido sobre o comportamento dos algoritmos, foi proposto um novo algoritmo. O novo algoritmo foi avaliado empiricamente e comparado com os existentes.

1.2 Estrutura da Dissertação

A estrutura do documento é a seguinte: no capítulo 2 serão discutidos os diferentes conceitos que serão utilizados neste trabalho. No capítulo 3, o problema da deteção de casos raros usando árvores de decisão é abordado, mostrando diferentes técnicas existentes e apresentando a nossa proposta. No capítulo 4 serão mostrados os diferentes aspetos do nosso problema e os resultados encontrados. Finalmente, no capítulo 5 é discutido o alcance dos objetivos e trabalho futuro.

Capítulo 2

Trabalho Relacionado

Neste capítulo serão discutidos os diferentes conceitos utilizados neste projeto. Em primeiro lugar, será mostrado em que consistem os processos de *data mining* e os algoritmos de *machine learning*. Em seguida, será abordado o problema da deteção de anomalias, mostrando algumas das técnicas existentes para resolver esse problema. Na secção seguinte, será exposto em que consiste uma árvore de decisão. Para isso, será apresentado o algoritmo por trás da sua indução, nomeadamente o algoritmo TDIDT, os diferentes critérios de divisão existentes usados no algoritmo e técnicas de poda de árvores de decisão. Por fim, serão mostradas diferentes medidas de avaliação, que tem como objetivo avaliar o desempenho de uma árvore de decisão.

2.1 *Data Mining* e *Machine Learning*

Data mining é um conjunto de processos para descobrir padrões num grande conjunto de dados, através de diferentes tipos de métodos, como por exemplo, o uso de algoritmos de *machine learning*. Um algoritmo de *machine learning*, é um algoritmo do ramo da inteligência artificial, que tem como propósito a construção e o estudo de modelos, através da análise de um conjunto de dados. Assim, o objetivo de um processo de data *mining* é extrair informação de dados e transformá-la, para posteriormente, se tomarem decisões com esse conhecimento.

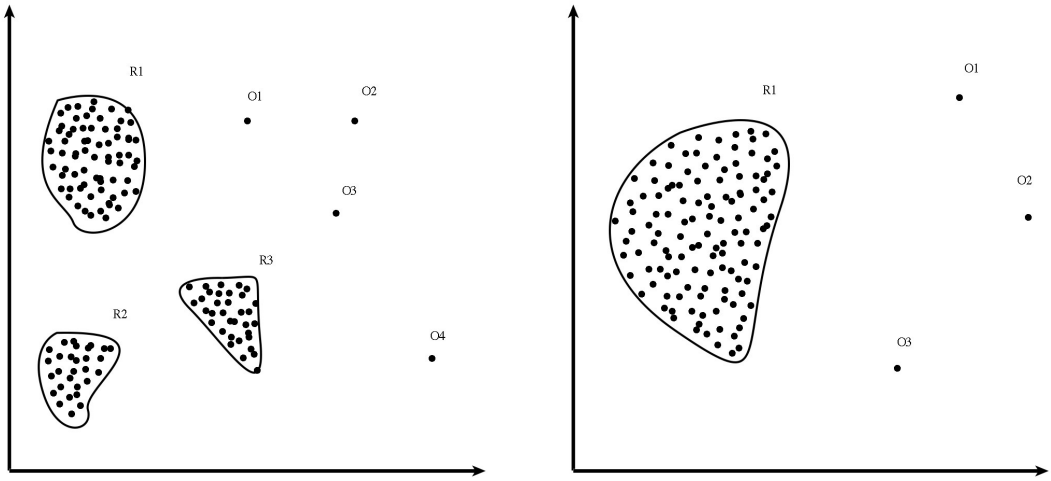
Dado um conjunto de dados $T = \{\overline{x}_i, y_i\}_{i=1}^{n}$, em que $\overline{x}_i$ é o vetor de variáveis, discretas ou contínuas, que representa os atributos de cada observação, y_i é a variável objetivo, e n é o número de observações. Os algoritmos de *machine learning* podem representar um problema de classificação ou um problema de regressão. O que distingue os dois problemas é o tipo da variável objetivo y_i. Se esta for categórica, estamos perante um problema de classificação, pois existe um número finito de valores que a variável pode assumir. O domínio de y será igual a $\{y_{C_1}, y_{C_2}, ..., y_{C_k}\}$, em que k é o número de classes existentes no conjunto de dados e y_{C_k} representa o valor de y para a classe C_k. Por exemplo, o conjunto de dados na Tabela 1, representa um problema de classificação, pois

o diagnóstico, que será a variável objetivo, assume apenas dois valores possíveis: doente ou saudável. Por outro lado, um problema é considerado de regressão quando y_i é contínua, assumindo valores numéricos. Um exemplo de um problema de regressão poderá ser o calculo do preço de uma casa, considerando diferentes fatores económicos e políticos.

A aprendizagem feita sobre conjunto de dados por algoritmos de *machine learning*, pode ser de três tipos diferentes: não supervisionada, semi-supervisionada e supervisionada. Os algoritmos não supervisionados recebem como input um conjunto de dados não classificado, ou seja, sem a variável objetivo y_i. Os algoritmos semi-supervisionados recebem como input um conjunto de dados em que apenas alguns exemplos estão classificados. Por fim, os algoritmos supervisionados recebem como input um conjunto de dados em que todos os seus exemplos estão classificados.

2.2 Deteção de Anomalias

Segundo Chandola [5] anomalias são observações de um conjunto de dados que não estão conforme o comportamento normal esperado. A Figura 2.1a ilustra um conjunto de dados de dimensão 2, contendo anomalias. Existem três regiões, R1, R2 e R3, que correspondem às regiões de maior concentração de observações, que terão o comportamento esperado para as três classes do problema. Os pontos O1, O2, O3 e O4 por se distanciarem dessas regiões, podem ser considerados anomalias. A Figura 2.1b representa o nosso problema, em que existe apenas uma classe normal, representada por R1 e anomalias representadas por O1, O2 e O3 que pretendemos detetar.



(a) Conjunto de dados com três classes normais

(b) Conjunto de dados com uma classe normal

Figura 2.1: Exemplo de dois conjuntos de dados com anomalias

Dentro das técnicas de deteção de anomalias, é do nosso interesse referir as técnicas baseadas em classificação. Estas, baseiam-se em, através de um conjunto de dados de treino, construir um modelo preditivo, que vai ser usado para classificar uma nova observação.

Essas técnicas podem assumir diferentes abordagens. Uma delas é baseada na construção de regras sobre os diferentes atributos x_i do conjunto de dados de treino, que através delas captura o comportamento normal de um sistema. A técnica de deteção de anomalias baseada em regras consiste em dois passos. O primeiro passo é a aprendizagem de regras através de um conjunto de dados de treino, usando um algoritmo de *machine learning*. O segundo passo é encontrar, para uma nova observação, a regra que melhor a captura.

Tal como qualquer técnica de *data mining*, as técnicas de deteção de anomalias, podem ser de três tipos diferentes, dependendo da variável objetivo y_i: não supervisionadas, semi-supervisionadas ou supervisionadas. As técnicas não supervisionadas por não receberem um conjunto de dados classificado, têm como objetivo encontrar padrões no conjunto de dados, partindo do princípio que observações que se desviam significativamente desses padrões são anomalias. As técnicas semi-supervisionadas recebem um conjunto de dados apenas com as classes normais classificadas. As técnicas supervisionadas, que serão o foco deste estudo, recebem um conjunto de dados classificado, que no nosso caso representa se uma observação é ou não uma anomalia.

2.3 Árvores de Decisão

As árvores de decisão são um dos algoritmos existentes de classificação. O modelo gerado assume a forma de uma árvore (Figura 2.2), que é construída através de diferentes testes que dividem os dados nos diferentes nós.

O problema de classificação da Tabela 2.1 representa um problema no diagnóstico da saúde de pacientes. Através do conjunto de variáveis $\bar{x}_i$, que serão os atributos que caracterizam cada observação, nomeadamente, febre, enjoo, manchas e dor, cada paciente é classificado em doente ou saudável, que será a variável objetivo, y_i, deste problema. Este conjunto de dados é o input dado ao algoritmo de indução da árvore de decisão, que constrói a árvore representada na Figura 2.2.

Segundo Tan et al. [15], uma árvore de decisão possui três tipos de nós:

- um nó raiz, que não possui nenhuma aresta de entrada e zero ou mais arestas de saída;
- nós internos, cada qual com exatamente uma aresta de entrada e duas ou mais arestas de saída;

Tabela 2.1: Conjunto de dados para o diagnóstico da saúde de pacientes

Exemplo	**Febre**	**Enjoo**	**Manchas**	**Dor**	**Diagnóstico**
T1	Sim	Sim	Pequenas	Sim	Doente
T2	Não	Não	Grandes	Não	Saudável
T3	Sim	Sim	Pequenas	Não	Saudável
T4	Sim	Não	Grandes	Sim	Doente
T5	Sim	Não	Pequenas	Sim	Saudável

Figura 2.2: Uma possível árvore de decisão para o problema de diagnosticar pacientes [1]

- nós folhas, cada qual com uma única aresta de entrada e nenhuma de saída, pois é o nó que determina a qual classe o exemplo pertence;

Dessa forma, é possível utilizar uma árvore de decisão para classificar um novo paciente como saudável ou doente. Para isso, basta partir do nó raiz da árvore e ir percorrendo-a através das respostas aos testes dos nós internos, até chegar a um nó folha, que indica a classe correspondente. Além da obtenção da classe, uma grande vantagem é que a trajetória percorrida até ao nó folha representa uma regra, facilitando a decisão pelo utilizador.

A construção de uma árvore de decisão é chamada de processo de indução, que vai ser aprofundado na secção seguinte.

2.3.1 Indução de Árvores de Decisão

O processo de indução de árvores de decisão exige um grande poder computacional, mas uma vez construída, o seu uso é imediato e muito rápido, tornando-se uma das vantagens da sua utilização. Na Figura 2.3 é ilustrado um diagrama desse processo. Dado um conjunto de dados de treino (A), já classificado, é construído o modelo através do algoritmo de aprendizagem (B). Em seguida, a esse modelo é dado como input um conjunto de dados de teste (C), que através do processo de dedução, classifica essas observações.

Segundo Rokach e Maimon [12], o processo de indução é, formalmente, definido como:

" Dado um conjunto de treino T composto por um vetor de atributos $\bar{x} = x_1, x_2, \ldots, x_n$ *e uma variável objetivo y com uma distribuição desconhecida, o objetivo é induzir um classificador ótimo com um mínimo de erro generalizado."*

O *Top-Down Induction of Decision Tree* (*TDIDT*) é um algoritmo bem conhecido e é utilizado como base para muitos algoritmos de indução de árvores de decisão. Entre eles os mais conhecidos são ID3 [9], C4.5 [10] e CART [2]. O *TDIDT* produz regras de decisão de forma implícita numa árvore de decisão, a qual é construída por sucessivas divisões das observações de acordo com os valores de seus atributos preditivos $\bar{x}$. Dado o conjunto de treino T contendo classes $C_1, C_2, \ldots, C_k$, o esqueleto do algoritmo de *TDIDT* é baseado nos seguintes pontos.

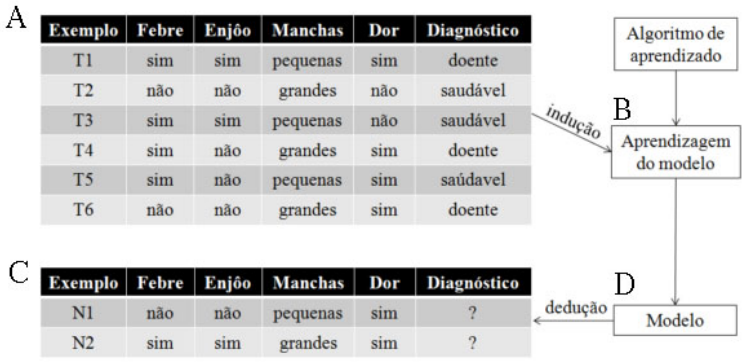


Figura 2.3: Indução de um classificador e dedução das classes para novas amostras [1]

1. Se todos os elementos num nó pertencerem à mesma classe C_j, então é criado um nó folha identificado pela classe C_j.

2. Se o ponto número 1 não se verificar, é então selecionado o "melhor" atributo preditivo x_i, de acordo com um dado critério de avaliação. Uma vez selecionado o atributo, o conjunto de dados T é dividido, de acordo com os valores de x_i, em subconjuntos $T_1, T_2, \ldots$ e são construidas as sub-árvores que definem esses conjuntos.

3. O algoritmo é chamado recursivamente até que se verifique o critério de paragem definido (por ex. todas as folhas são puras, ou seja, contêm observações da mesma classe).

Basicamente, o algoritmo *TDIDT* é um algoritmo recursivo de busca gulosa que procura, sobre um conjunto de atributos, aqueles que "melhor" dividem o conjunto de dados T em subconjuntos. Dessa forma o algoritmo em cada nó procura encontrar uma solução ótima local, na esperança que cada escolha leve até à solução ótima global. Para encontrar o atributo que conduz à "melhor" divisão, o algoritmo utiliza diferentes critérios de avaliação, que serão apresentados na secção que se segue.

2.3.2 Critérios de Divisão

O algoritmo de indução tem que escolher qual o atributo preditivo que será utilizado em cada nó da árvore. Essa escolha será baseada em diferentes critérios, tais como impureza, distância ou dependência. A maior parte dos algoritmos tenta dividir os dados de um nó de forma a minimizar o grau de impureza dos nós filhos. Se um nó for totalmente puro significa que todos os exemplos que fazem parte desse nó pertencem à mesma classe.

O algoritmo ID3 [9], pioneiro em indução de árvores de decisão, bem como o algoritmo C4.5 [10] que adveio do anterior, utiliza a entropia como medida de impureza. A entropia de um nó N é dada pela equação:

$$Entropia(N) = -\sum_{C=1}^{k} p(C|N) log_2 p(C|N))) \tag{2.1}$$

em que $p(C|N)$ é a fração de elementos pertencentes à classe C, no nó N, e k é o número de classes. Assumindo que estamos perante um problema de classificação, em que a variável objetivo y_i assume apenas duas classes possíveis, a entropia assume um valor no intervalo $[0,1]$, em que é 0 quando o nó em questão é totalmente puro e 1 quando o nó é totalmente impuro, ou seja, que existem tantos elementos de uma classe como de outra. Para determinar o quanto um atributo preditivo é bom, é necessário recorrer ao conceito de ganho que traduz a diferença entre a impureza do nó pai e a soma da impureza das partições resultantes, multiplicadas pelas suas probabilidades. O ganho associado a uma divisão S, é definido na seguinte equação:

$$GanhoEntropia(S) = Entropia(N_{pai}) - \sum_{j=1}^{n} \frac{|N_j|}{|N_{pai}|} Entropia(N_j) \tag{2.2}$$

em que n é o número de nós filhos resultantes da divisão, $|N_{pai}|$ é o tamanho da partição de dados associada ao nó pai e $|N_j|$ é o número de elementos associados ao nó filho N_j.

Outra medida também bastante conhecida é o *Gini*, utilizado no algoritmo CART [2], que é baseado, tal como a entropia, na pureza do nó. A medida *Gini* é dada pela seguinte equação:

$$Gini(N) = 1 - \sum_{C=1}^{k} p(C|N)^2 \tag{2.3}$$

Assim como no cálculo do ganho da entropia o ganho da medida *Gini* é dado por:

$$GanhoGini(S) = Gini(N_{pai}) - \sum_{j=1}^{n} \frac{|N_j|}{|N_{pai}|} Gini(N_j) \tag{2.4}$$

em que n é o número de nós filhos resultantes da divisão, $|N_{pai}|$ é o número de elementos do nó pai e $|N_j|$ é a soma de elementos associados ao nó filho N_j.

Uma outra medida que também pode ser usada, é o erro da classificação. Essa medida é dada por:

$$ErroClassificacao(N) = 1 - max_C(p(C|N)) \tag{2.5}$$

em que $max_C(p(C|N))$ é o máximo da fração de elementos das diferentes classes existentes no nó.

Na Figura 2.4 são apresentados os valores que estas medidas podem produzir, conforme a fração de elementos de uma classe, considerando um problema de classificação binária.

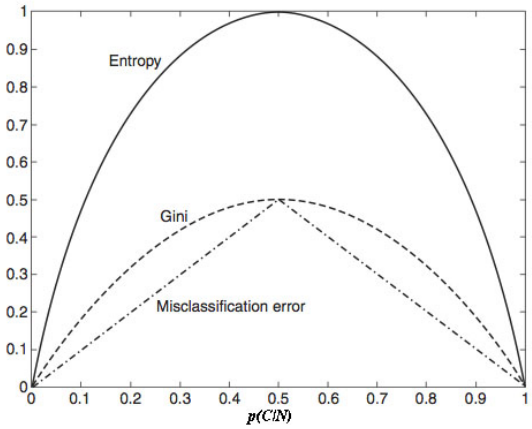


Figura 2.4: Comparação das medidas de impureza para um problema de classificação binária [16]

2.3.3 Poda

O *overfitting* é o sobre-ajuste excessivo do modelo ao conjunto de dados. Isto acontece sempre que a árvore estende a sua profundidade até ao ponto de classificar individualmente todos os exemplos do conjunto de dados de treino, reduzindo a sua capacidade de generalização. Segundo Breiman et al. [2] para ultrapassar este problema em árvores de decisão, deve-se utilizar técnicas de poda. Essas técnicas permitem detetar e excluir sub-árvores do modelo, com o objetivo de melhorar a taxa de acerto de novas observações. Na Figura 2.5 podemos ver a relação entre a taxa de acerto e o número de nós da árvore, considerando os dados de treino e os dados de teste.

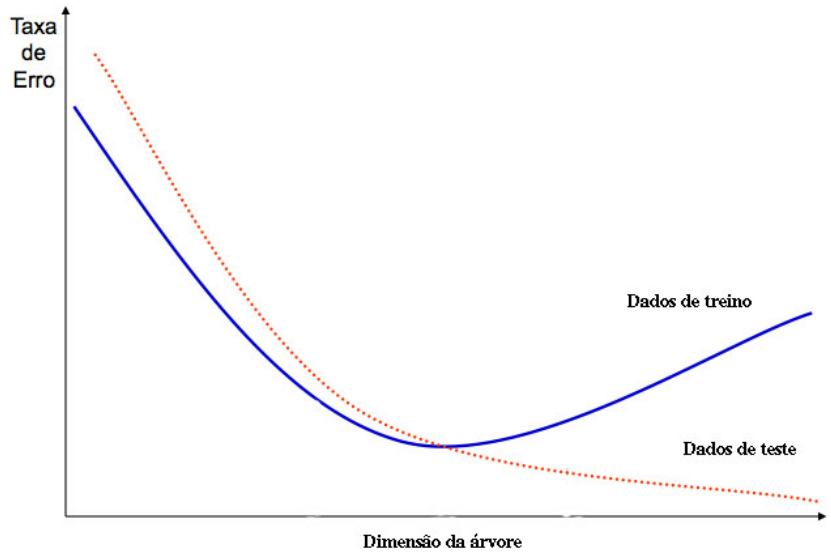


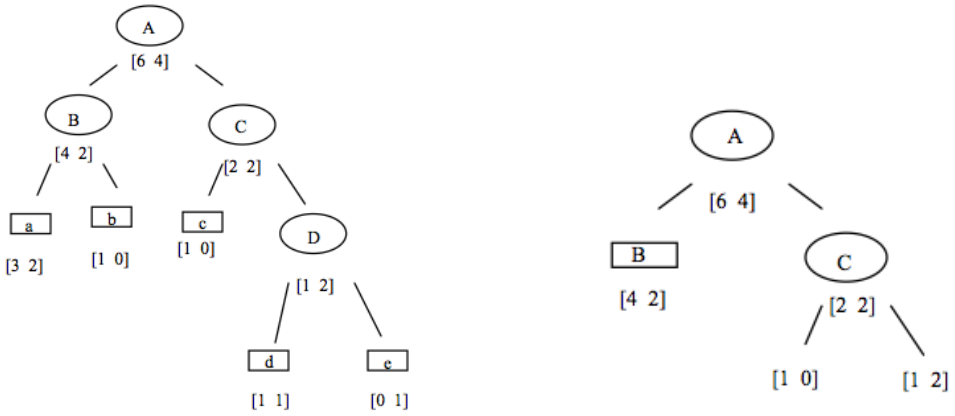
Figura 2.5: Taxa de erro em relação ao tamanho da árvore de decisão

Existem diversas técnicas de poda de árvores de decisão, dividindo-se em dois grupos: pré-poda e pós-poda. As técnicas de pré-poda são realizadas no momento em que se está a induzir a

árvore, e, por essa razão, essas técnicas começam pelo nó raiz e vão até aos nós folha à medida que a própria árvore é construída. Por exemplo, o ganho da entropia, apresentado na secção anterior, pode ser usado para podar a árvore. A estratégia passa por calcular o ganho associado ao split de cada nó. No caso desse ganho ser inferior ao um dado limite, esse nó torna-se um nó folha da árvore. Contudo, uma dificuldade destas técnicas é encontrar um limite que não seja nem muito grande, levando a uma poda mais radical, nem muito pequeno, pois faria com que quase nenhum nó tivesse um valor inferior a esse limite.

Por outro lado, as técnicas de pós-poda são realizadas apenas quando a árvore já se encontra construída. Após a sua construção, o algoritmo, começando pelos nós folha, calcula a soma das taxas de erros dos nós filho resultantes de uma divisão. Essa taxa será a fração de elementos pertencentes ao nó, que não pertencem à sua classe. Em seguida é calculada a taxa de erro do nó pai dessa divisão, e é comparada com a dos filhos. Se a taxa de erro do nó pai for igual ou inferior à soma da dos filhos, a árvore é podada, eliminando os nós filhos. Na Figura 2.6 podemos ver um exemplo de uma pós-poda. Olhando para a Figura 2.6a, a taxa de erro no nó B é igual a $\frac{2}{6}$, enquanto que nos nós filhos é de $\frac{2}{5}$ no nó "a"e 0 no nó "b". Assim, como $\frac{2}{6} < \frac{2}{5} + 0$, os nós "a"e "b"são eliminados, transformando-se apenas no nó B (Figura 2.6b). O mesmo acontece para o nó D.

As técnicas de pré-poda são mais rápidas, porém menos eficazes do que as de pós-poda, pelo facto de correrrmos o risco de interromper o crescimento da árvore ao selecionar uma árvore sub-ótima [2].



(a) Árvore original

(b) Árvore após a poda

Figura 2.6: Exemplo de uma poda numa árvore de decisão [7]

2.3.4 Medidas de Avaliação

Na literatura de recuperação de informação [4] a noção de relevância é definida como a utilidade de um output, no contexto da pesquisa efetuada por um utilizador. Duas medidas para definir a relevância são o *recall* e a *precision*. Ambas as medidas avaliam o modelo através de uma classe

positiva e uma classe negativa. A classe positiva é a classe de interesse ao problema, que no nosso caso representa um caso raro. A classe negativa será a que representa uma observação normal. Assim, T_p representa o número de exemplos da classe positiva verdadeiros e F_p o número de exemplos positivos falsos. Por outro lado, T_n representa o número de exemplos da classe negativa verdadeiros e F_n o número de exemplos negativos falsos. Na Tabela 2.2 é mostrada a matriz de confusão para perceber melhor estes conceitos.

Tabela 2.2: Matriz de confusão

	observado		
previsto		**Normal**	**Caso raro**
	Normal	T_n	F_n
	Caso raro	F_p	T_p

O *recall* é a fração de casos raros corretamente previstos pelo modelo, em relação ao número total de casos raros do nosso conjunto de dados. A *precision* é a fração de casos raros corretamente previstos pelo modelo, em relação ao número total de previsões de casos raros. As seguintes equações representam estas duas medidas.

$$Recall = \frac{T_p}{T_p + F_n} \tag{2.6}$$

$$Precision = \frac{T_p}{T_p + F_p} \tag{2.7}$$

A medida *F-measure* é uma medida que combina o *recall* e a *precision*, usando um parâmetro β, para controlar a importância do *recall* sobre a *precision*. A *F-measure* é definida na seguinte equação:

$$F - measure = \frac{(\beta^2 + 1) * precision * recall}{\beta^2 * precision * recall} \tag{2.8}$$

Uma das medidas mais utilizadas em relação à avaliação de modelos é a *accuracy*, que mede a taxa de acertos, independentemente da classe, ou seja:

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \tag{2.9}$$

Para além destas medidas, é importante referir a definição da medida que avalia o esforço. O esforço será a fração do número de casos raros previstos no nosso modelo em relação ao número total de observações. A seguinte equação representa o cálculo do esforço:

$$Esforco = \frac{T_p + F_p}{T_p + T_n + F_p + F_n} \tag{2.10}$$

Assim, um modelo que tenha um esforço muito grande representa um mau modelo, pois através dele serão gastos muitos recursos, nomeadamente, o tempo perdido na eliminação das observações que o modelo prevê como caso raro, mas que não o são na realidade. De notar que, um

modelo com um grande esforço pode não ser considerado um mau modelo, desde que a ele esteja associado um valor elevado de *recall*. Essa conclusão deve-se ao facto de, se um modelo tiver um *recall* elevado, o esforço será associado à eliminação de casos raros previstos que realmente são casos raros.

2.3.4.1 Curvas ROC

Uma alternativa à utilização de medidas é o uso de gráficos. Segundo Fawcett [6], o gráfico ROC é baseado na probabilidade de deteção, ou taxa de verdadeiros positivos, $tpr = \frac{T_p}{T_p+F_N}$, e na probabilidade de falsos alarmes, ou taxa de falsos positivos, $fpr = \frac{F_p}{F_p+T_n}$. Para se construir o gráfico, desenha-se fpr no eixo das abcissas e tpr no eixo das ordenadas. Para obter um ponto no espaço ROC, correspondente a um modelo de classificação, calculam-se essas duas variáveis, através de uma matriz de confusão.

No espaço ROC o ponto (0,0) representa a estratégia de nunca classificar um exemplo como positivo. O ponto (1,1) representa a estratégia de sempre classificar um exemplo como positivo. O ponto (0,1) representa o modelo perfeito, pois todos os exemplos positivos e negativos são classificados corretamente. Por fim, o ponto (1,0) representa o modelo que classifica erradamente todos os exemplos. A reta de declive 1 que passa na origem representa a estratégia de classificar aleatoriamente um exemplo. Isso significa que o modelo prevê o mesmo número de exemplos positivos e negativos corretamente.

Na Figura 2.6 é apresentada a performance de 5 classificadores no espaço ROC. O classificador C encontra-se na reta de declive 1 que passa na origem e, portanto, é considerado um classificador aleatório que não tem qualquer informação à cerca das classes. Pontos que estejam no triângulo superior dessa diagonal, por exemplo A, representam modelos que têm uma performance superior ao aleatório. Por outro lado, pontos que estejam no triângulo inferior, por exemplo E, representam modelos com má performance, pois apesar de possuírem informação sobre as classes, aplicam-na incorretamente [6].

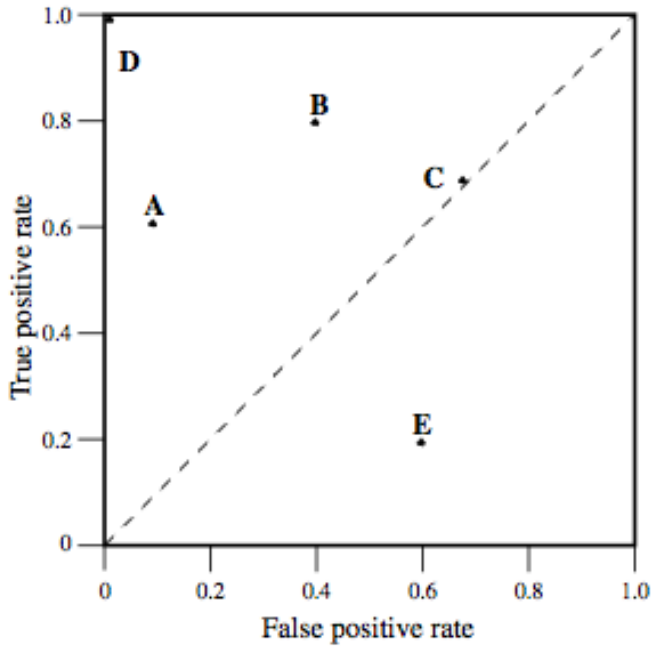


Figura 2.7: Performance de 5 classificadores no espaço ROC [6]

Capítulo 3

Árvores de decisão para deteção de casos raros

Neste capítulo serão apresentadas as árvores de decisão para a deteção de casos raros. Em primeiro lugar, será discutido o problema da existência de classes desbalanceadas e medidas para avaliar esses problemas. Em seguida, serão apresentados métodos existentes para a deteção de casos raros através de árvores de decisão. Por fim, será apresentada a nossa proposta de uma nova adaptação do algoritmo de árvores de decisão.

3.1 Medidas de avaliação para problemas com classes desbalanceadas

Um conjunto de dados diz-se desbalanceado se existir uma classe que contenha um número muito maior de exemplos do que as restantes classes, ou se, por outro lado, existir, pelo menos, uma classe que contenha um número muito reduzido de elementos em relação às restantes classes. No nosso problema essa distinção não é necessária por se tratar de um problema de classificação binária sobre um conjunto de dados desbalanceado onde $|C_0| \gg |C_1|$. C_0 é a classe negativa no nosso problema e representa uma observação normal. C_1 é a classe positiva e representa a classe de interesse no nosso problema, ou seja, um caso raro.

A existência de um conjunto de dados desbalanceado torna-se um problema na construção de uma árvore de decisão. Devido ao algoritmo procurar padrões frequentes, dá pouca relevância às classes minoritárias, pois elas, por representarem um número reduzido de exemplos, não influenciam a formação desses padrões.

Na secção 2.3.4, foi introduzida a definição de *accuracy*. Embora seja uma medida comum para a avaliação de modelos de classificação, não é adequada para o nosso problema, dado que o nosso conjunto de dados é desbalanceado. Como a nossa classe positiva representa uma minoria

no conjunto de dados, avaliando o modelo através da *accuracy*, poderia conduzir a um desempenho bom, devido à previsão correta de muitos exemplos da classe negativa. O nosso objetivo é a deteção de casos raros (classe positiva), e então, utilizar esta medida levaria a uma falsa avaliação dos modelos. Assim, ao invés de utilizar a *accuracy*, medidas como o *recall* e a *precision* representam uma melhor avaliação, por fazerem a distinção entre a classe positiva e negativa.

Os algoritmos de indução das árvores de decisão ID3 [9] e C4.5 [10] apresentados no capítulo anterior, utilizam como medida de ganho a definição de entropia. Considerando o nosso problema de classificação binária a equação de entropia é equivalente a:

$$Entropia(N) = -p(C_0|N)log_2p(C_0|N) - p(C_1|N)log_2p(C_1|N) \tag{3.1}$$

em que C_0 representa a classe negativa e C_1 representa a classe positiva. N_L e N_R representam os dois sub-nós da divisão, respetivamente, o nó esquerdo e o nó direito.

Numa árvore de decisão, a trajetória desde a raiz até uma folha pode ser vista como uma regra, sendo que várias medidas podem ser utilizadas para avaliar essa regra. Assim, segundo Liu e Chawla [8] e considerando o nosso problema, $Suporte(y \cup N)$ representa a proporção de elementos no conjunto de dados que contém C e N, ou seja:

$$Suporte(N \cup C) = \frac{|C_N|}{|T|} \tag{3.2}$$

em que $|C_N|$ é o número de elementos da classe C no nó N e $|T|$ é o número de elementos totais no conjunto de dados. O $Suporte(N)$ representa a proporção de elementos no nó N em relação ao conjunto de dados, ou seja:

$$Suporte(N) = \frac{|N|}{|T|} \tag{3.3}$$

em que $|N|$ é o número de elementos do nó N.

A confiança de uma regra de associação é dada por

$$Confianca(N \rightarrow C) = \frac{Suporte(N \cup C)}{Suporte(N)} \tag{3.4}$$

assim, através das equações 3.3 e 3.4 podemos afirmar que a probabilidade condicional $p(C|N)$ é equivalente à $Confianca(N \rightarrow C)$, através da seguinte dedução:

$$Confianca(N \rightarrow C) = \frac{|C_N|}{|N|} = p(C|N) \tag{3.5}$$

Considerando que C_1 é a classe menos frequente e que para um nó N, o $Suporte(N)$ é constante, podemos afirmar que $Confianca(N \rightarrow C_1)$ é tanto maior quanto maior for $Suporte(N \cup C_1)$. Dessa maneira se $C_1 \ll C_0$, então, $Confianca(N \rightarrow C_1) \ll Confianca(N \rightarrow C_0)$.

Com isto conclui-se que mesmo que a regra $N \rightarrow C_0$ não seja uma regra muito significativa, é fácil ter uma confiança alta, devido a C_0 ter um grande número de elementos. Assim torna-se

difícil encontrar um "boa" regra $N \rightarrow C_1$ que tenha uma confiança maior que uma "má" regra $N \rightarrow C_0$, levando a uma distribuição desequilibrada das classes.

3.2 Métodos Existentes

3.2.1 Class Confidence Proportion

Como vimos anteriormente, o maior problema na indução de uma árvore de decisão tradicional prende-se com o fator $p(C|N)$ que é igual a $Confianca(N \rightarrow C)$. Liu e Chawla [8] apresenta-nos uma alternativa a essa confiança, representada na seguinte equação:

$$CC(N \rightarrow C) = \frac{Suporte(N \cup C)}{Suporte(C)} \tag{3.6}$$

Num conjunto de dados desbalanceado, o nó N pode não "explicar", por exemplo, a classe C_1, e assim através da equação anterior, a classe C_0 em nada vai influenciar a obtenção de uma "boa" confiança para a regra $N \rightarrow C_1$. Mas, obter uma elevada *ClassConfidence* não é suficiente para conseguirmos uma boa divisão, é necessário que a classe em questão seja mais interessante que a correspondente classe alternativa. É proposta, então, a definição de *ClassConfidenceProportion* na seguinte equação:

$$CCP(N \rightarrow C_1) = \frac{CC(N \rightarrow C_1)}{CC(N \rightarrow C_0) + CC(N \rightarrow C_1)} \tag{3.7}$$

Uma regra com um elevado *ClassConfidenceProportion* significa que a classe C dessa regra, comparada com a sua classe alternativa, tem uma maior *ClassConfidence*, e assim é mais provável que a regra que dá origem ao nó N discrimine a classe C.

Como concluímos que o fator $p(C|N)$, presente na definição de entropia no algoritmo de indução de árvores de decisão tradicional, tem um mau desempenho quando se trata de um conjunto de dados desbalanceado, esse fator é substituído pela definição de *ClassConfidenceProportion*, pois toma valores compreendidos entre [0,1], tal como o fator $p(C|N)$. A função de entropia é então definida por:

$$Entropia_{CCP}(N) = -\sum_{C=1}^{k} CCP(N \rightarrow C) log_2 CCP(N \rightarrow C) \tag{3.8}$$

3.2.2 One-sided Purity e One-sided Extremes

Buja e Lee [3] apresenta dois algoritmos diferentes para substituir o algoritmo tradicional de árvores de decisão. Num problema de classificação binária, seja $p_L = \frac{|N_L|}{|N_{pai}|}$, a proporção de exemplos contidos na partição de dados do nó pai que, como resultado da divisão, são incluídos na partição de dados do nó filho esquerdo. Nestas condições, p_L e p_R são tais que, $p_L + p_R = 1$.

O problema que Buja e Lee [3] tentam solucionar é eliminar a dependência que existe entre os dois nós filhos quando calculamos o ganho da divisão, pois tipicamente, o critério que calcula a qualidade de uma divisão é dada por:

$$Ganho(N) = p_L loss_L + p_R loss_R \tag{3.9}$$

em que $loss_L$ e $loss_R$ representam o ganho em relação ao nó esquerdo e ao nó direito, respetivamente, através de um dos diferentes critérios apresentados na secção 2.3.2.

Com isso, através da eliminação de essa dependência, se for encontrado um nó que tenha um grau de pureza elevado, o algoritmo considera que se trata de uma boa divisão, mesmo que o outro nó resultante da divisão não tenha um grau de pureza muito bom. Dessa forma, o algoritmo tenta encontrar nós puros (*One-sided Purity*) e extremos (*One-sided Extremes*).

O *One-sided Purity* consiste em substituir o ganho da divisão, pelo máximo das seguintes probabilidades:

$$GanhoOSP(S) = max(p_L^0, p_L^1, p_R^0, p_R^1) \tag{3.10}$$

em que p_y^x, representa a probabilidade da classe x, no nó y. Este algoritmo valoriza as divisões que dão origem aos nós mais puros de qualquer uma das classes.

Na variante *One-sided Extremes* é importante escolhermos qual a classe de interesse. No nosso caso, a classe de interesse será a classe que caracteriza um caso raro, isto é, C_1, o algoritmo substitui o ganho da divisão, pelo máximo das seguintes probabilidades:

$$GanhoOSE(S) = max(p_L^1, p_R^1) \tag{3.11}$$

Como o nosso problema consiste em conseguir isolar os exemplos da classe positiva, a variante *One-sided Extremes* torna-se a melhor opção.

3.3 One-sided F-measure Maximization Classification

O método que se segue é a nossa proposta para o problema de deteção de casos raros e é uma adaptação do método usado para a deteção de valores extremos e regressão, apresentado por Torgo e Ribeiro [19], para um problema de classificação. Este método tem como base o princípio seguido por Buja e Lee [3], no sentido de não basear a escolha de uma boa divisão através da média dos pesos de cada sub-nó resultante da divisão. Ao invés disso, se encontrar um "bom" nó, a divisão será também considerada uma boa divisão.

O critério de divisão é baseado nas medidas de *recall* e *precision*, apresentadas na secção 2.3.4, para encontrar uma boa regra. Através dessas medidas, a *F-measure* de um nó é calculada e o ganho da divisão é dado pela seguinte equação:

$$GanhoOSF = max(F(N_L), F(N_R)) \tag{3.12}$$

em que N_L é o sub-nó esquerdo, enquanto que N_R o direito; $F(N)$ é a *F-measure* para o nó N.

Para o cálculo *da F-measure* é necessário calcular o *recall* e a *precision* do nó. No trabalho original foram usadas medidas de *recall* e *precision* para regressão. No nosso projeto, usamos as medidas tradicionais, que são mais adequadas para classificação. Assim o *recall* vai ser igual a:

$$Recall(N) = \begin{cases} \frac{|C_{1_N}|}{|C_1|} & \text{if } |C_{1_N}| > |C_{0_N}| \\ 0 & \text{if } |C_{0_N}| > |C_{1_N}| \end{cases} \tag{3.13}$$

em que $|C_{1_N}|$ é o número de exemplos da classe positivas no nó N e $|C_{0_N}|$ o número de exemplos da classe negativa. $|C_1|$ é o número de exemplos da classe positiva no conjunto de dados de treino.

A *precision* em cada nó é calculada da seguinte forma:

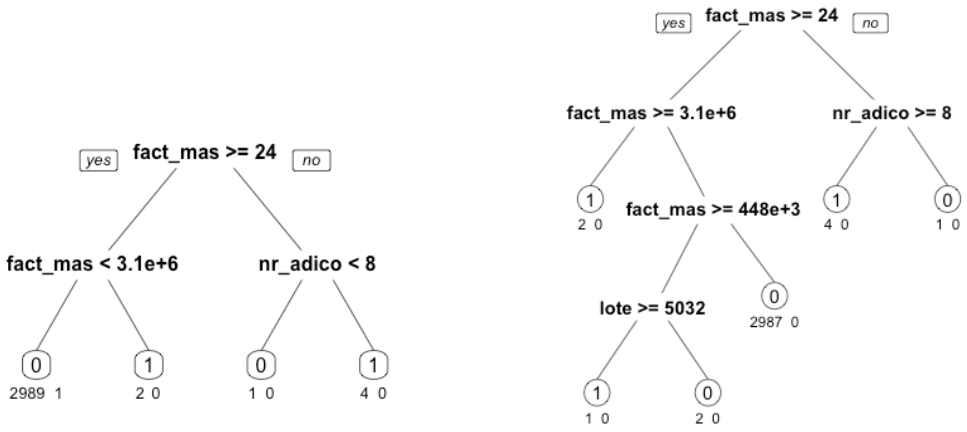
$$Precision(N) = \begin{cases} \frac{|C_{1_N}|}{|N|} & \text{if } |C_{1_N}| > |C_{0_N}| \\ 0 & \text{if } |C_{0_N}| > |C_{1_N}| \end{cases} \tag{3.14}$$

em que $|N|$ é o número total de observações no nó N.

O valor da *precision* é equivalente à fração de elementos da classe positiva no nó N, apresentada no método *One-sided Purity* e *One-sided Extremes* na secção 3.2.2. No nosso método combinamos essa medida com a medida de *recall*, através da *F-measure*.

Outro aspeto importante quando falamos da construção de uma árvore de decisão é o critério para a paragem do seu crescimento. Na secção 2.3.3 referimos os dois métodos para a poda de árvores de decisão, chegando à conclusão que a pós-poda seria mais eficaz. No entanto, exemplos da nossa classe positiva são insignificantes de uma perspetiva estatística, por existir um número muito reduzido no conjunto de dados. Assim, estratégias de pós-poda tornam-se muito difíceis de implementar considerando o nosso problema, pois o nosso interesse é maximizar a obtenção de nós que apenas contenham exemplos da classe positiva. Assim, o único critério que faz com que a árvore de decisão pare de crescer, é se não existir nenhum exemplo da classe positiva num nó, pois como o nosso interesse é isolar a classe positiva, de nada nos interessa continuar a dividir exemplos da classe negativa.

Na Figura 3.1 podemos ver um exemplo de uma árvore tradicional, resultante do algoritmo CART [2], em relação á árvore obtida com o novo critério proposto. Analisando as duas árvores podemos concluir que a nossa proposta é igual à árvore tradicional no 2º nível. Apesar disso, a árvore tradicional, através de métodos de poda, não isolou um exemplo da classe positiva, enquanto a nossa árvore o fez, tornando-se mais eficaz para o nosso objetivo.



(a) Árvore de decisão CART

(b) Árvore de decisão OSF

Figura 3.1: Comparação da nossa proposta de árvore de decisão e uma árvore tradicional

Capítulo 4

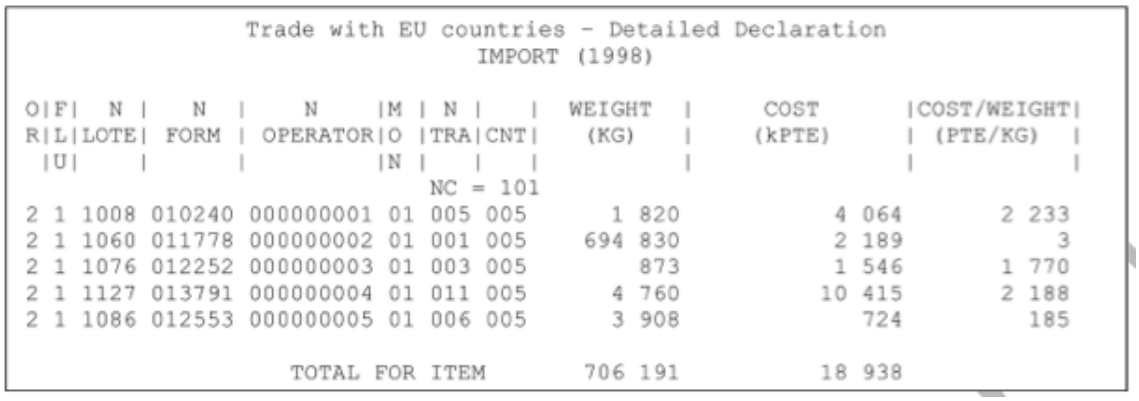
Resultados

Neste capítulo será descrito o nosso problema, bem como o conjunto de dados associado a ele. Em seguida, será apresentada a metodologia experimental usada para a nossa solução, onde a partir desse conjunto de dados foram induzidas árvores de decisão *CART*, árvores de decisão *One-sided Extremes* e árvores de decisão *One-sided F-measure Maximization Classification*. Foi estudado os resultados de cada uma e comparado o desempenho entre elas.

4.1 Descrição do Problema

O problema que pretendemos resolver, é detetar erros num conjunto de dados obtido do sistema INTRASTAT (Figura 4.1). O sistema INTRASTAT é referente a estatísticas das trocas de bens entre Estados-Membros, comunicadas ao Instituto Português de Estatística num formato específico. Esses dados podem conter erros. Por exemplo, na Figura 4.1, podemos ver que a segunda linha se deve tratar de um erro, pois o peso toma um valor muito grande, provavelmente por ter sido introduzido em gramas em vez de quilogramas. Alguns destes erros podem não ter efeito nas estatísticas finais, enquanto outros podem afetá-las significativamente.

Normalmente, quando todos os dados de um mês de transações são introduzidos na base de dados, estes são, posteriormente, analisados manualmente para a eliminação de possíveis erros existentes. Assim, o objetivo do nosso estudo passa por, usando algoritmos de árvores de decisão, detetar a maior parte dos erros existentes, para ser usado o menor trabalho manual possível na sua eliminação. Com isso espera-se diminuir o número de transações a serem analisadas, e que contenham o maior número de transações com erros. De frisar que a eficiência do algoritmo não é posta em causa, dado que dificilmente o algoritmo demorará mais tempo que a análise manual sobre o conjunto de dados completo.



Trade with EU countries - Detailed Declaration
IMPORT (1998)

O R	F L U	N LOTE	N FORM	N OPERATOR	M O N	N TRA	CNT	WEIGHT (KG)	COST (kPTE)	COST/WEIGHT (PTE/KG)
						NC = 101				
2	1	1008	010240	000000001	01	005	005	1 820	4 064	2 233
2	1	1060	011778	000000002	01	001	005	694 830	2 189	3
2	1	1076	012252	000000003	01	003	005	873	1 546	1 770
2	1	1127	013791	000000004	01	011	005	4 760	10 415	2 188
2	1	1086	012553	000000005	01	006	005	3 908	724	185
				TOTAL FOR ITEM				706 191	18 938	

Figura 4.1: Exemplo de conjunto de dados do sistema INTRASTAT [14]

4.2 Descrição dos Dados

O conjunto de dados é composto por dezassete variáveis que caracterizam cada transação, mais um último valor que corresponde à variável objetivo binária, que nos indica se estamos perante um erro ou não. Os atributos de cada transação são:

- Data – Corresponde à data do recebimento do ficheiro por parte do INESC;
- Origem – Tipo de meio usado para declarar a transação (disquete ou formulário);
- Fluxo – Diz-nos se a transação se trata de uma exportação ou de uma importação;
- Lote – Número do lote.
- Decl – Documento usado para fazer a declaração;
- Op – Declarante;
- Mês – Mês da transação;
- Adi – Número da adição na declaração;
- País – País de proveniência/destino;
- Nc- Código do produto;
- Massa – Massa líquida;
- Fact – Valor faturado;
- FactMassa – Valor faturado por unidade de massa;
- MediaNcValQuant – Média do valor faturado por unidade de massa para as adições neste ficheiro do mesmo código de mercadorias;

- DpNcValQuant – Desvio-padrão do valor faturado por unidade de massa para as adições neste ficheiro do mesmo código de mercadorias;
- DistanciaNcValQuant – Distância normalizada do valor faturado por unidade de massa ao atributo MediaNcValQuant;
- NrAdicoes – Número de adições deste código de mercadorias neste ficheiro;
- Erro – Valor binário que indica se estamos perante um erro.

O conjunto de dados usado é referente a 13 meses de transações, em que o número total de transações existentes é de 647.308, sendo 2222 correspondente a casos raros.

A Figura 4.2 representa uma análise estatística simples a cada uma das variáveis. No conjunto de dados existem duas variáveis binárias, Origem e Fluxo, e duas variáveis discretas, Mês e País. As restantes são variáveis contínuas. Podemos, também observar, que em apenas três variáveis existem valores em falta.

Atributo	Tipo	Moda	1º Quartil	Mediana	3º Quartil	Valor mínimo	Valor máximo	Média	Nº valores em falta
Origem	Discreta binária	2	-	-	-	1	2	-	0
Fluxo	Discreta binária	1	-	-	-	1	2	-	0
Lote	Contínua	-	1121	3023	60008	0	9500	3661	0
Decl	Contínua	-	20990	35056	70570	6	916210	47142	0
Op	Contínua	-	722	1568	3258	1	7985	2266	0
Mes	Discreta	2	2	3	7	1	11	-	0
Adi	Contínua	-	3	9	28	1	689	30,66	0
Pais	Discreta	11	4	5	11	1	959	-	0
Nc	Contínua	-	24751427	49899216	78324600	132716	99759927	51728408	0
Massa	Contínua	-	15	169	1535	0	101863488	11862	18692
Fact	Contínua	-	83	451	2177	0	3626518	4063	1621
Fact_massa	Contínua	-	804	2838	7916	1	862126000	59114	0
Media_nc_val_quant	Contínua	-	1746	4919	12094	1	516412000	59114	0
Dp_nc_val_quant	Contínua	-	1047	4067	16713	0	268165000	74355	0
Distancia_nc_val_quant	Contínua	-	0.2363	0.4724	0.8604	0	23.2494	0.6632	19818
Nr adicoes	Contínua	-	9	24	60	1	1032	51,48	0

Figura 4.2: Análise exploratória do conjunto de dados

Como vai ser do nosso interesse dividir o conjunto de dados em meses e em código de produto, os seguintes gráficos ilustram algumas das análises feitas a essas variáveis.

Na Figura 4.3 é mostrado o número de transações existentes em cada mês presente no conjunto de dados. Podemos observar, tal como é mostrado na Figura 4.2 através de o valor máximo da variável Mês, que não existem transações realizadas durante o mês de Dezembro. Novembro representa o mês com o menor número de transações e é no início do ano que existe um maior número.

A Figura 4.4 representa a proporção de casos raros que existem em relação ao número total de observações de cada produto. Para simplificar a análise, foram criados nove grupos de produtos, caracterizados pelo código de família pertencente aos intervalos apresentados no gráfico. Esta análise é importante devido à estratégia que vamos adotar para a metodologia experimental, pois através dela obtemos uma ideia de como a proporção de casos raros se distribui pelos diferentes códigos de família de produtos.

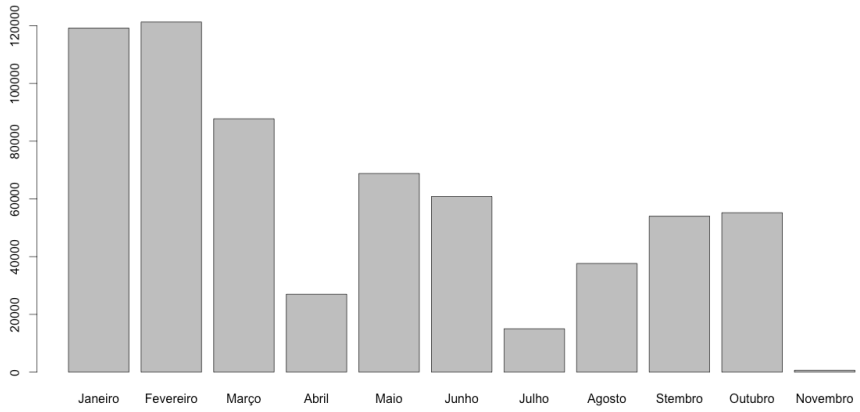


Figura 4.3: Número de transações em cada mês

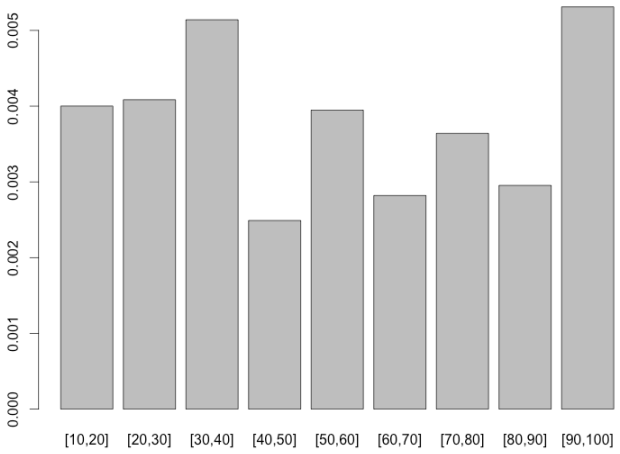


Figura 4.4: Número de casos raros em relação à família de produtos

4.3 Metodologia Experimental

Para a nossa experiência foi utilizado o software RStudio [13], que tem uma interface gráfica intuitiva, para desenvolvimentos de projetos de análise de dados em R [11]. A biblioteca utilizada foi o *RPART* [18], que é uma biblioteca para a construção de árvores de decisão. A sua escolha assentou no facto de possibilitar a fácil adaptação do critério de divisão. Essa manipulação é conseguida através de um *script* disponibilizado pelo *RPART* [17], que contém três funções necessárias para a definição de um novo critério de divisão: função de inicialização, função de avaliação e por último a função de split (Anexo 1). A função de inicialização, é usada para inicializar todas as variáveis necessárias para o critério de divisão. A função de avaliação constitui a função que avalia cada nó quando é feita uma divisão, comparando essa avaliação com o nó pai, vendo se a divisão representa uma "boa" divisão. A função de *split* é onde o trabalho computacional está

presente. É nesta função que a estratégia que mencionamos no capítulo 3.3 está a ser realizada.

Na criação de uma árvore de decisão através da biblioteca *RPART* é possível fornecer parâmetros de controlo, que constituem diferentes valores que caracterizam a construção da árvore. No nosso caso, o parâmetro cp, que controla a profundidade da árvore é igual a zero. Através desse valor, a paragem da construção da árvore é conseguida quando num nó não existirem exemplos da classe positiva, tal como foi referido na secção 3.2.2.

O pré-processamento dos dados assume um papel importante quando estamos perante técnicas de *data mining*. Para o processamento do nosso conjunto de dados, foi criado um objeto contendo todas as transações, para ser mais fácil a sua manipulação. Na primeira fase do processamento dos dados, os valores em falta foram alterados pelo valor 0, pois no domínio do problema é o significado de um valor em falta. Em seguida, apenas as transações referentes a importação foram consideradas, pois contêm um maior número de transações e as características das transações de importação e exportação são muito diferentes. Por fim, os dados foram divididos conforme o código de família do produto, que representa os dois primeiros dígitos do código do produto, para serem utilizados em separado. Devido a essa separação em vez de apenas uma árvore de decisão foram obtidas 89 árvores. Dessa maneira conseguimos ter resultados mais precisos, porque a análise reside, em cada árvore, apenas num tipo de produto. Os dados de treino são dados referentes a um ano completo, enquanto que os dados de teste são referentes a dois meses de um ano diferente.

4.4 Resultados

Através do parâmetro β presente na fórmula da *F-measure*, conseguimos controlar a importância do *recall* sobre a *precision*. Foi experimentado o valor de β igual a 1, que dá a mesma importância às duas variáveis, β igual a 0,5 que dá o dobro da importância à *precision* do que ao *recall* e β igual a 2 que dá o dobro da importância ao *recall* do que à *precision*. Com isto foi concluído que o valor de β igual a 0.5 é o que conduz a melhor resultados. Esse resultado era esperado, pois é através da *precision* que conseguimos isolar as observações da classe positiva.

No capítulo anterior foi referido que vamos obter 89 árvores de decisão, devido a existirem 89 códigos de família de produtos. Apesar de existirem 89 códigos de famílias de produtos, em apenas 62 deles existem casos raros no conjunto de teste. Assim, considerando as 62 árvores construídas para esses produtos, a Tabela 4.1 apresenta a média pesada do desempenho de cada uma das variantes de árvores de decisão.

Tabela 4.1: Desempenho médio das diferentes árvores

Árvore	**Erro médio**	***Recall* médio**	***Precision* médio**	**Esforço médio**
CART	**0.0054**	0.3257	0.4155	**0.0041**
One-sided Extremes	0.0055	**0.4342**	0.4252	0.0048
One-sided F-measure Maximization Classification	0.0055	0.3630	**0.4309**	0.0044

Através da Tabela 4.1 podemos concluir que ambas as árvores *One-sided Extremes* como a *One-sided F-measure Maximization Classification*, obtiveram melhores resultados em relação ao *recall* conseguido, do que a árvore CART. A nossa proposta não consegue um *recall* superior à *One-sided Extremes*, mas conseguimos ter uma *precision* superior, e mais importante, um esforço menor. Assim conseguimos ter um melhor equilíbrio entre o *recall* e o esforço, que é o nosso objetivo.

Tabela 4.2: Desvio padrão das diferentes medidas nas árvores

Árvore	$\sigma(Erro)$	$\sigma(Recall)$	$\sigma(precision)$	$\sigma(Esforco)$
CART	**0.0088**	**0.2931**	0.4044	0.0108
One-sided Extremes	0.0099	0.3315	**0.3736**	0.0111
One-sided F-measure Maximization Classification	0.0094	0.3074	0.4004	**0.0099**

Na Tabela 4.2 podemos concluir que a nossa proposta, em termos de esforço, tem uma variação menor que as outras duas variantes. Em relação ao *recall*, apesar de a nossa proposta ter um *recall* inferior à árvore *one-sided Extremes*, tal como vimos na Tabela 4.1, apresenta uma variação inferior. Estas duas conclusões representam uma mais-valia na nossa proposta por ser uma árvore com resultados mais robustos.

Comparando a nossa árvore com a árvore de decisão tradicional, obtivemos 56 árvores que têm um valor maior ou igual de *recall*. Dessas, 13 obtiveram um *recall* superior. Em relação à *precision*, obtivemos 49 árvores que têm um valor maior ou superior. Dessas, 16 obtiveram uma *precision* superior.

Na Figura 4.5 podemos analisar as curvas ROC para os três algoritmos de árvores de decisão que estamos a analisar. Através da Figura podemos ver que os resultados contradizem os encontrados através na Tabela 4.1, pois a linha ROC da árvore *one-sided Extremes*, bem como da árvore *One-sided F-measure Maximization Classification*, situam-se no triângulo inferior da reta de declive 1 que passa na origem 2.3.4.1. Para criar as curvas ROC, ao invés de se utilizar a classe de cada exemplo, que é o que acontece quando se constrói uma matriz de confusão para se realizar a análise presente na Tabela 4.1, é utilizada a probabilidade de cada exemplo ser da classe positiva. Para tentar perceber o porquê dessa contradição, foram consideradas duas hipóteses. A primeira tem a ver com a possibilidade de o RPART não criar corretamente essas probabilidades, quando é criada um novo critério de divisão, através do script disponibilizado pela biblioteca. A segunda tem a ver com o facto de, apesar a árvore *CART* apresentar piores resultados na classificação de casos raros, lidar melhor com as probabilidades em relação às outras duas variantes.

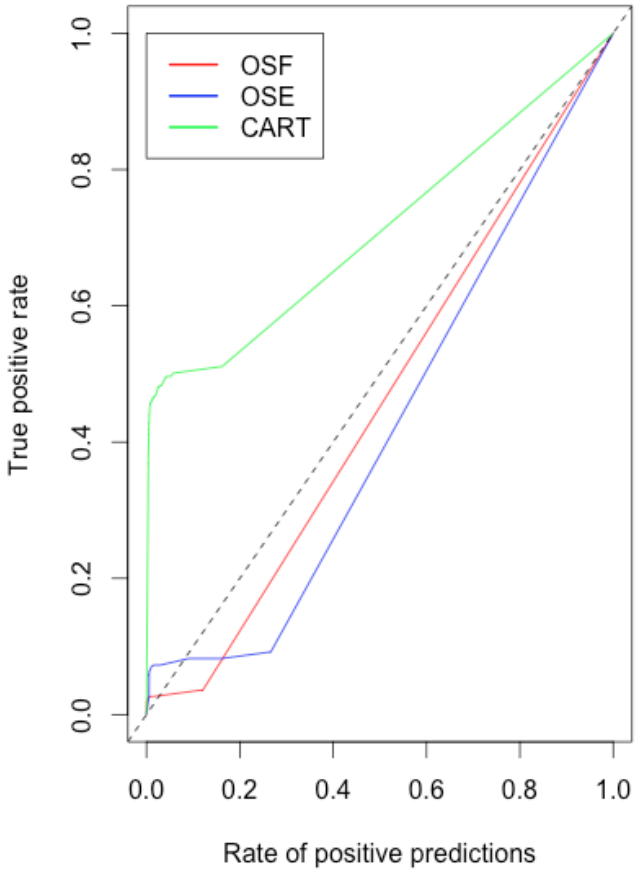


Figura 4.5: Gráfico ROC dos três algoritmos de árvores de decisão

Capítulo 5

Conclusões e Trabalho Futuro

O objetivo deste trabalho era desenvolver um novo critério de divisão na construção de uma árvore de decisão capaz de detetar casos raros, com um desempenho superior em relação a uma árvore de decisão tradicionais. Esse critério de divisão foi adaptado a partir de uma proposta existente para regressão [19]. Para validar o método proposto, foi realizado um estudo empírico comparando-o com algoritmos num caso de estudo de deteção de erros em transações de comércio externo (INTRASTAT). Esse objetivo foi conseguido, apesar de a nossa proposta ter tido resultados piores em certos aspetos em relação a algoritmos já existentes. Um dos objetivos que não conseguimos alcançar foi a obtenção de um *recall* superior em relação a algoritmos existentes. Apesar disso, conseguimos obter uma *precision* superior, o que traduz um esforço inferior necessário na deteção de casos raros, pois o método é mais preciso.

Uma das dificuldades encontradas foi o fato de a biblioteca RPART não ser completamente adaptável para a definição de uma novo critério de divisão por parte do utilizador, apesar de ter sido a melhor solução encontrada. Isso deveu-se principalmente por não criar certas variáveis, que na construção de uma árvore de decisão tradicional criaria e que são importantes para a nossa abordagem.

5.1 Trabalho Futuro

Apesar de termos alcançado o nosso objetivo, há certos aspetos que poderão ser melhorados na nossa proposta. Um aspeto que poderia ser melhorado é a possibilidade da deteção de casos raros num conjunto de dados que contenha mais que duas classes. Dessa forma, em vez de existirem apenas duas classes, uma que classifica um exemplo como normal e outra como caso raro, existiria um número de classes normais superior a 1 e uma classe para casos raros. Com isso, conseguiria-se, não só a deteção de casos raros, mas também a criação de padrões, considerando os exemplos das outras classes.

Um outro aspeto que poderia ser melhorado, era a aplicação da nossa proposta noutros conjuntos de dados, para podermos comparar resultados entre eles. Isso seria vantajoso pois é esperado que haja conjuntos de dados em que a nossa proposta tenha um desempenho superior, comparando com outros, e vice-versa. Um estudo empírico mais extenso permitiria perceber melhor as condições em que o método funciona e aquelas em que não funciona tão bem.

Anexos

Anexo 1

```
init <- function(y, offset, parms, wt) {
  if (is.matrix(y) && ncol(y) > 1)
    stop("Matrix response not allowed")
  if (!missing(parms) && length(parms) > 0)
    warning("parameter argument ignored")
  if (length(offset)) y <- y - offset
  sfun <- function(yval, dev, wt, ylevel, digits ) {
    paste(" mode=", round(yval[,1]), ", MSE=" , format(signif(dev/wt, digits)), sep
         = '')
  }
  environment(sfun) <- .GlobalEnv
  list(y = c(y), parms = parms, numresp = 6, numy = 1, summary = sfun)
}
```

Listing 5.1: Função de inicialização

```
eval <- function(y, wt, parms) {
  uy <- unique(y)
  mode <- uy[which.max(tabulate(match(y,uy)))]
  nFreq <- length(which(y == mode))
  FreqOutlier <- length(which(y == 1))
  rss <- (1 - (nFreq/length(y)))*length(y)
  list(label = c(mode,nFreq,length(y)-nFreq,nFreq/length(y),1-nFreq/length(y),
      length(y)/parms), deviance = rss)
}
```

Listing 5.2: Função de avaliação

```
split <- function(y, wt, x, parms, continuous)
{
    max <- 0
    l_one <- 0
```

```
    l_zero <- 0
    r_one <- length(which(y == 2))
    r_zero <- length(which(y == 1))
    n_left <- 0
    n_right <- length(y)

    for(value in y[-n]){
      n_left <- n_left + 1
      n_right <- n_right - 1

      if(value == 1){
        l_zero <- l_zero + 1
        r_zero <- r_zero -1
      }
      else if(value == 2){
        l_one <- l_one + 1
        r_one <- r_one -1
      }
      l_recall <- (l_one)/( (l_one+r_one))
      l_precision <- (l_one)/( l_one+l_zero )

      if(l_recall == 0 || l_precision == 0)
        l_f <- 0
      else
        l_f <- (1.25*l_precision*l_recall)/(0.25*l_precision+l_recall)

      r_recall <- (r_one)/( ((l_one+r_one)))
      r_precision <- (r_one)/( (r_one+r_zero) )

      if(r_recall == 0 || r_precision == 0)
        r_f <- 0
      else
        r_f <- (1.25*r_precision*r_recall)/(0.25*r_precision+r_recall)

      tempMax <- max(c(l_f,r_f))
      max <- c(max,tempMax)
    }

    signal <- si(max[-1])
    goodness <-max[-1]
    list(goodness = goodness, direction = signal)
}
```

Listing 5.3: Função de spliting

Referências

[1] Márcio Porto Basgalupp. *LEGAL-Tree: Um algoritmo genético multi-objetivo lexicográfico para indução de árvores de decisão*. tese de doutoramento, Instituto de Ciencias Matemáticas e de Computação - São Paulo, 2010.

[2] L Breiman, J H Friedman, R A Olshen e C J Stone. *Classification and Regression Trees*, volume 19. 1984.

[3] Andreas Buja e Yung-seop Lee. Data Mining Criteria for Tree-Based Regression and Classification. 1998.

[4] H. Schutze C. Manning, P. Raghavan. *An Introduction to Information Retrieval*. Number c. Cambridge University Press, Cambridge, England, 2009.

[5] Chandola. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41, 2009.

[6] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 2006.

[7] João Gama. Árvores de decisão. 2004.

[8] Wei Liu e Sanjay Chawla. A Robust Decision Tree Algorithm for Imbalanced Data Sets. 2006.

[9] J. R. Quinlan. Induction of decision trees, 1986.

[10] J R Quinlan. *C4.5: Programs for Machine Learning*, volume 1. 1993.

[11] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/. ISBN 3-900051-07-0.

[12] Lior Rokach e Oded Maimon. Top-Down Induction of Decision Trees Classifiers—A Survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 35, 2005.

[13] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2012. URL http://www.rstudio.com/.

[14] Carlos Soares, Pavel Brazdil e Liacc Costa, Joaquim. Error Detection in Foreign Trade Data using Statistical and Machine Learning Algorithms. 1999.

[15] Pang-Ning Tan, Michael Steinbach e Vipin Kumar. Introduction to Data Mining. *Journal of School Psychology*, 19, 2005.

[16] Pang-Ning Tan, Michael Steinbach e Vipin Kumar. Classification : Basic Concepts , Decision Trees , and. In *Introduction to Data Mining*, volume 67. 2006.

[17] Terry Therneau. User written splitting functions for RPART Anova function. páginas 1–10, 2014.

[18] Terry Therneau, Beth Atkinson e Brian Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2014. URL http://CRAN.R-project.org/package=rpart. R package version 4.1-8.

[19] Luis Torgo e Rita Ribeiro. Predicting Outliers. *Proceedings of Principles of Data Mining and Knowledge Discovery*, 2003.