Mineração de texto - Conceitos e aplicações práticas

Article · November 2012

CITATIONS
0

READS
75

1 author:

Eduardo Corrêa Gonçalves
Instituto Brasileiro de Geografia e Estatística
27 PUBLICATIONS **75** CITATIONS

SEE PROFILE

Mineração de Texto

Conceitos e Aplicações Práticas

Eduardo Corrêa Gonçalves – SQL Magazine, v. 105, 2012, p. 31-44

De que se trata o artigo:

Mineração de texto é um processo que utiliza algoritmos capazes de analisar coleções de documentos texto - tais como arquivos PDF, páginas Web, documentos XML e campos CLOB ou VARCHAR de tabelas relacionais - com o objetivo de extrair conhecimento valioso. Este artigo introduz os principais conceitos e aplicações práticas da mineração de texto.

Em que situação o tema útil:

Nos últimos anos a mineração de texto tem atraído o interesse não apenas dos pesquisadores em Ciência da Computação, mas também das empresas, que procuram extrair conhecimento a partir de texto livre ou semi-estruturado com o objetivo de conquistar um melhor posicionamento no mercado. Neste artigo, apresentamos um panorama geral da área de mineração de texto para alunos e profissionais de informática que desejem estudar, trabalhar ou simplesmente conhecer um pouco mais sobre o assunto.

Resumo SQL Magazine

Este artigo apresenta os conceitos fundamentais sobre mineração de texto. Inicialmente, o artigo descreve o cenário que motivou o surgimento da área no final dos anos 90. A seguir, mostra-se os problemas práticos que a mineração de texto vem ajudando a resolver, tanto em ambiente acadêmico como dentro das empresas. Por fim, o artigo apresenta uma descrição das principais etapas envolvidas em processos de mineração de texto reais, utilizando como exemplo a tarefa de classificação de texto.

A maior parte da informação disponível no mundo não está - e, de fato, jamais esteve! - armazenada em tabelas de bancos de dados relacionais. Ao invés disso, se encontra disponibilizada digitalmente como **texto**: livros, jornais, revistas, páginas Web, blogs, perfis de redes sociais, e-mails, arquivos PDF, documentos XML, arquivos JSON, etc. No final dos anos 90, esta situação foi percebida tanto por pesquisadores como pelas empresas. Mais ou menos nesta época surgiu a seguinte ideia: "que tal analisarmos estas 'montanhas de texto digital' para que novas informações sobre nossos clientes, fornecedores, produtos e serviços possam ser reveladas e, assim, utilizadas de forma estratégica em processos de tomada de decisões?".

A ideia é bastante atraente, entretanto, nada simples de ser colocada em prática. Para começar, quando trabalhamos com dados textuais precisamos lidar com informações que, na maioria das vezes, **não possuem um esquema** para descrever a sua estrutura. Ou seja,

ao contrário do que acontece com os "bem-comportados" dados estruturados em tabelas relacionais, os dados textuais normalmente **não** estão organizados em campos, cada qual com seu tipo, tamanho e faixas de valores possíveis. Sendo assim, comparada com a informação gravada em SGBD's relacionais, a informação em formato texto é bem mais difícil de coletar, tratar, analisar e sumarizar.

Esta situação motivou o surgimento da **mineração de texto** (*text mining*), uma subárea da mineração de dados interessada no desenvolvimento de técnicas e processos para a **descoberta automática de conhecimento valioso a partir de coleções de documentos texto**. Este artigo apresenta um panorama geral desta área para todos aqueles que desejem trabalhar com mineração de texto, tanto em pesquisas acadêmicas como nas empresas, ou que simplesmente tenham curiosidade em conhecer os conceitos básicos sobre o assunto. O artigo está dividido em duas partes. Na primeira, denominada "Tarefas de Mineração de Texto", são apresentadas as mais importantes **aplicações práticas** desta tecnologia nos dias atuais. A parte seguinte, denominada "Mineração de Texto Passo-a-Passo", é composta por um conteúdo mais técnico, consistindo na apresentação de um exemplo que descreve as etapas executadas durante um processo de mineração de texto.

Tarefas de Mineração de Texto

As tarefas de mineração de texto podem ser entendidas como as diferentes **categorias de problemas** que podem ser resolvidos através de processos de mineração de texto. Esta seção introduz as mais importantes através de uma abordagem simples e prática: apresentando não apenas as tarefas propriamente ditas, mas também uma série de exemplos de aplicações reais que podem ser solucionadas com o uso das mesmas. Durante a leitura da seção, você notará que o texto a ser minerado pode estar armazenado em dois diferentes formatos:

1. **Texto Livre**: trata-se de texto escrito em alguma linguagem natural - como Português, Inglês, Italiano, etc. – que contém pouca ou nenhuma marca de estruturação. Alguns exemplos: artigos de revista, capítulos de livro, texto do corpo de um e-mail, arquivos PDF, entre outros. Nesta categoria também podem ser incluídos os campos descritivos de tabelas de bancos de dados relacionais, como CLOB, Memo e VARCHAR, já que eles são criados com o intuito de armazenar texto livre. Em muitos sistemas reais, os textos armazenados nestes campos são consideravelmente longos e complexos (ex: um campo CLOB utilizado para armazenar a descrição de um atendimento em um sistema de *help desk*).

2. **Texto Semi-Estruturado**: documentos que, mesmo sem possuir um esquema rígido para validar seus dados, contêm alguma estrutura. Os dois principais exemplos são os documentos XML, onde as informações encontram-se demarcadas entre *tags*, e os arquivos JSON, que são compostos por pares atributo/valor.

Nos itens a seguir, as principais tarefas de mineração de texto são introduzidas e suas aplicações práticas são relacionadas e comentadas.

Classificação de Texto

Esta é, provavelmente, a tarefa de mineração de texto mais conhecida e utilizada. O objetivo é realizar a associação automática de documentos texto a uma determinada classe, pertencente a um conjunto pré-definido de classes. Um exemplo de classificador de texto bastante conhecido e bem-sucedido encontra-se nos programas para **filtragem de spam**. A partir da análise do assunto e do texto de uma mensagem, o programa utiliza um algoritmo classificador de texto para identificar automaticamente se esta mensagem deve ser classificada como "normal" ou "*spam*" (Figura 1).

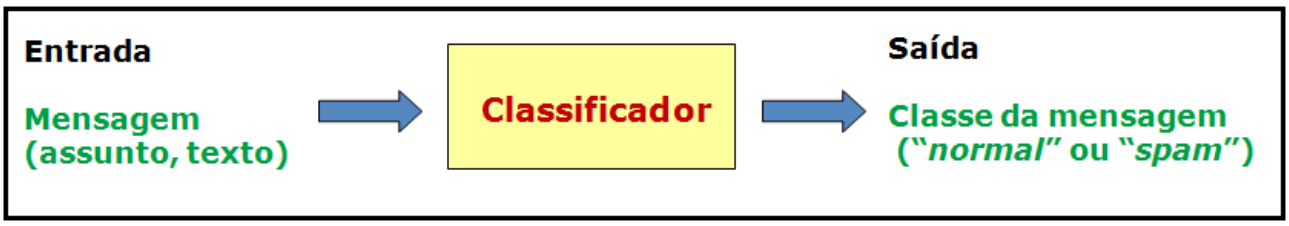

Figura 1. Classificador de *spam*.

Outro exemplo interessante de aplicação prática para a classificação de texto é descrito em [1]. Trata-se do algoritmo utilizado pelo Internet Explorer para **identificar o idioma** de uma página Web. Esse algoritmo é disparado sempre que o IE se depara com uma página que não possua informação sobre o seu *Content-Type*, ou seja, uma página que não informe explicitamente o idioma do texto e a codificação utilizada para representá-lo (UTF-8, latin-1, etc.). Se a página contiver caracteres específicos de uma determinada codificação X (ex: japonês) e o IE utilizar uma codificação Y (ex: russo) para interpretá-la, o resultado é bem conhecido de todos nós: a exibição de "lixo" na tela ou de um monte de pontinhos de interrogação "???? ??? ??". Mas como o IE procede para classificar o idioma da página? A ideia básica consiste em tentar "advinha-lo" baseado na frequência com que determinadas sequencias de bytes ocorram no texto. Pelo fato de cada linguagem humana possuir um padrão distinto de uso de letras, o IE "conhece" alguns conjuntos de sequencias associadas a textos escritos em diferentes idiomas. Deste modo, basta com que o IE compare as sequências encontradas na página com as sequências por ele conhecidas para classificar o idioma da página.

Na filtragem de *spam* e na classificação de idiomas podemos associar apenas uma classe ao documento. Um e-mail recebido deve ser classificado como "normal" ou "*spam*", mas nunca pode ser classificado com ambos os rótulos! Por isso, esse tipo de aplicação é chamado de **classificação monorrótulo** (*single-label classification*) [2], [3]. No entanto, também existem as aplicações de **classificação multirrótulo** (*multi-label classification*) [4], onde o texto pode ser associado a uma ou mais classes. Um exemplo é a **atribuição automática de tópicos** (*topic tagging*). Nesta aplicação, o objetivo é rotular um texto com um ou mais tópicos, de acordo com os assuntos por ele abordados. Na prática, um portal de notícias poderia empregar um algoritmo deste tipo para conseguir com que as notícias recebidas pudessem ser automaticamente distribuídas pelas seções mais adequadas. A Figura 2 ilustra um exemplo: o texto da reportagem sobre as Olimpíadas de 2016 é analisado pelo algoritmo classificador e rotulado com os seguintes tópicos: "Esporte", "Rio

de Janeiro" e "Economia", podendo assim ser enxergado e acessado pelos usuários através de três seções distintas do portal.

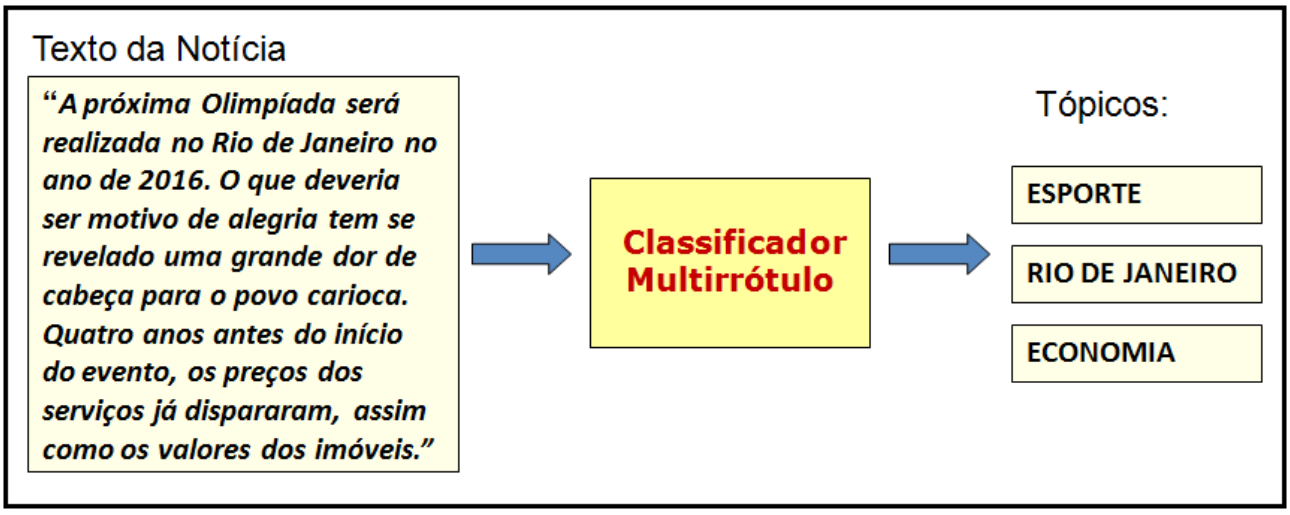

Figura 2. *Topic Tagging* – uma aplicação da classificação multirrótulo.

Além das aplicações que acabamos de apresentar, existem outros usos interessantes para a classificação de texto, conforme descrito nas referências [2], [5], [6]. Alguns exemplos são: **identificação de estilos de escrita** (o objetivo é identificar o autor de um texto anônimo), **atribuição de palavras-chave** (processo quase idêntico ao de *topic tagging*. Porém, neste caso, o algoritmo analisa um texto técnico ou científico e tenta associar o conjunto mais adequado de palavras-chave) e **classificação do propósito de hiperlinks** (documentos podem ser ligados por hiperlinks devido a vários propósitos, como recomendação, propaganda, crítica, etc.).

Nota SQL Magazine 1. Como Funciona a Classificação?

Você já sabe que os algoritmos de classificação (monorrótulo ou multirrótulo) podem ser utilizados em diversas aplicações práticas. Mas, talvez, esteja curioso para saber como esses algoritmos funcionam, ou seja, entender como o processo de classificação de textos é executado. De uma maneira simplificada, podemos dizer que o procedimento geral é composto pelas duas etapas descritas a seguir:

- Treinamento (Etapa 1): o algoritmo processa uma **base de treinamento**, composta por textos já classificados. Como resultado, ele constrói (ou "aprende") um modelo capaz de classificar novos textos (textos cuja classe é desconhecida).

- Teste (Etapa 2): após o treinamento, o modelo precisa ser testado para que sua **acurácia** seja avaliada. Isso normalmente é feito com o uso de uma **base de teste**, que também possui textos classificados. A acurácia do modelo de classificação representa a porcentagem de observações do conjunto de teste que são corretamente classificadas pelo modelo construído na fase de treinamento. Se a acurácia for alta, o classificador é considerado eficiente, podendo então ser colocado em produção.

Existem diversos algoritmos que podem ser utilizados para a construção de modelos classificadores, porém os mais comumente empregados na mineração de texto são os algoritmos SVM, k-NN e Naïve Bayes. Informações completas sobre estes algoritmos podem ser obtidas em [2], [3]. Na seção "Mineração de Texto Passo-a-Passo", apresentamos uma breve introdução ao algoritmo k-NN e a algumas técnicas adicionais que costumam ser empregadas em processos de classificação de texto.

Agrupamento de Documentos (*Clustering*)

Esta tarefa consiste em dividir automaticamente uma coleção de documentos texto em grupos (*clusters*) de acordo com algum **relacionamento de similaridade** entre os mesmos. A organização dos documentos em cada *cluster* deve ser feita de forma que haja:

- **Alta similaridade** entre os documentos pertencentes a um **mesmo *cluster***.
- **Baixa similaridade** entre elementos que pertencem a ***clusters* diferentes**.

Esta tarefa possui a **mineração de documentos XML** (*XML mining*) como uma de suas principais aplicações práticas [7]. É possível realizar tanto a mineração da estrutura (ex: agrupar documentos com DTD's similares), como a mineração do conteúdo de um documento (ex: analisar a similaridade de um conjunto de documentos em função do conteúdo dos elementos). Para que o conceito fique mais claro, apresentaremos um exemplo prático envolvendo os documentos XML representados nas Listagens 1, 2 e 3. Os documentos "Tao.xml" (Listagem 1) e "Utopia.xml" (Listagem 3) não possuem estrutura idêntica, porém ambos representam informações sobre livros. Já o documento "Brasil.xml" (Listagem 2) armazena informações sobre um conceito inteiramente distinto (dados geográficos de um país).

```
<?xml version="1.0"?>
<produto tipo="livro">
   <titulo>Tao Te Ching</titulo>
   <autor>Lao-Tsé</titulo>
   <assunto>Religião</assunto>
</produto>
```

Listagem 1. Documento "Tao.xml"

```
<?xml version="1.0"?>
<pais sigla="BR">
   <nome>Brasil</nome>
   <populacao>196.655.014</populacao>
</pais>
```

Listagem 2. Documento "Brasil.xml"

```
<?xml version="1.0"?>
<livro>
   <titulo>Utopia</titulo>
   <ano>1516</ano>
   <autor>
      <nome>Thomas More</nome>
      <pais>Inglaterra</pais>
   </autor>
   <assunto>Filosofia</assunto>
   <assunto>Política</assunto>
<livro>
```

Listagem 3. Documento "Utopia.xml"

Um algoritmo para determinação de agrupamentos poderia processar os três documentos e identificar que "Tao.xml" e "Utopia.xml" **possuem estrutura muito similar**, associando-os a um mesmo *cluster* (por exemplo, *cluster 1*). De maneira análoga, o algoritmo identificaria que o documento "Brasil.xml" não é similar a nenhum dos outros dois, alocando-o a um *cluster* diferente (*cluster 2*). Esta situação é representada na Figura 3.

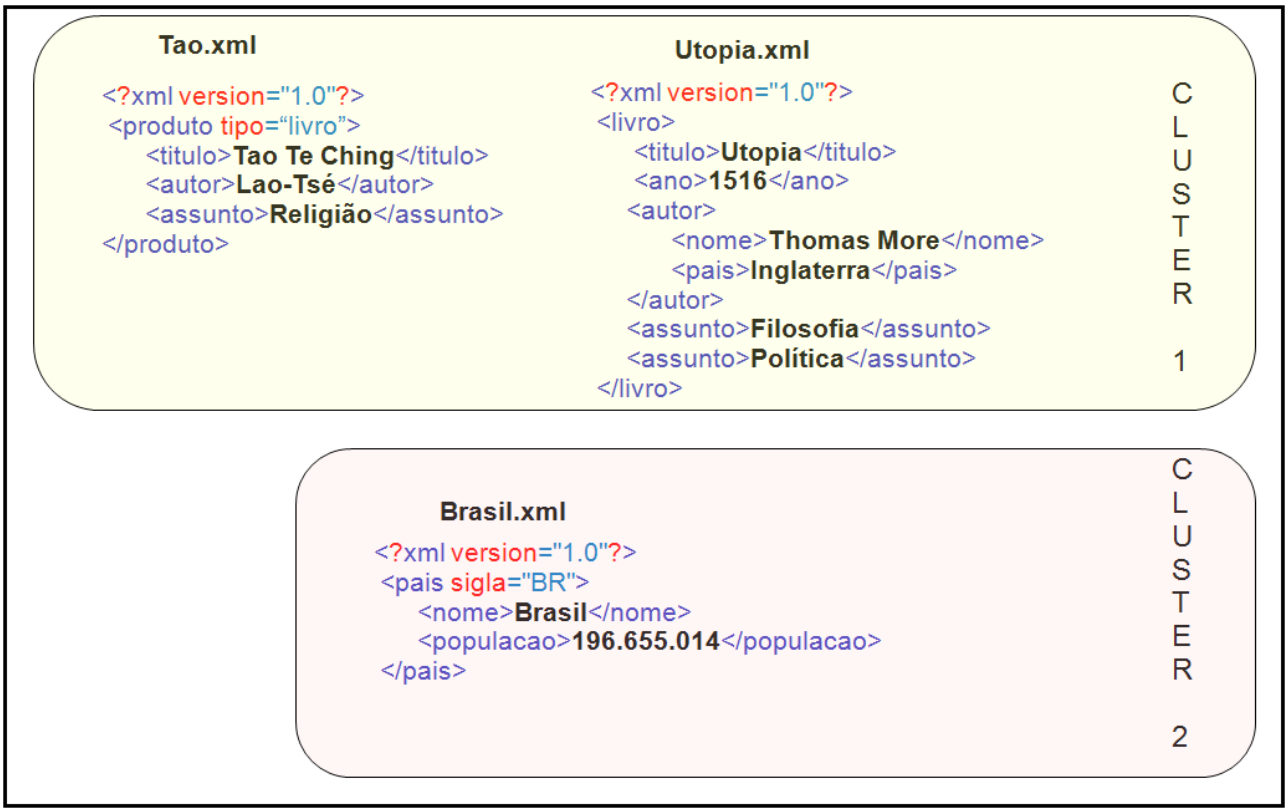

Figura 3. Agrupamento de documentos XML.

É importante observar que, na realidade, o algoritmo não "sabe" que os dois primeiros documentos contêm dados sobre livros e que o segundo contém informações

sobre um país. Na realidade, ele é absolutamente ignorante quanto ao tema de cada documento, pois ele não realizou nenhum tipo de análise semântica. A tarefa que, de fato, o algoritmo executa consiste em aplicar algum tipo de técnica capaz de **medir a similaridade** entre cada par de documentos. Quando ele faz o **cálculo da similaridade** entre "Tao.xml" e "Utopia.xml", obtém um **valor alto** e, por consequência, aloca ambos em um mesmo *cluster*. De maneira oposta, quando realiza-se o cálculo da similaridade entre "Brasil.xml" e "Tao.xml" e entre "Brasil.xml" e "Utopia.xml", o resultado é um valor baixo. Por isso, "Brasil.xml" não pode ser colocado no mesmo *cluster* dos outros dois documentos.

Embora o exemplo apresentado nesta seção tenha envolvido apenas 3 documentos, é evidente que na prática o objetivo é "clusterizar" um grande conjunto de documentos (centenas ou milhares). Também é importante deixar claro que a tarefa de determinação de agrupamentos não é necessariamente aplicada apenas sobre documentos XML. Ela também pode ser executada sobre texto livre. Um exemplo apresentado em [5] descreve o uso da tarefa aplicada à **detecção de plágio**. Como todos sabemos, os textos resultantes de plágio podem possuir muitas ou apenas algumas partes extraídas do texto original. Os plagiadores alteram certas palavras, assim como a ordem de alguns parágrafos ou frases. Mesmo com essas alterações, os algoritmos para análise de agrupamentos conseguem ser altamente eficientes para resolver esse problema (possuem alta taxa de acerto na tarefa de colocar os textos dos plagiadores no mesmo cluster do texto original).

Nota SQL Magazine 2. Medidas de Similaridade

O conceito de similaridade é fundamental na análise de agrupamentos. Por isso, os livros de mineração de dados dedicam um grande número de páginas descrevendo técnicas para medir similaridade entre objetos. Normalmente estes técnicas não são muito triviais, baseando-se nos mais diversos princípios matemáticos. Para aqueles que desejarem se aprofundar no assunto, destacamos o livro texto [5], um dos poucos que possui um capítulo inteiro sobre medidas de similaridade direcionadas para a mineração de texto.

Felizmente, existem algumas medidas de similaridade simples de serem utilizadas e entendidas. Um exemplo muito conhecido é a **Distância de Levenshtein** ou **Distância de Edição**. Esta medida compara duas *strings*, digamos str1 str2, e retorna o número mínimo de caracteres que devem ser modificados, inseridos ou excluídos para transformar str1 em str2. A ideia é que se duas *strings* possuem um valor baixo para a distância de edição então podemos considerá-las similares, já que transformar uma na outra requer um número baixo de operações. Veja alguns exemplos:

- str1 = '12345' e str2= '12345'
Dist. Edição = 0, já que as *strings* são iguais.

- str1 = 'baba' e str2= 'boba'
Dist. Edição = 1. Basta uma operação para transformar str1 em str2, que consiste em trocar o segundo caractere de str1 de "a" para "o".

- str1 = ‘edu’ e str2= ‘dudu’
Dist. Edição = 2. Duas operações precisam ser realizadas para transformar str1 em str2. A primeira é inserir a letra “d” no início de str1 e a segunda é trocar “e” por “u”.

Apesar de simples, a distância de edição tem se demonstrado bastante eficaz em certos tipos de aplicações práticas. Para citar um caso real, a medida foi utilizada em um sistema do IBGE para **identificar nomes de bairros** que estavam armazenados com a grafia diferente em um grande banco de dados (ex: “Jardim Europa” e “Jdim Europa”). Nesta aplicação, o algoritmo de distância de edição conseguiu identificar 90% dos casos.
Outro uso prático muito comum para a medida é a **identificação de erros de digitação** ou **erros ortográficos** em textos.

A distância de edição é tão popular que certas linguagens de programação, como o PHP, já possuem funções prontas para o seu cálculo. Para outras linguagens é fácil encontrar implementações prontas na Internet.

Extração da Informação

Consiste na tarefa de preencher *templates* a partir de documentos texto, conforme mostra o exemplo da Figura 4.

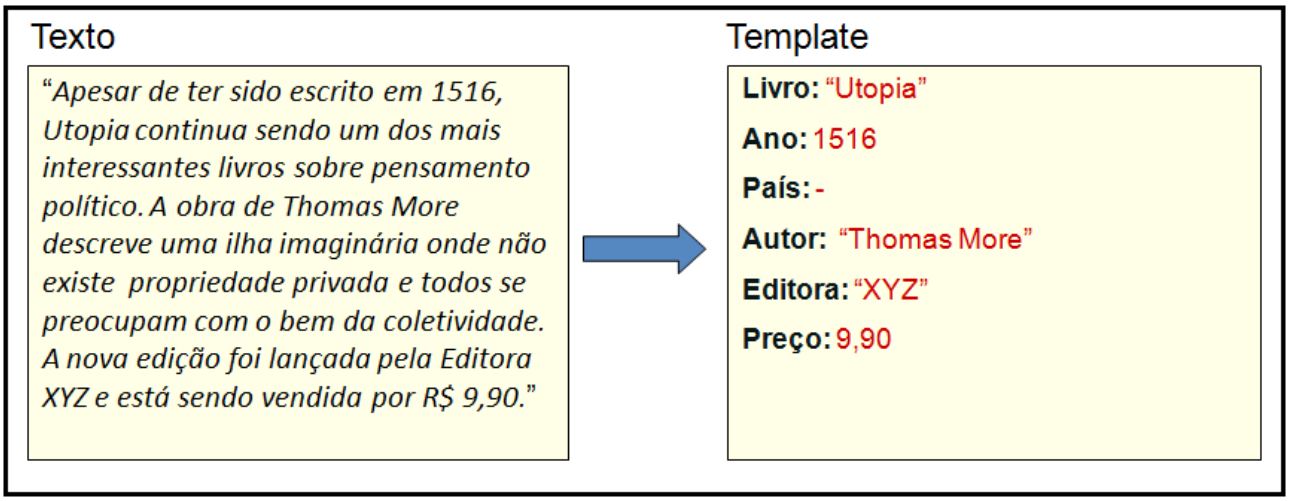

Figura 4. Extração da Informação.

Os sistemas de extração da informação utilizam técnicas para **interpretar** a informação contida em textos e, assim, **extrair as partes relevantes** para uma necessidade específica. As informações extraídas são geralmente inseridas em tabelas de bancos de dados, para que possam ser posteriormente analisadas através da SQL, planilhas, ou até mesmo da mineração de dados tradicional.

Para preencher os *templates* o sistema precisa identificar os “objetos” do mundo real dentro dos textos: datas, substantivos (nomes de pessoas, locais, empresas, coisas, etc.),

verbos, abreviações, endereços, URL's, etc. [6] Certos tipos de objetos podem ser reconhecidos com o uso de técnicas simples, como as **expressões regulares**, úteis para a identificação de datas, preços, números de telefone, CEPs, etc. Alguns tipos de substantivos, como nomes próprios e nomes de países, são identificados com o uso de **dicionários.** Também é comum a construção de dicionários de acrônimos e abreviações. Para melhorar a eficiência, o sistema pode também utilizar **regras de identificação de padrões**. Um exemplo de regra poderia ser: se existe no texto uma sequência com duas ou mais palavras possuindo apenas a primeira letra em maiúsculo, é muito provável que se trate de um nome de pessoa ou local (ex: Thomas More).

Grande parte dos sistemas de extração de informação atual se baseia no uso de **ontologias** para viabilizar a identificação dos "objetos". De maneira simples, uma ontologia pode ser definida como uma estrutura que representa as **relações entre conceitos de um determinado domínio** e os **temos associados a cada conceito** (vocabulário). O exemplo apresentado na Figura 5 mostra uma pequena ontologia sobre o domínio dos "livros".

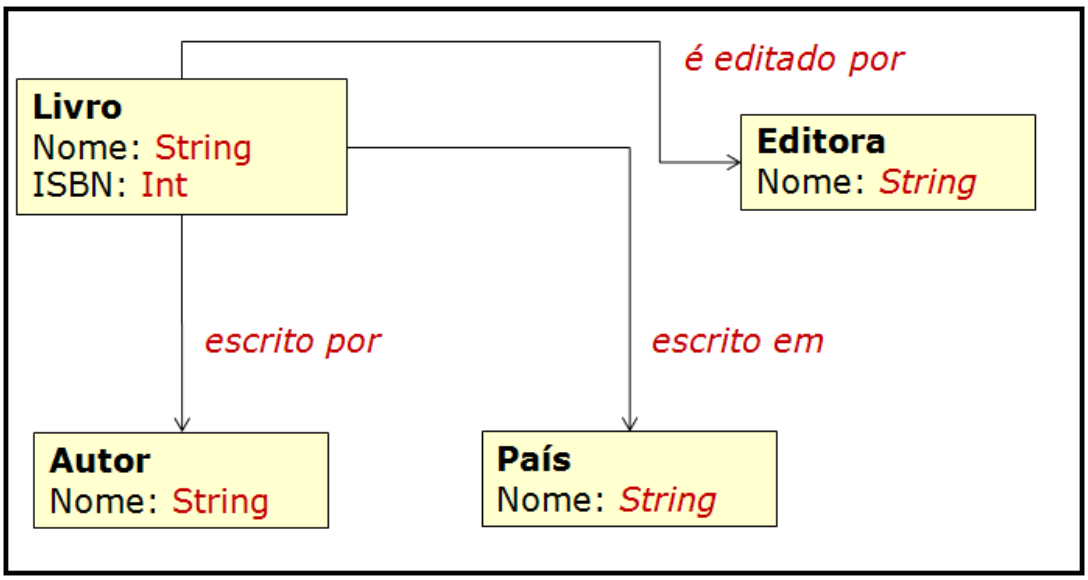

Figura 5. Uma ontologia simples

Evidentemente, o exemplo é bem simplificado. No entanto, é capaz de evidenciar o objetivo de uma ontologia: representar os conceitos e relacionamentos que são "senso comum" em um determinado domínio (no caso dos livros, são conceitos como autor, país, editora, etc.). É importante reforçar a ideia de que a ontologia sempre está representando um domínio específico e deve ser usada para resolver problemas associados a este domínio. Por exemplo: se temos uma coleção de textos sobre livros para ser submetida a um processo de extração da informação, então é necessário utilizar uma ontologia sobre livros (de nada adiantaria utilizar uma ontologia sobre esportes, por exemplo).

Na prática, as ontologias são estruturas sofisticadas, cujo processo de construção é bastante trabalhoso. Por esta razão, nos dias atuais, já existem diversas ontologias prontas para muitos domínios, como "genoma", "robótica", "terminologia médica", "biologia", entre outras. Infelizmente, a maioria destas ontologias prontas está em Inglês (ou seja, os

termos associados aos conceitos estão especificados em Inglês), não podendo ser utilizadas em processos de mineração de textos em Português. Para conhecer mais sobre ontologias consulte a referência [8].

Descoberta de Associações

Tem por objetivo descobrir combinações de palavras que ocorram com frequência significativa em uma coleção de documentos texto e realizar a análise da associação / correlação entre essas palavras. Os resultados obtidos são expressos na forma de **regras de associação** [2], [3], [5].

Considere, por exemplo, um *corpus* formado por textos sobre futebol (obs: na literatura, coleções de documentos texto são frequentemente referenciados pelo termo "*corpus*"). Um algoritmo para mineração de regras de associação poderia analisar esta coleção de textos e revelar os padrões apresentados na Figura 6:

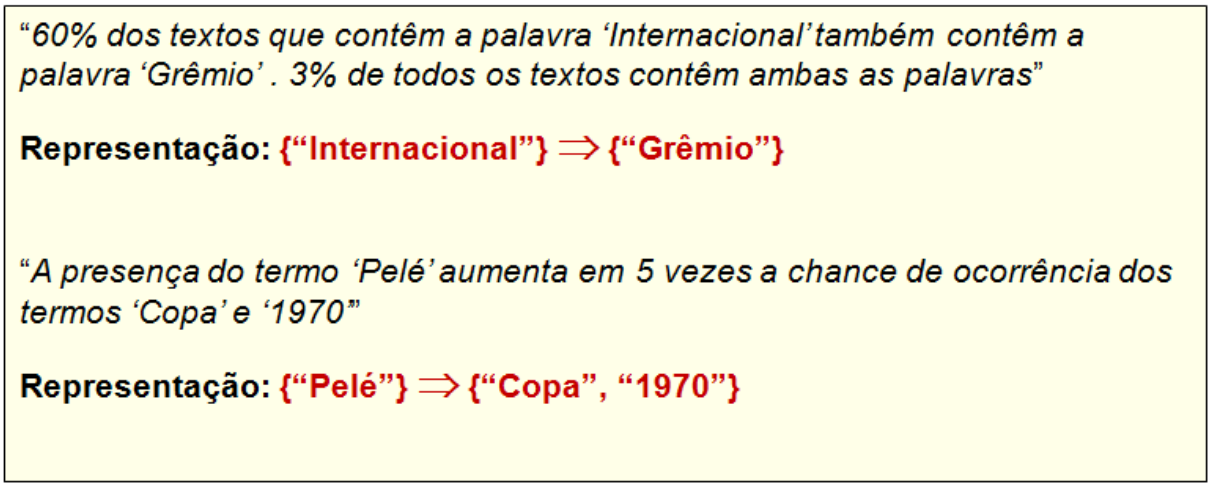

Figura 6. Regras de Associação.

Análise de Sentimentos (*Sentiment Analysis*)

Uma enorme quantidade de textos encontrados na Internet - blogs, redes sociais, twiter, etc. – reflete a **opinião de pessoas** a respeito de algum produto, serviço, programa de TV, jogo de futebol, filme, livro, discurso político, etc. A análise de sentimentos é uma nova tarefa de mineração de textos que tem por objetivo identificar a opinião, emoção e sentimento das pessoas sobre um determinado tema, a partir da análise de textos. Por esta razão, a tarefa também é comumente referenciada como *opnion mining*.

Tipicamente (mas não necessariamente), o objetivo final é **classificar a opinião** das pessoas em uma dos seguintes rótulos: "positiva", "negativa" ou "neutra" (neste caso, podemos entender a análise de sentimentos como uma aplicação da tarefa de classificação). A Figura 7 apresenta um sistema protótipo que executa este tipo de análise de sentimentos a partir de textos em inglês digitados em um formulário [9].

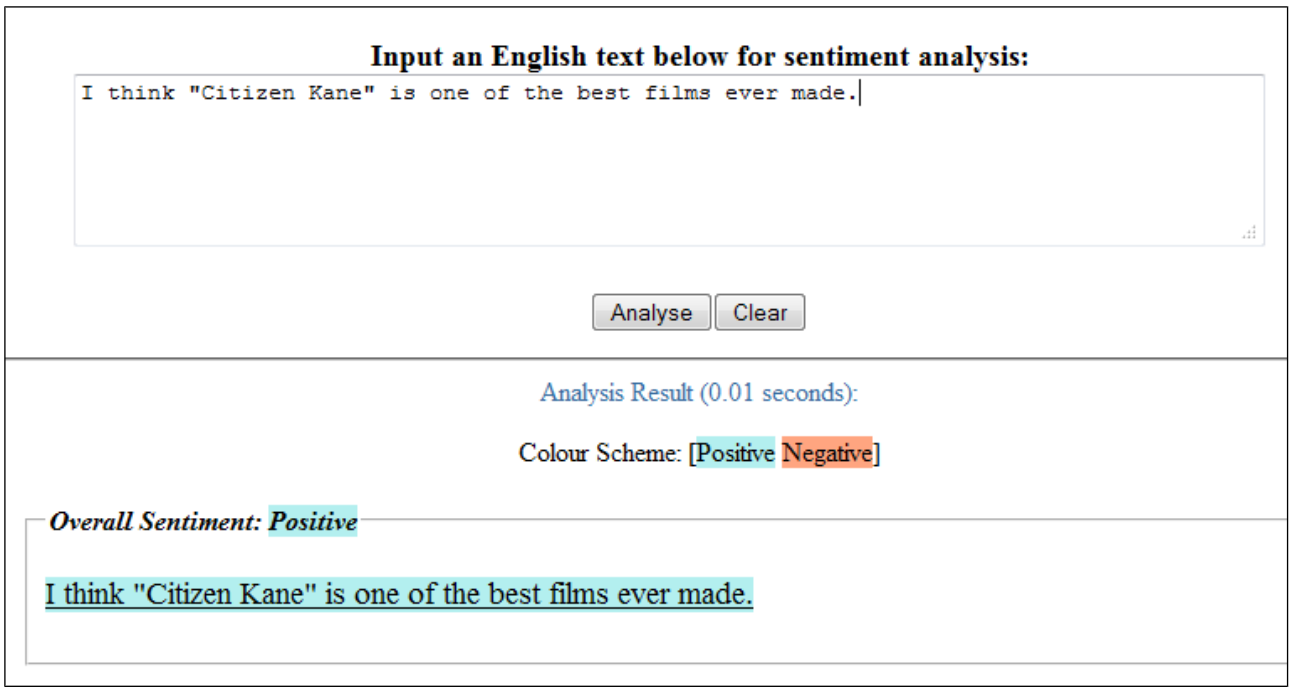

Figura 7. Análise de Sentimentos.

Casamento de Esquemas (*Schema Matching*)

Tarefa que consiste na identificação das **correspondências semânticas** existentes entre **elementos** de dois esquemas. O objetivo é mapear automaticamente os elementos que representem a mesma informação em ambos os esquemas (ou, ao menos, tornar esse processo de mapeamento semi-automatizado). Os algoritmos para casamento de esquemas são muito utilizados na prática, já que estão embutidos nos principais ***softwares* para ETL.** Neste tipo de aplicação, o objetivo é auxiliar os analistas de sistemas no árduo trabalho de integrar dados armazenados em diferentes fontes (etapa de pré-processamento do processo de construção de um *data warehouse*).

Para que o conceito de casamento de esquemas fique claro, considere o exemplo apresentado na Figura 8, onde o objetivo é casar os elementos (campos) das tabelas "FUNCIONARIO" (esquema S1) e "EMPREGADO" (esquema S2).

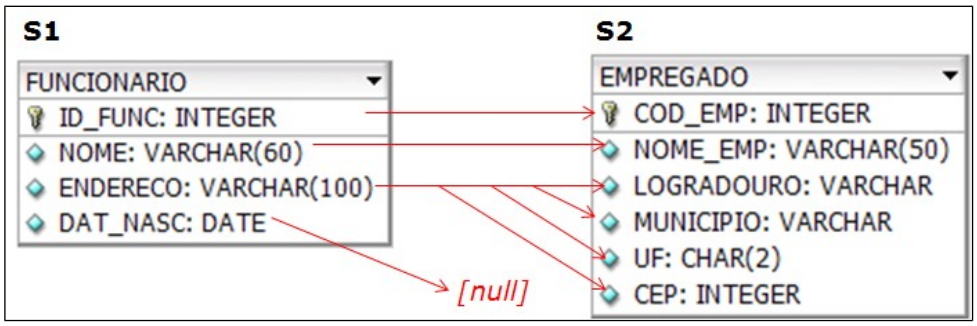

Figura 8. Casamento de Esquemas

No exemplo, o algoritmo para casamento de esquemas (denominado *matcher*) conseguiu identificar três mapeamentos:

- O campo ID_FUNC do documento S1 corresponde ao mesmo conceito representado pelo campo COD_EMP do documento S2.
- O campo NOME de S1 é equivalente à NOME_EMP de S2.
- O campo ENDERECO de S1 é equivalente à concatenação de LOGRADOURO + MUNICIPIO + UF + CEP de S2 (casamento um-para-muitos).
- Por sua vez, o campo DAT_NASC de S1 não possui correspondente em S2.

Nos processos de ETL, podemos utilizar *matchers* para nos ajudar em duas tarefas: (i) encontrar os atributos das fontes que irão compor atributos do DW e (ii) combinar atributos das fontes e criar as transformações necessárias. Por exemplo: o atributo CEP de uma tabela relacional chamada T_CLIENTE (fonte) poderia ser combinado com um arquivo texto dos correios (fonte), ambos sendo mapeados para o atributo REGIAO do DW (no caso da cidade do Rio de Janeiro, as regiões seriam: "Sul", "Norte", "Oeste" e "Centro").

As aplicações práticas do casamento de esquemas não se restringem aos processos de ETL e integração de dados. Existem inúmeras outras, sendo um exemplo interessante a aplicação conhecida como **processamento semântico de consultas**. Neste tipo de aplicação, o usuário especifica uma consulta utilizando **termos familiares** para ele e o sistema mapeia automaticamente estes termos para os objetos do BD que serão utilizados.

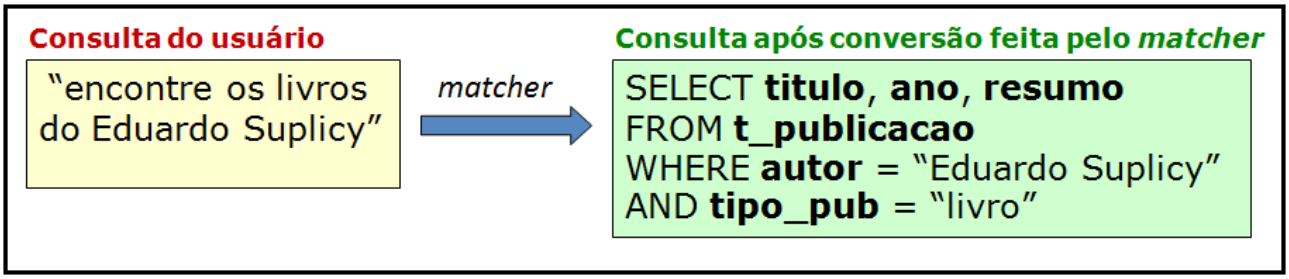

Figura 9. Processamento semântico de consultas

A princípio, pode parecer uma aplicação "boba", porém o processamento semântico pode ser usado para finalidades "nobres", por exemplo, em sistemas direcionados para pessoas com deficiência visual. Neste caso, o usuário pode falar o que deseja para o sistema. O sistema "ouve" o que foi dito pelo usuário e utiliza o *matcher* para identificar os objetos que precisarão ser acessados, podendo assim, montar a consulta SQL desejada.

Para conseguir efetuar os casamentos, o *matcher* precisa utilizar diversos tipos de técnicas, sendo algumas delas muito similares às utilizadas pelos sistemas de extração de informação (ex: dicionários de sinônimos, expressões regulares e outras). Um grande conjunto de técnicas para casamento de esquemas é descrito no trabalho [10] .

Nota SQL Magazine 3. Casamento de Dados

Além do casamento de esquemas, outra tarefa de **integração semântica** muito utilizada é o **casamento de dados** (*data matching*). Neste caso, o objetivo não é casar elementos de esquemas (campos), mas sim casar registros (conteúdo dos campos). Alguns exemplos:

- Decidir se duas tuplas de uma tabela possuem o mesmo valor.

- Decidir se os conteúdos de dois campos VARCHAR de duas diferentes tabelas de um BD possuem conteúdo similar.

A tarefa de identificação de bairros com o nome escrito em grafias diferentes, citada na nota anterior, é um exemplo prático de aplicação do casamento de dados nas empresas.

Recuperação da Informação (*Information Retrieval*)

Para encerrar nossa introdução às tarefas de mineração de texto, apresentamos a recuperação da informação (*information retrieval* – IR), cujo objetivo é **localizar** e **ranquear documentos relevantes** em uma coleção, **de acordo com as palavras-chave** digitadas em uma consulta feita por usuário. Em outras palavras: é exatamente o trabalho realizado pelos **sites de busca** da Internet.

Na realidade, a IR é uma área de pesquisa ativa desde muito tempo. Durante os anos 70, 80 e 90 esta área veio se desenvolvendo em paralelo à área de banco de dados e muitos algoritmos interessantes foram propostos. Porém, somente a partir da popularização da Web e – especialmente – da consolidação do Google como sua principal ferramenta de busca, é que estes "tesouros algorítmicos" da área de IR passaram a ser largamente utilizados. A seguir, abordamos dois conceitos importantes relacionados à tarefa de recuperação de informações: indexação e relevância.

Um mecanismo eficiente de **indexação de informações** forma o núcleo de qualquer sistema de recuperação de informações, incluindo o Google . Existem muitas técnicas para indexação de texto. Uma das mais simples consiste na construção de **índices invertidos** [2]. Uma representação bem simples para a estrutura é apresentada na Figura 10. Existe uma tabela *hash* de palavras, em que cada elemento possui uma lista que aponta para os *ids* dos documentos onde os termos aparecem. Com o uso deste tipo de estrutura é fácil resolver consultas como "encontre todos os documentos que possuam as palavras 'gol' e 'zagueiro'". Basta primeiro localizar as listas de ponteiros associados a estas palavras na tabela *hash* e fazer uma interseção dos valores constantes em cada lista.

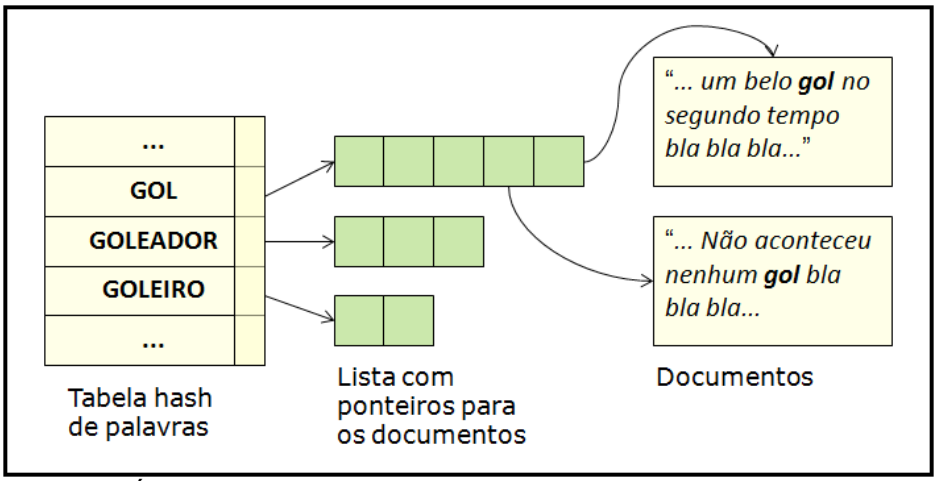

Figura 10. Índice invertido

Já o conceito de **relevância** dos resultados é bastante conhecido por nós: os sites de busca apresentam uma lista ordenada por relevância em resposta a uma consulta de usuário. Mas como os algoritmos conseguem obter esse ranking? A ideia básica consiste em casar as palavras-chave digitadas na consulta do usuário com as palavras presentes em cada documento do *corpus*. O algoritmo atribui uma **pontuação** para cada documento, baseado em quão bem ele "casa" com a consulta do usuário. Essa pontuação é tipicamente computada em função da frequência das palavras do documento e da coleção como um todo.

Nota SQL Magazine 4. Índices Full-Text

Apesar de parecer muito complexa, a tecnologia utilizada na recuperação de informações está ao alcance de qualquer desenvolvedor de sistemas, pois os SGDB's atuais permitem com que mecanismos de indexação similares aos dos sistemas de IR possam ser utilizados de forma simples e transparente. São os chamados **índices full-text**, que realizam a indexação de cada palavra que esteja armazenada em campos CLOB e VARCHAR de tabelas do banco de dados.

A criação destes índices é feita de maneira trivial (por exemplo, com um comando CREATE INDEX) e eles podem ser acessados através de funções especiais utilizadas em instruções SQL. As palavras-chave para a busca podem ser especificadas em conjunto com as cláusulas "OR", "AND" e "NOT".

Para exemplificar, considere um *schema* do banco Oracle que possua uma tabela chamada "EMPRESA". Suponha que exista um índice full-text associado ao campo "RAZAO_SOCIAL" desta tabela. Através da consulta mostrada abaixo, seria possível utilizar esse índice full-text para recuperar de forma eficiente todas as empresas cuja razão social contenha a palavra "BANCO" em conjunto com as palavras "RJ" ou "SP".

```
select * from EMPRESA
where contains(RAZAO_SOCIAL, 'BANCO and (RJ or SP)')
```

A função "**contains**" é a responsável por **acionar** o índice full-text (informar ao Oracle que desejamos fazer uso do índice).

Mineração de Texto Passo-a-Passo

Conforme mencionado na introdução deste artigo, as informações em texto plano ou semi-estruturado **não** estão representadas de uma forma que facilite o processo de análise automática, já que não dispomos de um esquema rígido para descrever os dados e guiar o processo de mineração. Além disso, para tornar a coisa mais difícil, quando se trabalha

com texto, é preciso conviver com uma série de problemas complicados para algoritmos computacionais, problemas estes inerentes aos processos de interpretação de texto: existência de sinônimos, erros ortográficos, diversidade de idiomas, recursos estilísticos (metáfora, metonímia, ironia, etc.), entre outros.

Este conjunto de dificuldades faz com que a mineração de textos precise utilizar técnicas específicas e diferentes das utilizadas na mineração de dados tradicional. Esta seção apresenta algumas destas técnicas, através de um exemplo que descreve as etapas envolvidas em um processo de classificação de texto. Porém, antes de entrar na parte prática, é importante introduzir as duas abordagens existentes para resolver problemas de mineração de texto:

- **Abordagem Estatística:** baseia-se principalmente na frequência dos termos dentro do texto, ignorando qualquer informação semântica.
- **Processamento de Linguagem Natural**: baseia-se no uso de algoritmos que tentam processar os textos da mesma forma que nós, seres humanos: através da interpretação sintática e semântica das frases. O Processamento de Linguagem Natural (PLN) é, na realidade, uma subárea da Inteligência Artificial que tenta fazer o computador entender comandos ou sentenças escritos em linguagens humanas, como Português, Inglês, etc. As origens desse campo remontam às décadas de 1940 e 1950, quando foram iniciados os primeiros esforços para a elaboração de sistemas para tradução automática [6].

A abordagem estatística é mais comumente empregada na prática e, muitas vezes, é a única apresentada em detalhes nos livros de mineração de dados [2], [5]. Isto não quer dizer que ela seja superior à PLN. Na realidade ela é menos trabalhosa de ser implementada e consegue resolver de forma eficiente a maioria das tarefas de mineração de texto, como a classificação, determinação de agrupamentos, descoberta de associações e recuperação de informação. Também existem os sistemas que combinam ambas as abordagens, PLN e estatística, para obter melhores resultados.

Esta seção introduz as principais técnicas empregadas na abordagem estatística. Não existe uma regra rígida para conduzir a execução do processo, mas é comum que ele seja dividido em três etapas: **pré-processamento**, **modelagem** e **mineração de texto** propriamente dita. Os itens a seguir apresentam exemplos de atividades executadas durante cada etapa, considerando um exemplo envolvendo a classificação de texto.

Etapa 1. Pré-Processamento

Na abordagem estatística, cada documento da coleção é modelado como um "**saco de palavras**" (***bag of words***): um vetor que registra o número de ocorrências (frequência) de cada palavra distinta do documento (Figura 11). As decisões tomadas pelo algoritmo de mineração de texto serão sempre baseadas nestes valores de frequência dos termos.

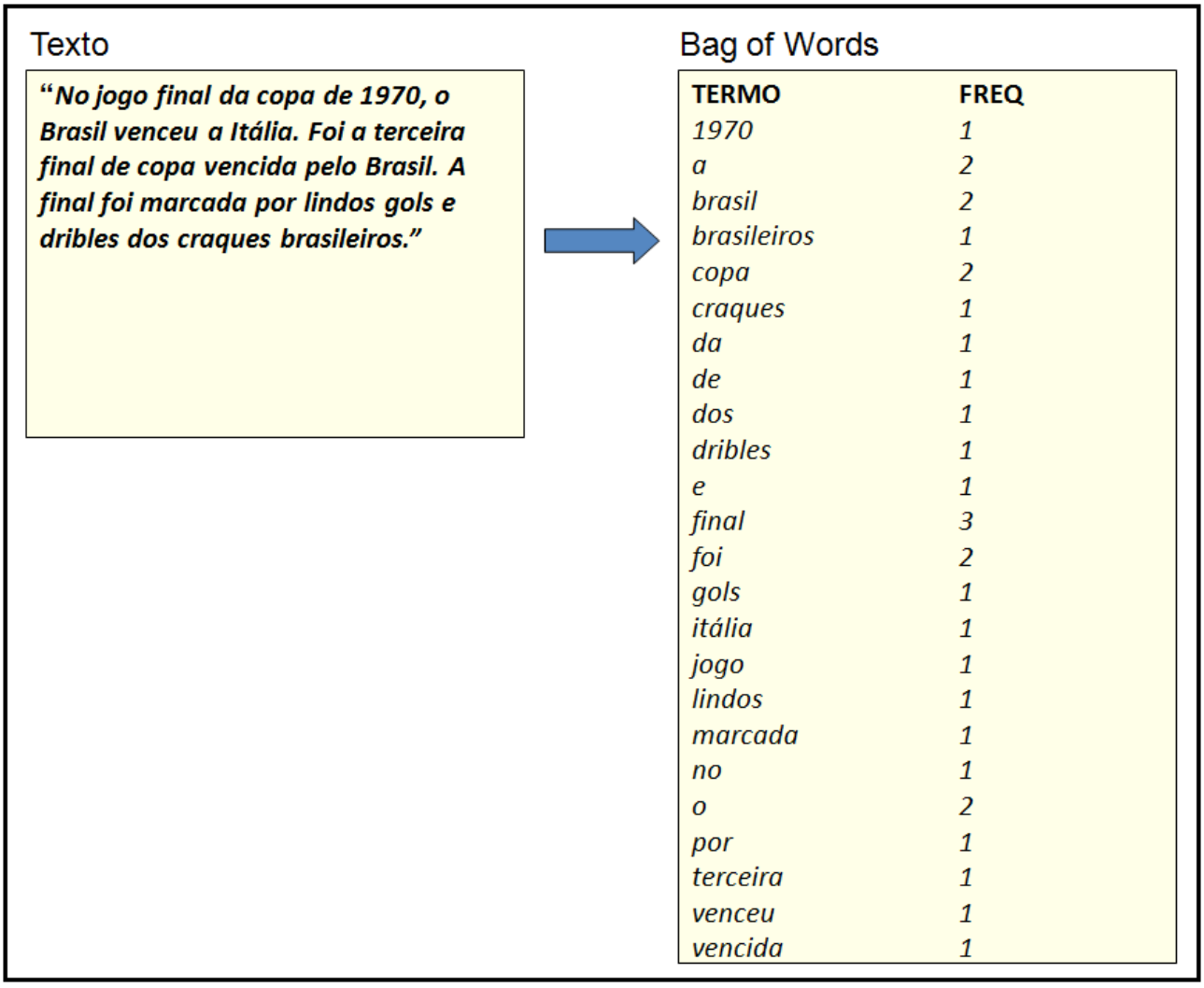

Figura 11. *Bag of words*

A etapa de pré-processamento dedica-se ao seguinte problema: "como gerar as *bag of words?"*. O primeiro passo é identificar e separar cada termo (palavras do documento), um processo conhecido como **tokenização**. A estratégia mais simples consiste em converter o texto para minúsculo (ou maiúsculo) e utilizar uma expressão regular para separar as palavras de acordo com os **sinais de pontuação** que forem encontrados. A Listagem 4 exibe um exemplo de código em Java que realiza a tokenização de uma string, transformando-a em um vetor de palavras. Note que o processo descrito causa alguns problemas para termos compostos, como "Rio de Janeiro" que seria quebrado em 3 palavras: "Rio", "de" e "Janeiro". Porém, a abordagem estatística ignora esse tipo de problema, pois ela evita ao máximo lidar com qualquer questão semântica que seja minimamente complicada.

```
String bagOfWords[] = texto.toLowerCase().split("[!\"'-\\[\\]#$%&()*+,./:;<=>?@^_`{|}~\\s]");
```

Listagem 4. Sinais de pontuação considerados no processo de tokenização

O segundo passo da etapa de pré-processamento consiste em garantir com que os

vetores *bag of words* não sejam preenchidos com palavras desnecessárias. Mas o que seriam estas palavras desnecessárias? São palavras muito frequentes, porém utilizadas apenas para ajudar a construir frases: artigos, preposições, pronomes, etc. No jargão da mineração de texto, essas palavras são denominadas ***stop words***. Isoladamente, elas não têm nenhum significado e, por isso, são consideradas irrelevantes. Na prática, os sistemas de mineração de texto utilizam ***stop lists*** – espécie de dicionário contendo a relação de *stop words*. Um exemplo de *stop list* para o idioma Português é mostrado na Listagem 5. É importante esclarecer que a definição das *stop list* é dependente do **idioma** dos textos que estão sendo analisados (uma *stop list* para Inglês teria palavras como "a", "and", "in", "I", "He", "this", enquanto a *stop list* em Português possuiria palavras como as da Listagem 5) e do **domínio** do problema (por exemplo, em uma coleção de textos sobre cinema, as palavras "filme" e "produção" poderiam ser consideradas *stop words*).

```
<?xml version="1.0"?>
<stoplist>
   <stopword>um</stopword>
   <stopword>e</stopword>
   <stopword>no</stopword>
   <stopword>eu</stopword>
   <stopword>ele</stopword>
   <stopword>este</stopword>
...
</stoplist>
```

Listagem 5. Stoplist

Um último passo que opcionalmente pode ser executado na etapa de processamento é a operação conhecida como ***stemming***. Trata-se do processo de transformar formas variantes das palavras (plural, feminino, gerúndio, etc.) para uma representação única e concisa, denominada ***stem*** (Figura 12).

Palavra	Stem
saltar	salt
saltaram	salt
saltos	salt
saltei	salt
saltou	salt
saltará	salt

Figura 12. Stems

Para obter o *stem* de uma palavra, é preciso utilizar um algoritmo conhecido como *stemmer*. Basicamente, o que o algoritmo faz é **identificar os sufixos** das palavras e removê-los. Um *stemmer* muito famoso é o Algoritmo de Porter [11], que funciona para o idioma Inglês e está embutido em muitos sistemas comerciais.

Na prática, *stemming* é um recurso bem menos usado do que as *stop lists*. Isto se justifica por dois motivos principais: (i) os *stemmers* nem sempre conseguem obter radicais corretos (ex: o algoritmo pode computar "brinc"para a palavra "brincar" e "brinq" para a palavra "brinquei") e (ii) cada idioma precisa de um *stemmer* próprio, já que cada um tem sufixos e as regras de formação diferentes para suas palavras (montar uma *stop list* é fácil, mas implementar um algoritmo de *stemming* é complicado).

Revisitando o exemplo da Figura 11, considere que agora, a *bag of words* seja gerada com a remoção das *stop words* e o uso da operação de *stemming* sobre o texto. Nesta situação, o vetor gerado no processo seria similar ao mostrado na Figura 13. Veja que estas técnicas de pré-processamento são capazes de diminuir enormemente o vetor de palavras, o que propicia ganho de desempenho para o processo de mineração de textos.

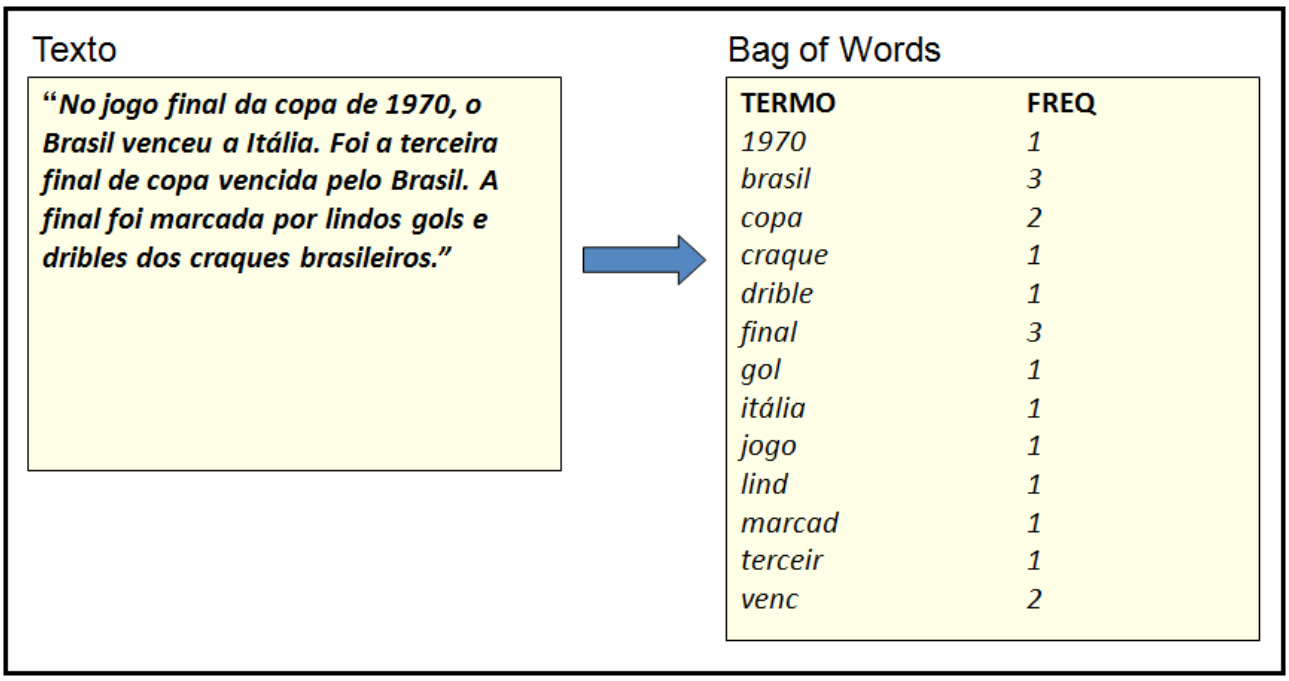

Figura 13. Bag of Words montada com remoção de *stop words* e operação de *stemming*.

Etapa 2. Modelagem

A forma mais comum de modelar as coleções de documentos que serão submetidas ao processo de mineração de texto é conhecida como *vector space model* [2]. Considerando um conjunto de **d documentos** e um total de **n termos** distintos considerando todos esses d documentos, os documentos são modelados como uma matriz de dimensão d linhas x n colunas. A Figura 14 mostra um exemplo deste tipo de matriz, considerando um exemplo envolvendo 3 documentos e 5 termos.

documentos / termos	t1	t2	t3	t4	t5
d1	6	14	0	5	15
d2	0	0	8	0	22
d3	17	10	1	0	6

Figura 14. Matriz com a frequência de termos por documentos

Etapa 3. Mineração de Texto

A partir da geração dessa matriz, suponha que nosso objetivo seja realizar a tarefa de *topic tagging*, onde o objetivo é atribuir um ou mais tópicos para uma notícia. Um algoritmo muito utilizado para este fim é k-NN (*k-Nearest Neighbours*). Ele realiza a classificação a partir de três passos muito simples:

- **Passo 1:** a base de dados de treinamento (base que contém documentos classificados – ver Nota 1) é importada para a memória.
- **Passo 2:** suponha que temos um novo documento para ser classificado (novoDoc). O algoritmo irá procurar os k documentos mais parecidos ou **similares** ao novoDoc na base de dados. A Figura 15 mostra um exemplo, considerando o valor de k como 3 (o valor de k é um parâmetro de entrada que pode ser especificado pelo usuário do sistema de mineração de dados). No exemplo, os 3 documentos mais similares a "novoDoc" são d2, d4 e d8.
- **Passo 3:** as classes mais frequentes entre os k documentos encontrados serão atribuídas para "novoDoc". Voltando ao exemplo da Figura 15, se as classes de d2 são "Futebol" e "Rio de Janeiro", as de d4 "Rio de Janeiro", "Política" e "Turismo" e a d8 apenas "Futebol", então as classes atribuídas para "novoDoc" seriam "Futebol" e "Rio de Janeiro"

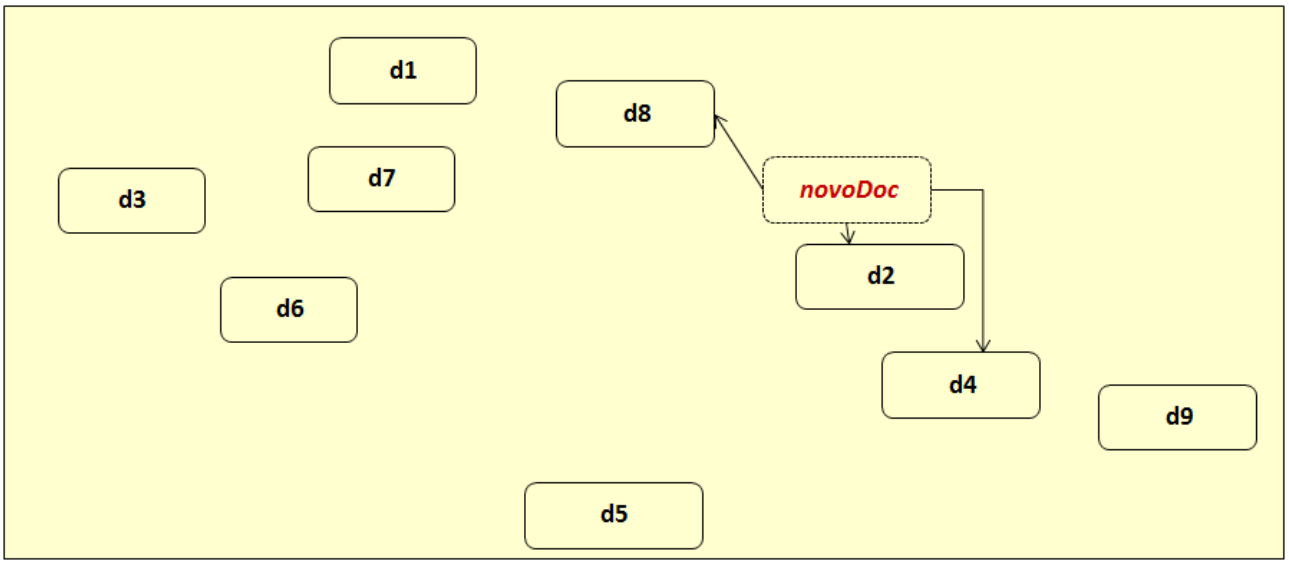

Figura 15. Classificação com o algoritmo k-NN

O algoritmo k-NN é muito simples, porém computar a similaridade entre documentos é algo bem mais difícil. A apresentação detalhada dessas técnicas encontra-se fora do escopo deste artigo, podendo ser obtidas em [5].

Conclusões

Este artigo apresentou os conceitos fundamentais sobre mineração de textos. Esta tecnologia utiliza técnicas e algoritmos provenientes de diferentes áreas com o intuito de viabilizar o processo de descoberta de conhecimento em informações textuais. Muitos algoritmos foram propostos há décadas atrás, quando ainda não havia evolução tecnológica suficiente e nem uma grande quantidade disponível de texto digitalizado (era pré-Internet). No entanto, agora chegou o momento

propício para o seu uso e a mineração de textos está se tornando cada vez mais utilizada em sistemas modernos, tanto os utilizados em ambiente acadêmico, como os sistemas comerciais.

Referências Bibliográficas

[1] J. Spolsky, "The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)," 2003. [Online]. Available: http://www.joelonsoftware.com/articles/Unicode.html.

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

[3] I. Witten, E. Frank, and H. Mark, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[4] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining Multi-Label Data," in *Data Mining and Knowledge Discovery Handbook*, 2nd ed., Springer, 2010, pp. 667–685.

[5] A. Rajaraman and J. Ullman, "Mining of Massive Datasets." [Online]. Available: http://i.stanford.edu/~ullman/mmds.html.

[6] I. Witten, "Text Mining," in *Practical handbook of internet computing*, Chapman & Hall/CRC Press, 2005.

[7] L. I. Rusu, "XML data mining, Part 3: Clustering XML Documents for Improved Data Mining," 2012. [Online]. Available: http://www.ibm.com/developerworks/library/x-datamine3/index.html.

[8] A. F. Martins, "Construção de Ontologias de Tarefa e sua Reutilização na Engenharia de Requisitos," Universidade Federal do Espírito Santo - Departamento de Informática, Vitória, 2009.

[9] "NaCTeM - National Centre for Text Mining," *Sentiment Analysis Tool*, 2012. [Online]. Available: http://www.nactem.ac.uk/software.php.

[10] E. Rahn and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *The VLDB Journal*, vol. 10, no. 4, 2001.

[11] M. Porter, "The Porter Stemming Algorithm," 1979. [Online]. Available: http://tartarus.org/martin/PorterStemmer/.