Capítulo 3

Introdução à Classificação Multirrótulo

Eduardo Corrêa Gonçalves

Abstract

Multi-label classification (MLC) is the task of assigning multiple class labels to an object based on the features that describe the object. There are many important and modern applications of MLC, such as text categorization (associating documents to various subjects) and semantic scene classification (categorizing images into concepts). This chapter is intended to those who wish to work with MLC in information systems. We present the basic approaches for the construction and evaluation of multi-label classifiers as well as examples of software libraries that support the development of MLC applications in Java and R.

Resumo

Classificação multirrótulo (CMR) pode ser definida como a tarefa de associar múltiplos rótulos de classe para um objeto com base nas características que o descrevem. Existem muitas aplicações modernas e importantes para a tarefa, tais como a categorização de textos (associar documentos texto a tópicos) e a classificação semântica de cenas (associar imagens a conceitos). Este capítulo é direcionado a pessoas que querem trabalhar com classificação multirrótulo em sistemas de informação. Ele apresenta as abordagens básicas para a construção e avaliação de classificadores multirrótulo e também exemplos de bibliotecas que oferecem suporte para o desenvolvimento de aplicações de CMR nas linguagens Java e R.

3.1. Introdução

Aprendizado de máquina (*machine learning*) é a linha de pesquisa que investiga como computadores podem aprender a executar tarefas baseando-se na utilização de dados [Han et al., 2011]. Dentre os diferentes tipos de tarefas que podem ser realizadas, a classificação é provavelmente a mais conhecida e utilizada. Seu objetivo é bastante simples: treinar um programa de computador para que o mesmo seja capaz de atribuir

automaticamente classes (ou "rótulos de classe") para objetos cujas classes sejam desconhecidas.

Para que o conceito fique claro, será apresentado um exemplo. Considere um programa que receba como entrada a fotografia do rosto de uma pessoa e que seja capaz de determinar automaticamente se o rosto pertence a um adulto ou a uma criança. Veja que o objetivo do programa é associar uma classe ("Adulto" ou "Criança") a um objeto (a fotografia de um rosto). Desta forma, trata-se de um programa que realiza a tarefa de classificação. Nos dias atuais, técnicas de classificação vêm sendo exploradas tanto pela comunidade acadêmica como pelas empresas para resolver diversos problemas importantes. Alguns deles são tradicionais e bastante conhecidos, como a detecção de *spam* (identificar um e-mail como "Spam" ou "Normal"), a detecção de fraudes (identificar se uma transação de cartão de crédito é "Fraudulenta" ou "Genuína") e a análise de crédito (classificar clientes que solicitaram um empréstimo bancário de acordo com os riscos de crédito "Baixo", "Médio" ou "Alto"). Já outras aplicações são mais recentes e menos conhecidas, entre elas a genômica funcional (determinar as funções biológicas de genes e proteínas) e a categorização automática de músicas (associar canções a estilos musicais).

Na maioria dos problemas de classificação, cada objeto deve ser associado a um e somente um rótulo pertencente a um conjunto predeterminado de rótulos de classe. Estes problemas são conhecidos como problemas de classificação tradicional ou monorrótulo (*single-label classification*) [Flach, 2012; Han et al., 2011; Marsland, 2015; Witten et al., 2016]. Por exemplo, no problema de classificação de *spam*, um e-mail recebido deve ser classificado como "Spam" ou "Normal", mas nunca com ambos os rótulos. Analogamente, no problema da análise de crédito, um candidato a empréstimo pode ser classificado em apenas um dos rótulos de classe possíveis ("Baixo", "Médio" ou "Alto"), jamais podendo ser classificado em dois ou três deles ao mesmo tempo.

Entretanto, nem todos os problemas reais de classificação são do tipo monorrótulo. Um dos exemplos é a categorização automática de músicas, onde o objetivo é associar canções a gêneros musicais. Neste problema, tem-se, por exemplo, que muitas músicas compostas pelo grupo brasileiro Novos Baianos podem ser classificadas como pertencentes aos gêneros "Rock", "MPB" e "Samba" ao mesmo tempo. De maneira similar, um grande número de composições do músico Tom Jobim consiste em uma mistura de dois gêneros: "Jazz" e "Bossa Nova". Por esta razão, o problema da categorização de músicas representa um problema de classificação multirrótulo (CMR – *multi-label classification*) [Gibaja and Ventura, 2015; Gonçalves et al., 2018; Tsoumakas et al., 2010; Zhang and Zhou, 2014], em que os objetos podem ser associados a múltiplos rótulos de classe. Ao longo dos últimos anos, diversas outras aplicações modernas e importantes de CMR surgiram. A listagem a seguir inclui apenas algumas delas, como forma de motivação para o leitor.

- Classificação de artigos de jornais ou portais de notícias em um número fixo categorias, como "Esporte", "Cultura", "Política", etc. [Tsoumakas et al., 2014];
- Genômica funcional: determinar as múltiplas funções biológicas de genes e proteínas [Diplaris et al., 2005; Elisseeff, and Weston, 2001; Huang et al., 2015];

- Classificação semântica de cenas: consiste em identificar os diferentes componentes de uma cena da natureza, como “Montanhas”, “Mar”, “Árvores”, etc. [Boutell et al., 2004];
- Classificação de músicas [Trohidis et al., 2011] ou textos curtos [Almeida et al., 2018] em um conjunto de emoções (ex.: “Triste”, “Relaxante”, “Alegre”, etc.);
- Predição do efeito colateral de drogas, ou seja, prever o conjunto de reações adversas que novas drogas podem causar [Zhang et al., 2015];
- Classificação de textos de laudos médicos em classes de diagnóstico [Cherman and Monard, 2009; Pestian et al., 2007]
- Categorização dos tipos de falhas de execução de programas a partir da análise de *crash reports* [Xia et al., 2014];
- Atribuição automática dos gêneros de filmes (ex.: “Drama”, “Romance”, “Ação”, etc.) em função do texto contendo o seu resumo [Ururahy et al., 2018];
- Sugestão de *tags*: processo de atribuir pequenas informações textuais – as chamadas *tags* ou palavras-chave – a objetos multimídia como vídeos e imagens [Xiao et al., 2016].

Este capítulo, embora de maneira resumida, tem por objetivo apresentar um panorama geral da área de classificação multirrótulo, focando nos aspectos essenciais para aqueles que desejam incorporar esta tecnologia em seus próprios aplicativos. O texto está divido em duas partes. A primeira, denominada “Classificação – uma Visão Geral”, apresenta a teoria necessária para habilitar o leitor a entender como funciona um sistema de classificação. Nesta primeira parte, o foco é a classificação monorrótulo. A segunda e principal parte do curso, “Classificação Multirrótulo”, introduz os principais conceitos sobre CMR, cobrindo os seguintes tópicos: abordagens básicas para a construção de classificadores multirrótulo, propriedades das bases de dados multirrótulo e medidas de avaliação de desempenho para CMR. O capítulo também aborda a classificação na prática, através da apresentação de exemplos de *scripts* em Java e R que descrevem os passos básicos para a criação e utilização classificadores nestas linguagens.

3.2. Classificação – uma Visão Geral

3.2.1. Como Construir um Classificador?

Para que seja possível construir um classificador, “apenas” duas coisas são necessárias:

1. Uma boa base de dados de treinamento (ou base de dados rotulada). Esta base deve conter objetos pré-classificados, de acordo com um número finito de classes pré-definidas.
2. Um bom algoritmo de classificação. Alguns exemplos: naïve Bayes, k-NN, SVM, entre outros.

Nesta seção, o problema da classificação será introduzido de forma prática, a partir de exemplos que envolvem a classificação monorrótulo (é importante observar que o entendimento dos conceitos básicos de classificação monorrótulo representa um pré-requisito para quem deseja trabalhar com classificação multirrótulo). O texto está dividido da seguinte forma. Inicialmente, as bases de dados de treinamento e os algoritmos de classificação são respectivamente cobertos nas Subseções 3.2.2 e 3.2.3. Em seguida, a Subseção 3.2.4 aborda as técnicas básicas para avaliação da qualidade de classificadores. Por fim, a Subseção 3.2.5 dedica-se à classificação em sistemas de informação, apresentando exemplos de programas que utilizam bibliotecas de aprendizado de máquina nas linguagens Java e R.

3.2.2. Bases de Dados de Treinamento

A Figura 3.1 apresenta um exemplo de base de dados de treinamento. Considere que o objetivo seja utilizá-la para possibilitar a criação de um classificador monorrótulo capaz de determinar se a foto do rosto de uma pessoa é de um “Adulto” ou de uma “Criança”.

X	Y
x_1: 	y_1: Adulto
x_2: 	y_2: Criança
x_3: 	y_3: Criança
x_4: 	y_4: Adulto
...	...
x_N: 	y_N: Adulto

Figura 3.1. Exemplo de base de dados de treinamento para classificação monorrótulo. Ela pode ser utilizada para treinar um classificador capaz de determinar se a foto de um rosto é de um “Adulto” ou de uma “Criança”

Em uma base de treinamento existem duas categorias de atributos, *X* e *Y*:

- *X*: conjunto de atributos preditivos. São os atributos que descrevem as características (*features*) dos objetos, podendo ser tanto numéricos como discretos. Neste exemplo, a descrição de cada objeto da base de dados é dada pela foto de seu rosto. É importante deixar claro que em uma base de treinamento real para classificação de imagens, cada foto (dado bruto) precisa ser

transformada em um vetor numérico (dado estruturado). Este vetor armazena um conjunto de medições que caracteriza as propriedades da imagem [Chen et al., 2018].

- *Y*: atributo especial, denominado atributo classe. Trata-se do atributo alvo da classificação. No exemplo apresentado, cada $y_i \in$ {"Adulto", "Criança"} (conjunto de rótulos de classe pré-definido). O atributo classe é sempre do tipo discreto.

Observe que cada objeto[1] da base rotulada é representado por um par (*x*,*y*), onde *x* é um vetor contendo valores para os atributos do conjunto *X* e *y* é um dos valores possíveis (pré-determinados) de *Y*. A criação de um classificador é realizada através do processamento da base por um algoritmo de classificação, que se encarregará de "aprender" a mapear as características dos objetos (*X*) em rótulos de classes (*Y*).

A seguir, apresenta-se a definição formal para o problema de classificação monorrótulo:

Definição 1 *(Classificação Monorrótulo). Seja* $X = \{X_1, \ldots, X_d\}$ *um conjunto de* d *atributos preditivos e* $L = \{l_1, \ldots, l_q\}$ *um conjunto de* q *rótulos de classe, onde* $q \geq 2$. *Considere uma base de dados de treinamento D composta por N objetos do tipo* $\{(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)\}$. *Nesta base de dados, cada* x_i *corresponde a um vetor* $\{x_1, \ldots, x_d\}$ *que armazena valores para os* d *atributos preditivos do conjunto* X *e cada* $y_i \in$ L *corresponde a um único rótulo de classe. O objetivo da tarefa de classificação monorrótulo é, a partir de* D*, realizar o aprendizado de uma função* f *(classificador ou modelo de classificação) que, dado um objeto não rotulado* t = (x,?)*, seja capaz de prever de forma efetiva o seu rótulo de classe* y.

Quando $q = 2$, o problema é chamado de problema de classificação binário. Este é o caso da detecção de *spam*, onde $L =$ {"Spam", "Normal"} e também da detecção de fraudes, em que $L =$ {"Fraudulenta", "Genuína"}. Por outro lado, se $q > 2$, o problema é chamado de problema de classificação multiclasse. Este é o caso da classificação do risco de crédito, onde $L =$ {"Baixo", "Médio", "Alto"}.

3.2.3. Algoritmos de Classificação

A construção de classificadores precisos e eficientes é considerada um dos grandes desafios na área de aprendizado de máquina. Por este motivo, foram desenvolvidas diversas técnicas para a execução desta tarefa, como naïve Bayes, k-NN, SVM, redes neurais e árvores de decisão, entre outros [Han et al., 2011; Witten et al., 2016].

Todas estas técnicas trabalham em duas etapas: treinamento (ou aprendizado) e classificação. A etapa de treinamento é aquela em que o modelo de classificação é

[1] Na literatura sobre classificação, as palavras "objeto", "instância", "tupla", "observação", "exemplo", "*data point*" e "registro" podem ser utilizadas como sinônimos.

construído a partir da base de dados rotulada. Já a classificação corresponde à etapa em que o modelo criado é utilizado para classificar novos objetos.

Esta seção apresenta um breve resumo sobre alguns dos principais algoritmos de classificação monorrótulo. Ênfase especial é dada ao algoritmo naïve Bayes (NB) que, a despeito de sua simplicidade, é apontado na literatura como um dos mais eficazes para problemas de classificação de texto [Witten et al., 2016] – a área de aplicação dominante e motivadora do surgimento da CMR [Tsoumakas et al., 2010].

Naïve Bayes (NB)

NB é um dos mais simples, populares e eficientes algoritmos de classificação. Trata-se de um classificador estatístico fundamentado em diversos conceitos da Teoria da Probabilidade, como probabilidade condicional, independência condicional, regra da multiplicação, distribuição conjunta de probabilidades e, especialmente, em uma importante fórmula conhecida como Fórmula de Bayes.

Esta seção explica, de forma resumida, o funcionamento do algoritmo a partir da descrição dos principais passos por ele empregados nas etapas de treinamento e classificação. Conforme introduzido na subseção anterior, a tarefa de classificação possui como objetivo associar objetos de classe desconhecida a um conjunto pré-definido de classes. Os algoritmos de classificação extraem os modelos classificadores a partir de bases de dados rotuladas, onde cada objeto é representado por um par (x,y). A base de dados de filmes apresentada na Tabela 3.1 é um exemplo de base rotulada. Considere que ela foi montada com o objetivo de viabilizar a construção de um classificador monorrótulo capaz de classificar o gênero de um filme como Romance (‘Sim’ ou ‘Não’) em função da ocorrência das palavras “love”, “people” e “relationship” em seu resumo.

Tabela 3.1. Base de dados de treinamento para criar classificador que determina se um filme é do gênero “Romance” em função da ocorrência das palavras “love”, “people” e “relationship” em seu resumo.

love	*people*	*relationship*	**Romance (classe)**
1	1	0	Sim
1	0	0	Não
1	0	1	Sim
0	0	1	Sim
1	1	1	Não
1	1	0	Sim
0	0	0	Não
0	1	0	Não
1	1	1	Sim
0	1	1	Não
1	0	1	Sim
1	1	0	Não
0	0	0	Não
1	1	1	Sim
1	0	1	Sim

A base de dados possui as seguintes características:

- Os atributos “love”, “people” e “relationship” descrevem as propriedades dos resumos (ou seja, eles formam o conjunto *X*). Estes atributos indicam se as respectivas palavras ocorrem ou não no resumo do filme.
- O atributo “Romance” é o atributo classe (*Y*). Ele registra o valor ‘Sim’ quando o filme é do gênero Romance e ‘Não’ caso contrário.

A base de dados possui informações sobre 15 filmes. Observe que o primeiro filme possui as palavras “love” e “people” em seu resumo, mas não possui “relationship”. Veja ainda que este filme está rotulado como Romance (Romance = ‘Sim’). Já o segundo filme possui a palavra “love”, não possui “people” e “relationship” e não é um Romance (Romance = ‘Não’). E assim por diante. A seguir, são descritas as etapas de treinamento (construção do modelo classificador) e classificação do algoritmo NB utilizando a base de filmes como exemplo.

A etapa de treinamento consiste basicamente em computar e armazenar em memória uma tabela de probabilidades condicionais que resume o conjunto de dados de treinamento. O modelo de classificação gerado pelo NB corresponde exatamente a esta tabela em memória. A Tabela 3.2 apresenta um exemplo que corresponde ao modelo que seria gerado para a base de dados de resumos de filmes da Tabela 3.1.

Tabela 3.2. Modelo de classificação gerado pelo naïve Bayes a partir da análise de base de dados de resumos de filmes.

Romance (classe)	***love***		***people***		***relationship***	
	0	**1**	**0**	**1**	**0**	**1**
Não 7/15 (46,67%)	4/7 (57,14%)	3/7 (42,86%)	3/7 (42,86%)	4/7 (57,14%)	5/7 (71,43%)	2/7 (28,57%)
Sim 8/15 (53,33%)	1/8 (12,50%)	7/8 (87,50%)	4/8 (50,00%)	4/8 (50,00%)	2/8 (25,00%)	6/8 (75,00%)

No modelo da Tabela 3.2, a primeira coluna apresenta as probabilidades “a priori” de um filme pertencer ao gênero Romance ou não (para Romance=‘Não’ o valor é 46,67% e para Romance=‘Sim’ o valor é 53,33%). Já as colunas 2 a 7 apresentam as probabilidades condicionais dos valores dos atributos preditivos “love”, “people” e “relationship” dados os dois rótulos possíveis da classe Romance. Considere, por exemplo, o valor 4/7 localizado na primeira linha e segunda coluna da tabela. Ele indica que existem 7 filmes que não são do gênero Romance (Romance=‘Não’) na base de treinamento e que 4 deles (57,14%) não possuem a palavra “love” em seu resumo (love=0). De maneira análoga, na célula localizada logo abaixo, o valor 1/8 indica que apenas 1 dos 8 filmes (12,50%) do gênero Romance (Romance=‘Sim’) não possuem a palavra “love” em seu resumo (love=0).

Uma vez criado o modelo de classificação, o NB está apto para realizar a classificação de novos objetos. Para tal, o algoritmo faz uso de uma adaptação Fórmula de Bayes, apresentada na Equação 1:

$$P(Y \mid X) = \frac{P(X \mid Y) \times P(Y)}{P(X)} \quad (1)$$

Suponha que o objetivo seja classificar um novo filme *t*, com as seguintes características: *t* = (*love=1, people=0, relationship=1, Romance = ?*). A questão é estimar se *t* é ou não um filme do gênero Romance utilizando a Equação 1. Para classificar *t*, o algoritmo NB faz o seguinte: ele observa as caraterísticas de *t* e, de acordo com estas, busca os valores de probabilidade correspondentes no modelo de classificação. Estes valores são então "plugados" na fórmula de Bayes para que seja possível realizar o cálculo da estimativa para os dois rótulos de classe possíveis (Romance='Não' e Romance='Sim'). A seguir, é mostrado como o cálculo dessas estimativas é realizado, considerando o novo objeto *t* e o modelo de classificação da Tabela 3.2:

- Estimativa 'Não':

P(Romance='Não' | *t*) = P(love=1| Romance='Não') ×
P(people=0| Romance='Não') ×
P(relationship=1| Romance='Não') ×
P(Romance='Não')

P (Romance='Não' | *t*) = 0,4286 x 0,4286 x 0,2857 x 0,4667 = **0,0245**

- Estimativa 'Sim':

P(Romance='Sim' | *t*) = P(love=1| Romance='Sim') ×
P(people=0| Romance='Sim') ×
P(relationship=1| Romance='Sim') ×
P(Romance='Sim')

P (Romance='Sim' | *t*) = 0,8750 x 0,5000 x 0,7500 x 0,5333 = **0,1750**

O resultado indica que há uma maior chance de o filme ser um Romance (pertencer à classe 'Sim'). Para que seja possível analisar o resultado de uma forma mais interessante, os valores calculados podem ser convertidos para probabilidades através da normalização da soma para 1. Isto é feito da seguinte forma:

- P('Não') = 0,0245 ÷ (0,0245 + 0,1750) = 12,28%
- P('Sim') = 0,1750 ÷ (0,0245 + 0,1750) = 87,72%

Desta forma, o rótulo de classe 'Sim' é atribuído ao novo objeto *t*. Observe que para realizar o cálculo não foi preciso utilizar o denominador da Fórmula de Bayes –

P(X) – uma vez que o seu valor seria o mesmo tanto para o cálculo da estimativa da classe 'Não' como da classe 'Sim'.

O algoritmo NB possui a palavra *naïve* (ingênuo) em seu nome porque se baseia em duas suposições:

1. Todos os atributos da base de dados são igualmente importantes.
2. Todos os atributos preditivos são condicionalmente independentes em relação ao atributo classe.

São exatamente estas suposições que fazem com que seja possível realizar a classificação com o uso de uma adaptação simples da Fórmula de Bayes.

Outros Algoritmos de Classificação Monorrótulo

Conforme mencionado anteriormente, além do NB, existem muitos outros algoritmos de classificação. O texto a seguir apresenta um breve resumo sobre quatro deles: *k-Nearest Neighbors* (k-NN), C4.5, *Sequential Minimal Optimization* (SMO) e *Multi-Layer Perceptron* (MLP). Explicações detalhadas sobre estes algoritmos e também sobre o NB podem ser obtidas em [Han et al., 2011; Witten et al., 2016].

- k-NN: este algoritmo executa a tarefa de classificação de um novo objeto *t* em dois passos: primeiro, o algoritmo encontra os *k* objetos da base de treinamento que sejam mais similares a *t*. Tipicamente, a similaridade é computada em termos de alguma medida de distância, como a distância Euclidiana ou a distância de Jaccard. Em seguida, *t* é classificado com a classe mais comum entre os *k* objetos.
- C4.5: constrói um modelo de classificação na forma de uma árvore de decisão. Em uma estrutura deste tipo, cada nó interno corresponde a um atributo preditivo (ex.: "love"), cada ramo representa um teste sobre um nó pai (ex.: o valor de love é 0?) e cada folha consiste em um rótulo de classe. Um novo objeto é classificado através da aplicação de sucessivos testes que encontrarão um caminho na árvore, desde o nó raiz até um nó folha. Para construir a árvore, o algoritmo C4.5 adota uma abordagem recursiva e gulosa que emprega a medida da entropia como critério de seleção de atributos. A cada passo, esta medida é aplicada para definir qual atributo melhor particiona os objetos da base de treinamento nas classes distintas.
- SMO: trata-se de um algoritmo para treinar classificadores SVM (*support vector machines*). Na técnica SVM, os dados de treino são separados em duas classes através da busca por uma fronteira de decisão que maximiza a distância entre os objetos de treino destas duas classes.
- MLP: algoritmo utilizado para treinar redes neurais. Uma rede neural MLP é uma estrutura formada por um conjunto de nós (denominados neurônios artificiais) e arestas estruturados em três camadas: de entrada, oculta (*hidden*) e de saída. Cada aresta possui um peso associado, cujo valor inicial é tipicamente determinado de forma aleatória, com base na distribuição normal. A técnica MLP emprega um processo de treinamento denominado *backpropagation* para

determinar o melhor conjunto de pesos para a rede. Primeiro, para cada objeto de treino, os valores de classe são determinados com base nos pesos correntes. Os erros de classificação são então computados e os pesos associados a cada aresta são atualizados. O processo é repetido até que um critério especificado para minimização do erro seja alcançado.

3.2.3. Como Avaliar a Qualidade de um Classificador?

Para criar um classificador, basta ter em mãos um bom algoritmo de classificação e uma boa base de dados de treinamento. Entretanto, para colocar o classificador em produção é preciso estimar a sua eficácia, isto é, estimar se ele terá um bom desempenho para classificar novos objetos. O princípio básico empregado na avaliação de classificadores consiste em utilizar um conjunto separado de dados de teste formado por objetos que não estiveram envolvidos no processo de treinamento do classificador. Esse conjunto também deverá conter objetos rotulados, ou seja, deve ser formado por pares (x,y).

O método *holdout* [Han et al., 2011] é o mais simples dentre os que empregam o princípio descrito acima. Na abordagem *holdout*, a base de dados rotulada é dividida de forma aleatória em dois conjuntos independentes: conjunto de treinamento e conjunto de teste. Tipicamente, dois terços dos dados são alocados para treino e o terço restante é alocado para teste (por exemplo, considerando a base de resumos de filmes apresentada na Tabela 3.1, tem-se que 10 objetos seriam alocados para o conjunto de treino e os 5 restantes para o conjunto de teste). Preferencialmente, a divisão deve ser feita de modo a garantir que cada classe seja adequadamente representada tanto no conjunto de treinamento como no de teste (processo conhecido como estratificação). O conjunto de treinamento é então usado para a construção do classificador, cuja acurácia é estimada com o conjunto de teste. A acurácia corresponde à porcentagem de objetos de teste corretamente classificados pelo modelo. A Figura 3.2 resume o processo *holdout*.

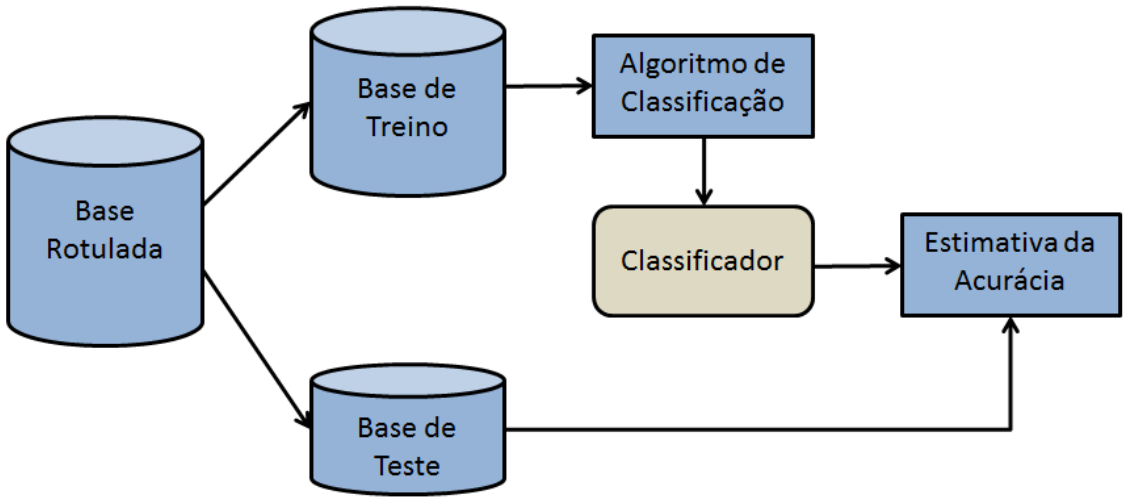

Figura 3.2. Estimando a acurácia de um classificador com o método *holdout*

O método *holdout* não é o mais utilizado na prática, pois existem técnicas capazes de obter estimativas mais confiáveis para o desempenho preditivo de um

classificador. Uma delas é a validação cruzada (*cross-validation*) [Han et al., 2011; Witten et al., 2016]. Neste método, a base rotulada é dividida em *k* partições D_1, D_2,..., D_k de tamanho aproximadamente igual (na Figura 3.3, apresenta-se um exemplo onde *k=10*, um dos valores mais comumente utilizados). Após a divisão, são realizadas *k* rodadas de treino e teste. A cada iteração *i*, a partição D_i é reservada para teste e as partições restantes são utilizadas para treinar o modelo. Considerando o exemplo onde *k*=10, na primeira rodada as partições D_2, D_3,... D_{10} seriam utilizadas para treinamento e a partição D_1 para teste. Na segunda rodada, as partições D_1,D_3... D_9 seriam utilizadas para treinamento e a partição D_2 para o teste. E assim por diante, até a décima rodada. A acurácia final estimada para o modelo será igual à acurácia média das 10 rodadas. Sendo assim, ao contrário do que ocorre com o método *holdout*, na validação cruzada a estimativa da acurácia não é feita "apostando-se todas as fichas" no resultado obtido sobre uma única base de teste, o que constitui uma importante vantagem.

É importante comentar que além da validação cruzada, existem outras técnicas para estimar a qualidade de classificadores, dentre as quais a validação cruzada repetida, *leave-one-out* e *bootstrap* [Witten et al., 2016].

Figura 3.3. Base rotulada dividida em 10 partições (*folds*). Na primeira rodada do método de validação cruzada, as partições D_2, D_3,..., D_{10} são utilizadas para treinar o modelo enquanto a partição D_1 é utilizada para testá-lo.

Matriz de Confusão

A matriz de confusão (MC) é uma das mais úteis ferramentas para a avaliação da qualidade de classificadores. Sua finalidade é armazenar todos os diferentes tipos de erros e acertos realizados pelo classificador ao processar um conjunto de objetos de teste. Uma MC é uma matriz quadrada *q* x *q*, onde *q* representa o número de rótulos de classe envolvidos no problema. Cada célula c_{ij} denota o número de objetos de teste que o classificador associou à classe *i* e que, de fato, pertencem à classe *j*. Desta forma, as células da diagonal principal sempre irão conter o número de objetos corretamente classificados pelo modelo. No exemplo da Figura 3.4, apresenta-se o formato da MC que seria produzida após o processo de teste do classificador de resumos de filmes (nesse problema, *q*=2).

		Rótulos Preditos	
	Classes	Romance='Não'	Romance='Sim'
Rótulos Reais	Romance='Não'	VN	FP
	Romance='Sim'	FN	VP

Figura 3.4. Matriz de confusão gerada após um processo de teste do classificador de resumos de filmes.

Na Figura 3.4, a célula $MC_{1,1}$ armazena o total de objetos da base de teste cujo rótulo era 'Não' e que foram corretamente classificados como 'Não', e por isso são chamados Verdadeiro Negativos (VN). Em $MC_{1,2}$ está representado o total de objetos de teste cujo rótulo era 'Não' e que foram incorretamente classificados como 'Sim', e que por isso são chamados Falso Positivos (FP). O valor em $MC_{2,1}$ indica o número de objetos de teste cujo rótulo era 'Sim' e que foram incorretamente classificados como 'Não', sendo então chamados de Falso Negativos (FN). Por fim, em $MC_{2,2}$ está armazenado o número de objetos de teste cujo rótulo era 'Sim' e que foram corretamente classificados como 'Sim', chamados de Verdadeiro Positivos (VP)

A partir da MC é possível calcular diversas medidas de desempenho [Flach, 2012; Han et al., 2011; Japkowicz and Shah, 2011]. Uma delas é a própria Acurácia, mencionada no início desta subseção, que computa a proporção de objetos de teste corretamente classificados. Sua fórmula é apresentada na Equação 2.

$$Acurácia = (VP + VN) \div (VP + FP + FN + VN) \quad (2)$$

A Acurácia é a medida de desempenho de classificadores mais empregada. Entretanto, não é a mais indicada em situações onde a base de dados é desbalanceada, isto é, quando existem muito mais objetos da classe 'Não' do que da classe 'Sim', ou vice-versa. Este é o caso típico de aplicações como a classificação de fraudes, onde normalmente vão existir muito mais objetos com a classe 'Não' (transações que não são fraudes, ou seja, transações genuínas) do que com a classe 'Sim' (transações fraudulentas). A Figura 3.5 mostra um exemplo de matriz de confusão hipotética que poderia ter sido gerada a partir de uma base de dados contendo 1000 objetos de teste em um problema de classificação de fraudes.

		Rótulos Preditos	
	Classes	Fraude='Não'	Fraude='Sim'
Rótulos Reais	Fraude='Não'	900	90
	Fraude='Sim'	4	6

Figura 3.5. Matriz de confusão gerada após um processo de teste de uma base de dados desbalanceada (classificador de fraudes).

A partir do conteúdo apresentado na MC da Figura 3.5, é possível saber que a base de teste possui 990 objetos com a classe Fraude= 'Não' (soma das colunas da primeira linha da matriz) e apenas 10 objetos com a classe Fraude='Sim' (soma das colunas da segunda linha). A Acurácia do classificador pode ser obtida com o uso da Equação 2:

$$Acurácia = (VP + VN) \div (VN + FP + FN + VP) = (906 / 1000) = 90{,}60\%$$

Este valor de Acurácia de 90,60% parece indicar que o desempenho preditivo do classificador na base de teste foi satisfatório. No entanto, o resultado é ilusório. Ao

observar a matriz, é trivial perceber que o classificador teve uma boa taxa de acertos para o rótulo ‘Não’, mas seu desempenho foi bem inferior para classe ‘Sim’ (algo que seria péssimo para um sistema real, pois ele não foi capaz de identificar grande parte das transações fraudulentas). Por este motivo, é interessante usar a Acurácia em conjunto com métricas capazes de avaliar o desempenho do classificador em cada rótulo de classe. Dois exemplos de medidas deste tipo são a Especificidade (também conhecida como *true negative rate*) e a Sensibilidade (também conhecida como *true positive rate*), respectivamente apresentadas nas Equações 3 e 4.

$$Especificidade = VN \div (VN + FP) \quad (3)$$

$$Sensibilidade = VP \div (VP + FN) \quad (4)$$

Com a Especificidade, torna-se possível avaliar o desempenho do classificador com relação aos verdadeiros negativos. No exemplo da Figura 3.5:

$$Especificidade = VN \div (VN + FP) = 900 \div (900 + 90) = 90{,}90\%.$$

Por sua vez, a Sensibilidade torna possível avaliar o desempenho do classificador com relação aos verdadeiros positivos:

$$Sensibilidade = VP \div (VP + FN) = 6 \div (6 + 4) = 60{,}00\%.$$

Os resultados demonstram que embora o classificador tenha tido um bom desempenho na classificação de casos que não são fraudes (90,90%), seu desempenho foi muito inferior para classificar casos de fraudes (60,00%). Além disso, o seu valor alto de Acurácia de 90,60% foi determinado unicamente pelo fato de a base de teste ter muito mais objetos da classe ‘Não’ (onde o classificador se sai bem) do que da classe ‘Sim’ (onde o desempenho do classificador é insatisfatório).

Além da Especificidade e Sensibilidade, existem diversas métricas que podem ser calculadas a partir dos dados de uma matriz de confusão. Os trabalhos de Flach (2012) e Japkowicz and Shah (2011) são indicados para aqueles que desejam se aprofundar no tema. Adicionalmente, para obter um exemplo de utilização da MC em um problema de classificação monorrótulo multiclasse, consulte [da Silva et al., 2015].

3.2.4. Classificação na Prática

Esta subseção apresenta a receita padrão para incorporar processos de classificação monorrótulo aos seus próprios programas. Os exemplos envolvem bibliotecas disponibilizadas para as linguagens Java e R. Estas duas linguagens foram escolhidas pelo fato de serem bastante diferentes entre si: enquanto Java é uma linguagem compilada e de propósito geral, R é uma linguagem interpretada e de propósito específico (serve basicamente para análise de dados). Os exemplos apresentados demonstram como executar as principais etapas de um processo de classificação: importação da base de dados rotulada, treinamento do modelo de classificação e classificação de novos objetos. Embora os exemplos envolvam o algoritmo NB, é

importante observar que, de uma forma geral, o mesmo modelo pode ser seguido com qualquer outro algoritmo (k-NN , C4.5, SMO, MLP, etc.).

Java (Weka API)

Weka [Frank et al., 2016; Witten et al., 2016] é uma popular ferramenta de aprendizado de máquina e mineração de dados *open-source* desenvolvida em Java. Esta ferramenta fornece uma API bastante poderosa e flexível que permite a integração de suas classes a qualquer tipo de sistema Java.

A Weka API trabalha preferencialmente com arquivos de entrada no formato ARFF (*Attribute Relation File Format*). Este formato corresponde a um arquivo texto contendo a base de dados propriamente dita, precedida por um pequeno cabeçalho que fornece informações a respeito de seus atributos. A Figura 3.6 apresenta a versão ARFF da base de dados de filmes apresentada na Tabela 3.1.

```
@relation filmes
@attribute love {0, 1}
@attribute people {0, 1}
@attribute relationship {0, 1}
@attribute Romance {Nao,Sim}

@data
1,1,0,Sim
1,0,0,Nao
1,0,1,Sim
0,0,1,Sim
1,1,1,Nao
1,1,0,Sim
0,0,0,Nao
0,1,0,Nao
1,1,1,Sim
0,1,1,Nao
1,0,1,Sim
1,1,0,Nao
0,0,0,Nao
1,1,1,Sim
1,0,1,Sim
```

Figura 3.6. Base de dados "filmes_treino.ARFF": versão ARFF da base de dados de filmes apresentada na Tabela 3.1.

O cabeçalho de um arquivo ARFF deve conter um apelido para o mesmo (*tag* @relation), seguido da especificação de cada atributo (*tag* @attribute). Para atributos categóricos é preciso especificar as categorias que ele pode assumir entre chaves (ex.: {0, 1}), enquanto que para os numéricos, utiliza-se a palavra "numeric". No exemplo da Figura 3.5, os atributos são "love", "people", "relationship" e "Romance" (este último, o atributo classe). Os dados propriamente ditos são precedidos por uma *tag* @data. Cada objeto é estruturado em uma linha do arquivo, com os valores separados por vírgula.

A Weka API possui uma classe denominada "NaiveBayes" que oferece uma implementação do algoritmo NB. A seguir será apresentado um exemplo que mostra o código mínimo para treinar um classificador NB e utilizá-lo para classificar dois novos

objetos (objetos de classe desconhecida). Os novos objetos foram estruturados em um arquivo ARFF denominado "filmes_novos.ARFF", que é apresentado na Figura 3.7. No arquivo de novos objetos, um ponto de interrogação ("?") é utilizado para expressar que o valor da classe "Romance" é desconhecido (trata-se da sintaxe adotada pela Weka API). Observe que as bases de dados "filmes_treino.ARFF" (que será usada para treinar o modelo) e "filmes_novos.ARFF" (base com os novos objetos) são compatíveis, isto é, elas possuem os mesmos atributos, dos mesmos tipos e declarados na mesma ordem. Isto costuma ser um pré-requisito exigido pelas bibliotecas para classificação em qualquer linguagem, como Java, R, Python, etc.

```
@relation filmes
@attribute love {0, 1}
@attribute people {0, 1}
@attribute relationship {0, 1}
@attribute Romance {Nao,Sim}

@data
1,0,1,?
0,1,0,?
```

Figura 3.7. Base de dados "filmes_novos.ARFF": contém dois novos objetos (objetos de classe desconhecida)

O código da Figura 3.8, apresenta um programa-exemplo que executa os dois passos necessários para a execução da tarefa de classificação com o algoritmo NB: (i) construção do modelo classificador e (ii) aplicação da Fórmula de Bayes para classificar novos objetos. O exemplo assume que tanto o programa Java, como as bases de dados e a biblioteca "weka.jar" (Weka API) estão localizados na mesma pasta, chamada "c:\cmr". Também é assumido que o código será compilado e executado em uma máquina que possua a JDK (*Java SE Development Kit*) versão 8 ou superior instalada. Informações detalhadas sobre como obter e configurar a Weka API podem ser obtidas em [Gonçalves, 2013].

Para compilar e executar o exemplo, salve o programa com o nome "NaiveBayesUsoBasico.java". Supondo que ele seja salvo dentro da mesma pasta onde estão localizados os arquivos "weka.jar" e as bases de dados ARFF, é possível realizar a compilação utilizando o comando abaixo (obs.: no ambiente Linux, é necessário trocar o ponto-e-vírgula ";" por dois pontos ":"):

```
> javac -cp .;weka.jar NaiveBayesUsoBasico.java
```

Para executar o programa, o processo é análogo, bastando digitar a linha de comando apresentada a seguir (o resultado da execução é mostrado na Figura 3.9):

```
> java -cp .;weka.jar NaiveBayesUsoBasico
```

```
import weka.classifiers.bayes.NaiveBayes;
import weka.core.DenseInstance;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class NaiveBayesUsoBasico {

public static void main(String[] args) throws Exception {

  // (i) Construção do Classificador

  // (i.1) importação da base de dados de treinamento
  DataSource source = new DataSource("c:\\cmr\\filmes_treino.arff");
  Instances D = source.getDataSet();

  // (i.2) espeficicação do atributo classe
  if (D.classIndex() == -1) {
      D.setClassIndex(D.numAttributes() - 1);
  }

  // (i.3) Treinamento do Modelo de Classificação
  //       (gera a "tabelona" de probabilidades condicionais)

  //basta instanciar a classe "NaiveBayes" e chamar o método
  //"buildClassifier" passando a base de treinamento como entrada
  NaiveBayes nbc = new NaiveBayes();
  nbc.buildClassifier(D);

  // (ii) Classificação dos Novos Objetos

  //(ii.1) importação da base de novos objetos
  DataSource source2 = new DataSource("c:\\cmr\\filmes_novos.arff");
  Instances novosObjetos = source2.getDataSet();

  //(ii.2) loop que classifica todos os novos objetos
  for (int i=0; i < novosObjetos.numInstances(); i++) {

      //(ii.2.1) recupera o objeto
      Instance t = novosObjetos.instance(i);

      //(ii.2.2) classifica esse objeto
      double probs[] = nbc.distributionForInstance(t);

      //(ii.2.3) imprime os resultado da classificação
      System.out.println("- Classificação do Novo Objeto " + i + ":");
      System.out.println("\tProbababilidade de Romance = 'Nao':" +
                                                        probs[0]);
      System.out.println("\tProbababilidade de Romance = 'Sim':" +
                                                        probs[1]);
      }
  }
}
```

Figura 3.8. NaiveBayesUsoBasico.java: programa que utiliza a Weka API para implementar um classificador monorrótulo NB em Java.

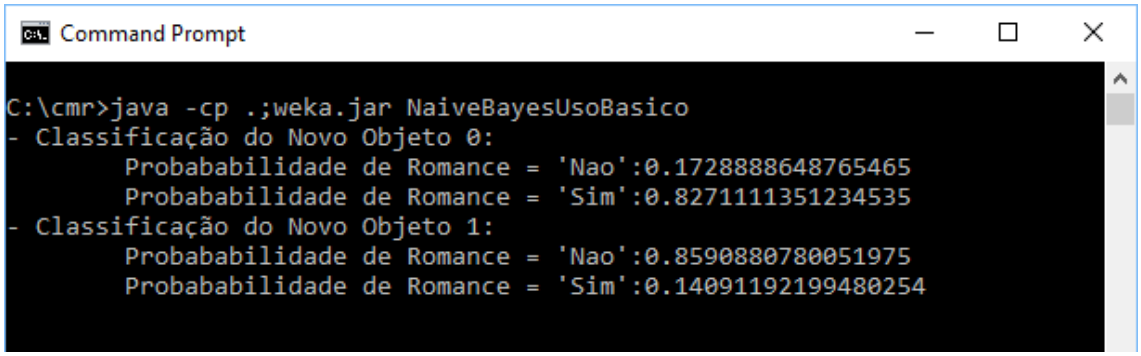

Figura 3.9. Resultado da execução do programa Java

A seguir apresenta-se uma breve explicação sobre o código. Observe que o programa-exemplo está dividido em duas seções: (i) Construção do Classificador e (ii) Classificação dos Novos Objetos. A seção (i) começa com a importação da base de dados de treinamento "filmes_treino.arff" para a memória através dos comandos mostrados na seção de código (i.1). Na Weka API, a importação de bases de dados ARFF é sempre realizada de forma padrão, com o uso das classes "DataSource", "Instances" e "DenseInstances" (ou "SparseInstances"). É importante comentar que a classe "Instances" é uma das mais importantes classes da Weka. Ela representa o conjunto de instâncias da base de dados que está armazenado em memória, oferecendo diversos métodos para a manipulação das mesmas. Consulte o JavaDoc da Weka [Weka, 2018a] e também as referências [Gonçalves, 2013] e [Weka, 2018b] para maiores detalhes.

Na seção (i.2), encontra-se o comando: "D.setClassIndex(D.numAttributes-1);". Este comando serve para informar a Weka que o último atributo da base de dados *D*, isto é, o atributo "Romance" deve ser tratado como atributo classe.

Fechando a primeira parte, a seção (i.3) é a responsável por instanciar a classe "NaiveBayes" e comandar a criação do classificador. Em outras palavras: fazer a Weka API criar a tabela de probabilidades condicionais em memória. Observe que apenas duas linhas de código são necessárias para realizar uma tarefa relativamente complexa. Para instanciar a classe "NaiveBayes" não é preciso passar nenhum parâmetro (basta fazer "new NaiveBayes()"). E para construir o classificador é preciso fazer uma chamada ao método "buildClassifier" passando a base de dados de treinamento como entrada.

A seção (ii) mostra um exemplo de código para classificação dos novos objetos. Em (ii.1) e (ii.2) encontra-se o código necessário para, respectivamente, importar a base de dados de novos objetos ("filmes_novos.ARFF") e depois percorrê-la em um laço. Dentro do laço, cada novo objeto é recuperado individualmente, sendo armazenado em *t*. Em (ii.2.2) está a parte onde a classificação NB é, de fato, realizada. É preciso uma única linha de código para executar a ação: "double probs[] = nbc.distributionForInstance(t);". Nessa linha, o método "distributionForInstance" da classe "NaiveBayes" é chamado passando como entrada o novo objeto *t*. O método "distributionForInstance" contém uma implementação da Fórmula de Bayes que efetua o cálculo das probabilidades associadas aos rótulos de classe. Como resposta, o método "distributionForInstance" retorna um vetor do tipo double, em que cada índice conterá a

probabilidade associada para um dos rótulos de classe. No exemplo, uma variável double [] chamada "probs" foi declarada para receber o retorno do método. O índice probs[0] armazenará a probabilidade computada para Romance='Não' e o índice probs[1] para Romance= 'Sim' (isto ocorre porque no cabeçalho do arquivo ARFF, o atributo classe Romance foi declarado com os valores {Nao,Sim}, ou seja, o 'Não' foi especificado antes do 'Sim').

Fechando o programa, a seção (ii.2.3) simplesmente imprime as estimativas associadas à cada classe, para os dois novos objetos. Observando a Figura 3.8, veja que para o primeiro objeto t_1 = (*love=1, people=0, relationship=1, Romance = ?*), o valor Romance ='Não' foi estimado com probabilidade de 17,29%, enquanto Romance='Sim' com probabilidade de 82,71%. Perceba que esses valores estão diferentes daqueles obtidos através do cálculo mostrado Subseção 3.2.3. Essa divergência ocorre apenas porque a implementação do NB presente na Weka é um pouco mais sofisticada e utiliza, de forma embutida, uma técnica conhecida como correção laplaciana [Han et al., 2011], que adiciona uma constante em todos os valores de probabilidade condicional mantidos na tabela de probabilidade com o intuito de evitar a presença de células contendo probabilidades condicionais com o valor zero.

Embora o exemplo tenha envolvido um programa muito simples, é importante deixar claro que o modelo básico para utilizar a Weka dentro de qualquer sistema será sempre similar. Primeiro será preciso importar a base de dados de treinamento, depois construir o classificador para, em seguida, aplicá-lo sobre novos objetos. Evidentemente, em um projeto real você precisará lidar com algumas questões adicionais. Em especial, será preciso executar um procedimento de teste do modelo de classificação antes de colocá-lo em produção usando o método *holdout* ou a validação cruzada para que seja possível computar os valores de Acurácia, Especificidade, Sensibilidade e outras métricas. Informações sobre a forma de estimar o desempenho preditivo de classificadores através da Weka API podem ser obtidas em [Weka, 2018b].

R

R [R, 2017] é um dos softwares *open-source* para análise de dados mais utilizados em todo o mundo. Existem diversos pacotes de classificação monorrótulo disponibilizados para este ambiente. Esta seção apresenta um exemplo de classificação com a técnica NB que utiliza o pacote "e1071" [Meyer, 2018]. Para obter detalhes sobre como instalar o ambiente R e qualquer um de seus pacotes, consulte [Lander, 2017].

O ambiente R suporta diferentes formatos de arquivo de entrada, sendo CSV um dos mais utilizados. Desta forma, para executar o exemplo a ser apresentado, é preciso converter as bases de dados apresentadas nas Figuras 3.6 e 3.7 do formato ARFF para o formato CSV. Para tal, basta retirar todo o conteúdo compreendido entre as *tags* @relation e @data e acrescentar uma linha de cabeçalho simples, com os nomes dos atributos separados por vírgula (love, people, relationship, Romance). Em seguida, os arquivos deverão ser salvos com a extensão CSV.

O código da Figura 3.10, apresenta um programa R para a classificação. Os passos são os mesmos que foram implementados no programa Java: (i) Construção do Classificador e (ii) Classificação dos Novos Objetos. Entretanto, observe que o código R

é bem menor, algo justificável tendo em vista que R é uma linguagem específica para análise de dados, enquanto Java é uma linguagem de propósito geral. O programa assume que o ambiente R possui o pacote "e1071" instalado e que os arquivos CSV estão localizados na pasta "c:\cmr".

```
#carrega o package "e1071"
library(e1071);

# (i) Construção do Classificador

#(i.1) importação da base de dados de treinamento para um data frame
D <- read.table("c:\\cmr\\filmes_treino.csv",header=TRUE,sep=",")

#(i.2) Treinamento do Modelo de Classificação
#      deve-se utilizar a função naiveBayes do pacote "e1071",
#      indicando o atributo classe ("Romance")
#      e a base de treinamento como parâmetros
nbc <- naiveBayes(Romance ~ ., data = D)

#visualização da tabela de probabilidades:
nbc

# (ii) Classificação dos Novos Objetos

#(ii.1) importação da base de novos objetos
novos <- read.table("c:\\cmr\\filmes_novos.csv",header=TRUE,sep=",")

#(ii.2) classificação dos novos objetos
probs = predict(nbc,novos[,-1], type="raw")

probs #mostra as estimativas para cada classe

preds = predict(nbc,novos[,-1])
preds #mostra apenas os rótulos classe preditos
```

Figura 3.10. NaiveBayesUsoBasico.R: programa que utiliza o pacote "e1071" para implementar um classificador monorrótulo NB em R.

A seção (i.1) começa com a importação da base de treinamento "filmes_treino.csv" para um *data frame* R, utilizando o método "read.table". A seção (i.2) mostra o código responsável por treinar o modelo de classificação com o uso do método "naiveBayes" do pacote "e1071". Observe que dois parâmetros são passados para o método:

- Romance ~ .: indica que o "Romance" é o atributo classe e que os demais atributos (representados por um ponto, ".") serão utilizados como atributos preditivos (essa é a notação, conhecida como "formula", é comumente utilizada na linguagem R para indicar a variável *Y* e as variáveis de *X* em diferentes pacotes para classificação e regressão).
- data = D: indica o *data frame* que armazena a base de dados de treinamento.

A seção (ii) contém o código para classificar novos objetos utilizando o modelo de classificação construído. Inicialmente, em (ii.1) importa-se a base de dados com os novos objetos para o *data frame* "novos". Então, em (ii.2) a classificação é realizada utilizando o método "predict" do pacote "e1071". Foram especificados três parâmetros:

- nbc: modelo de classificação treinado.
- novos[,-1]: *data frame* com os novos objetos. A especificação [,-1] indica quais são os atributos preditivos (todos menos o último).
- type= "raw": o método irá computar as estimativas de probabilidade para cada classe. Se for omitido, os apenas o rótulo de classe previsto é retornado.

Como resposta o método "prediction" retorna um vetor de reais, em que cada índice conterá a probabilidade associada para um dos rótulos de classe (Figura 3.11).

```
> probs = predict(nbc,novos[,-1], type="raw")
> probs #mostra as estimativas de cada classe

           Nao       Sim
[1,] 0.2234392 0.7765608
[2,] 0.7447740 0.2552260

> preds = predict(nbc,novos[,-1])
> preds #mostra apenas os rótulos de classe preditos

[1] Sim Nao
Levels: Nao Sim
```

Figura 3.11. Resultado da execução do programa R

A linguagem R também oferece pacotes para implementar o processo de teste do classificador através das técnicas *holdout*, validação cruzada e outras abordagens. Um dos mais utilizados para este fim é o pacote 'caret' [Kuhn, 2018].

3.3. Classificação Multirrótulo (CMR)

Esta seção dedica-se à apresentação dos conceitos fundamentais sobre CMR. O texto está estruturado da seguinte forma. A Subseção 3.3.1 introduz o problema da CMR, discutindo suas características gerais e seus principais desafios. Em seguida, a Subseção 3.3.2 apresenta as abordagens básicas para a construção de classificadores multirrótulo. A Subseção 3.3.3 discute as propriedades das bases de dados multirrótulo. A Subseção 3.3.4 introduz as principais medidas utilizadas para estimar a qualidade de classificadores multirrótulo. Por fim, a Subseção 3.3.5 cobre a CMR na prática, através da apresentação de *scripts* desenvolvidos em Java e R.

3.3.1. Como Construir um Classificador Multirrótulo?

Assim como ocorre com a classificação monorrótulo, duas coisas são necessárias para que seja possível construir um classificador multirrótulo:

1. Uma boa base de dados de treinamento.
2. Uma boa técnica de CMR.

Entretanto, em problemas de CMR, cada objeto pode estar associado a uma ou mais classes. Um exemplo de base de dados multirrótulo é apresentado na Figura 3.12. Suponha que o objetivo seja utilizá-la para possibilitar a criação de um classificador multirrótulo para categorização de músicas. Neste exemplo, tem-se que $L = \{$*"Metal", "Jazz", "Bossa", "Pop"*$\}$ representa um conjunto não-disjunto de rótulos de classe (gêneros musicais). Considere que cada objeto *i* é um par (x_i, Y_i), onde x_i armazena um número arbitrário de valores de atributos preditivos, enquanto $Yi \subseteq L$ armazena um subconjunto de rótulos de classe.

X	Metal	Jazz	Bossa	Pop
x_1	•			•
x_2		•	•	
x_3		•		
x_4	•			
x_5		•	•	•

Figura 3.12. Base de dados de treinamento de um classificador multirrótulo para a categorização de músicas

No exemplo da Figura 3.12, observe que o primeiro objeto (música) está associado a dois gêneros: "Metal" e "Pop". O segundo objeto também está associado a dois gêneros: "Jazz" e "Bossa". O terceiro objeto possui apenas um gênero ("Jazz") e o quarto também apenas um gênero ("Metal"). Por fim, o quinto objeto possui três gêneros ("Jazz", "Bossa" e "Pop"). A seguir apresenta-se a definição formal para o problema da classificação multirrótulo:

Definição 2 *(Classificação Multirrótulo). Seja* $X = \{X_1, \ldots, X_d\}$ *um conjunto de* d *atributos preditivos e* $L = \{l_1, \ldots, l_d\}$ *um conjunto de* q *rótulos de classe, onde* $q \geq 2$. *Considere uma base de dados de treinamento D composta por N objetos do tipo* $\{(x_1,Y_1), (x_2,Y_2), \ldots, (x_N,Y_N)\}$. *Nesta base de dados, cada* x_i *corresponde a um vetor* $\{x_1, \ldots, x_d\}$ *que armazena valores para os* d *atributos preditivos do conjunto* X *e cada* $Y_i \subseteq L$ *corresponde a um subconjunto de rótulos de classe. O objetivo da tarefa de classificação multirrótulo é, a partir de* D, *realizar o aprendizado de uma função* h *(classificador) que, dado um objeto não rotulado* t = (x,?), *seja capaz de prever de forma efetiva o seu labelset Y (conjunto de rótulos de classe).*

Os problemas de classificação multirrótulo costumam ser muito mais desafiadores do que os de classificação monorrótulo. Isto se deve as seguintes razões [Gonçalves et al., 2018]:

- As aplicações CMR normalmente precisam lidar com um número enorme de combinações de rótulos. Considerando um problema que envolva *q* rótulos de classe distintos, o tamanho do espaço de resultados em um problema CMR é 2^q, enquanto em um problema de classificação monorrótulo é de apenas *q*.
- As aplicações típicas de CMR referem-se a problemas modernos de *big data*, onde o dado a ser processado costuma ser semi-estruturado ou não estruturado: dados multimídia, dados biológicos, etc. Bases de dados reais para CMR costumam ser muito volumosas, tanto em número de atributos como no de objetos.
- Na CMR, é muito comum a existência da correlação entre rótulos de classe. Por exemplo: é improvável que uma canção seja classificada ao mesmo tempo com os rótulos "Heavy Metal" e "Jazz" dado que estes dois estilos musicais possuem uma forte correlação negativa. De maneira análoga, a probabilidade de uma canção ser rotulada como "Pop" torna-se mais forte caso ela tenha sido rotulada como "Dance" e "R&B". Consequentemente, é intuitivo esperar que algoritmos capazes de capturar e modelar as correlações entre rótulos sejam mais acurados. De fato, explorar a dependência entre rótulos tem sido tratado como uma das principais linhas de pesquisa na área de CMR [Boutell et al., 2004; Gibaja and Ventura, 2015; Gonçalves et al., 2013; Hüllermeier et al., 2008; Read et al., 2011; Zhang and Zhou, 2014].
- Em problemas monorrótulo, a classificação de um novo objeto pode estar apenas correta ou errada. Por exemplo, se um classificador monorrótulo classifica um e-mail normal como *spam*, esta situação claramente representa um erro. De maneira oposta, em uma aplicação CMR, os resultados podem estar parcialmente corretos, isto é, o modelo pode classificar corretamente alguns rótulos, mas pode errar outros. Por exemplo, se os rótulos reais de uma música são "Pop" e "Jazz" e o classificador rotulou essa música como "Pop" e "Bossa", este resultado está parcialmente correto. Devido a este fato, a avaliação de classificadores multirrótulo precisa empregar métricas diferentes das utilizadas na avaliação de classificadores monorrótulo.

3.3.2. Técnicas de Classificação Multirrótulo

Nos últimos anos diversas estratégias distintas foram propostas na literatura para a construção de classificadores multirrótulo [Gibaja and Ventura, 2014; Gonçalves et al., 2018; Tsoumakas et al., 2010]. Estes métodos podem ser divididos em duas categorias gerais: Adaptação de Algoritmo (AA) e Transformação do Problema (TP). Esta seção discute ambas as categorias com ênfase dada à segunda, uma vez que os métodos de TP costumam ser mais utilizados na prática.

Os métodos da família AA estendem ou adaptam um algoritmo de classificação monorrótulo existente para que ele possa executar a tarefa de CMR. Por exemplo, o

trabalho de Zhang and Zhou (2007) introduz o método Multi-label kNN (ML-kNN), que é derivado de dois algoritmos monorrótulo: k-NN e NB. Nesta abordagem, o processo de classificação multirrótulo de um novo objeto *t* é realizado em duas etapas. Primeiro, os *k* vizinhos mais próximos de *t* na base de treinamento são identificados. Então, para cada rótulo de classe l_j presente no *labelset* destes *k* vizinhos, a Fórmula de Bayes é empregada para estimar se *t* deverá ou não ser rotulado com l_j. Outros exemplos de métodos da família AA são apresentados e discutidos em [Gibaja and Ventura, 2014].

Os métodos da categoria TP trabalham transformando o problema de CMR original em um ou mais problemas de classificação monorrótulo. Isto possibilita com que qualquer algoritmo de classificação monorrótulo (como NB, k-NN, C4.5, etc.) possa ser diretamente aplicado através do mapeamento das predições monorrótulo para predições multirrótulo. Os métodos de transformação do problema são flexíveis, uma vez que permitem a abstração do algoritmo de classificação base (monorrótulo). Esta é uma vantagem importante, uma vez que diferentes algoritmos monorrótulo são mais ou menos efetivos de acordo com os diferentes domínios de aplicação.

Existem algumas estratégias distintas para executar a transformação de um problema multirrótulo em um ou mais problemas monorrótulo [Gibaja and Ventura, 2015; Tsoumakas et al., 2010]. Entretanto, três delas são principais, de onde as demais foram derivadas: *Label Powerset* (LP), *Ranking by Pairwise Comparison* (RPC) e *Binary Relevance* (BR). A seguir, estas três estratégias são explicadas com ajuda da base de dados para categorização de músicas ilustrada na Figura 3.12.

Label Powerset (LP)

A abordagem LP [Boutell et al., 2004; Tsoumakas et al., 2010] − também referenciada na literatura como "Label Combination" e "Label Creation" − reduz o problema multirrótulo a um único problema monorrótulo do tipo multiclasse. Isto é realizado através da definição de um novo atributo classe composto cujos valores correspondem a todas as combinações distintas de rótulos de classe presentes na base de treinamento. A Figura 3.13 ilustra o processo de transformação LP da base de dados de músicas originalmente apresentada na Figura 3.12.

X	Metal	Jazz	Bossa	Pop
x_1	•			•
x_2		•	•	
x_3		•		
x_4	•			
x_5		•	•	•

X	Y
x_1	Metal-Pop
x_2	Jazz-Bossa
x_3	Jazz
x_4	Metal
x_5	Jazz-Bossa-Pop

Figura 3.13. Transformação LP da base de dados de categorização de músicas

Observe que os dois rótulos associados ao primeiro objeto da base de dados original ("Metal" e "Pop") foram combinados para criar uma nova classe composta chamada "Metal-Pop" na base de dados transformada. De maneira análoga, as novas

classes monorrótulo “Jazz-Bossa” e “Jazz-Bossa-Pop” foram criadas, respectivamente, a partir dos dois rótulos associados ao segundo objeto e aos três rótulos associados ao quinto objeto. É interessante observar que as transformações afetam apenas os rótulos, isto é, os atributos preditivos são mantidos em sua forma original (de fato, esta é uma característica comum a todos os métodos da família TP).

Uma vez que a transformação tenha sido aplicada à base de dados original, a criação de um modelo LP pode ser feita de maneira trivial: basta treinar um classificador monorrótulo multiclasse sobre a base transformada. A classificação de uma nova instância também é trivial: o modelo LP simplesmente gera como saída uma classe composta que, de fato, corresponde a um conjunto de rótulos no problema original.

O método LP é simples e oferece a vantagem de implicitamente levar a correlação entre rótulos em consideração. Entretanto, possui dois problemas principais. Primeiro, ele não é capaz de prever *labelsets* que não estejam presentes na base de treino original. Por exemplo, o modelo LP gerado a partir da base de dados transformada da Figura 3.12 jamais será capaz de classificar uma nova música como “Jazz-Pop” ou somente como “Bossa” porque essas combinações não existem na base de dados multirrótulo original. Segundo, e mais importante, a transformação LP pode gerar um número exponencial de classes compostas, algumas delas associadas a um número reduzido de objetos. O número máximo de classes compostas é dado por $min(N, 2^q)$, onde N corresponde ao número de objetos da base de treino e q representa o número de rótulos. Embora na prática o número real de combinações seja usualmente bem menor do que o limite superior possível, ele é normalmente muito maior do que q (como será mostrado na Subseção 3.3.3). Desta forma, o método LP costuma ser inviável para muitos problemas reais.

Ranking by Pairwise Comparision (RPC)

A abordagem RPC [Hüllermeier et al., 2008] transforma a base de dados multirrótulo original em diversas bases de dados monorrótulo. Cada base de dados derivada é associada a um par distinto de rótulos $\{l_i, l_j\}$, $1 \leq i < j \leq q$ e deve conter os objetos da base de treinamento D que estejam associados ao rótulo l_i ou l_j, mas nunca associados a ambos. A Figura 3.14 apresenta o resultado da transformação RPC da base de dados de categorização de músicas. Veja que seis bases de dados binárias são produzidas como resultado da transformação.

Observe que a primeira base de dados binária se refere ao par de rótulos “Metal” e “Jazz”. Todos os objetos da base de dados podem estar associados a um dos seguintes rótulos de classe: “Metal(1)-Jazz(0)” ou “Metal(0)-Jazz(1)”. Nesta base binária, o primeiro e o quarto objetos possuem a classe “Metal(1)-Jazz(0)” porque na base de dados multirrótulo original, os respectivos objetos estão rotulados como “Metal” e não estão como “Jazz”. De maneira análoga, o segundo, terceiro e quinto objetos da base binária possuem a classe “Metal(0)-Jazz(1)” porque, na base de dados multirrótulo original estão associados ao rótulo “Jazz”, mas não estão associados ao rótulo “Metal”.

A construção de um modelo RPC consiste no treinamento de um classificador monorrótulo binário para cada uma das bases de dados derivadas. Portanto, cada classificador é treinado para um par de rótulos $\{l_i, l_j\}$, formando uma fronteira de

decisão entre ambos. Ao contrário do que ocorre no método LP, a etapa de classificação do RPC não pode ser executada diretamente. No método RPC, um novo objeto *t* a ser classificado deve ser primeiro submetido a todos os classificadores binários. O conjunto de rótulos final predito para *t* será obtido através da aplicação de uma função de limiar capaz de separar os rótulos com maior número de vitórias daqueles com menor número.

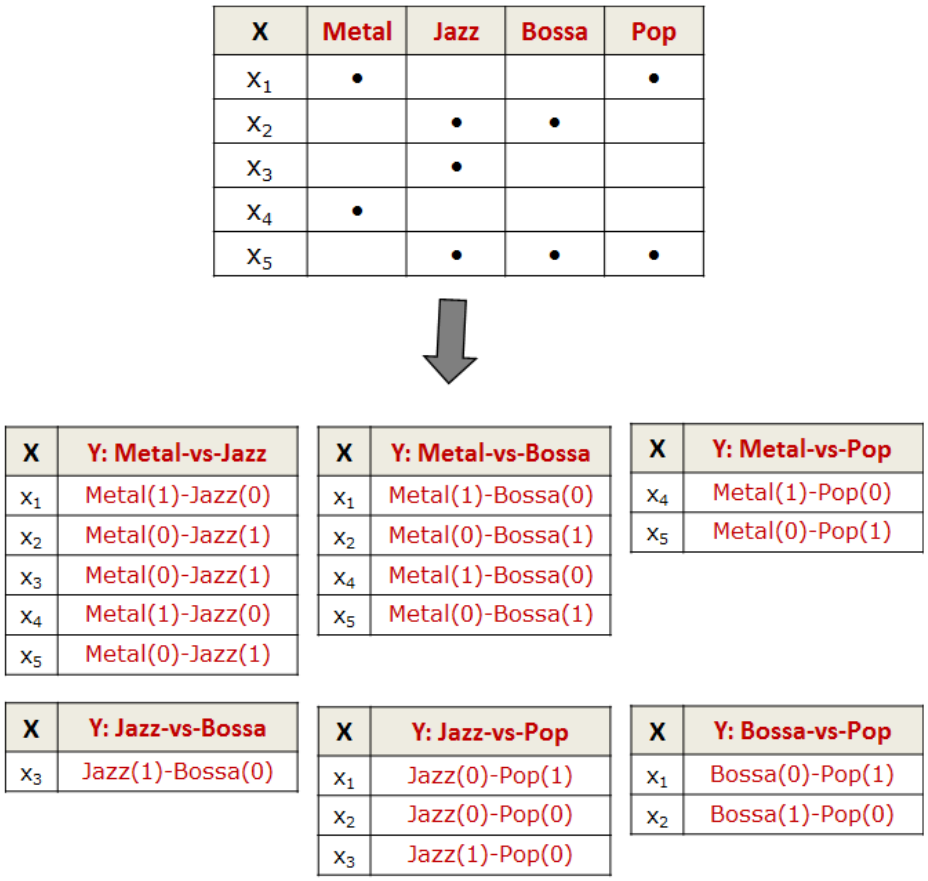

Figura 3.14. Transformação RPC da base de dados de categorização de músicas

O método RPC possui a vantagem de conseguir prever combinações de rótulos que não estão presentes na base de dados original, sendo ainda capaz de incorporar implicitamente as correlações entre rótulos no processo de construção do modelo de classificação. Entretanto, ele possui duas desvantagens importantes: primeiro o seu processo de classificação é dependente de uma função de limiar que, na prática, precisa ser calibrada de acordo com as diferentes bases de dados. Segundo, ele pode atingir complexidade quadrática em termos de espaço e tempo, uma vez que, no pior caso, um total de $q(q-1) / 2$ classificadores binários precisarão ser treinados e mantidos em memória. Todos eles precisam ser consultados no momento da classificação. Devido a este fato, o RPC acaba sendo intratável em alguns problemas reais onde q é alto.

Binary Relevance (BR)

BR [Godbole and Sarawagi, 2004; Tsoumakas et al., 2010] é o método de classificação multirrótulo mais conhecido e adotado. Nesta abordagem, a base de dados multirrótulo original é decomposta em *q* bases de dados binárias, uma para cada rótulo distinto. A Figura 3.15 apresenta o resultado da transformação BR da base de dados de categorização de músicas.

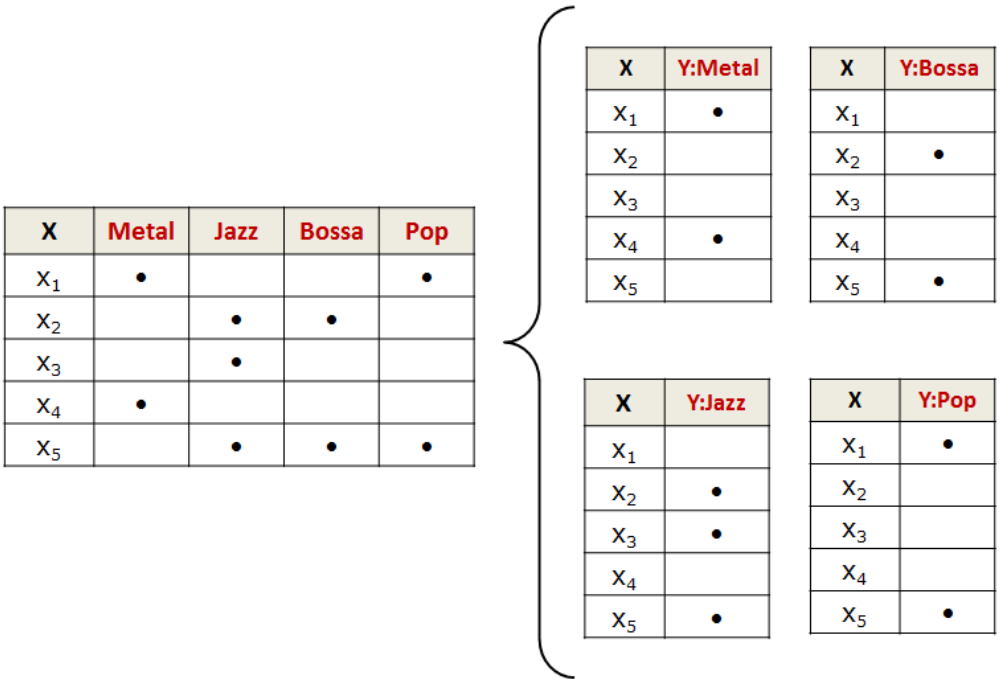

Figura 3.15. Transformação BR da base de dados de categorização de músicas

A indução de um modelo BR consiste no treinamento de um classificador binário para cada base de dados derivada. Consequentemente, no exemplo da categorização de músicas, quatro modelos binários independentes precisariam ser treinados, sendo cada um deles exclusivamente responsável pela classificação de um dos gêneros.

Tendo o modelo BR sido treinado, o processo de classificação é bastante simples: o conjunto de rótulos de um novo objeto é predito através da simples combinação das saídas produzidas por cada um dos classificadores binários. Este processo é ilustrado no exemplo da Figura 3.16. Ela mostra o processo de classificação de um novo objeto *t* (a música "Garota de Ipanema") considerando um modelo BR hipotético treinado utilizando o grupo de bases de dados derivadas apresentado na Figura 3.14. Nesta figura, f_1, f_2, f_3 e f_4 representam, respectivamente, os classificadores binários treinados para classificar os gêneros "Metal", "Jazz", "Bossa" e "Pop", enquanto *x* representa o conjunto de atributos preditivos que descrevem *t*.

t = "*Garota de Ipanema*" – Tom Jobim & Frank Sinatra

Classificadores	Classificações
f_1: x → {Metal, ~Metal}	~Metal
f_2: x → {Jazz, ~Jazz}	Jazz
f_3: x → {Bossa, ~Bossa}	~Bossa
f_4: x → {Pop, ~Pop}	Pop
Conjunto de Rótulos Predito	***{Jazz, Pop}*** $\subseteq L$

Figura 3.16. Classificação da música "Garota de Ipanema" por um modelo BR

Observe que para realizar a classificação multirrótulo do novo objeto *t*, o modelo BR agrega as predições de todos os classificadores binários independentes. No exemplo, "Metal" e "Bossa" foram preditos como não-relevantes, enquanto "Jazz" e "Pop" foram considerados relevantes. Desta forma, o *labelset* {"Jazz", "Pop"} será associado a *t*.

A estratégia BR oferece vantagens importantes. Primeiro, assim como a abordagem LP, trata-se de um método simples e intuitivo. Segundo, assim como a abordagem RPC, é capaz de prever conjuntos de rótulos que não estão presentes na base de dados multirrótulo original. Terceiro, ao contrário do LP e RPC, o método BR possui complexidade linear em *q*. Entretanto uma desvantagem óbvia e importante se encontra no fato de que um classificador BR ignora inteiramente as possíveis correlações entre os rótulos de classe, uma vez que os classificadores binários tomam decisões independentes. Apesar disso, o método costuma apresentar desempenho satisfatório, sendo muitas vezes capaz de superar os resultados obtidos por técnicas bem mais sofisticadas, como foi demonstrado em [Madjarov et al., 2012].

Como comentário final, é importante mencionar que a partir dos métodos LP, RPC e BR, diversos métodos derivados foram propostos na literatura. Algumas destas variações alcançaram grande destaque, como, por exemplo, *Classifier Chains* [Read et al., 2011], *Hierarchy of Multi-Label Classifiers* [Tsoumakas et al., 2008] e *Random k-Labelsets* [Tsoumakas et al., 2007].

3.3.3. Propriedades das Bases de Dados Multirrótulo

O desempenho dos diferentes métodos de CMR pode ser afetado pelas características da base de dados de treinamento. Esta subseção apresenta um estudo das propriedades que caracterizam e definem a complexidade de uma base de dados multirrótulo. Para facilitar a discussão, são apresentados exemplos obtidos a partir de bases de dados *benchmark* comumente utilizadas em experimentos de classificação multirrótulo (estas bases contêm dados reais de diferentes domínios de aplicação). A relação a seguir apresenta o nome de cada base, seguido de seu domínio de aplicação e uma breve descrição de suas características. A maioria delas está disponibilizada no *Web site* da ferramenta Mulan[2] e maiores informações sobre as mesmas podem ser obtidas consultando-se as referências indicadas no texto.

[2] http://mulan.sourceforge.net/datasets-mlc.html

- Emotions [Throdis et al., 2008] (classificação de músicas em emoções): classificação de músicas em 6 tipos de emoções: “sad-lonely”, “angry-aggressive”, “amazed-surprised”, “relaxing-calm”, “quiet-still” e “happy-pleased”. A base de dados possui 593 músicas diferentes descritas por 72 atributos preditivos numéricos.
- Scene [Boutell et al., 2004] (classificação semântica de cenas): classificação de imagens em 6 diferentes contextos: “beach”, “sunset”, “field”, “fall-foliage”, “mountain” e “urban”. A base de dados é formada por 2407 imagens descritas por 294 atributos preditivos numéricos.
- Flags [Gonçalves et al., 2013] (geografia): base de dados simples contendo informações sobre 194 países e suas bandeiras nacionais. Cada objeto é descrito por 19 atributos discretos e numéricos (como, por exemplo, área em km^2, idioma e religião predominante). A tarefa de CMR consiste em prever as cores presentes nas bandeiras dos países (“red”, “green”, “blue”, “yellow”, “white”, “black” e “orange”).
- Yeast [Elisseeff and Weston, 2001] (genômica funcional): esta é uma base de dados biológica onde genes de levedura podem estar associados a até 14 categorias funcionais. Possui 2417 objetos descritos por 103 atributos numéricos que representam expressões de *microarray* e perfis filogenéticos dos genes da levedura.
- POF-16 [Gonçalves et al. 2006] (pesquisa social): contém dados de 904 famílias entrevistadas pela Pesquisa de Orçamentos Familiares. Cada família é caracterizada por apenas 3 atributos: renda mensal, número de membros e cidade de residência. A tarefa CMR consiste em prever a coleção de produtos adquiridos por cada família em sua última visita a um supermercado.
- Birds [Briggs et al. 2013] (classificação de áudio e biologia): classificação das espécies de pássaros presentes em áudios de 10 segundos. Contém 645 objetos, 260 atributos preditivos e 19 rótulos de classe (espécies de pássaros).
- Genbase [Diplaris et al., 2005] (genômica funcional): classificação de proteínas em 27 funções. A base mantém dados de 662 cadeias de proteínas caracterizadas por um vetor binário de comprimento 1186.
- Medical [Pestian et al., 2007] (categorização de texto e diagnose médica): classificação de relatórios clínicos em 45 códigos distintos que representam diferentes doenças ou condições clínicas. Mantém informação sobre 978 relatórios, cada um associado a um paciente. Cada relatório é descrito por 1149 atributos binários que indicam se um termo (palavra) específico está presente ou ausente do texto (formato similar ao da base de dados de resumos de filmes, apresentada na Tabela 3.1).
- Enron [Klimt and Yang, 2004] (categorização de texto): classificação de e-mails em 54 categorias. A base contém 1702 mensagens de e-mail trocadas entre empregados da Enron Corporation que foram tornadas públicas durante uma investigação. Cada e-mail é descrito por 1001 atributos binários que indicam se um termo específico está presente ou ausente.

- Cal500 [Turnbull et al. 2008] (categorização de músicas): classificação de músicas em 174 diferentes tipos de conceitos, que, entre outros, podem representar gêneros musicais (ex.: “Soul”, “Jazz”, “Pop”, etc.) e a presença de certos instrumentos (ex.: “Synthetizer”, “Piano”, “Saxophone”, etc.). A base contém 502 músicas populares gravadas nos últimos 50 anos. Cada música é descrita por 68 atributos numéricos.

A Tabela 3.3 apresenta uma coleção de propriedades que podem ser utilizadas para indicar, de uma forma simplificada, a complexidade associada a cada uma das bases de dados *benchmark* recém-apresentadas. A primeira coluna informa o nome de cada base, enquanto a segunda, terceira e quarta colunas (N, d e q) apresentam, respectivamente, o número de objetos, atributos preditivos e rótulos de classe. Observe que as bases de dados estão apresentadas em ordem crescente de quantidade de rótulos. A quinta coluna indica a complexidade da base de dados, como proposta em [Read et al., 2011], que é obtida pelo produto $N \times d \times q$. Entretanto, é importante observar que métodos baseados na transformação do problema multirrótulo em diversos problemas monorrótulo, como o RPC e o BR, podem ser mais afetados pelo tamanho de q (parâmetro que, de fato, irá definir o número de classificadores binários que precisarão ser treinados) do que pela complexidade propriamente dita.

Tabela 3.3. Lista de bases de dados multirrótulo benchmark e as suas estatísticas com respeito à complexidade.

Base de Dados	*N*	*d*	*q*	*Complexidade*
Emotions	593	72	6	256.176
Scene	2.407	294	6	4.245.948
Flags	194	19	7	25.802
Yeast	2.417	103	14	3.485.314
POF-16	904	3	16	43.392
Birds	645	260	19	3.186.300
Genbase	662	1.186	27	21.198.564
Medical	978	1.449	45	63.770.490
Enron	1.702	1.001	53	90.296.206
Cal500	502	68	174	5.939.664

A Tabela 3.4 apresenta uma série de indicadores utilizados para fornecer uma razoável indicação das “propriedades multirrótulo” de uma base de dados [Charte and Charte, 2015]. A primeira coluna informa o nome da base, enquanto a segunda e terceira reproduzem, respectivamente o número de objetos (N) e rótulos (q). Os valores na quarta coluna (*LCard*) apresentam a cardinalidade associada a cada base de dados, que corresponde ao número médio de rótulos por objeto. O menor valor, próximo a 1,0 é da base de dados “Birds”, onde a maioria dos objetos está associada a apenas um rótulo. Por outro lado, o maior valor é superior a 26,0 na base de dados “Cal500”. Complementarmente, a quinta coluna (*LDens*) fornece o valor da densidade, que corresponde a *LCard* / q. Estas duas métricas foram introduzidas em [Tsoumakas et al., 2010]. A sexta coluna (*NC*) apresenta o número de combinações de rótulos distintas presentes em cada base. A sétima coluna (*NU*) apresenta o número de combinações

únicas, isto é, o número de combinações de rótulos que possuem frequência igual a 1 em cada base de dados. Estes dois índices foram propostos em [Read et al., 2011]. O presente trabalho introduz dois novos índices: O primeiro é *NU/NC* (coluna 8) que informa a proporção de *labelsets* únicos em relação ao número total de *labelsets* distintos. O segundo é o número de pares RPC (*NP*), apresentado na última coluna. Este índice indica o número total de pares rótulos de classe distintos $\{l_i, l_j\}$, $1 \leq i < j \leq q$ que estão associados a pelo menos um objeto da base de dados, considerando que o objeto deve possuir o rótulo l_i mas não o l_j e vice-versa. Em outras palavras: este índice fornece o número de classificadores binários que precisariam que precisariam ser treinados para construir um classificador multirrótulo através do método RPC.

Tabela 3.4. Lista de bases de dados multirrótulo benchmark e as suas estatísticas com respeito às propriedades multirrótulo.

Base de Dados	*N*	*q*	*LCard*	*LDens*	*NC*	*NU*	*NU/NC*	*NP*
Emotions	593	6	1,87	0,31	27	4	0,15	15
Scene	2.407	6	1,07	0,18	15	3	0,20	15
Flags	194	7	3,39	0,49	54	24	0,44	21
Yeast	2.417	14	4,24	0,30	198	77	0,43	91
POF-16	904	16	3,17	0,20	475	358	0,75	120
Birds	645	19	1,01	0,05	133	73	0,55	171
Genbase	662	27	1,25	0,05	32	10	0,31	350
Medical	978	45	1,25	0,03	94	33	0,35	984
Enron	1.702	53	3,38	0,06	753	573	0,76	1.378
Cal500	502	174	26,04	0,15	502	502	1,00	15.028

Conforme mencionado no início desta subseção, as propriedades de cada base de dados exercerão influência no desempenho de cada um dos diferentes métodos de CMR. Por exemplo, considere o método LP que trata cada combinação distinta de rótulos de classe existente na base de treinamento como uma nova classe monorrótulo composta. As informações apresentadas na Tabela 3.4 demonstram que este método é inviável para a maioria das bases de dados. Isto se deve a dois motivos: primeiro, o valor da propriedade *NC* (número de combinações distintas de rótulos) é superior a 30 na maioria das bases, o que pode ser considerado um número de rótulos de classes consideravelmente alto para um problema de classificação multiclasse. Segundo, observe que muitas bases possuem um número alto de combinações que estão associadas a apenas um objeto (propriedade *NU*). No caso das bases "POF-16", "Birds", "Enron" e "Cal500", onde a proporção *NU/NC* é superior a 50%, o que significa que entre o total de combinações de rótulos distintas, mais de 50% estão associadas a um único objeto da base de treinamento. Portanto, o método LP teria que lidar com um grande número de classes compostas que ocorrem apenas uma vez na base resultante da transformação. No caso extremo da "Cal500", o número de combinações distintas de rótulos é igual ao número de objetos (*NC = N*), e, desta forma, o método LC teria que lidar com apenas um objeto por classe composta.

As informações da Tabela 3.4 também possibilitam uma comparação entre os métodos RPC (onde um classificador binário deve ser treinado para cada par de rótulos de interesse) e BR (onde um classificador binário é treinado para cada rótulo do

problema) em termos de complexidade de tempo e espaço. Neste sentido, a propriedade *NP* revela que o número de classificadores binários que precisariam ser treinados e mantidos em memória para o método RPC é igual ou muito próximo ao limite superior de *q(q-1) / 2* para todas as bases de dados. Por outro lado, o método BR requer o treinamento de apenas *q* classificadores.

Em resumo: o método BR costuma ser computacionalmente viável para qualquer problema real de classificação. Por este motivo, tornou-se muito difundido e utilizado na prática, representando ainda o principal método *benchmark* na área de CMR (isto é: novos métodos propostos costumam ser comparados com o BR, tanto em termos de eficácia como de eficiência).

3.3.4. Medidas de Desempenho para Classificadores Multirrótulo

Nos últimos anos, diversas medidas de desempenho preditivo especialmente projetadas para CMR foram propostas na literatura. As plataformas Mulan [Tsoumakas et al., 2011] e Meka [Read et al., 2016], por exemplo, disponibilizam mais de vinte diferentes métricas aos seus usuários. Esta subseção introduz e compara algumas das mais comumente utilizadas. Nos exemplos e definições, a seguinte notação foi utilizada:

- n: número de objetos de teste;
- q: número de rótulos;
- Y_i: labelset real do objeto i da base de teste.
- Z_i: labelset predito para o objeto i da base de teste.

A medida mais trivial para avaliar o desempenho de métodos CMR é o Casamento Exato (*exact match*), definida na Equação 5. Esta medida determina a proporção de objetos que tiveram todos os seus rótulos preditos de maneira correta na base de teste. Considere que *I(VERDADEIRO) = 1* e *I(FALSO) = 0*.

$$CE = \frac{1}{n}\sum_{i=1}^{n} I\left(Y_i = Z_i\right) \tag{5}$$

Apesar de fornecer uma informação de grande importância, o Casamento Exato é uma medida considerada rigorosa demais para ser empregada sozinha. Isto porque, em problemas de CMR o resultado de uma classificação pode frequentemente estar parcialmente correto. Portanto, torna-se necessário empregar outros índices em processos de avaliação de classificadores multirrótulo. Um dos mais simples é a Acurácia Multirrótulo (*multi-label accuracy*), apresentada na Equação 6. Esta medida pode ser vista como uma versão "mais leve" do Casamento Exato. Ela provê informação sobre a proporção de rótulos classificados de maneira correta, levando em consideração resultados parcialmente corretos.

$$AC = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \quad (6)$$

A Tabela 3.5 ilustra o uso da Acurácia Multirrótulo em um problema hipotético envolvendo 6 exemplos (objetos de teste) e 4 rótulos de classe [Gonçalves et al., 2013]. Observe que a medida é simples e intuitiva: quanto maior o seu valor, melhor a classificação. Entretanto, os casos ilustrados nos exemplos E_5 e E_6 evidenciam uma desvantagem associada a esta medida: ela não é muito severa para penalizar predições binárias erradas (sejam falsos positivos ou falsos negativos).

Tabela 3.5. Ilustração da Acurácia Multirrótulo para 6 objetos de teste.

	Y_i	Z_i	*Acurácia Multirrótulo*	*Comentários*
E_1	l_1, l_2, l_3, l_4	l_1, l_2, l_3, l_4	1,00	Classificação perfeita
E_2	l_1	l_2, l_3, l_4	0,00	Pior caso
E_3	l_1	l_2	0,00	Pior caso
E_4	l_1, l_3	l_1, l_2	0,33	
E_5	l_1	l_1, l_2	0,50	
E_6	l_1, l_2	l_1, l_2, l_3, l_4	0,50	Número maior de falsos positivos em relação à E_5, mas o valor continua sendo 0,50

Para lidar com esse problema, é possível empregar a medida conhecida como Hamming Loss, definida na Equação 7. Esta medida informa o número médio de predições binárias incorretas por objeto de teste. A expressão $|Y_i \Delta Z_i|$ representa a diferença simétrica entre Y_i e Z_i, a qual é equivalente à operação booleana XOR. Quanto menor o valor do Hamming Loss, melhor o desempenho. O uso da medida Hamming Loss em um novo problema com 6 exemplos e quatro rótulos de classe é ilustrado na Tabela 3.6 [Gonçalves et al., 2013].

$$\mathrm{HL} = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \Delta Z_i|}{q} \quad (7)$$

Tabela 3.6. Ilustração do Hamming Loss para 6 objetos de teste.

	Y_i	Z_i	*Acurácia Multirrótulo*	*Comentários*
E_1	l_1, l_2, l_3, l_4	l_1, l_2, l_3, l_4	0,00	Classificação perfeita
E_2	l_1	l_2, l_3, l_4	1,00	Pior caso
E_3	l_1	l_2	0,50	
E_4	l_1, l_3	l_1, l_2	0,50	Esta classificação é melhor do que a de E_3, mas o valor continua sendo 0,50
E_5	l_1	l_1, l_2	0,25	
E_6	l_1, l_2, l_3	l_1, l_2, l_3, l_4	0,25	Número maior de predições corretas em relação à E_5, mas o valor continua sendo 0,25

Observe que a mais importante vantagem do Hamming Loss sobre a Acurácia Multirrótulo é o fato de que ele é mais adequado para penalizar predições binárias incorretas. Contudo, em muitas situações práticas essa característica pode se tornar uma desvantagem. Por exemplo, considere os quatro últimos exemplos da Tabela 3.6. Eles mostram que, considerando um par de exemplos com o mesmo número de predições binárias incorretas (como E_3 e E_4 ou E_5 e E_6), a medida de Hamming Loss não diferencia o caso com um número maior de classificações corretas do caso que possui um menor número.

É importante enfatizar que, a despeito de suas idiossincrasias, todas as três medidas (Casamento Exato, Acurácia Multirrótulo e Hamming Loss) são importantes, uma vez que fornecem informações complementares sobre o desempenho de um classificador multirrótulo. Outro importante aspecto relacionado a estas três medidas é elas são computadas levando em conta todos os rótulos reais e previstos de um exemplo (objeto de teste). Por este motivo, são chamadas de medidas baseadas em exemplo (*example-based measures*).

Entretanto, em certas situações, pode também ser interessante avaliar o classificador em cada rótulo de classe separadamente. Isto é: criar uma matriz de confusão para cada rótulo e computar a Sensibilidade, Especificidade e outras medidas para cada *label*. Desta forma, torna-se possível verificar se o desempenho do classificador está equilibrado entre os vários rótulos.

3.3.5. Classificação Multirrótulo na Prática

Esta subseção apresenta exemplos de *scripts* para CMR desenvolvidos em Java e R. Nestes exemplos, um classificador BR que utiliza o NB como algoritmo base (BR + NB) é criado a partir de uma base de treinamento multirrótulo. Em seguida, o modelo BR + NB é utilizado para classificar uma base de novos objetos. O objetivo principal é dar uma noção sobre como a CMR funciona na prática.

Java (Weka API)

As plataformas Mulan [Tsoumakas et al., 2011] e Meka [Read et al., 2016] são muito referenciadas na literatura de CMR. Ambas são ferramentas *open source*, criadas no topo da Weka API e voltadas para a elaboração de pesquisas científicas: elas foram projetadas de modo a facilitar o desenvolvimento de protótipos e a implementação de experimentos de avaliação de classificadores. Além disso, representam ótima fonte de consulta para aqueles que desejam aprender aspectos avançados de programação relacionados à Weka API. No entanto, nem a Mulan e nem a Weka foram construídas com o propósito de serem integradas a aplicativos desenvolvidos em Java. Sendo assim, nesta seção apresenta-se um exemplo que depende apenas da Weka API para ser executado e que, com poucas modificações, poderá ser integrado diretamente a qualquer sistema desenvolvido em Java (é importante observar que o código apresentado tem por base a implementação do método BR disponível na Mulan).

O exemplo utiliza a base de dados “Flags” [Gonçalves et al., 2013], descrita na Subseção 3.3.3. Esta base pode ser obtida no formato ARFF a partir do Web site da ferramenta Mulan. Ela armazena informações sobre países, que são caracterizados por

19 atributos preditivos. A tarefa de CMR consiste em prever as cores presentes nas bandeiras dos países, onde L = {“red”, “green”, “blue”, “yellow”, “white”, “black”, “orange”}. No exemplo, serão utilizadas as partições de treino para criar o classificador (“flags-train.ARFF”) e de teste para classificar novos objetos (“flags-test.ARFF”) – ambas as partições são fornecidas no arquivo de *download* da base de dados. O cabeçalho da base “flags-train.ARFF” e as seus três primeiros objetos são apresentados na Figura 3.17. Observe que os sete últimos atributos representam os rótulos de classe[3].

```
@relation flags_ml

@attribute landmass {1,2,3,4,5,6}
@attribute zone {1,2,3,4}
@attribute area numeric
@attribute population numeric
@attribute language {1,2,3,4,5,6,7,8,9,10}
@attribute religion {0,1,2,3,4,5,6,7}
@attribute bars numeric
@attribute stripes numeric
@attribute colours numeric
@attribute circles numeric
@attribute crosses numeric
@attribute saltires numeric
@attribute quarters numeric
@attribute sunstars numeric
@attribute crescent {0,1}
@attribute triangle {0,1}
@attribute icon {0,1}
@attribute animate {0,1}
@attribute text {0,1}
@attribute red {0,1}
@attribute green {0,1}
@attribute blue {0,1}
@attribute yellow {0,1}
@attribute white {0,1}
@attribute black {0,1}
@attribute orange {0,1}

@data
4,1,164,7,8,2,0,0,2,1,0,0,0,1,1,0,0,0,0,1,0,0,0,1,0,0
4,4,111,1,10,5,0,11,3,0,0,0,1,1,0,0,0,0,0,1,0,1,0,1,0,0
1,4,0,0,1,1,0,0,3,1,0,0,0,7,0,1,0,1,0,1,1,0,1,0,0,0
...
```

Figura 3.17. Base de dados “flags-train.ARFF”

A seguir, apresenta-se um programa-exemplo que treina um modelo de classificação BR + NB utilizando a base de dados apresentada na Figura 3.17. Em seguida, o modelo é utilizado para classificar objetos armazenados no arquivo “flags-test.ARFF”. Como o código é bem maior do que o do exemplo envolvendo a classificação monorrótulo, ele foi dividido em duas figuras: Figura 3.18 (etapa de treinamento) e Figura 3.19 (mostra a etapa de classificação).

[3]As bases de dados ARFF utilizadas pela ferramenta Mulan são sempre fornecidas em conjunto com um arquivo XML que indica quais atributos representam os rótulos de classe.

```
import weka.core.converters.ConverterUtils.DataSource;
import weka.core.DenseInstance;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.Attribute;
import weka.classifiers.AbstractClassifier;
import weka.classifiers.Classifier;
import weka.classifiers.meta.FilteredClassifier;
import weka.filters.unsupervised.attribute.Remove;
import weka.classifiers.bayes.NaiveBayes;

public class BR {

public static void main(String[] args) throws Exception {

  // (i) Construção do Classificador BR
  //  (implementação baseada no código da ferramenta MULAN)

  // (i.1) importacao da base de dados de treinamento
  DataSource source = new DataSource("c:\\cmr\\flags-train.arff");
  Instances D = source.getDataSet();
  int q = 7;                    //número de rótulos
  int d = D.numAttributes()-q; //número de atributos preditivos

  //(i.2) cria um array h[] de classificadores NB (um para cada label)
  NaiveBayes baseClassifier = new NaiveBayes();
  FilteredClassifier[] h = new FilteredClassifier[q];
  for (int yj = 0; yj < q; yj++) {
    h[yj] = new FilteredClassifier();
    h[yj].setClassifier(AbstractClassifier.makeCopy(baseClassifier));
    }

  // (i.3) treina o modelo BR
  // Este laço treina um classificador binário para cada label
  for (int yj = 0; yj < q; yj++) {

    // define os labels que serão removidos (todos, exceto yj!)
    int l_index = d + yj;
    int [] labelsToRemove = new int[q - 1];
    int aux = 0;
    for (int k = 0; k < q; k++) {
      if (k != yj) {
        labelsToRemove[aux] = d + k;
        aux++;
      }
    } //--for k

    // treina o "filtered classifier" associado a yj
    Remove remFilter = new Remove();
    remFilter.setAttributeIndicesArray(labelsToRemove);
    remFilter.setInputFormat(D);
    remFilter.setInvertSelection(false);
    h[yj].setFilter(remFilter);
    D.setClassIndex(l_index);
    h[yj].buildClassifier(D);
  } // --for yj
```

Figura 3.18. BR.java (parte 1 – etapa de treinamento): programa que utiliza a Weka API para implementar um classificador multirrótulo BR + NB

```
  // (ii) Classificação dos Novos Objetos

  //(ii.1) importação da base de novos objetos
  DataSource source2 = new DataSource("c:\\cmr\\flags-test.arff");
  Instances T = source2.getDataSet();
  int n = T.numInstances();

  //(ii.2) loop que classifica todos os novos objetos
  for (int k = 0; k < n; k++) {

    Instance t = T.instance(k); //obtém o objeto corrente

    int Z[] = new int[q];  //vetor que armazenará as predições
                           // de cada label para o objeto corrente
    System.out.println();

    for (int yj = 0; yj < q; yj++) {

      double probs[] = h[yj].distributionForInstance(t);

      Z[yj] = (probs[0] > probs[1]) ? 0 : 1;

      System.out.print(Z[yj]); //imprime a predição para o label yj
    } // --for yj
  } // --for k

  }
}
```

Figura 3.19. BR.java (parte 2 – etapa de classificação): programa que utiliza a Weka API para implementar um classificador multirrótulo BR + NB

Para compilar o programa, é preciso juntar o código apresentado nas duas figuras e salvar o arquivo com o nome "BR.java". Mais uma vez, o exemplo assume que o código será criado e executado em uma máquina que possua a JDK versão 8 e superior e que o programa Java, as bases de dados e a biblioteca "weka.jar" (Weka API) estão localizados na mesma pasta (neste caso, pasta "c:\cmr").

A seguir apresenta-se uma breve explicação sobre o *script*, começando pela etapa de treinamento (Figura 3.18). Neste programa, o método BR é implementado nas seções de código (i.2) e (i.3). Na Seção (i.2) cria-se um vetor "h", do tipo *filtered classifier* (classificador baseado em filtros) com dimensão *q* (número de rótulos de classe). Cada índice de "h" irá referenciar um classificador binário a ser treinado para um *label* específico. Inicialmente, associa-se um classificador NB "vazio" a cada posição, ou seja, apenas instanciado, mas não treinado.

A Seção (i.3) é a maior e mais importante do código. Nesta seção, é implementado um laço com *q* iterações. Um vetor denominado "labelsToRemove[]" de dimensão *q-1* é criado. Este recebe os índices de todos os rótulos de classe que deverão ser removidos para a criação de um classificador binário específico. Por exemplo, a primeira iteração é responsável por criar o classificador binário para o primeiro rótulo de classe, "red". Sendo assim, nesta iteração, o vetor "labelsToRemove" recebe os índices de todos os outros rótulos ("green", "blue", "yellow", "white", "black" e "orange"). Esse vetor é então utilizado em um filtro do tipo "Remove" (um filtro é um recurso da Weka

API para transformar bases de dados em memória) para possibilitar com que, no primeiro elemento "h" seja treinado um classificador binário para o label "red". Mais especificamente, o filtro remove todos os outros rótulos, exceto "red" e treina o classificador considerando apenas os 19 atributos preditivos da base "flags-train.ARFF". Nas iterações seguintes, o mesmo processo é realizado para os labels restantes. Para maiores informações sobre o uso de filtros na Weka, consulte [Weka, 2018a; Weka, 2018b].

Terminado o treinamento, o processo de classificação de novos objetos com o modelo BR + NB é trivial, como mostra a Figura 3.19. Para cada novo objeto, basta implementar um laço com *q* iterações que acionará cada classificador binário. A classificação multirrótulo final é obtida através da união das saídas geradas por todos os *q* classificadores binários. Os resultados da classificação multirrótulo dos 65 objetos da base "flags-test.ARFF" são apresentados na tela.

R

As Figuras 3.20 e 3.21 apresentam o *script* que reproduz na linguagem R os mesmos passos executados pelo programa Java recém-apresentado. Para utilizá-lo, é necessário converter as bases "flags-train.ARFF" e "flags-teste.ARFF" para o formato CSV. O programa assume que os arquivos CSV estão localizados na pasta "c:\cmr".

```
#carrega o package "e1071"
library(e1071);

# (i) Construção do Classificador BR

# (i.1) importacao da base de dados de treinamento
D <- read.table("c:\\cmr\\flags-train.csv",header=TRUE,sep=",")


#(i.2) cria uma lista h[] de classificadores NB (um para cada label)
h = list()
q = 7
d = ncol(D) - q

for (yj in 1:q) {
  #converte o atributo classe para "factor"
  D[,ncol(D)-q+yj] = factor(D[,ncol(D)-q+yj])

  #monta a formula para treinar o classificador binário yj
  aux = paste(names(D[ncol(D)-q+yj]),'~',names(D[1]))
  for (x in 2:d) {
    aux = paste(aux,"+",names(D[x]))
  }
  print(aux) #apenas para visualizar a formula na tela...

  f = as.formula(aux)
  h[[length(h)+1]] <- naiveBayes(f, data=D)
}
```

Figura 3.20. BR.R (parte 1 – etapa de treinamento): *script* R que implementa um classificador multirrótulo BR + NB

```
# (ii) Classificação dos Novos Objetos


#(ii.1) importação da base de novos objetos
novos <- read.table("c:\\cmr\\flags-test.csv",header=TRUE,sep=",")


#(ii.2) classificação dos novos objetos
preds = list()
for (yj in 1:q) {
  preds[[length(preds)+1]] <- predict(h[[yj]],novos[,1:d])
}


#(ii.3) imprime as classificações para todos os novos objetos
n = nrow(novos)
for (i in 1:n) {
  Z=""
  for (yj in 1:q) {
    Z = paste(Z,preds[[yj]][i])
  }

  print(Z)
}
```

Figura 3.21. BR.R (parte 2 – etapa de classificação): *script* R que implementa um classificador multirrótulo BR + NB

A ideia utilizada no *script* R é basicamente a mesma utilizada no programa Java. Na etapa de treinamento (Figura 3.20), cria-se uma lista de classificadores binários chamada "h" de tamanho *q*. Cada posição da lista é associada a um classificador NB treinado para um rótulo de classe específico (na posição 1, classificador para a classe "red", na posição 2 para a classe "green" e assim por diante). Em seguida, na etapa de classificação (Figura 3.21), os novos objetos são importados. A classificação multirrótulo de todos esses objetos é executada em apenas 4 linhas, no código apresentado na seção (ii.2). Mais uma vez, implementa-se um laço com *q* iterações para que todos os classificadores binários sejam acionados e realizem a classificação de cada rótulo. Fechando o programa, na seção (ii.3), as classificações multirrótulo são impressas na tela.

3.4. Considerações Finais

Este capítulo apresentou os conceitos fundamentais sobre classificação multirrótulo e uma noção geral dos problemas que esta tecnologia ajudou e está ajudando a solucionar, tanto em ambiente acadêmico como nas empresas. Sem ter a pretensão de cobrir toda a área, o texto teve como objetivo principal apresentar o conteúdo necessário para aqueles que pretendem começar a estudar o tema e que, futuramente, intencionam incorporar processos de classificação aos seus próprios sistemas de informação. A seguir, são sugeridos tópicos para pesquisas futuras, destinados aos que se interessarem em conhecer mais sobre CMR:

- O presente trabalho apresentou as três abordagens elementares para a construção de classificadores multirrótulo: LP, RPC e BR. No entanto, muitas outras técnicas importantes foram propostas ao longo dos últimos anos. O trabalho de Zhang and Zhou (2014) apresenta uma comparação entre as oito técnicas para CMR consideradas mais representativas na atualidade, ou seja, as técnicas mais citadas na literatura (entre elas, BR e ML-kNN). Já o artigo de revisão de Gibaja and Ventura (2014) propõe uma taxonomia que categoriza dezenas de métodos distintos. Conhecer algumas destas técnicas é importante para aqueles que desejam atuar na área de CMR.
- De maneira análoga, o texto apresentou apenas os métodos básicos para avaliar a qualidade de classificadores (*holdout* e validação cruzada) e introduziu algumas das medidas de desempenho mais comumente utilizadas. Informações detalhadas a respeito dos métodos *holdout*, validação cruzada e outros podem ser obtidas em [Flach, 2012; Han et al., 2016; Japkowicz and Shah, 2011; Witten et al., 2016]. Para um estudo detalhado sobre medidas de desempenho preditivo de classificadores multirrótulo, recomenda-se a referência [Pereira et al., 2018], onde uma análise de 16 diferentes medidas é realizada com o intuito de permitir com que usuários de sistemas CMR possam melhor entende-las.
- Ao longo do texto, foram apresentados exemplos simples de *scripts* para classificação multirrótulo elaborados em Java. Aqueles que desejam trabalhar com esta linguagem, poderão consultar as referências [Gonçalves, 2013; Read et al., 2016, Tsoumakas et al., 2011; Weka, 2018a, Weka, 2018b] para se aprofundar no tema.
- No caso da linguagem R, alguns pacotes específicos para CMR foram recentemente disponibilizados. Por exemplo, o pacote ‘mldr’ [Charte and Charte, 2015] fornece uma série de métodos para explorar e transformar bases de dados multirrótulo, permitindo a visualização de suas principais propriedades em tabelas e gráficos. Já o pacote ‘utiml’ [Rivolli, 2018] foi projetado com o objetivo de servir como uma plataforma para o desenvolvimento de protótipos e para a avaliação de classificadores multirrotulo (mesmo papel desempenhado pelas plataformas Mulan e Meka na linguagem Java).

Referências

Almeida, A. M. G., Cerri, R., Paraiso, E. C., Mantovani, R. G. and Junior, S. B. (2018). Applying multi-label techniques in emotion identification of short texts. In: Neurocomputing, 2018, preprint.

Boutell, M. R., Luo, J., Shen, X. and Brown, C. M. (2004). Learning multi-label scene classification. Pattern Recognition, v. 37, p. 1757−1771.

Briggs, F. et al. (2013). “The 9th Annual MSLP Competition: New Methods for Acoustic Classification of Multiple Simultaneous Bird Species in a Noisy Environment”. In: MLSP 2013, pages 1–8.

Charte, F. and Charte, D. (2015). Working with multi-label datasets in R: the mldr package. The R Journal, v. 7(2), p. 149−162.

Chen, L., Fan, C., Yang, H., Hu, S., Zou, L. and Deng, D. (2018). Face age classification based on a deep hybrid model. In Signal, Image and Video Processing, 12(8), pages 1531–1539

Cherman, E. and Monard, M. C. (2009) "Um Estudo sobre Métodos de Classificação Multirrótulo", In: Anais do IV Congresso da Academia Trinacional de Ciências, PTI, p. 1–10.

Diplaris, S.; Tsoumakas, G. and Mitkas, P. A. (2005) "Protein Classification with Multiple Algorithms", In: Advances in Informatics (LNCS 3476), Edited by P. Bozanis and E. Houstis, Springer, USA.

Elisseeff, A. and Weston, J. (2001). A kernel method for multi-labelled classification. In Advances in Neural Information Processing Systems, 14, pages 681–687.

Flach, P., Machine Learning: The Art and Science of Algorithms that Make Sense of Data, Cambridge University Press, 2012.

Frank, E., Hall, M. A. and Witten, I. H. (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, 4th ed., 2016. Disponível em: < https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf >. Acesso em: 25 set. 2018.

Gibaja, E. and Ventura, S. (2014). Multi-label learning: a review of the state of the art and ongoing research. In: WIREs Data Mining and Knowledge Discovery, 4(6), pages 411–444,Wiley.

Gibaja, E. and Ventura, S. (2015). A tutorial on multi-label learning. In: ACM Computing Surveys (CSUR), 47(3), pages 52:1–52:38, ACM.

Godbole, S. and Sarawagi, S. (2004). Discriminative methods for multi-label classification. In: Advances in Knowledge Discovery and Data Mining, 3056, pages 22–30.

Gonçalves, E. C. (2013). Mineração de dados na prática com a weka api. In: SQL Magazine, 107, pages 32–40, Devmedia.

Gonçalves, E. C., Freitas, A. A. and Plastino, A. (2018) "A Survey of Genetic Algorithms for Multi-Label Classification", In: Proc. of the 2018 IEEE Congress on Evolutionary Computation, IEEE, p. 981–988.

Gonçalves, E. C., Plastino, A. and Freitas, A. A. (2013) "A Genetic Algorithm for Optimizing the Label Ordering in Multi-Label Classifier Chains", In: Proc. of the 2013 IEEE 25th Int'l Conf. on Tools with Artificial Intelligence, IEEE, p. 469–476.

Gonçalves, E. C., Rabelo, F. A., Souza, I., Oliveira, M. R., Pereira, M. A. L., Araújo, R. B. S. (2006) "Construção de um Data Mart para a Análise dos Hábitos de Compra das Famílias Brasileiras", In: Anais do III Simpósio Mineiro de Sistemas de Informação, PUC-MG, p. 1–9.

Han, J., Kamber, M. and Pei, J., Data Mining: Concepts and Techniques, 3rd ed., Morgan Kaufmann, 2011.

Huang, Y-H., Hung, C-M. and Jiau, H. C. (2015). A multi-label approach using binary relevance and decision trees applied to functional genomics. In: Journal of Biomedical Informatics, v. 54, p. 85−95.

Hüllermeier, E., Fürnkranz, J., Cheng, W. and Brinker, K. (2008). Label ranking by learning pairwise preferences. In: Artificial Intelligence, 172(16-17), p. 1897−1916.

Japkowicz, N. and Shah, M., Evaluating Learning Algorithms: A Classification Perspective, Cambridge University Press, 2011.

Klimt, B. and Yang, Y. (2004) "The Enron Corpus: A New Dataset for Email Classification", In: Proc. of the ECML 2004, Springer, p. 217–226.

Kuhn, M. (2018). Package 'caret'. Disponível em: <https://cran.r-project.org/web/packages/caret/caret.pdf>. Acesso em: 25 set. 2018.

Lander, J. P. R for Everyone. 2nd ed., Addison-Wesley, 2017.

Madjarov, G., Kocev, D., Gjorgjevikj, D. and Džeroski, S. and Brinker, K. (2012). An extensive comparison of methdos for multi-label learning. In: Pattern Recognition, 45(9), p. 3084−3104.

Marsland, S., Machine Learning: An Algorithmic Perspective, 2nd ed., CRC Press, 2015.

Meyer et al. (2018). Package 'e1071'. Disponível em: <https://cran.r-project.org/package=e1071 >. Acesso em: 25 set. 2018.

Pereira, R. B., Plastino, A., Zadrozny, B. and Merschmann, L. H. C. (2018). Correlation analysis of performance measures for multi-label classification. In: Information Processing and Management, 54(3), pages 359–369.

Pestian, J. P. et al. (2007). "A Shared Task Involving Multi-Label Classification of Clinical Free Text". In: Proc. of the Workshop on BioNLP 2007, pages 87–102.

R Core Team (2017). R: A Language and Environment for Statistical Computing. Disponível em: <https://www.R-project.org/>. Acesso em: 25 set. 2018.

Read, J., Pfahringer, B., Holmes, G. and Eibe, F. (2011). Classifier chains for multilabel classification. In Machine Learning, 85(3), pages 333–359.

Read, J., Reutemann, P., Pfahringer, B., Holmes, G. (2016). Meka: a multi-label/multi-target extension to Weka. In Journal of Machine Learning Research, 17, pages 1–5.

Rivolli, A. (2018). Package 'utiml'. Disponível em: <https://cran.r-project.org/web/packages/utiml>.

da Silva, P. N. et al. (2015). Automatic classification of carbonate rocks permeability from 1H NMR relaxation data. In: Expert Systems with Applications, 42(9), pages 4299–4309.

Trohidis, K., Tsoumakas, G., Kalliris, G. and Vlahavas, I (2011). Multi-label classification of music by emotion. EURASIP J. Audio Speech Music Process., 2011:4.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2008) "Effective and Efficient Multilabel Classification in Domains with Large Number of Labels", In: Proc. of the ECML/PKDD 2008 Workshop on Mining Multidimensional Data, Springer.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010) "Mining Multi-Label Data", In: Data Mining and Knowledge Discovery Handbook, Edited by O. Maimom and L. Rokach, Springer, USA.

Tsoumakas, G. and Vlahavas, I. (2007) "Random k-Labelsets: An Ensemble Method for Multi-Label Classification", In: Proc. of the ECML 2007, Springer, p. 17–21.

Tsoumakas, G., Xioufis, E. S., Vilcek, J. and Vlahavas, I. P. (2011). Mulan: A java library for multilabel learning. In Journal of Machine Learning Research, 12, pages 2411–2414.

Tsoumakas, G. et al. (2014). "WISE 2014 Challenge: Multi-label Classification of Print Media Articles to Topics", In: Proc. of the Int'l Conf. on Web Information Systems Engineering, Springer-Verlag, p. 541–548.

Turnbull, D., Barrington, L., Torres, D. and Lanckriet, G. (2008). Semantic annotation and retrieval of music and sound effects. In IEEE Transactions on Audio, Speech, and Language Processing, 16(2), pages 467–476.

Ururahy, J. M. K., Bastos, J. L. G. V., Federici, A. C. M., Gonçalves, E. C. (2018) "Classificação Multirrótulo de Documentos Texto Utilizando a Relevância Binária e o Algoritmo Naïve Bayes", In: Anais do III Seminário de Estatística com R, Galoá.

Weka Javadoc (2018). Disponível em: <http://weka.sourceforge.net/doc.stable/>. Acesso em: 19 set. 2018.

Weka Wiki (2018). Using weka in your java code. Disponível em: <https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/>. Acesso em: 19 set. 2018.

Witten, I., Frank, E., Hall, M. and Pal, C. Data Mining: Practical Machine Learning Tools and Techniques. 4th ed., Morgan Kaufmann, 2016.

Xia, X., Feng, Y., Lo, Z., Chen, Z. and Wang, X. (2014) "Towards More Accurate Multi-Label Software Behavior Learning", In: Proc. of the 2014 CSMR-WCRE, IEEE, p. 134–143.

Xiao, J., Ehinger, K. A., Hays, J., Torralba, A. and Oliva, A. (2016). SUN database: Exploring a large collection of scene categories. In: International Journal of Computer Vision, 116(1), pages 3–22.

Zhang, W., Liu, F., Luo, L. and Zhang, J. (2015). Predicting drug side effect by multi-label learning and ensemble learning. In: BMC Bioinformatics, 16(365), pages 1–11.

Zhang, M.-L. and Zhou, Z.-H. (2007). ML-kNN: A lazy learning approach to multi-label learning. In: Pattern Recognition, 40(7), pages 2038–2048.

Zhang, M.-L. and Zhou, Z.-H. (2014). A review on multi-label learning algorithms. In: IEEE Transactions on Knowledge and Data Engineering, 26(8), pages 1819–1837.

Autor

Eduardo Corrêa Gonçalves cursou Doutorado em Ciência da Computação pela Universidade Federal Fluminense (UFF) com período de sanduíche na University of Kent, no Reino Unido. Atualmente, trabalha como desenvolvedor de banco de dados no Instituto Brasileiro de Geografia e Estatística (IBGE) e como professor colaborador na Escola Nacional de Ciências Estatísticas (ENCE/IBGE). Atua principalmente nas seguintes linhas de pesquisa: Big Data, Banco de Dados, Algoritmos e Processamento de Linguagem Natural. Endereço do Currículo Lattes:

http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4137241P3