

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA  
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GUILHERME HAAG RIBACKI

**Um framework para agrupamento de dados**

Trabalho de Graduação.

Prof. Dr. Leandro Krug Wives  
Orientador

Porto Alegre, janeiro de 2013.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. Carlos Alexandre Netto

Vice-Reitor: Prof. Rui Vicente Oppermann

Pró-Reitor de Graduação: Prof. Sérgio Roberto Kieling Franco

Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb

Coordenador do CIC: Raul Fernando Weber

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro



## **AGRADECIMENTOS**

Gostaria de agradecer à minha família por todo apoio que sempre recebi, principalmente durante o desenvolvimento deste trabalho; à Aline Lermen por ter estado presente durante toda a graduação, me ajudando e compartilhando desta aventura; aos meus amigos por jamais me abandonarem, mesmo nos momentos complicados do curso que exigiram um afastamento maior de minha parte; aos meus colegas e amigos de curso por terem tornado essa experiência mais interessante de diversas formas; e aos professores que mostraram dedicação excepcional à tarefa de ensinar e aprender com seus alunos.

Por último, gostaria de agradecer ao meu orientador por todo apoio e auxílio no desenvolvimento deste trabalho, sem jamais permitir que eu perdesse a motivação.

# SUMÁRIO

<b>AGRADECIMENTOS</b> .....	<b>4</b>
<b>SUMÁRIO</b> .....	<b>5</b>
<b>LISTA DE ABREVIATURAS E SIGLAS</b> .....	<b>7</b>
<b>LISTA DE FIGURAS</b> .....	<b>8</b>
<b>LISTA DE TABELAS</b> .....	<b>9</b>
<b>RESUMO</b> .....	<b>10</b>
<b>ABSTRACT</b> .....	<b>11</b>
<b>1 INTRODUÇÃO</b> .....	<b>12</b>
<b>2 AGRUPAMENTO DE DADOS</b> .....	<b>14</b>
<b>2.1 Definição</b> .....	<b>14</b>
<b>2.2 Etapas do processo de agrupamento</b> .....	<b>15</b>
2.2.1 Seleção de características .....	16
2.2.2 Cálculo de similaridade .....	16
2.2.3 Identificação de aglomerados .....	17
2.2.3.1 <i>Agrupamento por partição total</i> .....	18
2.2.3.2 <i>Agrupamento hierárquico</i> .....	19
2.2.4 Avaliação dos resultados .....	19
<b>2.3 Algoritmos de agrupamento</b> .....	<b>21</b>
2.3.1 Cliques .....	21
2.3.2 <i>Single Link (connected components)</i> .....	22
2.3.3 <i>Stars</i> .....	22
2.3.4 <i>Best-star e Full-stars</i> .....	23
2.3.5 <i>Strings</i> .....	23
2.3.6 K-médias.....	24
2.3.7 DBSCAN ( <i>Density-Based Spatial Clustering of Applications with Noise</i> ) .....	24
<b>2.4 Trabalhos relacionados</b> .....	<b>25</b>
<b>3 PROPOSTA DE FRAMEWORK PARA AGRUPAMENTO DE DADOS</b> .....	<b>26</b>

<b>3.1 Frameworks .....</b>	<b>26</b>
<b>3.2 Padrões de projeto .....</b>	<b>27</b>
<b>3.3 A Arquitetura.....</b>	<b>28</b>
3.3.1 Objetos e suas características .....	28
3.3.2 O processo e seus passos .....	29
3.3.3 Os observadores.....	31
<b>4 INSTANCIÇÃO DO FRAMEWORK PARA O AGRUPAMENTO DE DOCUMENTOS.....</b>	<b>35</b>
<b>4.1 Agrupamento de documentos usando o <i>framework</i> proposto.....</b>	<b>35</b>
4.1.1 Os documentos e seus termos .....	36
4.1.2 As estratégias .....	36
4.1.3 Exibindo progresso e resultados .....	37
<b>4.2 Resultados .....</b>	<b>39</b>
4.2.1 Coleção de artigos da Wikipedia.....	39
4.2.2 Subconjunto de documentos da coleção Reuters-21578 .....	41
<b>5 CONCLUSÃO.....</b>	<b>43</b>
<b>REFERÊNCIAS .....</b>	<b>45</b>
<b>APÊNDICE: UTILIZAÇÃO DO FRAMEWORK .....</b>	<b>47</b>

## **LISTA DE ABREVIATURAS E SIGLAS**

GSM	Grau de Similaridade Mínima
API	Application Programming Interface (Interface de Programação de Aplicativo)

## LISTA DE FIGURAS

Figura 3.1: Comparativo entre uso de <i>framework</i> e bibliotecas para o desenvolvimento de uma aplicação .....	28
Figura 3.2: Diagrama de classes com os conceitos básicos.....	30
Figura 3.3: Diagrama de classes exemplificando o <i>design pattern Strategy</i> .....	31
Figura 3.4: Versão simplificada do diagrama de classes das estratégias de agrupamento (sem os métodos).....	32
Figura 3.5: Diagrama da relação entre <i>ClusteringProcess</i> e as estratégias .....	32
Figura 3.6: Diagrama de classes exemplificando o <i>design pattern Observer</i> .....	33
Figura 3.7: Diagrama de sequência da troca de mensagens entre <i>subject</i> e <i>observers</i> ...	34
Figura 3.8: Diagrama de sequência simplificado da relação entre uma classe observadora do cliente e as entidades do <i>framework</i> .....	35
Figura 4.1: Visualização dos grupos resultantes na janela da classe <i>Inspector</i> .....	39
Figura 4.2: Gráfico de quantidade de objetos por grupo .....	39
Figura 4.3: Gráfico dos resultados para o algoritmo <i>Best-star</i> .....	41
Figura 4.4: Gráfico dos resultados para os algoritmos <i>Cliques</i> e <i>Star</i> .....	41
Figura 4.5: Gráfico dos resultados para o algoritmo <i>Full-stars</i> .....	42



## **LISTA DE TABELAS**

Tabela 2.1: Matriz de similaridade .....	17
--	----

## RESUMO

Com a evolução tecnológica, cada vez mais se tem acesso a grandes volumes de dados através da Internet. Para que se possa usufruir desses dados, algumas técnicas são aplicadas para extrair informações relevantes em uma determinada busca, sendo uma dessas técnicas o agrupamento de dados. O agrupamento de dados (*data clustering*) é utilizado para criar partições de objetos semelhantes entre si, independente dos tipos desses objetos, para facilitar a recuperação de informação. Este trabalho propõe uma arquitetura de *framework* que, através do uso de padrões de projeto (*design patterns*) e outras práticas da Engenharia de Software, permite que se implementem diferentes técnicas de agrupamento para tipos de dados variados de forma a maximizar o reuso de código. Uma instância desse *framework* é proposta para o agrupamento de textos de forma a demonstrar o uso do *framework* e comparar a eficácia de alguns algoritmos. Uma comparação é feita entre os algoritmos implementados com o *framework* e alguns resultados usando a ferramenta Eureka. Os agrupamentos resultantes são avaliados através de métricas como Medida-F (*F-Measure*) e Silhueta (*Silhouette*). Duas coleções de documentos são usadas nos testes – uma pequena (12 documentos) e uma grande (722 documentos). Os algoritmos *Cliques*, *Stars*, *Full-stars* e *Best-star* foram usados com a coleção pequena, e o último deles se mostrou o mais eficiente. A coleção grande foi testada com esse mesmo algoritmo, porém os resultados, avaliados com a técnica Silhueta, não obtiveram resultados tão bons.

**Palavras-Chave:** agrupamento de dados, análise de agrupamentos, algoritmos, *framework*, padrões de projeto.

## **A framework for data clustering**

### **ABSTRACT**

With the technological evolution, more and more there is access to large data volume through the internet. To be able to use this data, some techniques are used to extract relevant information from a search, one of these techniques being the data clustering. Data clustering is used to create partitions of similar objects, independently of their type, to make it easier to retrieve information. This work propose a framework architecture that, through the use of design patterns and other Software Engineering practices, allows different clustering techniques to be implemented for varying data types, in a way to maximize code reuse. An instance of this framework is proposed for text clustering in a way to show the framework's use and to compare the effectiveness of some algorithms. A comparison is made between the implemented algorithms in the framework and some results of the Eureka tool. The resulting clusters are evaluated through metrics like F-measure and Silhouette. Two collections are used in the tests – a small one (12 documents) and a large one (722 documents). The Cliques, Stars, Full-stars and Best-star algorithms were used with the small one, and the last had the best results. The big collection was tested with this algorithm, but the results, evaluated with the Silhouette technique, didn't reach as good results.

**Keywords:** data clustering, cluster analysis, algorithms, framework, design patterns.

# 1 INTRODUÇÃO

Hoje em dia os usuários de sistemas computacionais possuem acesso a uma quantidade enorme de informações e documentos, seja através da Internet ou de arquivos locais. Isso se deve à evolução tecnológica que permite a aquisição de componentes de hardware para armazenamento de dados muito mais baratos e conexões de Internet muito mais rápidas e acessíveis. Cada vez mais são necessárias ferramentas específicas para encontrar informações a respeito de algum assunto dentre uma coleção de documentos, sejam arquivos de texto ou páginas na Web. Estas informações são dos tipos mais variados e vão desde documentos textuais até imagens, vídeos ou gravações sonoras.

Para resolver os problemas provenientes do acesso a coleções extensas de documentos, diversas áreas acadêmicas se dedicaram ao estudo da análise de agrupamentos de dados. Segundo Jain (1999), análise de agrupamentos (*cluster analysis*) é a organização de uma coleção de padrões em grupos baseando-se na similaridade entre eles. Em se tratando de textos, isso quer dizer que se pode ter grupos onde todos os documentos em um mesmo grupo tratam de um assunto semelhante, podendo-se assim extrair um subconjunto de dados pertinentes para análise mais detalhada.

O processo de agrupamento de dados (*data clustering*) é constituído por passos que podem ser combinados de diversas formas a fim de buscar uma mais eficiente para criar partições de um determinado conjunto de dados. Com o objetivo de facilitar a análise comparativa entre diferentes métodos, Wives (1999) desenvolveu o Eureka, uma ferramenta de análise de agrupamentos de documentos textuais, porém a mesma é incompatível com os sistemas operacionais atuais e não foi projetada prevendo extensões, além de ser limitada ao uso de dados do tipo texto apenas.

Com essas limitações em mente, este trabalho tem como objetivo a especificação de um *framework* de agrupamento de dados flexível, que permita a implementação e combinação de diversos métodos de agrupamento que possam ser aplicados a diferentes tipos de dados, sem se limitar a documentos textuais. O presente trabalho também se propõe a instanciar o *framework* proposto para aplicação com textos, de forma a demonstrar o uso do mesmo em uma aplicação real.

Adicionalmente, este trabalho apresenta um estudo teórico sobre o processo de agrupamento, seus passos, técnicas, métodos existentes e métricas para avaliação dos resultados. Alguns algoritmos são comparados através de implementações na instância do *framework* para agrupamento de textos e os resultados são avaliados utilizando algumas métricas estudadas. Também é feita uma comparação entre os resultados obtidos com a ferramenta implementada e os resultados obtidos com a execução do Eureka.

Para isso, primeiramente são expostos os conceitos básicos de agrupamento de dados e avaliação dos resultados e, logo após, alguns algoritmos de agrupamento. Depois a arquitetura do *framework* é descrita e uma instância dele é definida para agrupamento de dados do tipo texto, onde alguns algoritmos de agrupamento são utilizados. Alguns testes são feitos com os algoritmos implementados e os resultados são discutidos e comparados através das métricas de avaliação estudadas.

## 2 AGRUPAMENTO DE DADOS

Agrupamento de dados, ou análise de agrupamentos (*cluster analysis*), tem como objetivo descobrir o agrupamento natural de conjuntos de padrões, pontos ou objetos (JAIN, 2009). É muito útil em situações de análise exploratória de padrões, de agrupamento, de tomada de decisões e aprendizado de máquina, incluindo mineração de dados, recuperação de documentos, segmentação de imagens e classificação de padrões. Isso se deve ao fato de que nessas situações se possui pouquíssima informação sobre os dados e o tomador de decisões precisa fazer a menor quantidade de suposições possíveis. A natureza da metodologia de agrupamentos permite que se faça uma avaliação, mesmo que preliminar, sobre a estrutura desses dados.

Nas próximas seções serão explicados os conceitos básicos envolvidos no processo de agrupamento de dados, definidos os tipos de algoritmos existentes e explorados alguns dos algoritmos mais conhecidos.

### 2.1 Definição

A análise de agrupamentos é o estudo formal de algoritmos e métodos para o agrupamento, ou classificação, de objetos (JAIN, 1988). Segundo Kowalski (2002), o objetivo do agrupamento de dados é fornecer um conjunto de objetos similares sob um título mais geral, também chamado de “classe”. Porém, para alguns autores, como Rubinov (2005), o conceito de “classe” representa os grupos de dados resultantes da análise supervisionada de um conjunto de dados, portanto, para evitar ambiguidade, o termo “classe” não será usado neste trabalho quando considerado o contexto de análise não supervisionada de dados.

Os grupos resultantes desse processo não possuem qualquer rótulo definido previamente, eles devem ser formados de acordo com o conteúdo da coleção de objetos utilizada, sem a intervenção de um especialista ou usuário. Porém, podem possuir um rótulo identificador chamado “descriptor”, que é determinado após o processo, para representar cada grupo de forma a facilitar a compreensão e a navegação nos resultados pelo usuário.

Segundo Jain (1999), é importante entender a diferença entre agrupamento (classificação não supervisionada) e análise discriminante (classificação supervisionada): no processo de classificação supervisionada tem-se como objetivo identificar a qual classe cada objeto pertence, dentre classes pré-definidas, enquanto que o processo de agrupamento busca separar os documentos em grupos relevantes sem o conhecimento prévio de classes, excluindo a necessidade de se ter conhecimento aprofundado da coleção previamente.

O agrupamento pode ser definido formalmente, segundo Everitt (1974), como:

1. Um aglomerado é um conjunto de entidades similares, e entidades de diferentes aglomerados não são similares;
2. Um aglomerado é uma agregação de pontos no espaço tal que a distância entre dois pontos em um mesmo aglomerado é menor do que a distância entre qualquer ponto do aglomerado e qualquer ponto não pertencente a ele;
3. Aglomerados podem ser descritos como regiões conexas de um espaço multidimensional contendo uma densidade relativamente alta de pontos, separadas umas das outras por uma região contendo uma relativamente baixa densidade de pontos.

## 2.2 Etapas do processo de agrupamento

O processo típico de agrupamento de dados pode ser descrito em cinco passos (JAIN, 1999):

1. Identificação, seleção e/ou extração de características;
2. Cálculo de similaridades;
3. Identificação de aglomerados;
4. Abstração dos dados (caso necessário);
5. Avaliação dos resultados.

O primeiro passo é constituído de três subetapas: as características de cada objeto são identificadas, aquelas que caracterizam melhor o objeto são selecionadas e a relevância de cada uma dentro do contexto é calculada. A relevância pode ser considerada no escopo da coleção inteira ou apenas do objeto, dependendo da técnica utilizada. Como resultados desta etapa, são geradas listas de características de cada objeto.

O segundo passo faz um cálculo de similaridade entre os objetos, utilizando as listas de características identificadas no passo anterior. O resultado desta etapa é uma matriz de duas dimensões, que contém os valores de similaridades entre todos os objetos. Quanto mais características em comum dois objetos possuírem, mais similares eles serão.

O terceiro passo é o agrupamento propriamente dito. Ele consiste em identificar quais objetos são mais similares entre si, a partir da matriz resultante do passo anterior, e reuni-los em grupos. Esta etapa tem como resultado os grupos e seus respectivos elementos, concluindo o processo de agrupamento.

A abstração de dados é o passo em que os grupos de dados recebem uma representação, seja para facilitar a leitura pela máquina para futuros processamentos ou para facilitar o entendimento por parte do usuário. Geralmente é atribuído um rótulo que identifique os objetos dentro do grupo em questão, mas não é um passo obrigatório. Essa etapa não será aprofundada neste trabalho e pode ser explorada em um trabalho futuro.

A avaliação dos resultados é a etapa em que a qualidade dos grupos resultantes é avaliada. Essa etapa pode considerar ou não um fator chamado “*cluster tendency*”, que indica o quanto um conjunto de dados é inerentemente agrupável, ou seja, possui um agrupamento natural. Caso o conjunto de dados de entrada seja completamente variado,

sem similaridades, o resultado da avaliação poderá variar drasticamente de acordo com o algoritmo de agrupamento usado.

Cada etapa ou subetapa é independente uma da outra, exigindo apenas que o resultado de uma esteja em um formato aceitável para servir como dado de entrada da próxima, permitindo que se possa usar diferentes combinações de algoritmos em uma execução do processo. Este é um dos focos da implementação deste trabalho e será melhor explorado adiante.

### 2.2.1 Seleção de características

O primeiro passo do agrupamento de dados é constituído basicamente pela seleção de características de cada objeto para que se possa calcular a similaridade entre eles na etapa seguinte. O processo de seleção de característica inclui três subetapas (JAIN, 1999):

1. Identificação de características;
2. Seleção de características;
3. Extração de características.

A identificação de características visa transformar ou extrair os dados do objeto original, colocando-os em um formato apropriado para que se possa trabalhar com eles. Na área de recuperação de informações textuais, por exemplo, pode-se identificar cada palavra utilizada em um documento como uma característica diferente, e associar a elas uma frequência, que poderá ser usada como critério no cálculo de similaridade. Em processamento de imagens, as características podem ser os pixels, associados a sua a intensidade, matiz e saturação.

A seleção de características é opcional e tem como responsabilidade filtrar as características que poderão contribuir melhor no processo de forma a otimizar os resultados. Como exemplo, tem-se a eliminação de *stop-words* no agrupamento de documentos textuais. *Stop-words* são palavras indesejadas que devem ser eliminadas do cálculo de similaridade por contribuírem de forma negativa. Por exemplo, são consideradas *stop-words* palavras muito frequentes que auxiliam na construção de orações (preposições, conjunções e artigos, por exemplo), aparecem em quase todos os elementos e não ajudam a distinguir o conteúdo dos textos (MANNING, 2008). Outro problema é que os objetos podem possuir muitas características, deixando o processo muito lento. Uma solução consiste em limitar o número de características que irão entrar no cálculo de similaridade, escolhendo as de maior impacto no objeto, ou seja, as de maior peso. No caso do agrupamento de documentos textuais, pode-se escolher as palavras mais frequentes de cada objeto, deixando de fora as menos frequentes.

A extração de características também é opcional e é a ação de transformar uma ou mais características dos dados de entrada para produzir novas características relevantes. O conjunto extraído pode ser menor que o original e ao mesmo tempo representar o objeto de forma eficiente, permitindo que se reduza o tempo total do processo de agrupamento (GUYON, 2006). Por exemplo, a eliminação de características redundantes através da unificação de características similares pode resultar em um conjunto de características menor sem perda de representação.

### 2.2.2 Cálculo de similaridade



A similaridade entre os objetos é geralmente calculada usando-se uma função que analisa todas as características que eles possuem e retorna o grau de semelhança entre eles. Existem várias classes de funções de similaridade, sendo as principais representadas por medidas de distância (WIVES, 1999).

As medidas de distância normalmente consideram cada objeto como um ponto em um espaço euclidiano de  $n$  dimensões, onde  $n$  é o número de características máximas que um objeto pode possuir. Cada característica do objeto é representada por uma componente de coordenada e a similaridade é dada pela proximidade dos objetos neste espaço.

Algumas dessas funções são consideradas binárias, pois utilizam apenas dois valores: 0, caso a característica não esteja presente e 1, caso contrário. Outras funções consideram o quanto uma característica influencia no objeto, e levam em conta quantas vezes ela aparece nele, algumas vezes considerando ainda o número total de objetos em que a característica está presente. Nesse caso os valores são geralmente normalizados no intervalo  $[0,1]$ , onde zero indica que a característica não tem relevância nenhuma para o objeto e um indica que o objeto é fortemente caracterizado pela variável em questão. A forma como esses valores são calculados é chamada de esquema de ponderação, ou *weighting scheme*, e se for escolhida adequadamente pode melhorar a precisão dos resultados.

Alguns algoritmos trabalham em cima de uma matriz de similaridades pré-calculada sobre o conjunto original de dados ao invés de trabalhar em cima dos dados diretamente. Um exemplo de matriz de similaridade pode ser observado na Tabela 2.1. Como a matriz é simétrica, a metade inferior pode ser ignorada.

Tabela 2.1: Matriz de similaridade

	OBJ1	OBJ2	OBJ3	OBJ4	OBJ5
OBJ1	1.0	0.3	0.5	0.2	0.7
OBJ2	0.3	1.0	0.6	0.2	0.3
OBJ3	0.5	0.6	1.0	0.9	0.5
OBJ4	0.2	0.2	0.9	1.0	0.3
OBJ5	0.7	0.3	0.5	0.3	1.0

Fonte: elaborada pelo autor.

O escopo deste trabalho se limita aos algoritmos que utilizam esta matriz de similaridades, pois o *framework* proposto supõe a construção desta matriz no cálculo da similaridade (passo dois) e o seu uso no algoritmo de agrupamento (passo três).

### 2.2.3 Identificação de aglomerados

Os agrupamentos podem ser classificados em dois tipos, dependendo da forma como os grupos são construídos: por partição total ou hierárquica. A forma por partição total produz uma série de grupos dispostos sequencialmente, enquanto a forma hierárquica produz séries aninhadas, em forma de árvore.

Segundo Jain (1999), pode-se ainda classificar os algoritmos de acordo com uma série de outras características: aglomerativos ou divisivos, monotéticos ou politéticos, fortes ou difusos, determinísticos ou estocásticos, incrementais ou não incrementais.

Os algoritmos aglomerativos consideram inicialmente cada objeto em seu próprio grupo e une pares de grupos sucessivamente até que um determinado critério de parada seja satisfeito, enquanto que os algoritmos divisivos consideram todos os objetos inicialmente em um mesmo grupo e vai separando-os em grupos menores até que se atinja o critério de parada.

A maioria dos algoritmos são politéticos, ou seja, consideram todas as características simultaneamente para calcular as distâncias entre os objetos, e as decisões são tomadas levando essas distâncias em consideração. Entretanto, também existem algoritmos monotéticos, que consideram as características individual e sequencialmente para dividir os objetos em grupos.

Os algoritmos também podem ser rígidos ou difusos (*fuzzy*). Um algoritmo de agrupamento rígido aloca cada objeto em apenas um aglomerado distinto, enquanto que um algoritmo de agrupamento difuso atribui graus de associação em vários aglomerados para cada objeto, permitindo que um mesmo objeto possa pertencer a mais de um grupo em graus diferentes. Ao atribuir cada objeto ao seu grupo de maior grau, pode-se converter o agrupamento difuso em um agrupamento forte.

Grande parte dos algoritmos é determinística, porém alguns podem ser estocásticos, isto é, podem utilizar métodos probabilísticos para otimizar os resultados. Exemplos de aplicações de métodos probabilísticos podem ser encontrados em algoritmos de partição total projetados para otimizar funções de erro quadrático (melhor explicados mais adiante).

Algoritmos incrementais consideram lotes limitados de objetos em cada processamento, de forma a otimizar o uso de recursos computacionais limitados, tais como memória ou tempo, e são utilizados nos casos em que a coleção de objetos é grande. Inicialmente não existiam muitos exemplos de algoritmos incrementais, porém a Mineração de Dados incentivou o desenvolvimento de algoritmos de agrupamento que minimizam o número de passadas pelo conjunto de objetos, reduzem o número de objetos examinados durante a execução ou reduzem o tamanho de estruturas de dados utilizadas pelas operações do algoritmo.

### *2.2.3.1 Agrupamento por partição total*

O agrupamento por partição total gera grupos distintos sem nenhuma representação de relação entre eles. Os objetos podem estar em um ou em mais de um grupo, dependendo do algoritmo, mas os aglomerados sempre são isolados e não possuem qualquer tipo de relação entre eles.

Geralmente os objetos são alocados apenas no grupo de maior similaridade. Quando isso acontece, diz-se que o processo é disjuncto. Caso os objetos possam ser atribuídos a mais de um grupo, o processo não é disjuncto, e o agrupamento pode ser difuso, ou seja, os objetos podem possuir graus de associação com cada grupo. Em ambos os casos não existe hierarquia, e os aglomerados são completamente isolados.

A principal desvantagem deste método é que, com a ausência de uma organização hierárquica, o usuário fica impossibilitado de navegar entre um número grande de aglomerados a fim de localizar alguma informação que julgue relevante. A principal vantagem é que ele é executado em apenas um passo, diferente do agrupamento hierárquico, que exige passos iterativos.

### 2.2.3.2 Agrupamento hierárquico

No agrupamento hierárquico os objetos são analisados recursivamente a fim de identificar as relações entre os diferentes aglomerados. Para alcançar este resultado, pode-se usar algoritmos específicos de agrupamento hierárquico ou algoritmos de agrupamento por partição total aplicados recursivamente.

O agrupamento hierárquico pode ter duas formas: aglomerativa ou divisiva (WIVES, 1999). Na forma aglomerativa, tem-se inicialmente cada objeto compondo um grupo. Estes grupos são comparados aos pares, e o par que possuir os grupos mais similares é agregado em um novo grupo que passa a representar os dois. O processo é repetido iterativamente até que se tenha apenas um grupo contendo todos os objetos, e então se tem uma estrutura hierárquica de grupos, onde este grupo final é a raiz. Na forma divisiva, todos os objetos começam em um mesmo grupo e são divididos de forma similar ao agrupamento por partição total. Os grupos resultantes são particionados novamente de forma recursiva descendo na hierarquia.

A principal desvantagem do agrupamento hierárquico é que ele deve ser executado iterativamente, enquanto o agrupamento por partição total é executado em apenas um passo. Em contrapartida, a maior vantagem dele é a organização hierárquica em forma de árvore, que permite ao usuário navegar entre assuntos mais abrangentes - nodos mais próximos da raiz - até os assuntos mais específicos - nodos mais próximos das folhas. No entanto, a atribuição de rótulos para cada aglomerado pode ser problemática, uma vez que a atribuição de um rótulo de forma ineficiente pode prejudicar a experiência do usuário ao realizar uma busca ou navegação.

### 2.2.4 Avaliação dos resultados

Como descrito na Seção 2.2, a última etapa do agrupamento de dados é a avaliação. Existem várias formas de avaliar a qualidade dos agrupamentos resultantes do processo e nesta seção serão apresentadas algumas delas.

Segundo Tan (2006), duas medidas de avaliação comumente utilizadas são a revocação (*recall*) e a precisão (*precision*). Ambas as medidas consideram que existe um agrupamento ideal dos dados em questão, um conjunto de classes, e que este é conhecido previamente. Em razão disto, esses métodos de avaliação são considerados supervisionados e não são aplicados em casos reais de agrupamento de dados em que não se tem nenhuma informação das classes de cada objeto, mas podem ser utilizados para comparar algoritmos usando dados de teste conhecidos e já classificados.

A revocação avalia, para cada grupo resultante, quantos objetos pertencentes a ele são naturalmente do grupo sobre o total de objetos que deveriam fazer parte do grupo. Para se atingir um resultado de 100% é trivial: basta retornar apenas um grupo com todos os objetos existentes. Justamente por isso, esta métrica deve ser utilizada em conjunto com alguma outra, como por exemplo, a precisão.

A precisão avalia quantos objetos de um grupo resultante do processo fazem parte da mesma classe sobre a quantidade total de objetos deste grupo. Novamente, esta métrica é normalmente usada em conjunto com outras por ter uma solução trivial: se o processo retornasse  $N$  agrupamentos de um objeto cada, a precisão de todos eles seria 100%.

Para resolver esses problemas, a Medida-F (*F-measure*) combina as métricas de revocação e precisão de forma a medir o quanto os agrupamentos possuem apenas os

objetos que fazem parte de uma mesma classe e todos os objetos desta classe (TAN, 2006). Essa métrica deve ser calculada pela fórmula:  $F(i,j) = (2 \times precision(i,j) \times recall(i,j)) / (precision(i,j) + recall(i,j))$ , onde  $i$  é o agrupamento em questão e  $j$  a classe respectiva a ele. Nos três casos (revocação, precisão e Medida-F), para se obter o resultado final, basta somar os valores calculados para cada grupo e tirar a média, técnica também conhecida como *Macroaveraging* (LEWIS, 1992).

Outras duas métricas de avaliação supervisionada são entropia (*entropy*) e pureza (*purity*). A entropia representa o quanto os objetos de um agrupamento pertencem a uma mesma classe, considerando a probabilidade de cada objeto de um grupo  $i$  fazer parte de uma classe  $j$ . A entropia total de um conjunto de agrupamentos é o somatório da entropia de cada agrupamento ponderada pelo tamanho de cada agrupamento. A pureza é o somatório da quantidade de objetos que foram corretamente agrupados dividido pelo total de objetos (MANNING, 2008). Um objeto foi corretamente agrupado se ele faz parte da classe com a qual o seu agrupamento tenha o maior número de objetos em comum, em comparação com as outras classes.

Também existem métodos de avaliação não supervisionados, como coesão e separação de *clusters* (TAN, 2006). Nesses métodos, é utilizada uma função de validade, que atribui um grau de aceitação para cada agrupamento. Em geral se calcula a validade total dos agrupamentos, considerando  $K$  grupos, através do somatório:

$$validade\ total = \sum_{i=1}^K w_i\ validade(C_i)$$

O peso  $w_i$  varia dependendo da medida de validade utilizada, mas em alguns casos pode ser 1 ou o tamanho do grupo  $C_i$ . A função de validade pode ser uma função de coesão, onde valores mais altos representam um agrupamento melhor, ou uma função de separação, onde os valores mais baixos são melhores, ou ainda uma combinação das duas.

$$coesão(C_i) = \sum_{\substack{x \in C_i \\ y \in C_i}} proximidade(x, y)$$

$$separação(C_i, C_j) = \sum_{\substack{x \in C_i \\ y \in C_j}} proximidade(x, y)$$

As funções de coesão e separação usam uma função de proximidade para calcular a distância entre dois objetos do mesmo grupo ou de grupos diferentes. Essa função pode ser de similaridade, dissimilaridade ou uma função simples desses valores. A função de coesão determina quão relacionados estão os objetos de um agrupamento, enquanto que a função de separação determina quão distintos os objetos de um agrupamento são dos objetos de outro agrupamento qualquer.

O método Silhueta (*Silhouette*), elaborado por Rousseeuw (1987), calcula, para cada objeto agrupado, a similaridade média deste com todos os outros objetos do mesmo grupo em comparação com a similaridade média dele com todos os objetos do agrupamento vizinho mais próximo, combinando as medidas de coesão e separação. O valor resultante é chamado de  $s(i)$  e é um número no intervalo  $[-1,1]$ , onde números mais próximos de -1 indicam que o objeto provavelmente deveria estar no agrupamento vizinho e números mais próximos de 1 indicam que o objeto está no agrupamento

correto. Resultados próximos de zero indicam que o objeto está próximo à fronteira entre dois aglomerados. O valor de  $s(i)$  pode ser calculado através da fórmula abaixo:

$$s(i) = \begin{cases} 1 - b'(i)/a'(i) & \text{if } a'(i) > b'(i) \\ 0 & \text{if } a'(i) = b'(i) \\ a'(i)/b'(i) - 1 & \text{if } a'(i) < b'(i) \end{cases}$$

Onde  $a'(i)$  é a média de similaridades entre o objeto  $i$  e todos os outros objetos do próprio grupo de  $i$  e  $b'(i)$  é a média de similaridades entre  $i$  e todos os objetos do aglomerado mais próximo de  $i$  que não é o próprio aglomerado dele.

Uma característica interessante desse método é a possibilidade de se extrair uma representação gráfica das “silhuetas” dos *clusters* de forma a descobrir qual a quantidade ideal de aglomerados para particionar os dados em questão. A média de todos os  $s(i)$  mede o quão apropriadamente os dados foram particionados, onde valores mais próximos de 1 são bons resultados.

## 2.3 Algoritmos de agrupamento

Nesta seção serão explicados alguns dos algoritmos de agrupamento mais conhecidos. Os algoritmos recebem como entrada a matriz de similaridade calculada no passo anterior e devem retornar um conjunto de aglomerados de objetos. É considerado ainda um fator de aceitação mínimo (*threshold*), também conhecido como Grau de Similaridade Mínima (GSM), onde são ditos similares apenas os objetos com o grau de similaridade igual ou maior que este número. Esse parâmetro é geralmente utilizado na classe de algoritmos chamados *graphic-theoretic*, que inclui os algoritmos *Cliques*, *Single Link*, *Stars* (e suas variações) e *Strings*, e é baseado em grafos. O algoritmo K-médias é baseado em minimização do erro quadrático e o DBSCAN é baseado em densidade, portanto seguem padrões bem diferentes de raciocínio e foram incluídos neste trabalho com a intenção de mostrar a diversidade dos algoritmos existentes.

### 2.3.1 Cliques

Neste algoritmo, todos os objetos são comparados uns com os outros, e são atribuídos ao mesmo grupo os que são similares entre si. Um objeto é similar ao outro se o grau de similaridade for maior ou igual ao fator de aceitação mínimo, definido previamente. Este algoritmo é mais restritivo, de forma que os elementos de um mesmo grupo devem possuir uma relação mais forte, porém é o mais demorado porque todos os objetos precisam ser comparados entre si. Os passos do algoritmo são:

1. Selecionar um objeto que não está em nenhum grupo e adicioná-lo a um novo grupo;
2. Selecionar outro objeto e compará-lo a ele;
3. Se o objeto selecionado for similar a todos os outros objetos do grupo, adicioná-lo ao grupo;
4. Enquanto houver objetos a serem comparados, voltar ao passo 2;
5. Enquanto houver objetos não agrupados, voltar ao passo 1.

Este algoritmo pode ser disjuncto ou não. Como mencionado anteriormente, caso ele seja disjuncto, um mesmo objeto pode fazer parte de mais de um aglomerado, porém

pode-se optar por atribuir cada objeto apenas ao *cluster* onde seu grau de similaridade é maior.

A aplicação da versão disjunta deste algoritmo com a matriz de similaridade da Tabela 2.1 e um GSM de 0.6 resulta nos seguintes grupos:

Grupo 1: OBJ1, OBJ5;

Grupo 2: OBJ2, OBJ3;

Grupo 3: OBJ4.

### 2.3.2 *Single Link (connected components)*

Neste algoritmo, basta que o objeto seja similar a pelo menos um objeto de determinado aglomerado para fazer parte do mesmo aglomerado, não precisando que ele seja similar aos outros objetos dela. Novamente, se usa um GSM determinado previamente para indicar se dois objetos são similares um ao outro. Os seguintes passos compõem o algoritmo:

1. Selecionar um objeto ainda não agrupado e criar um novo grupo com ele;
2. Colocar no mesmo grupo todos os objetos similares a ele;
3. Para cada um dos objetos adicionados ao grupo, realizar o passo 2;
4. Quando não forem identificados novos termos no passo 2, ir para o passo 1.

Neste caso, um objeto não pode pertencer a mais de um grupo. O algoritmo é considerado guloso (*greedy*) por ser satisfeito com as primeiras soluções encontradas, sendo mais rápido do que o algoritmo Cliques. Devido a isso, os agrupamentos produzidos são mais simples.

Com a matriz da Tabela 2.1 e um GSM de 0,5, os seguintes grupos são encontrados através do algoritmo *Single Link*:

Grupo 1: OBJ1, OBJ5;

Grupo 2: OBJ2, OBJ3, OBJ4.

### 2.3.3 *Stars*

Neste algoritmo, um objeto é selecionado e todos os objetos similares a ele são identificados e conectados a ele, formando assim um grafo em forma de estrela (por isso o nome *Stars*) onde todos os objetos de um *cluster* estão conectados a um objeto central. Os passos simples são:

1. Selecionar um objeto não atribuído a algum cluster como centro de um novo cluster;
2. Colocar todos os objetos não agrupados similares a ele neste mesmo cluster;
3. Se ainda houver objetos não agrupados, repete os passos 1 e 2;
4. Quando não houver mais objetos não classificados, o processo termina.

Executando-se o algoritmo Stars com GSM de 0.5 e a matriz da Tabela 2.1, se obtém os seguintes grupos:

Grupo 1: OBJ1, OBJ3, OBJ5;

Grupo 2: OBJ2, OBJ4.

Assim como o algoritmo *Single Link*, o *Stars* é um algoritmo guloso, sendo mais rápido do que os algoritmos mais elaborados, mas obtendo resultados mais relaxados e simples. Para resolver esse problema, Wives (1999) descreveu os algoritmos *Best-star* e *Full-stars*, definidos a seguir.

### 2.3.4 *Best-star* e *Full-stars*

O algoritmo *Stars* atribui um objeto ao primeiro *cluster* que satisfaça o fator de aceitação mínimo, ignorando a possibilidade de existirem outros *clusters* onde o grau de similaridade de um objeto seja maior. Os algoritmos *Best-star* e *Full-star* foram desenvolvidos a fim de resolver este problema.

No algoritmo *Best-star*, os objetos identificados como mais similares a determinado grupo, mesmo que já agrupados, são removidos do *cluster* atual e colocados no grupo de maior afinidade. Dessa forma, o usuário não precisa mais se preocupar em encontrar um bom grau de aceitação mínimo para o processo e os grupos resultantes são mais naturais, pois os objetos são atribuídos automaticamente aos grupos com que possuem uma relação mais forte. Aplicando-se o algoritmo à matriz da Tabela 2.1 com GSM de 0,5, os seguintes grupos são encontrados:

Grupo 1: OBJ1, OBJ5;

Grupo 2: OBJ2;

Grupo 3: OBJ3, OBJ4;

Grupo 4: OBJ5.

Alternativamente, o algoritmo *Full-stars* também considera os objetos já agrupados no momento de determinar os objetos similares a um objeto centro de outro *cluster*, porém atribui os objetos a todos os *clusters* em que sua similaridade satisfizer o *threshold*, sem removê-lo dos *clusters* dos quais já faz parte. Dessa forma, percebe-se que o algoritmo *Full-stars* é disjunto, enquanto o *Best-star* não. Novamente, o exemplo considerando a matriz 2.1 e GSM de 0.5:

Grupo 1: OBJ1, OBJ3, OBJ5;

Grupo 2: OBJ2, OBJ3;

Grupo 3: OBJ3, OBJ2, OBJ4;

Grupo 4: OBJ4, OBJ3.

### 2.3.5 *Strings*

A ideia deste algoritmo é construir cadeias objetos, de forma que o próximo objeto na sequência seja similar ao anterior, sem necessidade de ser similar aos objetos não adjacentes a ele. Os passos são os seguintes:

1. Selecionar um objeto não classificado e adicionar ele a um novo *cluster*;
2. Localizar o objeto mais similar ao objeto atual e adicioná-lo no mesmo *cluster*;
3. O objeto atual passa a ser o recém adicionado e repete-se o passo 2;
4. Se não houver objetos similares, volta-se ao passo 1;
5. Quando todos os objetos fizerem parte de algum *cluster*, o processo termina.

Novamente existe a necessidade de um fator mínimo de aceitação para determinar se um par de objetos é similar ou não. Aplicando-se este algoritmo à Tabela 2.1 com um GSM de 0,6, chega-se ao seguinte resultado:

Grupo 1: OBJ1, OBJ5;

Grupo 2: OBJ2, OBJ3;

Grupo 3: OBJ4.

### 2.3.6 K-médias

Neste algoritmo, deve-se inicialmente estabelecer um número  $k$  de grupos que se deseja obter para que ele busque a melhor forma de separar os objetos nestes  $k$  grupos. A atribuição dos objetos aos  $k$  grupos é um problema *NP-hard*, portanto é geralmente resolvido com heurísticas eficientes que acabam encontrando resultados localmente ótimos.

O algoritmo utiliza o conceito de centroides como objetos abstratos, que não fazem parte do conjunto de dados em questão, mas fazem parte do seu domínio, para representar os centros dos  $k$  grupos. Os centroides são calculados pela média de todos os objetos de um grupo.

Os passos a seguir descrevem o algoritmo k-médias mais utilizado e descrito por MacKay (2003):

1. Selecionar  $k$  centroides aleatórios;
2. Comparar cada objeto com os  $k$  centroides e atribuí-lo ao grupo do centroe mais similar a ele;
3. Calcular um novo centroe para cada um dos  $k$  grupos;
4. Repetir os passos 2 e 3 até que os resultados convirjam.

A maior desvantagem deste algoritmo é a necessidade de se selecionar um número  $k$  de grupos previamente, o que exige que se saiba de antemão quantos grupos os objetos do contexto devem formar, ou processar os objetos algumas vezes variando o valor de  $k$  até que se encontre o valor ideal. Assim, compreende-se que o valor ideal de  $k$  é extremamente dependente da coleção de objetos que se deseja processar, porém, não é necessário definir um GSM de antemão.

### 2.3.7 DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*)

Este algoritmo utiliza o conceito de vizinhança-épsilon para determinar os agrupamentos. Estão na vizinhança-épsilon de um objeto todos aqueles objetos que estão a uma distância de até épsilon dele, e se diz que estes objetos são alcançáveis dentro da densidade épsilon. Considera-se que um objeto é “de núcleo” (*core object*) se ele possui um número mínimo de objetos em sua vizinhança-épsilon. O objeto de núcleo e os seus vizinhos então formam um novo agrupamento. Os objetos que fazem parte de um agrupamento e não possuem o número mínimo de vizinhos são chamados “objetos de fronteira” (*border objects*). Objetos sem o número mínimo de vizinhos que não sejam vizinhos de um objeto de núcleo são chamados de ruído (*noise*) e não entram em nenhum agrupamento. O valor de épsilon e o número mínimo de vizinhos necessários são parâmetros do agrupamento.

O algoritmo pode ser definido pelos passos a seguir (ESTER, 1996):



1. Selecionar um objeto não visitado e encontrar todos os objetos vizinhos dele dentro da densidade  $\epsilon$ ;
2. Se o número de vizinhos é maior ou igual ao número mínimo, marca-se o objeto como de núcleo e atribui-se ele e todos os seus vizinhos a um novo grupo, removendo a marca de “ruído” de eventuais vizinhos;
3. Se o número de vizinhos é inferior ao número mínimo e ele não faz parte de nenhum grupo, marca-se o objeto como ruído;
4. Caso dois grupos possuam algum objeto em comum, unificam-se os dois grupos;
5. Enquanto houver objetos que não tenham sido visitados ainda, volta-se ao passo 1; caso contrário, termina-se o algoritmo.

Uma vantagem do DBSCAN é que não se precisa definir previamente o número de *clusters* a serem gerados, porém é necessário determinar o valor de  $\epsilon$  e o número mínimo de vizinhos. Isso pode ser problemático nos casos em que não exista uma combinação desses valores que satisfaça todos os grupos de um conjunto de dados a ser classificado. Por outro lado, um grande diferencial do algoritmo é a capacidade de encontrar grupos com formas arbitrárias, podendo inclusive encontrar grupos completamente cercados por outros grupos.

## 2.4 Trabalhos relacionados

Existem diversos *softwares* de mineração de dados que implementam o processo de agrupamento de dados. O *Waikato Environment for Knowledge Analysis* (WEKA) é uma coleção de ferramentas de aprendizado de máquina e pré-processamento de dados que fornece algoritmos para regressão, classificação, agrupamento, mineração de regras de associação e seleção de atributos (HALL, 2009). Em termos de agrupamento de dados, ele permite a execução de algoritmos variados sobre dados pré-processados, a análise desses processos e, quando aplicável, a visualização dos grupos resultantes. O fato de o WEKA trabalhar sobre dados pré-processados é uma vantagem, pois se pode transformar qualquer tipo de dado para uso na ferramenta, porém a curva de aprendizado no uso da ferramenta é um pouco íngreme em decorrência disso.

O Eureka é um *software* de agrupamento de documentos que implementa alguns algoritmos de agrupamento e permite a entrada de arquivos de texto puros e outros parâmetros através de uma interface gráfica simples e de fácil aprendizado. Porém, como mencionado no Capítulo 1, a ferramenta é incompatível com os sistemas operacionais atuais, não aceita extensões e é limitada ao agrupamento de textos.

Dito isso, o objetivo deste trabalho é a criação de uma ferramenta dedicada para agrupamento de dados de qualquer tipo de forma que se possa adicionar módulos e algoritmos a ela facilmente para experimentos e avaliação de técnicas, sem a necessidade de pré-processar os dados em algum formato específico. O uso da linguagem Java garante o funcionamento da ferramenta em diversas plataformas e o uso de padrões de projeto provê maior flexibilidade ao estender e reaproveitar módulos implementados por outros usuários.

### 3 PROPOSTA DE FRAMEWORK PARA AGRUPAMENTO DE DADOS

Para facilitar o desenvolvimento de novos algoritmos e a comparação entre diferentes métodos de agrupamento foi desenvolvido um *framework*. O objetivo principal foi criar uma implementação flexível, com foco no reuso de código e na possibilidade de se montar diferentes configurações apenas trocando uma classe por outra. Para que isso fosse possível, alguns padrões de projeto, ou *design patterns*, foram adotados.

Os passos implementados no *framework* foram os três primeiros do processo de agrupamento (definidos na Seção 2.2): seleção de características, cálculo de similaridade e agrupamento de objetos. O projeto foi modelado em torno do agrupamento de documentos textuais, porém de forma a permitir que generalizações sejam feitas e possa-se usá-lo para agrupar outros tipos de dados, como, por exemplo, imagens, em uma versão futura. A implementação foi feita na linguagem de programação Java.

Antes de entrar em detalhes da arquitetura proposta, serão apresentadas e discutidas as definições de *framework* e *design patterns*.

#### 3.1 Frameworks

*Frameworks* são projetos abstratos para um determinado tipo de aplicação. Segundo Johnson (1988), *frameworks* são constituídos por classes abstratas que representam cada componente do sistema e as interfaces entre essas classes, que descrevem como os componentes se comunicam entre si. Essas definições compõe uma *Application Programming Interface* (API), que define como as classes concretas devem ser implementadas para que possam ser usadas em conjunto com o *framework*.

Uma das principais características de um *framework* é a inversão de controle: o usuário deve implementar subclasses específicas para a aplicação derivadas de classes abstratas do *framework* (GAMMA, 2000). Isso impõe uma série de restrições ao seu uso, de forma a maximizar o reuso tanto do *framework* quanto dos componentes desenvolvidos para uso com o mesmo. Esta também é a principal diferença entre o uso de *frameworks* e o uso de bibliotecas de *software*: no caso de uma aplicação comum, que não utiliza um *framework*, o controle é ditado pelo programa e não pelas bibliotecas usadas. Essa diferença é ilustrada na Figura 3.1.

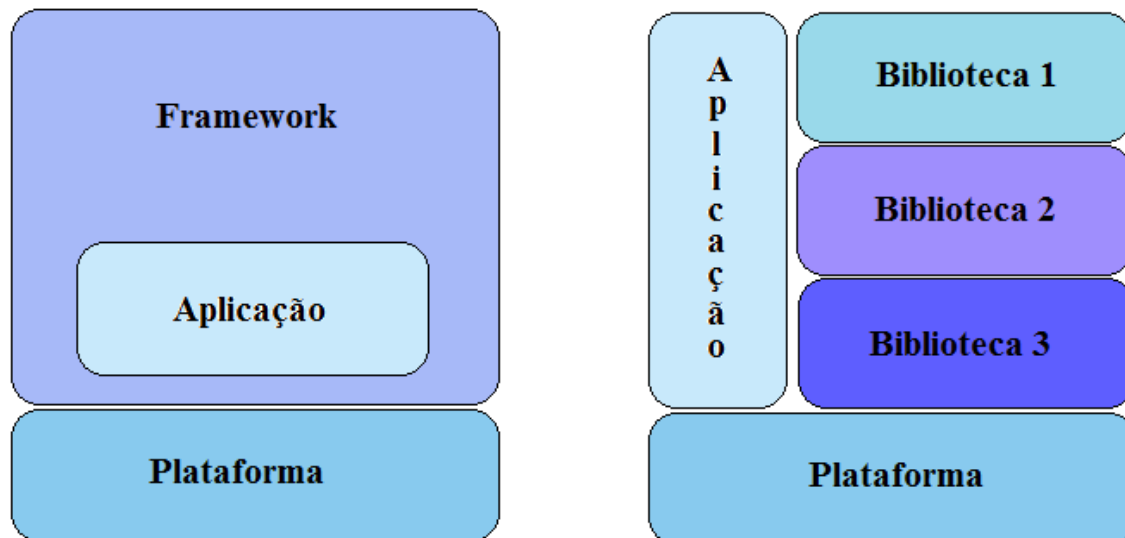


Figura 3.1: Comparativo entre uso de *framework* e bibliotecas para o desenvolvimento de uma aplicação.

Fonte: elaborada pelo autor.

Outra característica importante é sua capacidade de extensão, ou seja, sua capacidade de ser facilmente complementado por código de usuários. Geralmente *frameworks* são utilizados como uma caixa-preta, onde o conhecimento de seu funcionamento interno não deve ser um requisito para o uso e seu código não deve ser modificado. Essa política permite que componentes personalizados sejam reaproveitados em diferentes aplicações que utilizem o mesmo *framework* sem precisar se preocupar com a compatibilidade do mesmo, já que o código dele não será modificado.

Existem outras vantagens no uso de *frameworks*: muitas decisões de projeto já foram tomadas e a arquitetura já está definida, de forma que o usuário possa se focar na lógica da aplicação que deseja desenvolver. Em contrapartida, essas decisões podem tornar o desenvolvimento inflexível, exigindo um esforço maior para implementar determinadas partes da aplicação na arquitetura do *framework*, ou tornando impossível que estas sejam implementadas. Segundo Gamma (2000), essa inflexibilidade pode ser reduzida usando-se padrões de projeto, ou *design patterns*, no desenvolvimento do *framework*, aumentando a probabilidade de atingir níveis altos de reusabilidade de projeto e código. O uso de padrões de projeto ajuda a tornar a arquitetura adequada a diversas aplicações, sem necessidade de reformulação.

### 3.2 Padrões de projeto

Padrões de projeto são “descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular” (GAMMA, 1988, p. 20). Os padrões não são soluções que podem ser convertidas diretamente para código, devem ser obrigatoriamente adaptadas para o contexto da implementação antes de serem usadas. Isso se deve ao fato de serem genéricos o suficiente para que possam ser aplicados em contextos variados, sempre de uma forma diferente, específica ao contexto.

Um padrão é composto por um nome, um problema, uma solução e as suas consequências. O nome de um padrão ajuda a descrever um problema de projeto, suas soluções e consequências em uma ou duas palavras, de forma a facilitar o seu uso no projeto e em comunicação entre os membros de uma equipe de desenvolvimento. Um bom nome de padrão é facilmente reconhecido dentro de um projeto e permite um entendimento imediato de seu funcionamento na arquitetura do mesmo. O problema descreve uma situação na qual existe vantagem na aplicação do padrão. A solução descreve os elementos que compõem o padrão, seus relacionamentos, suas responsabilidades e colaborações, sem entrar em detalhes de implementação, pois o padrão deve ser genérico para poder ser aplicado em contextos diferentes. As consequências são os resultados e análises das vantagens e desvantagens da aplicação do padrão.

A seguir, a arquitetura proposta será descrita, assim como os padrões utilizados e a justificativa para o seu uso. Para uma descrição mais extensa e completa de padrões de projeto, recomenda-se a leitura de (GAMMA, 1988).

### 3.3 A Arquitetura

A decisão de se implementar um *framework* pareceu natural ao se considerar a sequência de passos que compõem o processo de agrupamento de dados. Os passos devem ser feitos sempre na mesma ordem e, em geral, o resultado de um passo é o parâmetro do passo seguinte. Como essa linha de execução é estática e apenas os componentes utilizados devem ser variáveis, decidiu-se usar padrões de projeto para definir como esses componentes devem ser implementados, permitindo que o foco do usuário possa permanecer nos métodos de agrupamento.

Adicionalmente, deseja-se poder visualizar o progresso do agrupamento e os resultados finais no formato que for mais conveniente para o usuário. Para isso, consideraram-se meios para permitir que sejam acopladas diferentes classes personalizadas ao *framework* e que recebam notificações de mudanças no processo, através do padrão de projeto *Observer*.

As duas etapas finais do processo, abstração dos dados e avaliação dos resultados, não foram implementadas para simplicidade do *framework*, e devem ser adicionadas em uma versão futura.

O *framework* possui três módulos principais: a definição dos objetos e suas características, o processamento e as estratégias de seus passos, e os objetos observáveis. Como o *framework* por si só não faz nada ao ser executado, é usada uma API para que o usuário defina o comportamento do mesmo com classes próprias. Essa API é composta das classes abstratas definidas nas seções 3.3.1 a 3.3.3.

Para que seja útil, o *framework* deve ser instanciado através da implementação de classes específicas que definirão o seu comportamento e irão gerar informações úteis para o usuário. Uma instância será descrita no Capítulo 4 para exemplificar o uso do *framework* e demonstrar parte de sua flexibilidade.

#### 3.3.1 Objetos e suas características

Os conceitos mais básicos envolvendo o agrupamento de dados são os objetos e suas características. Um objeto representa uma entidade a ser agrupada com outras entidades similares durante o processo de agrupamento. Uma característica é a representação de

um padrão de um objeto. Um objeto possui uma coleção de características que o definem dentro do domínio da aplicação. Dentro do contexto de agrupamento de documentos textuais, um documento pode ser considerado um objeto, e suas características podem ser o conjunto das palavras que o compõem. Assim, tem-se duas classes abstratas e duas subclasses concretas: *DataObject* e *DataFeature*, *TextDocument* e *Term*, cujas relações podem ser observadas no diagrama da Figura 3.2.

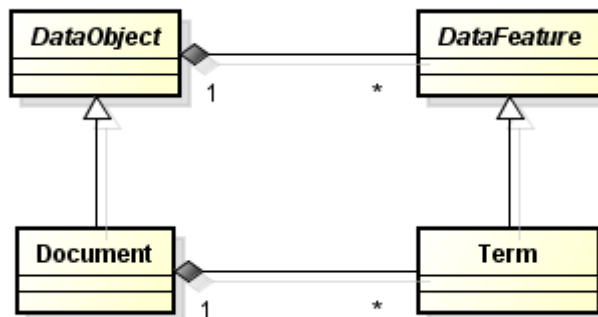


Figura 3.2: Diagrama de classes com os conceitos básicos.

Fonte: elaborada pelo autor.

Apesar de o *framework* proposto ter o objetivo de trabalhar com o agrupamento de qualquer tipo de objeto, o escopo deste trabalho se limitou à implementação de um protótipo do *framework*, de forma que apenas pudessem ser feitos agrupamentos de documentos textuais. Em decorrência disso, alguns ajustes nas classes *DataObject* e *DataFeature* poderão ser necessários para que se possa generalizar outros tipos de dados, porém esses ajustes são temas para um trabalho futuro.

### 3.3.2 O processo e seus passos

Para a linha principal de execução, considerou-se o padrão de projeto *Template Method*. Cada passo do agrupamento de dados seria um método abstrato e o método principal do processo chamaria esses métodos. Porém, isso deixaria o usuário preso à implementação de uma classe que definisse todos os passos, impossibilitando que se usassem dinamicamente combinações diferentes de algoritmos para cada passo.

Para resolver este problema, optou-se então pelo padrão *Strategy*. Segundo Gamma (1988), o objetivo do padrão *Strategy* é definir uma família de algoritmos, encapsular cada um deles e torná-los intercambiáveis, independente do cliente que os usa. Define-se uma interface comum, uma classe abstrata com apenas um método abstrato, e cada algoritmo será a implementação de uma classe diferente, que respeita esta interface. Um modelo genérico de aplicação do padrão de projeto pode ser visto na Figura 3.3.

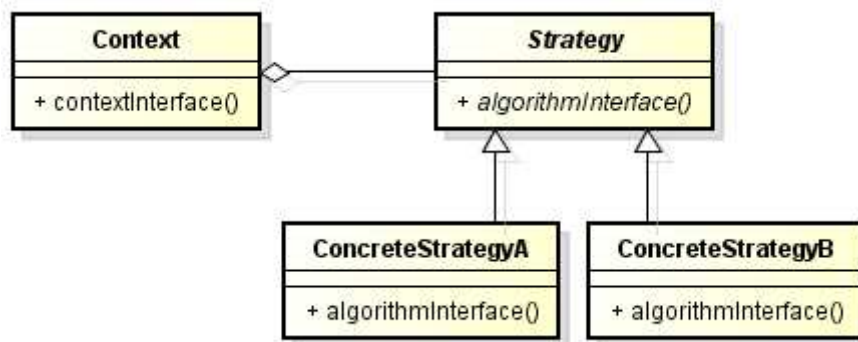


Figura 3.3: Diagrama de classes exemplificando o *design pattern* Strategy.

Fonte: elaborada pelo autor.

No caso do agrupamento de dados, tem-se cinco passos distintos que podem ser representados por cinco famílias de algoritmos diferentes. Cada passo do processo é definido por uma classe abstrata diferente, que define um método abstrato com os parâmetros e o tipo de retorno adequado para cada um. As estratégias criadas foram:

- *FeatureSelectionStrategy*: recebe como argumento uma coleção de objetos do tipo *DataObject* e retorna a mesma coleção, porém cada *DataObject* populado com objetos do tipo *DataFeature*;
- *SimilarityStrategy*: recebe como argumento uma coleção de objetos do tipo *DataObject*, que estão populados com seus objetos do tipo *DataFeature*, e retorna um objeto do tipo *Matrix2D*, a matriz de similaridade entre todos os objetos;
- *ClusteringStrategy*: recebe como argumento uma coleção de objetos do tipo *DataObject* e o respectivo objeto do tipo *Matrix2D*, e retorna uma coleção de agrupamentos entre os objetos, representados por objetos do tipo *DataCluster*.

Os algoritmos propriamente ditos serão implementados pelo usuário em subclasses concretas destas três classes abstratas. O uso dessas classes abstratas permite que se tenham diferentes implementações para cada passo e se possa trocar uma pela outra em tempo de execução, podendo-se comparar o desempenho de diferentes métodos ou apenas reaproveitar dois passos ao implementar dois algoritmos diferentes para o passo restante. Por exemplo, no caso do agrupamento de textos, tem-se as implementações *TermSelectionStrategy*, *FuzzyMeansSimilarityStrategy*, *CliquesClusteringStrategy* e *StarsClusteringStrategy*, como visto no diagrama da Figura 3.4: à esquerda estão as classes abstratas que definem as interfaces das estratégias e à direita estão as implementações das mesmas. Assim, tem-se uma opção de estratégia para seleção de características, outra para o cálculo da similaridade e duas opções de algoritmos de agrupamento, de forma que se pode usar ou uma, ou outra, podendo-se reaproveitar as implementações dos dois passos anteriores.

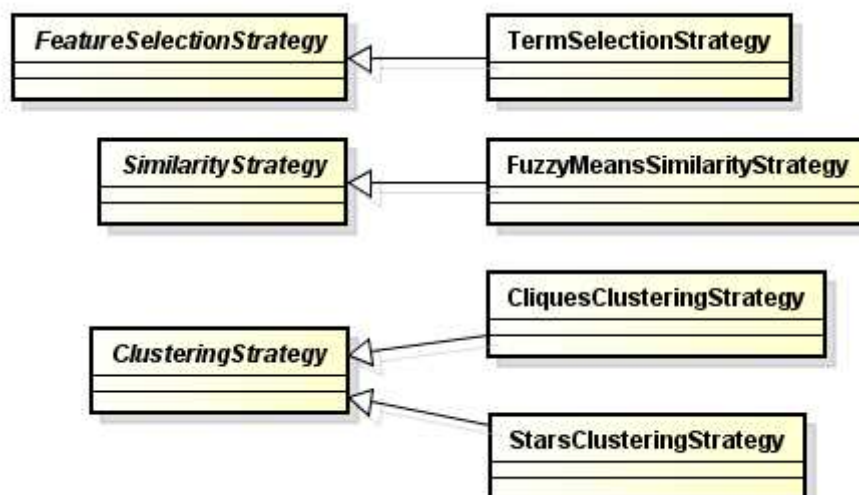


Figura 3.4: Versão simplificada do diagrama de classes das estratégias de agrupamento (sem os métodos).

Fonte: elaborada pelo autor.

Definidas as estratégias que se quer usar, precisa-se guardá-las em um objeto que irá executar o processo completo, chamando os métodos de cada passo e fazendo o processamento necessário entre passos. Esse objeto será a linha principal de execução do agrupamento e por isso ele será chamado de *ClusteringProcess*. A relação deste objeto com as estratégias de agrupamento pode ser visto no diagrama da Figura 3.5.

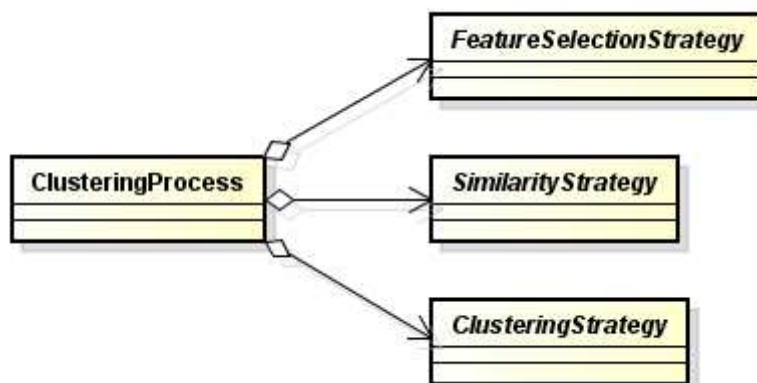


Figura 3.5: Diagrama da relação entre *ClusteringProcess* e as estratégias.

Fonte: elaborada pelo autor.

Tanto as classes concretas das estratégias quanto a classe *ClusteringProcess* devem definir objetos observáveis, para que componentes externos ao *framework* possam observar o estado destes objetos e exibir visualizações relevantes a respeito dos dados do processo.

### 3.3.3 Os observadores

O padrão de projeto *Observer* define uma dependência de um-para-muitos entre objetos, de forma que, quando um objeto muda de estado, todos os seus dependentes, "observadores", são notificados e atualizados (GAMMA, 1988). A classe "observada" é chamada de *subject*, e mantém uma lista de observadores que devem se registrar para

receber notificações toda vez que alguma coisa mude. Ao receber uma notificação de atualização, o observador pode então utilizar métodos da classe *subject* para obter informações. Um diagrama de classes genérico ilustrando o padrão de projeto *Observer* pode ser visto na Figura 3.6 e um diagrama de sequência exemplificando a comunicação existente entre as entidades envolvidas pode ser visto na Figura 3.7.

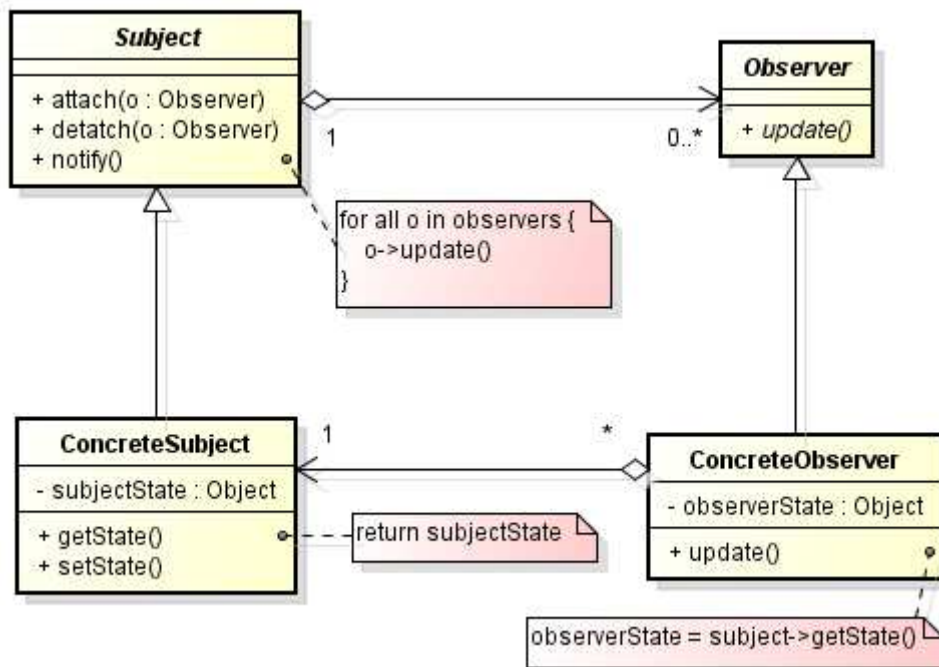


Figura 3.6: Diagrama de classes exemplificando o *design pattern Observer*.

Fonte: elaborada pelo autor.

Na Figura 3.7 *subject*, *obs1* e *obs2* são objetos em memória durante a execução de um programa. As setas preenchidas representam invocações de métodos dos objetos destino das mesmas, enquanto as setas tracejadas representam o retorno do método.



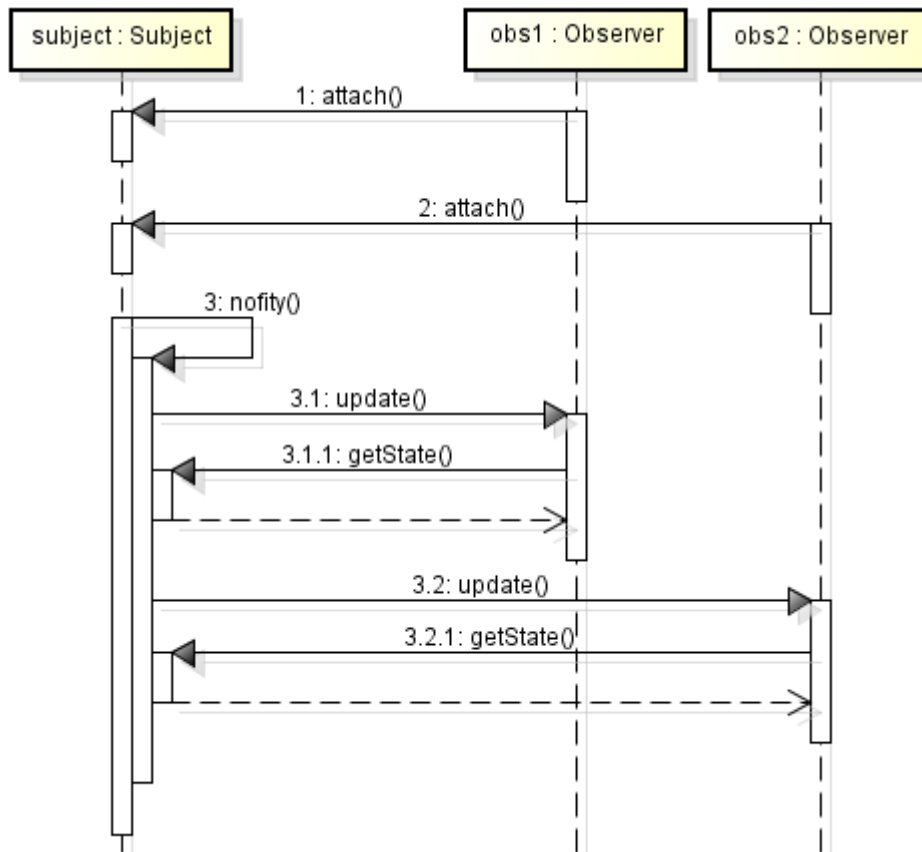


Figura 3.7: Diagrama de sequência da troca de mensagens entre *subject* e *observers*.

Fonte: elaborada pelo autor.

Como mencionado previamente, o objetivo ao aplicar este *design pattern* é poder observar a execução dos passos do agrupamento de dados mostrando alguma visualização gráfica para o usuário, logo, é compreensível ter a classe *ClusteringProcess* como *subject*. Porém, a granularidade desta classe é muito alta, já que ela abstrai os algoritmos aplicados em cada passo, podendo fornecer apenas a informação de qual passo já foi executado ou será executado em seguida. Para que se possa observar detalhes de execução dos algoritmos de cada passo, os mesmos deverão ser *subjects* também.

Já está claro que a classe *ClusteringProcess* deve ser observável pelas classes do usuário do *framework*, de forma que elas possam mostrar o progresso e os resultados (parciais e finais) visualmente em tela. Porém, os algoritmos implementados como estratégias do agrupamento podem possuir estruturas completamente diferentes entre si, e não há como prever que tipo de informação pode ser interessante para um observador. Por isso, as estratégias concretas devem poder ser observadas diretamente, de forma que os observadores possam adquirir conhecimentos específicos do processo em execução, caso seja desejado, como pode ser visto no diagrama de sequência da Figura 3.8.

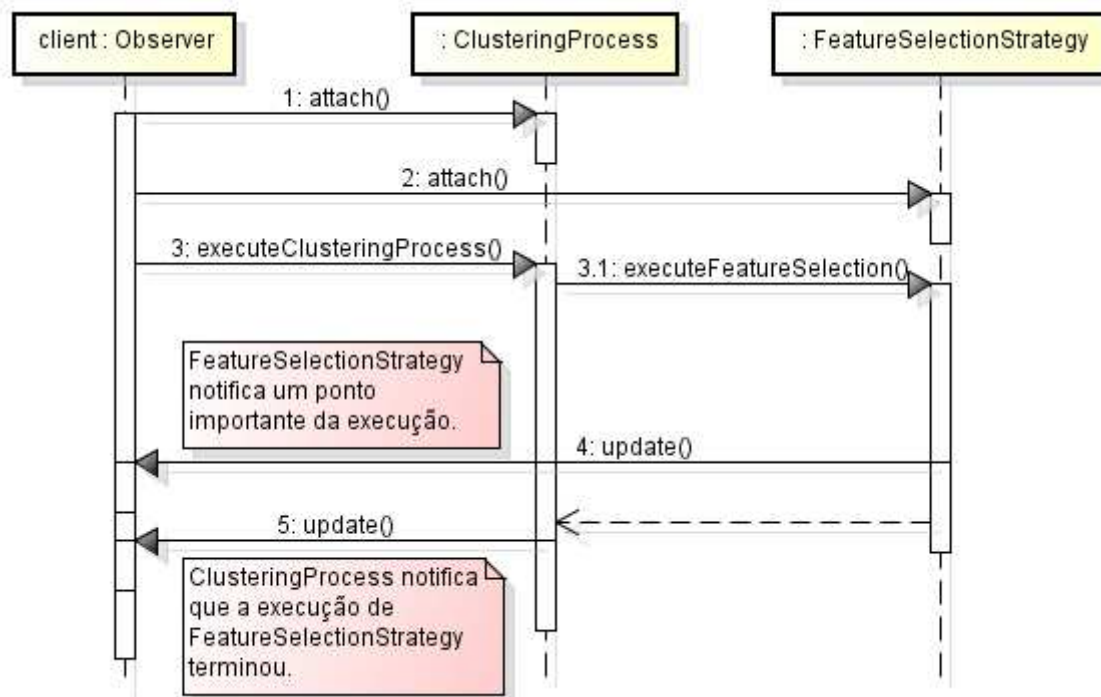


Figura 3.8: Diagrama de sequência simplificado da relação entre uma classe observadora do cliente e as entidades do *framework*.

Fonte: elaborada pelo autor.

Foram usadas a classe abstrata *Observable* e a interface *Observer* do pacote *java.util* da API da linguagem Java para implementar este padrão, reaproveitando código e simplificando a implementação do *framework*.

## 4 INSTANCIÇÃO DO FRAMEWORK PARA O AGRUPAMENTO DE DOCUMENTOS

Para fins de demonstração do *framework*, foi instanciada uma aplicação que o usa para agrupamento de textos. Para demonstrar a capacidade do *framework* de aceitar diferentes implementações que respeitem suas interfaces, foram implementados alguns algoritmos diferentes para cada passo do processo.

Nas próximas seções, esta instância será descrita juntamente com sua relação com a arquitetura do *framework*. Logo após, os resultados de alguns testes com coleções de documentos serão comparados, a fim de verificar quais dos algoritmos implementados são mais eficientes.

### 4.1 Agrupamento de documentos usando o *framework* proposto

Alguns detalhes de implementação ficam a cargo do usuário do *framework*, como por exemplo, a conversão dos dados que serão usados para um formato compatível, usando subclasses de *DataObject* e *DataFeature*. Outro detalhe importante é que, fora o número de passos completados durante a execução, o *framework* não provê outros tipos de *feedback* ao usuário - as classes que implementam as estratégias de cada passo devem especificar como seus observadores podem ser atualizados a respeito do estado interno delas.

No caso dos textos, o usuário deve implementar uma forma de carregar os conteúdos dos documentos para dentro da classe *Document*, seja lendo de um banco de dados ou uma coleção de arquivos. Logo após, a estratégia *FeatureSelectionStrategy* é responsável em transformar esse conteúdo em objetos *DataFeature*, que nesse caso são palavras (ou termos).

Caso seja do interesse do usuário, pode-se salvar os resultados de cada etapa em um arquivo de texto. Essa responsabilidade deve ser de uma ou mais classes observadoras das estratégias e, dependendo do nível de abstração dos resultados desejados, essas classes podem servir para quaisquer estratégias implementadas (se o dado desejado é o tempo de execução, por exemplo) ou para implementações específicas (caso o objetivo seja saber quanto tempo foi necessário para executar subtarefas específicas, por exemplo).

Nas seções seguintes as implementações de cada um desses módulos serão descritas de forma a demonstrar o uso do *framework* em uma aplicação de agrupamento de documentos. Primeiro são definidas classes concretas baseadas nas classes abstratas *DataObject* e *DataFeature* do *framework*. Depois as estratégias são implementadas usando as interfaces *FeatureSelectionStrategy*, *SimilarityStrategy* e *ClusteringStrategy*. Por último, é definida uma classe observadora *ConsoleObserver* que salva os resultados

do processo em arquivos de texto para que se possa inspecionar os dados posteriormente, e um programa *Inspector* para que se possa visualizar esses dados.

#### 4.1.1 Os documentos e seus termos

No contexto do agrupamento de documentos textuais, os documentos são os objetos do agrupamento, enquanto as palavras, ou termos, do documento são as características. Para isso, foram implementadas as classes *Document* e *Term*.

A classe *Document* possui dois campos: título e conteúdo. O título será usado para identificar os documentos agrupados ao final da execução da aplicação, enquanto o conteúdo será a fonte de suas características. Além dos campos específicos, a classe *Document* herda uma lista (vazia) de características do tipo *DataFeature* de sua classe pai, *DataObject*. Essa lista será preenchida com objetos do tipo *Term* pela implementação de *FeatureSelectionStrategy*.

A classe *Term*, por sua vez, possui os campos valor, frequência absoluta e frequência relativa. O valor é a palavra representada pelo objeto, em forma de um objeto do tipo *String*, a frequência absoluta é um número inteiro que indica quantas vezes a palavra aparece no documento e a frequência relativa é um número real que representará a influência do termo dentro do documento ao qual ele pertence. A frequência relativa pode ser calculada de diversas formas e, nesta implementação, ela é a frequência absoluta dividida pela quantidade total de termos do documento.

#### 4.1.2 As estratégias

Inicialmente, deve-se extrair os termos de um documento a partir de seu conteúdo. Para isso, foi implementada a estratégia *TermSelectionStrategy*, que extrai as palavras do documento e as coloca em uma coleção de palavras, contando quantas vezes cada palavra aparece. É usado um conjunto de *stop-words*, que deve ser passado como parâmetro do construtor da classe, para filtrar as palavras que serão extraídas do texto. A partir dessa coleção de palavras são gerados os objetos *Term*, tendo como valor cada palavra encontrada e a sua frequência absoluta como peso. Por último, são calculadas as frequências relativas de cada termo do documento. O processo é repetido para todos os documentos do conjunto de entrada.

A seguir, foi implementada uma estratégia para o cálculo de similaridade: *FuzzyMeansSimilarityStrategy*. Esta estratégia foi implementada originalmente por Wives (1999), baseada na função *média por operadores fuzzy* apresentada por Oliveira (1996). A função de similaridade utilizada é definida pela fórmula a seguir:

$$gs(X, Y) = (\sum_{h=1}^k gi_h(a, b)) / n$$

$X$  e  $Y$  são os dois documentos que serão comparados;  $gs$  é o grau de similaridade entre eles;  $k$  é o número de termos em comum que os dois documentos possuem;  $n$  é a quantidade de termos existentes nos documentos; e  $gi$  é o grau de igualdade entre os pesos  $a$  (no documento  $X$ ) e  $b$  (no documento  $Y$ ) do termo  $h$ . O grau de igualdade  $gi$  é calculado através da seguinte fórmula:

$$gi_h(a, b) = \frac{1}{2} [(a \rightarrow b) \wedge (b \rightarrow a) + (\bar{a} \rightarrow \bar{b}) \wedge (\bar{b} \rightarrow \bar{a})]$$

onde  $\bar{x} = 1 - x$ ,  $a \rightarrow b = \max\{c \in [0,1] | a * c \leq b\}$  e  $a \wedge b = \min\{a, b\}$ .

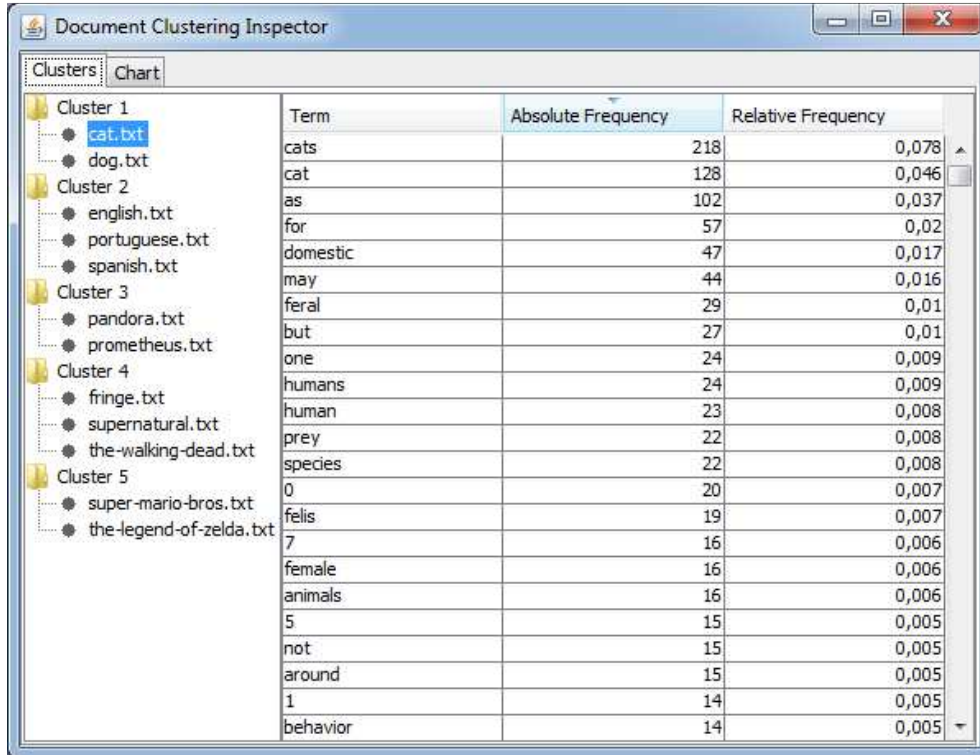
De forma a dar mais ênfase à comparação entre diferentes algoritmos de agrupamento neste trabalho, apenas essa função de similaridade foi implementada. Outras funções podem ser facilmente adicionadas à aplicação e podem ser implementadas em trabalhos futuros.

Por último, foram implementados alguns algoritmos de agrupamento: *CliquesClusteringStrategy*, *StarsClusteringStrategy*, *BestStarClusteringStrategy* e *FullStarsClusteringStrategy*. Inicialmente considerou-se a implementação dos algoritmos K-médias e DBSCAN, porém o *framework* foi construído com base nos algoritmos da classe *graphic-theoretic* e precisaria de algumas modificações para ser compatível com os demais. O algoritmo K-médias, por exemplo, precisa computar a similaridade dos documentos com os  $K$  centroides durante o processo de agrupamento, pois esta informação não está na matriz de similaridades, necessitando de acesso à função de similaridade utilizada. Como o objetivo é descrever o *framework* e mostrar sua aplicação, essas modificações podem ser abordadas em trabalhos futuros. Os algoritmos implementados já foram explicados na Seção 2.3 e não serão descritos aqui.

#### 4.1.3 Exibindo progresso e resultados

Para possibilitar o exame detalhado dos resultados, foi implementada a classe *ConsoleObserver*, que observa o processo em alto nível, registrando em arquivos de texto os resultados de cada etapa: as características extraídas de cada objeto, a matriz de similaridade e os grupos resultantes. Esses arquivos podem posteriormente serem usados para analisar os resultados e permitir a visualização gráfica de estatísticas do processo.

Justamente para esta finalidade foi criado o programa *Inspector*, que lê os arquivos gerados pela classe *ConsoleObserver* e exibe os resultados em um formato amigável. O programa é composto por uma janela com duas abas: uma exibe os grupos resultantes do processo e a outra, um gráfico estatístico a respeito dos mesmos. Na primeira aba, os grupos e seus objetos são exibidos em uma estrutura de árvore. Clicando-se em um dos objetos, é exibida uma tabela com as características utilizadas no processo, assim como as frequências absoluta e relativa de cada uma, como pode ser visto na Figura 4.1. Na segunda aba, pode ser visualizado um gráfico com a quantidade de objetos por grupo (Figura 4.2).



The screenshot shows the 'Document Clustering Inspector' window. On the left, a tree view lists five clusters and their associated documents. On the right, a table displays the terms and their frequencies for each cluster.

Cluster	Term	Absolute Frequency	Relative Frequency
Cluster 1	cat.txt	218	0,078
	dog.txt	128	0,046
Cluster 2	english.txt	102	0,037
	portuguese.txt	57	0,02
	spanish.txt	47	0,017
Cluster 3	pandora.txt	44	0,016
	prometheus.txt	29	0,01
Cluster 4	fringe.txt	27	0,01
	supernatural.txt	24	0,009
	the-walking-dead.txt	24	0,009
Cluster 5	super-mario-bros.txt	23	0,008
	the-legend-of-zelda.txt	22	0,008
	0	22	0,008
	1	20	0,007
	7	19	0,007
	behavior	16	0,006
	female	16	0,006
	animals	16	0,006
	5	15	0,005
	not	15	0,005

Figura 4.1: Visualização dos grupos resultantes na janela da classe *Inspector*.

Fonte: elaborada pelo autor.



Figura 4.2: Gráfico de quantidade de objetos por grupo.

Fonte: elaborada pelo autor.

## 4.2 Resultados

Após instanciar o *framework* implementando uma aplicação de agrupamento de documentos, foram feitos alguns testes com conjuntos reais de dados, utilizando várias estratégias diferentes a fim de comparar o desempenho delas e mostrar o *framework* em funcionamento. Adicionalmente, foi feita uma comparação entre os resultados obtidos com o *framework* proposto e com o programa Eureka a fim de demonstrar a eficácia dos algoritmos implementados em comparação com uma ferramenta já existente e com bons resultados conhecidos (WIVES, 2008).

Foram executados testes com uma coleção pequena de doze (12) artigos selecionados da Wikipedia que foram previamente classificados, e um subconjunto da coleção de documentos Reuters-21578, contendo 722 documentos. A primeira coleção foi avaliada através das métricas de revocação, precisão e Medida-F, enquanto a segunda coleção foi avaliada com o método Silhueta, de forma a mostrar a aplicação de várias métricas diferentes.

Como parâmetros para o agrupamento destas coleções foram usados quatro conjuntos de *stop-words*, selecionados por Wives (1999): “consoantes”, “vogais”, “english” e “sgml”. Os 4 conjuntos são descritos em (WIVES, 1999). Outro parâmetro é o GSM – vários valores foram utilizados para cada método para que se pudesse compará-los melhor, pois determinado algoritmo pode ter um desempenho melhor com um certo GSM. Os valores utilizados foram 0,05, 0,1, 0,15, 0,2, 0,5 e 1,0, pois foi observado que quanto maior o GSM, menos diferenças os resultados apresentam. O melhor algoritmo será aquele cuja média dos resultados obtidos com esses valores é melhor.

### 4.2.1 Coleção de artigos da Wikipedia

Foram selecionados doze (12) artigos da Wikipedia, previamente classificados em cinco classes diferentes, de acordo com o assunto abordado no texto: *animals* (2 artigos), *languages* (3 artigos), *moons* (2 artigos), *tv-shows* (3 artigos) e *video-games* (2 artigos). Esses artigos foram processados pela instância do *framework* para textos e foram usados os algoritmos Cliques, *Stars*, *Full-stars* e *Best-star*. Os resultados foram avaliados através do *macroaverage* das medidas de revocação, precisão e Medida-F.

Como esperado, os melhores resultados surgiram com o uso do algoritmo *Best-star*, com GSM entre 0 e 0,10. A partir do GSM 0,15 a qualidade dos resultados começa a cair para esse algoritmo, enquanto que o último GSM bom para ele (0,10) mostrou-se o melhor para os outros algoritmos, resultando em agrupamentos 100% corretos – exceto para o *Full-stars*, cujo resultado foi o melhor usando o algoritmo. A partir disso conclui-se que o *Best-star*, além de dar ótimos resultados sem necessidade de especificar um GSM (usando-se o valor 0), também pode ser utilizado para descobrir o GSM ótimo para uma determinada base de dados a serem agrupados com outros algoritmos.

No gráfico da Figura 4.3 pode se ver os resultados obtidos com o algoritmo *Best-star*, onde o eixo horizontal indica o GSM utilizado e o vertical representa os graus de qualidade obtidos por cada métrica de avaliação utilizada.

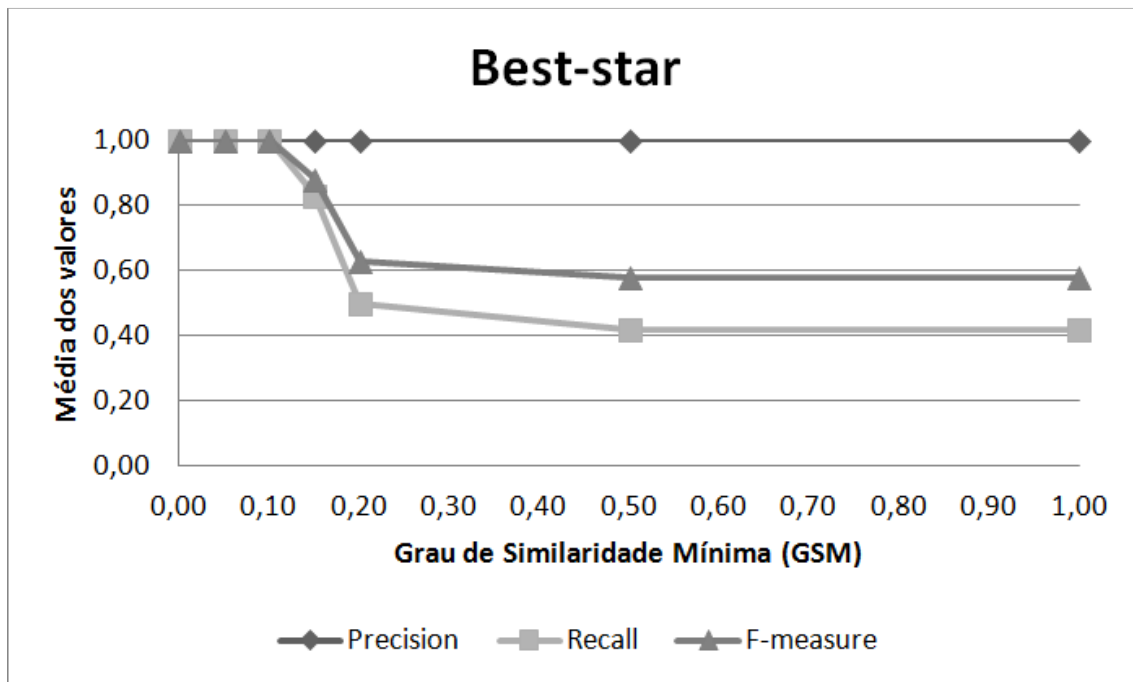


Figura 4.3: Gráfico dos resultados para o algoritmo *Best-star*.

Fonte: elaborada pelo autor.

Os algoritmos *Cliques* e *Stars* obtiveram os mesmos resultados, porém esse é um caso específico para os documentos utilizados nesse estudo de caso e nem sempre isso acontece. Geralmente o algoritmo *Cliques* obtém grupos mais rigorosos do que o *Stars* por ser um algoritmo mais rigoroso que exige que cada documento seja similar a todos os outros documentos do mesmo grupo. Porém, como os grupos obtidos com o *Stars* já respeitavam essa restrição, os resultados foram os mesmos. Estes resultados podem ser vistos no gráfico da Figura 4.4.

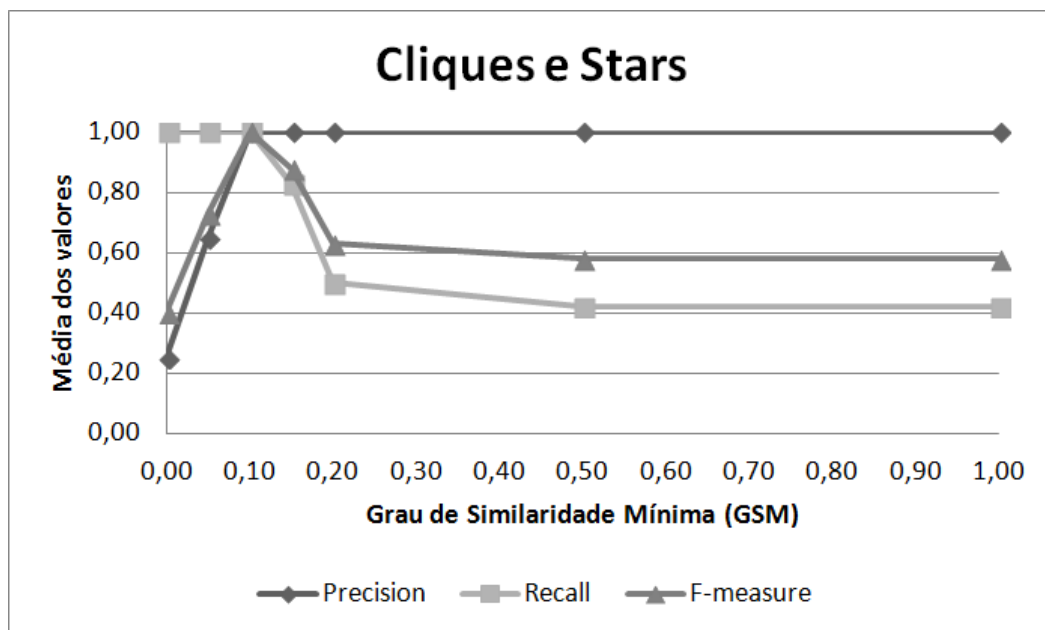


Figura 4.4: Gráfico dos resultados para os algoritmos *Cliques* e *Star*.

Fonte: elaborada pelo autor.



O algoritmo *Full-stars* foi o que obteve os piores resultados como se pode ver no gráfico da Figura 4.5. Os valores obtidos com GSM entre 0 e 0,5 foram inferiores aos outros algoritmos, enquanto os valores obtidos com GSM de 0,5 até 1,0 foram idênticos.

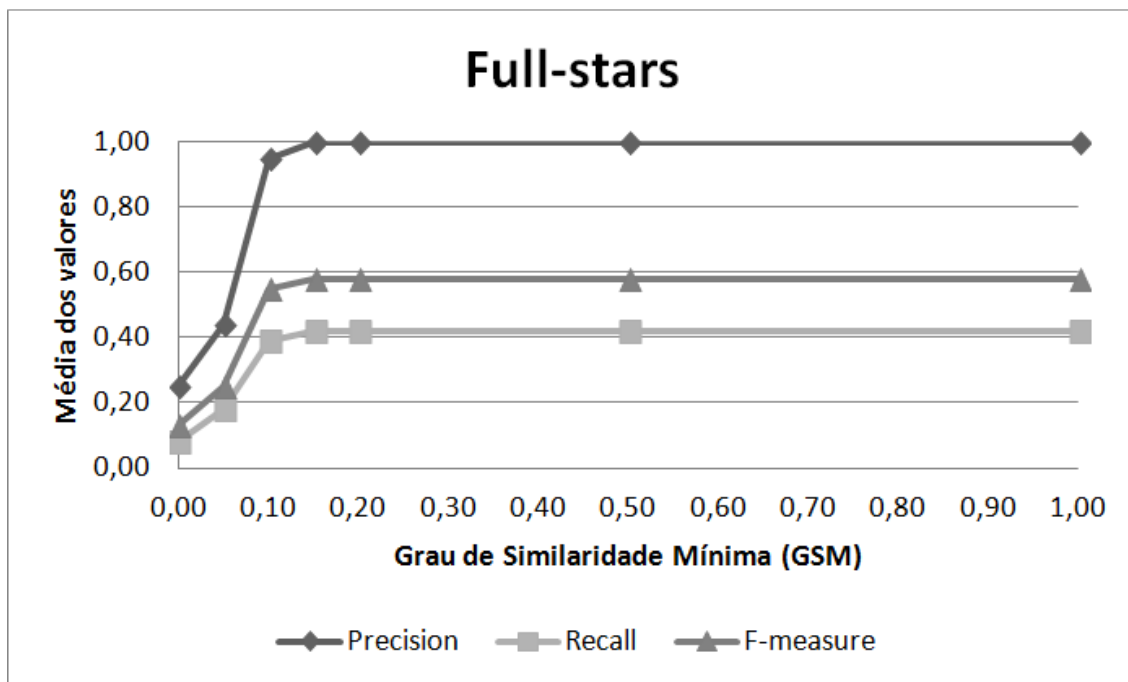


Figura 4.5: Gráfico dos resultados para o algoritmo *Full-stars*.

Fonte: elaborada pelo autor.

Os mesmos testes foram feitos usando a ferramenta Eureka e, para a coleção de artigos selecionada, todos os resultados foram os mesmos. Os fatores que poderiam criar uma diferença nos resultados seriam a extração de características e o cálculo de similaridade. O método de extração utilizado nesse trabalho é baseado no do Eureka, por isso é aceitável que se obtenha resultados similares. Já a função de similaridade utilizada é a mesma, então é esperado que os resultados sejam os mesmos.

#### 4.2.2 Subconjunto de documentos da coleção Reuters-21578

Para um segundo teste, foi escolhido o menor subconjunto de documentos da coleção Reuters-21578, por ter sido utilizada por Wives (1999) e já possuir resultados para comparação. A versão utilizada da coleção é a *Reuter-21578, distribution 1.0*, que foi obtida através do site <http://www.daviddlewis.com/resources/testcollections/reuters21578/> e o subconjunto de documentos utilizado é composto pelos 722 documentos de teste da subcoleção *HAYES SPLIT*.

A coleção foi processada com o algoritmo Best-star com um GSM de 0,0, pois essa configuração se mostrou a mais eficiente na seção anterior. O processo resultou em 653 dos documentos agrupados em 245 grupos. Com 90% dos documentos agrupados, isto é, documentos que estão em um grupo com pelo menos outro documento, o resultado foi abaixo do esperado, pois usando os mesmos parâmetros no Eureka obteve-se 95%, com 687 documentos agrupados, porém, ainda é um resultado aceitável.

Em seguida, utilizou-se o método Silhueta para avaliar os agrupamentos resultantes. Como descrito na Seção 2.2.4, este método combina as métricas de coesão e separação e

resulta em um valor no intervalo  $[-1,1]$ , onde, para um determinado documento, valores mais próximos de 1 indicam que o documento está em seu grupo natural valores mais próximos de -1 indicam que ele provavelmente deveria estar no agrupamento vizinho, e zero indica que ele está na fronteira entre dois grupos naturais.

Calculando-se a Silhueta média por grupo, obteve-se um coeficiente negativo para 209 *clusters* e positivo para 36, com uma média total de -0,51. Isso significa que a maioria dos grupos possuem documentos que são mais similares a outro grupo em comparação com o próprio. Os grupos calculados pelo algoritmo *Best-star* são pequenos e por isso é possível que os grupos naturais tenham ficado fragmentados em grupos menores, e até mesmo misturados com fragmentos de outras classes. Um estudo mais detalhado pode ser feito a respeito desse resultado em trabalhos futuros.

## 5 CONCLUSÃO

O desenvolvimento desse trabalho teve como objetivos apresentar um estudo sobre técnicas de agrupamento de dados e propor uma solução de implementação para experimentos através de um *framework* de forma a facilitar o desenvolvimento de novos algoritmos. Além disso, uma instância do *framework* foi desenvolvida com aplicação para agrupamento de textos e, através de métricas de avaliação, alguns resultados foram comparados entre os algoritmos implementados e a ferramenta Eureka.

O *framework* foi desenvolvido com algumas limitações em mente: existem muitas classes diferentes de algoritmos e não foi possível estudar todas, por isso o *framework* precisará ser adaptado para funcionar com algoritmos que seguem uma lógica diferenciada. A maioria dos algoritmos estudados é da classe *graphic-theoretic* e um estudo mais aprofundado a respeito das outras classes existentes é necessário para se chegar em um modelo mais flexível. Por outro lado, o *framework* se mostrou bastante eficiente para a implementação dos algoritmos propostos sem maiores problemas.

A aplicação para agrupar textos utilizando o *framework* proposto foi de fácil implementação, visto que o agrupamento de textos foi fortemente considerado quando do desenvolvimento da arquitetura do *framework* e fazia parte da proposta desse trabalho desde o início. O *framework* foi desenvolvido com base nos algoritmos da classe *graphic-theoretic* e, portanto, suporta os algoritmos dessa classe, mas algumas adaptações podem ser necessárias para que algoritmos de outras classes possam ser usados. Um estudo mais abrangente pode ser feito a respeito das técnicas de agrupamento existentes e melhorias podem ser feitas na arquitetura proposta, para que seja o mais flexível possível, em um trabalho futuro.

Um problema previsto durante o desenvolvimento do *framework* foi o fato de ele suportar qualquer tipo de dados que se deseje agrupar. Para não impedir que outros tipos de dados fossem utilizados, decidiu-se que as peculiaridades da aplicação com dados do tipo texto seriam implementadas de forma que as estratégias deveriam ser implementadas especificamente para esse tipo, sem ter o nível de generalização desejado inicialmente. Para se ter esse nível de generalização no nível das estratégias, algumas modificações devem ser feitas, como por exemplo, o padrão de projeto *Abstract Factory* pode ser usado para a criação de novos objetos que possuam um comportamento genérico especificado que possa ser usado pelas estratégias independente do tipo implementado. Uma solução alternativa pode ser extrair as características de um objeto de qualquer tipo que não seja texto (imagens, por exemplo), e transformá-las em termos, de forma que as soluções implementadas para os dados do tipo texto possam ser reaproveitadas.

Os resultados obtidos com a coleção de artigos da Wikipedia foram satisfatórios, as medidas utilizadas para avaliar os agrupamentos foram eficientes e indicaram o

esperado. Porém, o agrupamento feito com os documentos da coleção Reuters não obteve a qualidade esperada, diferindo dos resultados obtidos com a ferramenta Eureka, embora a aplicação desenvolvida neste trabalho fosse baseada na mesma. Da mesma forma, a métrica utilizada, Silhueta, não indicou um bom agrupamento, e um estudo mais aprofundado do caso é necessário para se descobrir se há como melhorar o agrupamento resultante. Esse pode ser um problema inerente à base de dados utilizada e independente dos algoritmos utilizados a métrica pode não obter resultados satisfatórios. Uma experiência que pode ser feita em trabalhos futuros é calcular a Silhueta da coleção considerando as classes que foram definidas em outros trabalhos para comparar com o resultado obtido. Como essas classes já foram validadas, sabemos que elas estão corretas, portanto se a Silhueta dessa classificação for similar à obtida neste trabalho, significa que os grupos obtidos não são necessariamente ruins. Outra possibilidade pode ser que alguma peculiaridade do algoritmo utilizado acabou causando esse resultado para essa coleção específica de documentos, mesmo obtendo bons resultados com a outra coleção. Essas hipóteses podem ser testadas em trabalhos futuros.

Alguns tópicos não puderam ser explorados neste trabalho e podem ser tratados em trabalhos futuros. Como mencionado, fazer um estudo mais abrangente sobre as classes de algoritmos de agrupamento de dados existentes de forma a generalizar a arquitetura do *framework*; usar o padrão de projeto *Abstract Factory* e/ou algum outro para generalizar as classes básicas que representam os objetos e suas características; implementar outros algoritmos e funções de similaridade aqui descritas; e incluir os passos de abstração dos grupos, através da atribuição de um descritor para cada um, e de avaliação, utilizando as métricas estudadas e outras.

## REFERÊNCIAS

- ESTER, M. et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. **Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)**. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.1980>> Acesso em: out. 2012.
- EVERITT, Brian. **Cluster Analysis**. New York: John Wiley & Sons, 1974.
- GAMMA, E. et al. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.
- GOLDBERG, J.; GORDON, S.; GREENSPAN, H. Unsupervised image-set clustering using an information theoretic framework. **IEEE Trans. on Image Processing**, v. 15, n. 2, p. 449-458, 2006.
- GUYON, I.; ELISSEEFF, A. An Introduction to Feature Extraction. **Feature Extraction: Foundations and Applications Studies in Fuzziness and Soft Computing**. Springer Verlag, p. 1 25. ISBN 3540354875. Disponível em: <<http://eprints.pascal-network.org/archive/00002475/01/IntroFS.pdf>> Acesso em: nov. 2012.
- HALL, M. et al. The WEKA Data Mining Software: An Update. **SIGKDD Explorations**, v. 11, n. 1, p. 10-18, 2009. Disponível em: <<http://dl.acm.org/citation.cfm?id=1656278>> Acesso em: dez. 2012.
- JAIN, A. K. Data Clustering: 50 Years Beyond K-Means. **To appear in Pattern Recognition Letters**, 2009.
- JAIN, A. K.; DUBES, R. C. **Algorithms for Clustering Data**. New Jersey: Prentice-Hall, 1988. Disponível em: <<http://www.dleex.com/read/4451>>. Acesso em: set. 2012.
- JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data Clustering: A Review. **ACM Computing Surveys**, v.31, n.3, p. 264 323, set. 1999.
- KOWALSKI, G. J. MAYBURY, M. T. **Information storage and retrieval systems: Theory and Implementation**. New York: Kluwer Academic Publishers, 2002.
- LEWIS, D. D. **Representation and Learning in Information Retrieval**. Amherst: University of Massachusetts, Department of Computer and Information Science, 1992. PhD Thesis. Disponível em: <<http://ciir.cs.umass.edu/pubfiles/UM-CS-1991-093.pdf>> Acesso em: dez. 2012.

MACKAY, D. Chapter 20: An Example Inference Task: Clustering. **Information Theory, Inference and Learning Algorithms**. Cambridge University Press. p. 284-292. ISBN 0-521-64298-1. MR 2012999. Disponível em: <<http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/284.292.pdf>> Acesso em: out. 2012.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to Information Retrieval**. Cambridge: Cambridge University Press, 2008.

ROUSSEEUW, P. J. Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. **Computational and Applied Mathematics**, Amsterdam, v. 20, p. 53-65, 1987. Disponível em: <<http://www.sciencedirect.com/science/article/pii/0377042787901257>> Acesso em: dez. 2012.

RUBINOV, A. M. et al. **Classes and clusters in data analysis**. Preprint submitted to Elsevier Science, 11 July 2005. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.1833>> Acesso em: dez. 2012.

TAN, P. STEINBACH, M. KUMAR V. Chapter 8: Cluster Analysis: Basic Concepts and Algorithms. **Introduction to Data Mining**. Boston: Addison-Wesley, p. 487-568, 2006. Disponível em: <<http://www-users.cs.umn.edu/~kumar/dmbook/index.php>> Acesso em: dez. 2012.

WIVES, L. K. **Um estudo sobre Agrupamento de Documentos Textuais em Processamento de Informações não Estruturadas Usando Técnicas de "Clustering"**. Porto Alegre: Instituto de Informática da UFRGS, 1999. Dissertação de mestrado.

WIVES, L. K.; LOH, S.; OLIVEIRA, J. P. A comparative Study of Clustering versus Classification over Reuters Collection. **Proceedings of the 8th International Workshop Pattern Recognition in Information Systems**, v. 1, p. 231-236, 2008.

## APÊNDICE: UTILIZAÇÃO DO FRAMEWORK

O framework proposto neste trabalho foi desenvolvido utilizando-se a linguagem de programação Java, portanto é necessária uma instalação do ambiente de desenvolvimento Java (JDK) para que seja possível utilizá-lo. O instalador pode ser baixado gratuitamente a partir do link <<http://www.oracle.com/technetwork/java/javase/downloads/>>. Adicionalmente, é recomendado o uso de um ambiente integrado de desenvolvimento, como Eclipse ou Netbeans. Foi utilizado o ambiente Eclipse para o desenvolvimento deste trabalho, portanto esse tutorial irá se basear neste ambiente. O Eclipse pode ser baixado a partir do link <<http://www.eclipse.org/downloads/>>. Por último, deve-se baixar o pacote do *framework* a partir do link <<http://inf.ufrgs.br/~ghribacki/clustering/>>.

Após a instalação do ambiente, deve-se criar um novo projeto Java no *workspace* do Eclipse e incluir o arquivo *jar* do *framework* como biblioteca referenciada. Para fazer isso, clique no menu “File”, “New” e “Java Project”, coloque um nome no projeto e clique em “Finish”. Na aba “Package Explorer” clique com o botão direito do mouse em cima do projeto e selecione “Build Path” e “Configure Build Path”. Na janela que abrir, selecione a aba “Libraries” e clique no botão “Add External JARs...”. Selecione o pacote do *framework* que foi baixado e clique em “Abrir” e depois em “OK”. O pacote do *framework* deve aparecer na árvore do projeto, abaixo do nodo “Referenced Libraries”, como mostra a Figura 1. Nesse momento, o *framework* já estará acessível dentro do projeto e as suas classes podem ser importadas normalmente.

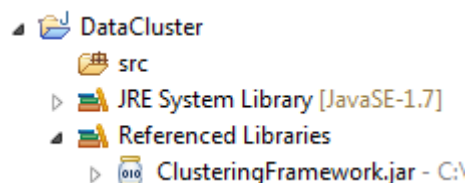


Figura 1: *ClusteringFramework* referenciado no projeto.

O primeiro passo para utilizar o *framework* é criar uma instância da classe *ClusteringProcess*. Essa é a classe principal e deve ser configurada com as implementações dos algoritmos para que possa processar os dados de entrada. Os algoritmos devem ser implementados cada um em uma classe própria e herdar das classes abstratas *FeatureSelectionStrategy*, *SimilarityStrategy* e *ClusteringStrategy*. Com a implementação pronta, o objeto *ClusteringProcess* pode ser configurado através dos métodos *setFeatureSelectionStrategy*, *setSimilarityStrategy* e *setClusteringStrategy*. É recomendado que se configure as estratégias implementadas no seu construtor ou, no mínimo, antes de se atribuí-las ao *ClusteringProcess*. Por fim, pode-se registrar classes observadoras no *ClusteringProcess* ou nas estratégias configuradas. A Figura 2 mostra um exemplo de código que configura um objeto *ClusteringProcess* e inicia a sua execução.

```

// Cria uma instância do framework.
ClusteringProcess process = new ClusteringProcess();

// Registra as estratégias que serão usadas.
process.setFeatureSelectionStrategy(new TermSelectionStrategy());
process.setSimilarityStrategy(new FuzzyMeansSimilarityStrategy());
process.setClusteringStrategy(new BestStarClusteringStrategy(0.0));

// Registra os observadores.
process.addObserver(new ConsoleObserver());

// Adiciona os objetos que serão agrupados.
// ...

// Inicia o processo de agrupamento.
process.executeClusteringProcess();

```

Figura 2: Exemplo de configuração do *framework*.

Após configurar a instância do *framework*, é necessário criar a coleção de objetos que serão processados. Seja lendo de arquivos ou de um endereço na internet, o usuário do *framework* é encarregado de converter esses dados para objetos. Esses objetos devem ser representados por uma subclasse de *DataObject* e podem ser adicionados ao *ClusteringProcess* através do método *addDataObject*. Após todos os objetos serem adicionados, o processo é iniciado através de uma chamada ao método *executeClusteringProcess*.

Uma vez iniciado o processo, o objeto *ClusteringProcess* irá notificar os seus observadores sempre que uma etapa for concluída. Quando isso acontecer, seus observadores podem usar os métodos *getProcessedSteps* para saber quantos passos já foram concluídos, *getDataObjects* para recuperar os objetos e suas características selecionadas na primeira etapa, *getSimilarityMatrix* para recuperar a matriz de similaridades calculada na segunda etapa, e *getDataClusters* para recuperar os grupos resultantes do processo. Adicionalmente as estratégias implementadas podem definir meios para os observadores recuperarem informações mais detalhadas a respeito da sua execução, mas o *framework* não especifica como isso deve ser feito e fica a cargo do usuário definir as interfaces para observação.