Representação e Extração de Métricas em Bases de Características: Uma Abordagem Orientada a Objetos

Fausto Neri da Silva Vanin [1]

Resumo: Este artigo apresenta uma abordagem orientada a objetos à leitura e extração de métricas de bases de características. Aspectos estruturais são apresentados inicialmente e, em seguida, os recursos de modelagem da Programação Orientada a Objetos (POO) são utilizados para representar os principais elementos presentes neste contexto. Questões de implementação deste modelo são discutidas e uma aplicação em C++ é apresentada. Para validar o modelo é feita sua aplicação sobre algumas bases da University of Carolina, Irvine Machine Learning Database (UCI).

Palavras-chave: Bases de Características, Programação Orientada a Objetos, Aprendizagem de Máquina, UML

Abstract: This article presents an object oriented approach to data reading and metrics extraction from feature bases. Structural issues are first discussed, and then the Object Oriented Programming (OOP) modelling tools are used to represent the main elements in this context. Implementation issues are then discussed and a C++ application is presented. In order to validate the proposed model, it is applied on some feature bases from the University of Carolina, Irvine Machine Learning Database (UCI).

Keywords: Feature Bases, Object Oriented Programing, Machine Learning, UML

1 Introdução

As bases de características (BCs) constituem um papel extremamente importante à Aprendizagem de Máquina (AM) por conterem intrínseco em seus exemplos conhecimento importante sobre o domínio. Em muitos casos o modelo de Representação do Conhecimento (RC) é obtido destas bases por técnicas de aprendizagem supervisionada, não supervisionada ou de mineração de dados.

A criação de uma BC passa por diversas fases e obedece a um conjunto de procedimentos criteriosamente especificados. Tipos de atributos, valores ausentes, bias e underfitting/overfitting são elementos estruturais que permitem avaliar o quanto uma dada BC é representativa no domínio abordado. Este conhecimento estrutural pode ser obtido pela análise de métricas extraídas dos valores existentes na BC e pode ser utilizado, por exemplo, na redução do espaço dimensional (seleção e extração de características) [13], amostragem, validação cruzada e *bootstrap*[14].

Analisando de uma forma geral, BCs possuem estruturas semelhantes, o que nos permite generalizar estas estruturas por modelos programáticos reutilizáveis (como classes, por exemplo). Da mesma forma, métricas que descrevem estas BCs têm uma grande utilidade, independentemente do contexto no qual o processo de aprendizagem será executado. As abordagens programáticas existentes em AM são voltadas principalmente a linguagens de programação e funcionalidades, tornando difícil especificar estes modelos a contextos específicos. Outra característica marcante é a de que a maioria das abordagens concentra-se em métodos de indução, não abordando a extração de métricas das BCs como uma fase do processo de AM.

Este artigo apresenta um conjunto de modelos abstratos de representação de BCs e extração de métricas baseado em Programação Orientada a Objetos (POO). Estes modelos são apresentados em *Unified Modeling Language* (UML) e implementados utilizando a linguagem de programação C++. Esta abordagem aplica-se ao contexto da AM supervisionada, em BCs com atributos numéricos.

[1]Universidade Tuiuti do Paraná. Faculdade de Ciências Exatas e de Tecnologia. Rua Sidney A. Rangel Santos, 238. CEP 82.010-330. Santo Inácio. Curitiba, PR.
{fausto.vanin@utp.br}

doi: 10.5335/rbca.2010.012

Este artigo está dividido da seguinte forma: inicialmente são apresentados os elementos estruturais básicos em BCs (seção 2); em seguida, na seção 3, são apresentadas métricas úteis à avaliação de BCs; trabalhos relacionados são discutidos na seção 4; na seção 5 é introduzido o modelo abstrato para a solução e as questões de implementação deste modelo na linguagem C++ são apresentadas na seção 6; experimentos com a base de dados UCI são mostrados na seção 7 e, na seção 8, é discutida a abordagem proposta e seus possíveis trabalhos futuros.

2 Bases de Características e a Representação do Conhecimento

Os processos existentes em AM têm como principal objetivo a obtenção de uma RC em um determinado domínio. Este modelo muitas vezes é utilizado como ferramenta para solucionar problemas dentro deste mesmo domínio, o que conhecemos como Reconhecimento de Padrões (RP) [9].

Segundo [20], a AM, dado um contexto, pode ser decomposta nas seguintes fases:

1. Escolha da experiência de treinamento;
2. Definição da função alvo;
3. Forma de representação da função alvo;
4. Escolha de um algoritmo de aproximação desta função;
5. Generalização.

De acordo com essa definição, a criação da BC faz parte da fase de "Escolha da experiência de treinamento" e acontece no início do processo. Por esse motivo, os esforços empreendidos nesta fase terão grande influência nas fases seguintes do processo.

Uma BC é composta por diversos exemplos representativos extraídos do domínio da aplicação. Esses exemplos são apresentados ao **indutor**[2] durante a aprendizagem. Cada exemplo é definido por um conjunto de atributos, que nada mais são do que características descritivas desses exemplos. Em **aprendizagem supervisionada** cada exemplo contém, além dos atributos, uma **categoria** (ou **rótulo**).

Categorias são estados da "natureza" associados a conceitos ou protótipos. Assume-se um conjunto de c categorias denotadas $\omega_i \in \Omega, (i = 1, ..., c)$, onde Ω é o conjunto de todas as categorias, conhecido como **espaço de interpretação**[9]. Um conjunto não ambíguo $\mathbf{A}$ de exemplos de uma determinada categoria forma uma **distribuição**. O vetor $\mathbf{a} = [a_1, ..., a_{n_i}] \in \mathbf{A}$ representa todo os valores de uma determinada característica associada à categoria ω_i, sendo n_i o número de exemplos desta categoria e N representa o total de exemplos da BC.

Tomemos por exemplo um problema de representação de roupas. A Tabela 1 contém um exemplo de BC para este problema.

Tabela 1. Exemplo - Base de Características "Roupas"

Botões	Bolsos	Peso (g)	Categoria
5	1	120.2	Camisa
2	0	80.4	Camiseta
1	4	152.3	Calça

A Tabela 1 está rotulada (coluna `Categoria`), portanto contém uma categoria associada a cada exemplo. Já os outros elementos são as características (ou atributos) que descrevem cada um dos exemplos. Em geral as características podem ser de três tipos:

1. Discreta: números inteiros;

[2]**Indutor**: elemento responsável pela obtenção do modelo de representação do conhecimento a partir da experiência de treinamento.[21]

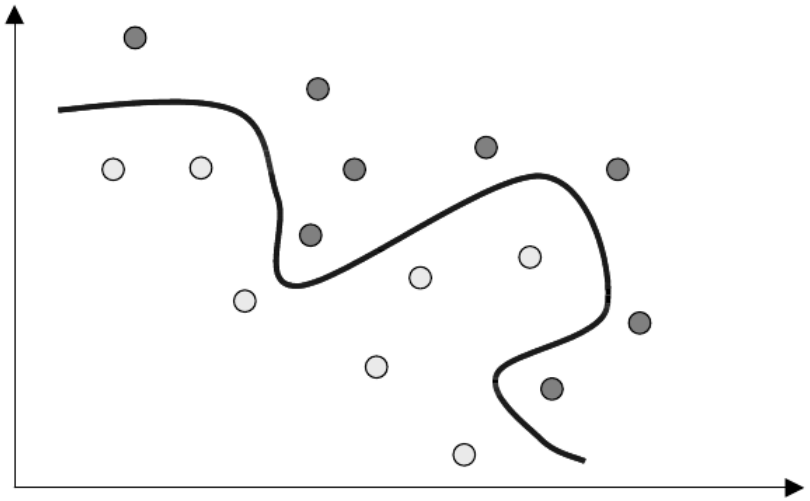


Figura 1. Separação do espaço de características - Fonte: [18]

2. Contínua: números decimais;
3. Simbólica: conjuntos de caracteres.

Cada exemplo ω_i possui associado a si valores para suas d características. Este conjunto de valores é chamado **vetor de características** e é representado por $\mathbf{x} = [x_1, ..., x_d] \in X$, onde X é o domínio d-dimensional dos vetores de características. Este domínio descreve um espaço euclidiano conhecido por **espaço dimensional de características**. Cada exemplo da BC representa um ponto neste espaço dimensional.

Neste caso, a tarefa do indutor é encontrar um modelo de RC capaz de dividir este espaço de características de forma a separar adequadamente as categorias existentes. A Figura 1 apresenta a divisão de um espaço 2-dimensional de características com duas categorias. Muitas vezes, durante o processo de criação da BC, diversas questões devem ser consideradas. Entre elas podemos citar o protocolo experimental, a existência de *bias*, *underfitting*, *overfitting* e ausência de valores.

O **protocolo experimental** nada mais é do que o conjunto predefinido de regras a serem seguidas durante um experimento. Tomando a construção de uma BC como experimento, grande parte do esforço deve ser o de garantir a representatividade das amostras obtidas em relação à população existente.

O ***bias***, ou viés, é justamente o predomínio característico de uma categoria sobre outra na BC. Neste caso, a representatividade das distribuições está desbalanceada, seja pelo tamanho da distribuição, seja pela variância dos exemplos, o que pode provocar um erro estrutural na RC, priorizando uma categoria em detrimento de outra.[21]

No que diz respeito à representatividade da BC em relação ao domínio do conhecimento, ***underfitting*** é o caso em que a variância dos exemplos é tão baixa que não é possível garantir que o modelo de RC será capaz de generalizar uma solução a partir de novos exemplos, ou seja, classificar corretamente uma entrada desconhecida. Já o ***overfitting*** acontece quando os exemplos existentes na base aumentam a complexidade do modelo a ponto de induzi-lo ao erro. O grande desafio é encontrar o ponto onde o indutor é capaz de generalizar o problema adequadamente, minimizando o erro de aprendizagem [18].

A seção 3 apresenta métricas que, ao serem extraídas de BCs, podem prover conhecimento estrutural valioso, que vão desde a análise técnica desses valores até sua aplicação prática em atividades como a redução do espaço dimensional (seleção e extração de características) [13], amostragem, validação cruzada e *bootstrap*[14].

3 Métricas Importantes sobre Bases de Características

Como visto anteriormente, a AM busca modelos de RC que melhor dividam o espaço de características dada uma BC. Para que seja possível projetar, ou até mesmo melhorar, os resultados da aprendizagem é importante

Tabela 2. Notação utilizada

Elemento	Descrição
$\mathbf{X}$	Espaço dimensional de características
$\mathbf{x}$ e $\mathbf{y}$	Vetores de características
ω_i	Uma dada categoria i da BC
$\mathbf{A}$	Conjunto de exemplos de uma categoria ω_i
$\mathbf{a}$	Vetor que contém valores para uma determinada categoria em A
N	Quantidade total de exemplos contidos na BC
n_i	Quantidade de exemplos de uma dada categoria ω_i

ter um bom conhecimento sobre a BC escolhida. Este conhecimento pode ser obtido pela extração de métricas, que irão descrever as categorias existentes no espaço e suas relações de similaridade.

A seguir são apresentadas três diferentes tipos de métricas: 1) globais; 2) por categoria; 3) entre categorias. A notação utilizada para representar as BCs, apresentada na seção 2, está sumarizada na Tabela 2.

3.1 Métricas Globais

Métricas globais dão uma visão geral da BC, relacionando cada distribuição com o todo.

A **distribuição de categorias** avalia a proporção de cada categoria ω_i em relação à quantidade n_i de exemplos. Pode ser representada em termos de um histograma normalizado de c canais, sendo que cada canal i deste histograma possui distribuição dc, como mostrado na equação 1[21].

$$dc(\omega_i) = \frac{n_i}{N} \tag{1}$$

Uma distribuição uniforme das categorias de uma BC denota condições iguais ao indutor no processo de aprendizagem. Já o caso contrário pode denotar a priorização de uma categoria em detrimento de outras, ou seja, a existência de *bias*.

3.2 Métricas por Categoria

Métricas por categoria descrevem individualmente as categorias de uma BC.

O **centro de massa** (ou centroide) é o ponto no espaço de características que representa o núcleo da distribuição de uma categoria ω_i. A equação 2 mostra o cálculo do centroide para uma característica a_j, onde j representa cada um dos n_i exemplos e n_i, o conjunto dos exemplos da categoria ω_i.

$$cm(\mathbf{a}) = \frac{1}{n_i} \sum_{j=1}^{n_i} a_j \tag{2}$$

Medidas de dispersão são recursos estatísticos que permitem avaliar o quanto os exemplos de uma categoria estão "espalhados" no espaço de características. Essas métricas, basicamente, contabilizam a distância entre os n_i exemplos de uma categoria ω_i em relação ao seu centroide $cm(\mathbf{a})$. Entre as principais técnicas podemos citar o desvio médio, desvio padrão, e a variância, dada pela equação 3[24].

$$\sigma^2(\mathbf{a}) = \frac{1}{n_i} \sum_{j=1}^{n_i} (a_j - cm(\mathbf{a}))^2 \tag{3}$$

Se assumirmos que para um dado atributo os exemplos descrevem uma distribuição simétrica[3] ω_i no espaço de características $\mathbf{X}$, esta área pode ser representada por uma esfera de diâmetro igual à variância $\sigma^2(\mathbf{a})$ em torno do centroide $cm(\mathbf{a})$. Se tomarmos duas categorias ω_i e ω_j, a existência de sobreposição entre essas áreas pode denotar ambiguidade, ou seja, o dado atributo não é capaz de separar as categorias em questão.

3.3 Métricas entre Categorias

Para obtermos métricas entre as categorias existentes podemos utilizar algumas medidas de **distância**. Segundo [9], uma distância $d(\mathbf{x}, \mathbf{y})$ entre dois vetores $\mathbf{x}$ e $\mathbf{y}$ é considerada uma métrica se respeitar as seguintes propriedades:

$$d(\mathbf{x}, \mathbf{y}) \geq d_0; \tag{4}$$

$$d(\mathbf{x}, \mathbf{y}) = d_0 \quad \text{se e somente se} \quad x = y; \tag{5}$$

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \tag{6}$$

$$d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}) \tag{7}$$

A mais comum dessas distâncias é a norma euclidiana dada pela equação 8. Além desta existem distâncias como a quarteirão, por exemplo (equação 9) [9].

$$\| \mathbf{x} - \mathbf{y} \|_e = \sqrt{\sum_{i=0}^{d} (x_i - y_i)^2} \tag{8}$$

$$\| \mathbf{x} - \mathbf{y} \|_c = \sum |x_i - y_i| \tag{9}$$

4 Trabalhos Relacionados

Em [26] discute-se a forma como soluções computacionais são apresentadas em publicações científicas. Muitas vezes se faz uso de metalinguagens para representar algoritmos, porém o caráter informal desta representação pode induzir os autores a erro ou à omissão de partes importantes em suas soluções[26]. Nota-se que o mesmo se aplica a modelos programáticos abstratos, que, mesmo quando não acompanham algoritmos, servem como recurso útil à comunidade científica.

A biblioteca MLC++ [17] abstrai diferentes classificadores em AM usando POO e C++ e permite representar atributos numéricos. As estruturas de dados utilizadas para representar as coleções são específicas da biblioteca, ou seja, não oferecem compatibilidade com outros formatos suportados por bibliotecas padrão da linguagem. Não possuem recursos para extração de métricas, mas implementam uma estimativa de *bias*-variância para validação cruzada.

Em [6] é apresentada a linguagem de POO Lush, voltada a implementações científicas. A linguagem é similar ao Lisp, mas permite que seu código seja embutido em algoritmos C/C++. Oferece compatibilidade com uma série de bibliotecas gráficas (OpenGL, GLUT, OpenGLU, OpenRM), numéricas (*Gnu Scientific Library* (GSL), LAPACK e BLAS), de AM, processamento de imagens, áudio e vídeo. A biblioteca para AM, chamada GBLearn2, modela o processo de indução para alguns classificadores, como redes neurais, redes de convolução e funções de base radial. Possui uma representação de BC, mas não há abordagem à extração de métricas destas BCs. Essas métricas podem ser extraídas pela utilização de uma das bibliotecas de análise numérica suportada.

Em [1] é apresentada a biblioteca JavaML, voltada à AM e que utiliza a POO da linguagem Java. Suporta algoritmos de classificação e agrupamento (*clustering*), seleção de características, diversas métricas de distância e correlação, tratamento de valores ausentes e normalização de dados, amostragem, validação cruzada e *bootstrapping*. Não extrai métricas de dispersão, nem calcula a distribuição de categorias.

[3] Para representar distribuições multinormais recomenda-se o uso da distância de Mahalanobis[9]

Além dos trabalhos detalhados aqui existem diversas outras abordagens relacionadas, como as bibliotecas SHOGUN[23], Dlib-ml[16], PRTools[12], InfoSel++[15], entre outras. Uma característica comum a todos esses trabalhos é o forte direcionamento a métodos e linguagens. Alguns trabalhos possibilitam suporte a mais de uma linguagem, mas possuem modelos difíceis de estender. Muito se deve ao fato de essas bibliotecas serem construídas para serem apenas utilizadas; em muitos casos, os pesquisadores precisam desenvolver modelos mais específicos, o que implica dominar o modelo adotado a ponto de estendê-lo de acordo com as especifidades do domínio de aplicação. Outra característica marcante é a de que a maioria das abordagens concentra-se em métodos de indução, não abordando a extração de métricas das BCs como uma fase do processo de AM.

Por esses motivos percebe-se a importância de uma abordagem que aborde as questões estruturais da AM, especialmente em BCs, de modo a prover modelos reutilizáveis, para que, em casos práticos, pesquisadores possam acrescentar novas funcionalidades ao modelo, relegando as questões de implementação para fases posteriores do trabalho.

5 Modelo Proposto

A solução aqui apresentada é um modelo abstrato reutilizável à extração de métricas em BCs. Utilizou-se o paradigma de POO como abordagem ao problema pelos benefícios que este paradigma pode trazer às soluções computacionais nesse contexto. Entre esses benefícios destacam-se a reutilização de código, organização hierárquica de tipos de dados e a abstração de processos.

O modelo proposto tem como prioridade explorar os seguintes itens:

1. Representar categorias e seus atributos;
2. Representar atributos e seus tipos;
3. Representar a estrutura hierárquica das BCs;
4. Extração de métricas da base (globais, por categoria e entre categorias);
5. Leitura de fontes de dados (arquivos, bancos de dados, etc.).

As próximas seções descrevem as seguintes itens da metodologia: a) modelo de alto nível; b) leitura dos dados; c) extração de métricas.

5.1 Modelagem de Alto Nível

Em um nível de abstração mais alto, podemos identificar os principais elementos presentes na estrutura de uma base de carcterísticas como sendo: valor, tipo, atributo, categoria, exemplo e a própria base. Esses elementos se relacionam por uma relação de todo/parte, do mais específico para o mais abrangente, respectivamente. A Figura 2 apresenta uma representação[4] desta estrutura. Esta parte do modelo é composta, basicamente, por:

- Classe abstrata `Valor`;
- Enumeração `Tipo` que possui três valores: `Discreto`, `Continuo` e `Simbolico`;
- Classe `Atributo` com a composição de `Tipo` e `Valor` em um relacionamento `1-1`. Desta classe podemos derivar uma classe `Rotulo`, que representa o atributo especial de rotulação de cada exemplo;
- Classe `Exemplo` composta de `Atributo` (`1-*`) e `Rotulo` (`1-1`);
- Classe `Base` composta de `Exemplo` (`1-*`).

[4]**Representação UML**: todos os diagramas aqui mostrados seguirão as orientações da *Unified Modeling Language*. Para mais sobre o assunto pode-se consultar [3, 5, 19].

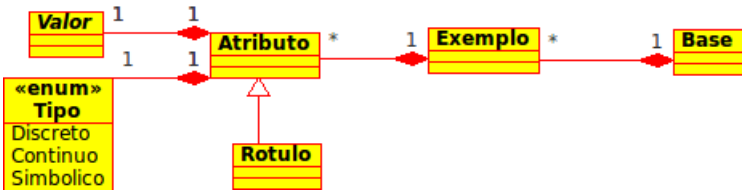


Figura 2. Diagrama de classe - Estrutura Proposta

5.2 Leitura dos dados

A estrutura do arquivo deve ser conhecida para que, posteriormente, esses dados possam ser lidos corretamente. Entre as diversas formas possíveis de representação desses descritores[5] podem-se citar arquivos CSV (*Comma-Separated Values*) e XML (*eXtensible Markup Language*).

Define-se, então, uma classe abstrata `Metadados` para dar acesso às descrições da base em suas classes herdeiras (`MetadadosCSV`, `MetadadosXML` e outras mais). Esta classe contém objetos `Atributo` na ordem em que devem ser lidos da base e compõe a classe `Base` em um relacionamento `1-1`, conforme apresentado na Figura 3.

O formato CSV pode conter em cada linha dados como ordem, nome do atributo e tipo de dado, como no exemplo abaixo:

```
1, qtd_botoes, discreto
2, qtd_bolsos, discreto
3, peso, continuo
4, tipo, rotulo
```

Já no caso do XML pode-se ter uma estrutura hierárquica mais complexa - consequentemente mais rica -, como mostra o exemplo abaixo:

```
<base nome="Roupas">
    <atributo ordem="1" nome="qtd_botoes" tipo="discreto"/>
    <atributo ordem="2" nome="qtd_bolsos" tipo="discreto"/>
    <atributo ordem="3" nome="peso" tipo="continuo"/>
    <rotulo ordem="4"/>
</base>
```

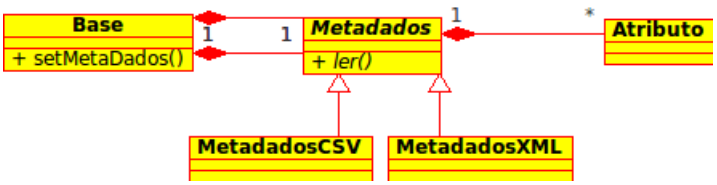


Figura 3. Diagrama de classe - Metadados

5.3 Extração de Métricas

No que diz respeito à extração das métricas, estas podem ser concebidas como métodos da classe `Base` (Figura 4), conforme a lista abaixo:

[5]Metadados: Dependendo do contexto em que se irá trabalhar mais ou menos metadados serão necessários. A linguagem PMML (*Predictive Model Markup Language* [10] é um exemplo de modelo de metadados mais completo do que os aqui mencionados.

- `getDistribuicao(Rotulo)`: retorna um valor entre 0 e 1 correspondente à distribuição da categoria dada;
- `getCentroide(Rotulo)`: retorna um objeto `Exemplo`, que corresponde ao centro de massa para a categoria dada;
- `getDispersao(Rotulo, Dispersao)`: dado um valor da enumeração `Dispersao`, retorna a métrica para uma dada categoria;
- `getDistancia(Rotulo, Rotulo, Distancia)`: dado um valor da enumeração `Distancia`, retorna a distância entre os elementos de duas categorias;

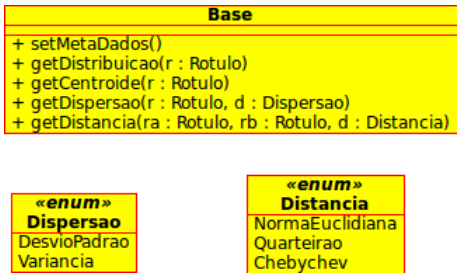


Figura 4. Diagrama de classe - Métricas

Dessa forma, de acordo com o escopo proposto, o diagrama de classe com todos os tipos propostos fica conforme a Figura 5.

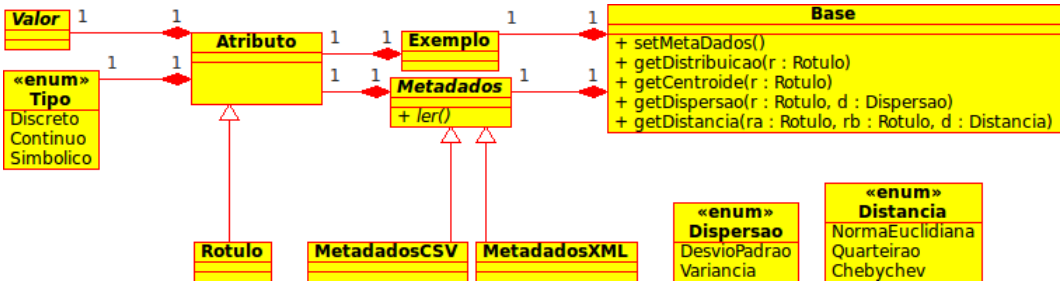


Figura 5. Diagrama de classe - Global

O modelo aqui proposto, por ser uma representação formal abstrata, pode ser implementado em qualquer linguagem de POO. Analisam-se as questões de implementação segundo as características da linguagem C++[25, 11, 7] na seção 6.

6 Implementação do Modelo

Entre os principais recursos e características da linguagem C++ os que mais merecem destaque na implementação desta solução são:

1. Sobrecarga de operador: permite que o cálculo das métricas seja executado usando operadores aritméticos padrão sobre os objetos criados e oferece conformidade com as bibliotecas padrão da linguagem;
2. *Template*: reduz o esforço de codificação pela criação de métodos independentes de tipo;
3. STL (*Standard Template Library*): esta biblioteca padrão de *templates* possui diversas estruturas de dados (vetores, listas, mapas, filas, etc.) implementadas e faz o gerenciamento de memória necessária ao armazenamento dos valores. Essas estruturas de dados são utilizadas para compor as coleções de objetos na relações de composição existentes no modelo;

4. Biblioteca `algorithm`: oferece uma série de métodos úteis para operar com coleções;
5. Biblioteca `iostream`: define diversas classes para operar com *buffers* de dados, sejam estes em memória ou arquivos;
6. Biblioteca `functional`: define elementos úteis para encapsular chamadas de funções.

Para a leitura de dados no formato CSV (classe `MetadadosCSV`) utilizou-se o método `getline` da classe `fstream` (permite definir um delimitador) combinado com *template*, como mostrado na porção de código a seguir. Note-se o uso da classe `stringstream` para fazer a conversão de dados para o tipo desejado (sobrecarga dos operadores << e >>).

```
 1 class MetadadosCSV : public Metadados {
 2         private:
 3                 std::ifstream arquivo;
 4                 template<typename T>
 5                 void lerDado(T &valor) {
 6                         char linha[256];
 7                         arquivo.getline(linha, 256, ',');
 8                         std::stringstream ss;
 9                         ss << linha;
10                         ss >> valor;
11                 }
12 (...)
13 }
```

A chamada deste método `lerDado` é feita na sobrescrita do método `ler` definido na classe pai Metadados, como mostrado abaixo:

```
 1 void ler() {
 2     unsigned ordem;
 3     std::string nomeAtributo, tipoAtributo;
 4
 5     while(!arquivo.eof()) {
 6         lerDado(ordem);
 7         lerDado(nomeAtributo);
 8         lerDado(tipoAtributo);
 9
10         Tipo tipo;
11         if(tipoAtributo == "discreto")  tipo = Discreto;
12         if(tipoAtributo == "continuo")  tipo = Continuo;
13         if(tipoAtributo == "simbolico") tipo = Simbolico;
14
15         Atributo atributo(nomeAtributo, tipo);
16             listaAtributo.push_back(atributo);
17         }
18         arquivo.close();
19 }
```

O objeto `listaAtributo` é um membro da classe `Metadados` e representa uma coleção de dados STL (`list` ou `vector`, por exemplo).

Para fazer a extração das métricas podem-se utilizar os métodos da biblioteca `algorithm`.

A extração da **distribuição de categorias** é feita utilizando-se o método `size_type count(first, last, value)`, que percorre a coleção do início ao fim contando quantos objetos são iguais a `value`, como mostrado a seguir:

```
1 double getDistribuicao(Rotulo r) {
2     return count(listaExemplo.begin(), listaExemplo.end(), r)
3         /listaExemplo.size();
4 }
```

Para que essa comparação funcione corretamente é necessário que as classes `Exemplo` e `Rotulo` sobrecarreguem o operador de comparação ==, a primeira repassando a chamada para a segunda, como mostrado abaixo:

```
 1 class Exemplo {
 2     private:
 3         Rotulo rotulo;
 4 (...)
 5     public:
 3         bool operator==(Rotulo &r) {
 4             return rotulo == r;
 5         }
 6 (...)
 7 };
 8 class Rotulo : public Atributo {
 9         private:
10             std::string nome;
11 (...)
 9         public:
10                 bool operator==(Rotulo &r) {
11                         return nome == r.nome;
12                 }
13 };
```

O **cálculo do centroide** opera apenas com exemplos de uma determinada categoria. O código abaixo implementa a métrica utilizando *template* para os valores de entrada e de retorno, respectivamente `R` e `T`:

```
1 template<typename T, typename R=T>
2 R &mean(T first, T last) {
3     R *m = new R();
4     unsigned n=0;
5     while(first != last) {
6         *m += *first;
7         ++first;
8         ++n;
9     }
10    *m /= n;
11    return *m;
12 }
```

Para que esta função funcione adequadamente é necessário que a classe `Exemplo` e, consequentemente, as classes `Atributo` e `Valor` sobrecarreguem os operadores += e /=. Considerando que `listaR` é um objeto `list<Exemplo>` com exemplos de uma determinada categoria, a obtenção do centroide pode ser representada por:

```
1 Exemplo centroide =
2    mean<list<Exemplo>::iterator, Exemplo>(
3        listaR.begin(), listaR.end()
4    );
```

No cálculo das **medidas de dispersão** é extraído um valor contínuo (aqui `double`) de um grupo de exemplos. Para tanto, é necessário que as classes `Atributo` e `Valor` implementem um método de conversão para `double`. O cálculo da **variância**, portanto, pode ser feito sobre coleções usando *template*, como segue:

```
1 template<typename T>
2 double variance(T first, T last) {
3     double v, m = mean<T, double>(first, last);
4     while(first != last) {
5         v += ( *first - m ) * ( *first - m );
6     }
7     return v;
8 }
```

A **norma euclidiana**, da mesma forma, pode ser concebida independente de tipo para oferecer compatibilidade com diferentes tipos de coleções, como mostrado abaixo:

```
38 template<typename T>
39 double euclidian_norm(T firstA, T firstB, unsigned size) {
40     double en=0, a, b;
41     for(int i=0; i<size; i++) {
42         a = *firstA;
43         b = *firstB;
44         en += fabs(a-b);
45         ++firstA;
46         ++firstB;
47     }
48     return en/size;
49 }
```

A aplicação prática desta implementação dependerá do tipo de BC a ser lida. Portanto, as particularidades do formato a ser lido podem ser implementadas em uma classe herdeira da classe `Base`. A seção 7 apresenta a aplicação deste modelo.

7 Experimentos Realizados e Resultados Obtidos

Para validar o modelo proposto foi feita uma implementação para ler BCs da *UCI Machine Learning Repository*[2]. Foram escolhidas quatro bases para extração das métricas e implementada uma classe chamada `BaseUCI`, que especifica a classe `Base` e implementa métodos de leitura dos arquivos. As bases a seguir foram escolhidas pelo fato de não terem valores ausentes e possuírem apenas atributos numéricos.

1. *Wine Quality*[8];
2. *Breast Cancer*[27];
3. *Iris*[2];
4. *Glass*[2].

Os experimentos consistiram em extrair as métricas de cada BC para fazer uma análise técnica projetando como este conhecimento pode ser utilizado no processo de indução. A seguir são apresentadas as bases e suas respectivas análises.

7.1 Base Glass

A base Glass possui um total de sete categorias diferentes de vidros e suas respectivas características. A Figura 6 mostra que a **distribuição das categorias** na base não é equilibrada, o que pode significar a existência de um ***bias*** para as categorias 1 e 2.

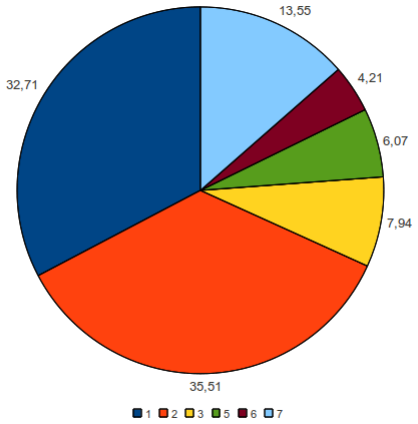
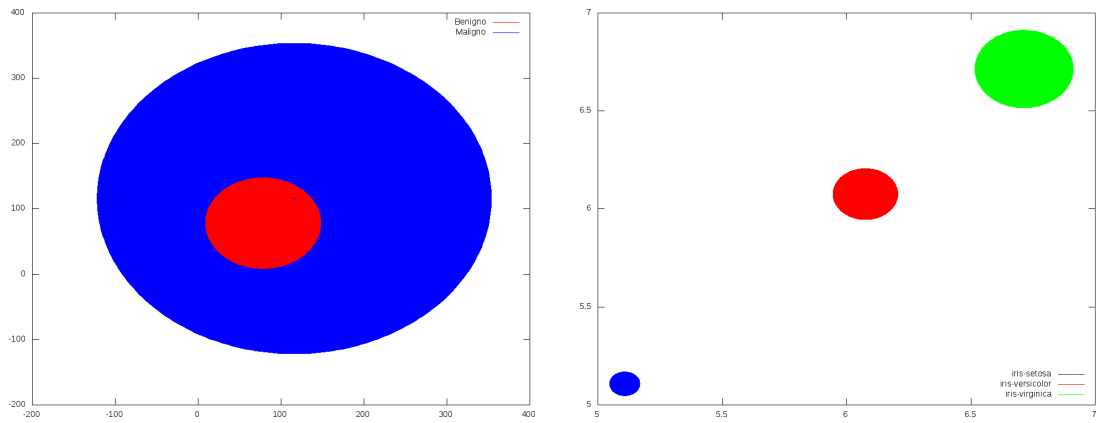


Figura 6. Distribuição de categorias - Base Glass

Um futuro processo de amostragem nesta base deve levar em consideração esta distribuição não uniforme, pois categorias minoritárias podem ficar acidentalmente fora do grupo amostral.

7.2 Bases Breast Cancer e Iris



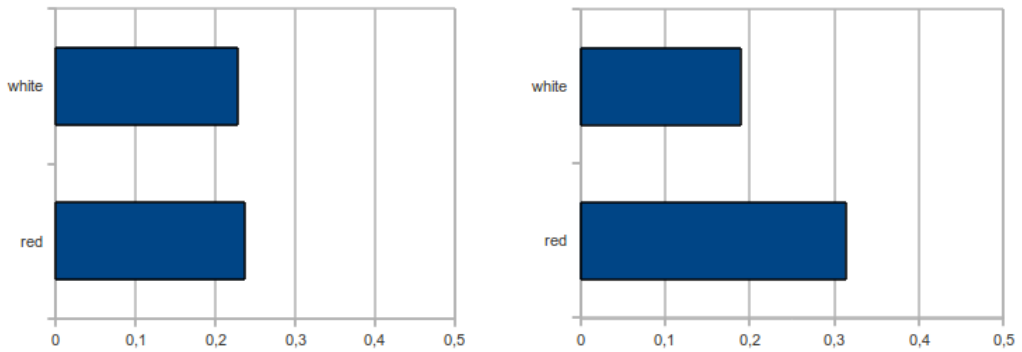
(a) Variância atributo perimeter - Base Breast Cancer

(b) Variância atributo sepal_length - Base Iris

Figura 7. Variância de Atributos

Com as métricas extraídas das bases *Breast Cancer* (atributo `perimeter`) e *Iris* (atributo `sepal_length`) podem-se ter dois exemplos de atributos que possuem áreas de **sobreposição** (conforme seção 3) e que não as possuem, apresentadas nas figuras 7(a) e 7(b), respectivamente. Esta área de interseção significa que elementos com os mesmos valores para este atributo podem ser associados a categorias diferentes, ou seja, há **ambiguidade** nos valores existentes.

No caso da base *Breast Cancer* (Figura 7(a)) o atributo não é capaz de separar tumores malignos de tumores benignos; ao passo que no caso da base *Iris* (Figura 7(b)) não existe ambiguidade entre os atributos.



(a) Normalização pH (0-14) (b) Normalização pH (3-4)

Figura 8. Normalização atributo pH - Base Wine Quality

7.3 Base Wine Quality

Da base *Wine Quality* pode-se analisar o centroide do atributo `pH` utilizando uma **normalização de dados**. Este atributo em particular possui valores contínuos, que, na prática, estão dentro de uma escala definida. O pH, ou pontencial de hidrogênio, define soluções como sendo ácidas ou básicas. Os valores para soluções em geral variam de 0 (soluções ácidas) a 14 (soluções básicas)[22]. Já no contexto específico da vinicultura, a variação de pH vai de 3 a 4, conforme [4].

Dessa forma, esses valores foram normalizados segundo as duas escalas. As Figuras 8(a) e 8(b) apresentam as diferenças desses valores para pH entre 0 e 14 e entre 3 e 4, respectivamente. É facilmente perceptível como o **conhecimento do domínio** da aplicação pode influenciar positivamente na análise dos dados neste caso.

8 Conclusões e Trabalhos Futuros

Este artigo apresenta um modelo orientado a objetos à representação de BCs. O modelo contempla a extração de métricas a serem obtidas de atributos numéricos. Uma implementação em C++ do modelo foi criada e aplicada a algumas bases UCI. A partir desses resultados pode-se analisar esta abordagem por diversos aspectos.

O modelo proposto explora as relações todo/parte existentes nas estruturas das BCs. Representar separadamente tipos, atributos e exemplos possibilita uma organização hierárquica dos dados. Essa característica mostra-se vantajosa em comparação a outras abordagens, pois permite que o modelo seja facilmente especificado de acordo com o domínio de aplicação, não restringindo sua aplicação a um determinado grupo de funcionalidades ou linguagens de programação.

O suporte à especificação de metadados possibilita definir, além das característica existentes na base, possíveis relações hierárquicas entre elas e, também, questões estruturais, como a normalização de dados e o tratamento de valores ausentes, por exemplo. Este modelo pode ser implementado em qualquer linguagem de POO, como Java ou C#, por exemplo. Neste caso optou-se pela linguagem C++.

Diversos recursos da linguagem C++ facilitam o trabalho de codificação, como a sobrecarga de operdor e o uso de *template*. Dessa forma, foi possível criar implementações independentes de tipos, que podem, inclusive, operar com dados que não sejam oriundos do modelo proposto, como é o caso dos métodos para cálculo da média, variância e norma euclidiana, que requerem apenas que os objetos a serem utilizados sobrecarreguem os operadores `+=` e `/=`. A biblioteca STL também é um recurso que facilita a organização da massa de dados, possuindo funções específicas para operar com essas coleções.

A aplicação do modelo sobre um conjunto de BCs permitiu avaliar diversos itens. O esforço de codificação necessário para estender o modelo concentra-se especialmente na leitura dos dados para popular as coleções de objetos. Dessa forma, é possível criar classes novas para cada formato de BC. A possibilidade de lançar visões transversais sobre os dados lidos permite segmentar esses dados por rótulo, por atributo, ou até mesmo por uma determinada métrica, o que facilita a auditoria de dados. O conjunto de métricas obtidas também permite analisar

a BC de forma global (distribuição de categorias), por categoria (medidas de dispersão) ou entre as categorias (cálculo de distâncias).

Como trabalhos futuros este modelo dá a possibilidade de diversas aplicações, algumas citadas abaixo:

1. Operar com atributos simbólicos, extraindo métricas como ganho de informação, por exemplo;
2. Suportar operações de amostragem visando segmentar a base (treinamento e teste, por exemplo). Neste caso as características de distribuição e dispersão podem ser utilizadas para avaliar a representatividade de cada segmento obtido em relação à base toda;
3. Suportar técnicas de seleção e extração de características, importantes também à análise de BCs.

Referências

[1] Thomas Abeel, Yves Van de Peer, and Yvan Saeys. Java-ml: A machine learning library. *Journal of Machine Learning Research*, 10:931–934, April 2009.

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] Michael Blaha and James Rumbaugh. *Object Oriented Modeling and Design with UML*. Pearson Prentice Hall, 1991.

[4] Tereza Cristina Blasi. *Análise do consumo e constituintes químicos de vinhos produzidos na quarta colônia de imigração italiana do Rio Grande do Sul e sua relação com as frações lipídicas sanguineas*. PhD thesis, Programa de Pós-Graduação em Ciência e Tecnologia dos Alimentos da Universidade Federal de Santa Maria, 2004.

[5] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, May 2005.

[6] Léon Bottou and Yann LeCun. Lush: Reference manual. Technical report, NEC Laboratories America, January 2003.

[7] Frank B. Brokken and Karel Kubat Until Version. C++ annotations version 4.4.2. online: http://sourceforge.net/projects/cppannotations/, 2001.

[8] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, June 2009.

[9] J. P. Marques de Sá. *Pattern Recognition: concepts, methods and applications*. Springer, 2001.

[10] Data Mining Group DMG. Predictive modeling markup language. On line, 8 2008.

[11] Adam Drozdek. *Estruturas de dados e algoritmos em C++*. Thomson Learning, São Paulo, 2005.

[12] Robert P. W. Duin, P. Juszczak, D. de Ridder, Pavel Paclik, E. Pekalska, and D.M.J. Tax. Prtools, a matlab toolbox for pattern recognition, 2004.

[13] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003.

[14] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, corrected edition, July 2008.

[15] Adam Kachel, Jacek Biesiada, Marcin Blachnik, and Wlodzislaw Duch. Infosel++: Information based feature selection c++ library. In Leszek Rutkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada, editors, *ICAISC (1)*, volume 6113 of *Lecture Notes in Computer Science*, pages 388–396. Springer, 2010.

[16] Davis E. King. Dlib-ml: A machine learning toolkit. *J. Mach. Learn. Res.*, 10:1755–1758, 2009.

[17] Ron Kohavi, Dan Sommerfield, and James Dougherty. Data mining using mlc++, a machine learning library in c++. In *ICTAI '96: Proceedings of the 8th International Conference on Tools with Artificial Intelligence*, page 234, Washington, DC, USA, 1996. IEEE Computer Society.

[18] Daniel T. Larose. *Discovering Knowledge in Data: an introduction to data mining*. John Wiley & Sons, New Jersey, 2005.

[19] Richard C. Lee and William M. Tepfenhart. *Uml and C++: A Practical Guide to Object-Oriented Development*. Pearson, 2nd edition, 2002.

[20] Tom M. Mitchel. *Machine Learning*. Mc-Graw Hill, 1997.

[21] Solange Oliveira Rezende. *Sistemas inteligentes : fundamentos e aplicações*. Manole, Barueri, SP, 2003.

[22] Carlos A. Richter and José Martiniano de Azevedo Netto. *Tratamento de água : tecnologia atualizada*. Edgard Blucher, São Paulo, 1991.

[23] Sören Sonnenburg, Gunnar Rätsch, Sebastian Henschel, Christian Widmer, Jonas Behr, Alexander Zien, Fabio de Bona, Alexander Binder, Christian Gehl, and Vojtech Franc. The SHOGUN machine learning toolbox. *Journal of Machine Learning Research*, 11:1799–1802, June 2010.

[24] Murray Ralph Spiegel and Pedro Cosentino. *Estatística*. Makron Books, São Paulo, SP, 1993.

[25] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, third edition, June 1997.

[26] H. Thimbleby. Explaining code for publication. *Software — Practice & Experience*, 33(10):975–1001, 2003.

[27] W. H. Wolberg and O. L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences of the United States of America*, 87(23):9193–9196, December 1990.