UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA

Trabalho de Conclusão de Curso

Uso de Reconhecimento de Padrões na Classificação de Documentos de Texto

André Dieb Martins

Prof. Dr. Bruno Barbosa Albert

Professor Orientador

Campina Grande, Agosto de 2011

Universidade Federal de Campina Grande

André Dieb Martins

Uso de Reconhecimento de Padrões na Classificação de Documentos de Texto

Trabalho de Conclusão de Curso apresentado à Unidade Acadêmica de Engenharia Elétrica como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador:

Prof. Dr. Bruno Barbosa Albert

Campina Grande

Dezembro de 2011

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Na atualidade, é notável o crescimento massivo do volume de dados e de seu fluxo na Internet. Essa tendência é alimentada em grande parte pelo uso da rede para uma grande quantidade de aplicações: pesquisas e estudos, comunicações, negócios, redes sociais, compras, dentre outras. Dessa tendência, reforça-se ainda mais uma necessidade geral de uma compreensão desses dados, para as mais diversas aplicações.

Nesse âmbito, nos posicionamos na área de reconhecimento de padrões, que lida especificamente com o problema da classificação de objetos em categorias (ou *classes*). Esses objetos, denominados na literatura como *padrões*, podem ser imagens, formas de onda, palavras, textos ou qualquer tipo de coisa que possa ser classificada.

A área de reconhecimento de padrões tem uma longa história, mas foi apenas em meados de 1960 que, com o surgimento do Minicomputador, começaram a surgir maiores demandas de aplicações práticas (THEODORIDIS & KOUTROUMBAS 2006). Com o passar do tempo, da era industrial a pós-industrial, essa área tem ganhado grande prestígio e como parte integral dos sistemas inteligentes feitos para tomada de decisões.

Atualmente, essa área possui inúmeras pesquisas em andamento e aplicações na engenharia. Por exemplo, podemos citar os sistemas de reconhecimento ótico de caracteres (do inglês *Optical Character Recognition - OCR*) já empregados em *Scanners* comerciais modernos, ou até sistemas de detecção de face (ZHAO et al. 2003). Não obstante, o reconhecimento de padrões também tem sido aplicado em diversas áreas da engenharia elétrica, como nos sistemas de potência, por exemplo, no estudo de descargas parciais (CHANG & YANG 2008) e também no processamento digital de sinais para medicina (GUPTA et al. 2007).

Uma outra aplicação prática importante é a classificação de documentos de texto (LI & JAIN 1998) – objeto de estudo deste trabalho. Esta trata-se da designação de uma categoria a um documento de texto, por exemplo determinar de maneira automática que uma dada notícia faz parte da categoria *esportes* (WU, FULLER & ZHANG 2009) ou

até que uma mensagem instantânea como "*Estou ótimo!*" reflete uma emoção positiva (ABBASI, CHEN & SALEM 2008).

Nesse trabalho foi projetado e implementado um sistema de classificação, utilizando o classificador *Naive Bayes* (**NB**) e implementado um *web crawler* específico para obtenção dos documentos de treinamento e testes. Nesse sistema foi observada a relação entre a quantidade de características, isto é, o tamanho do vetor de características, e a qualidade resultante do classificador.

1.1 Objetivos

Este trabalho tem como objetivo a criação de um sistema de classificação de documentos acadêmicos, cujo propósito específico é classificar teses de mestrado e/ou doutorado em seu respectivo curso baseado na análise comparativa de seu conteúdo e de um aprendizado prévio sobre os cursos conhecidos. Propõe-se também uma análise dos resultados e da performance do sistema construído.

Como objetivos secundários, visa-se fomentar conhecimentos e habilidades práticas na construção de *web crawlers* (ou *web spiders*), ferramenta necessária para *download* automático das teses de bancos de teses de universidades brasileiras.

2 Revisão Bibliográfica

Nessa seção é feita uma revisão bibliográfica com os conceitos básicos para a implementação do classificador.

2.1 Sistemas de Classificação Artificial

Os sistemas de classificação artificial são sistemas em que ocorre a classificação um padrão em duas ou mais classes (ou categorias) de forma automática. Para tal, o sistema deve ser capaz de adquirir valor da variável através de um sensor, medir suas características e a partir de algum conhecimento prévio, decidir a qual classe pertence o padrão.

Essas medidas são comumente chamadas de *características* do padrão. Além disso, chama-se *vetor de características* $\mathbf{x}$ o vetor composto pelas características x_i de um determinado padrão.

$$\mathbf{x} = [x_1, x_2, ..., x_k] \tag{2.1}$$

Um vetor de características identifica unicamente um padrão, sendo assim uma maneira bastante conveniente de representação. Mostramos a seguir um exemplo que ilustra os conceitos acima apresentados.

Exemplo 1 (Largo ou Comprido)**.** *Suponha que necessitamos classificar objetos em duas classes: Largo (L) ou Comprido (C). Em nosso contexto, um objeto é dito L se sua largura for maior que seu comprimento, caso contrário ele é dito classe C.*

Considere que nosso sensor é capaz de medir a largura l *e comprimento* c *de um objeto, isto é, os objetos medidos são representados pelo vetor de características* $\mathbf{t_i} = (l_i, c_i)$*.*

Considere agora que dois objetos foram medidos e obtivemos os seguintes vetores: $\mathbf{t_1} = (1, 5)$ *e* $\mathbf{t_2} = (3, 2)$*. Nesse caso, é obvio que o objeto* $\mathbf{t_1}$ *é da classe C e* $\mathbf{t_2}$ *da classe L.*

No exemplo acima, a distinção entre as duas classes é clara: se $l_i > c_i$, o objeto é da classe L, caso contrário, é da classe C. Na figura 1, mostramos as regiões correspondentes às classes, assim como a linha que as divide, chamada de *linha decisória*.

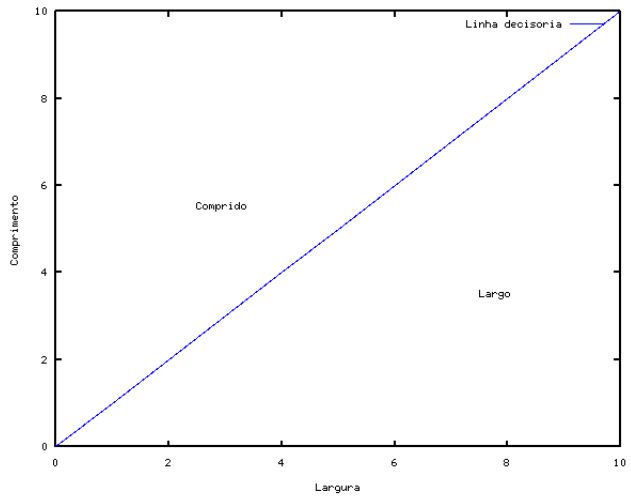

Figura 1 *– Linha decisória entre classes Largo e Comprido*

Essa *linha decisória* define os limites onde acaba uma classe e onde começa a outra, constituindo assim o *classificador*, cujo papel é dividir o *espaço de características* em regiões que correspondem às classes.

Vale notar que nesse exemplo temos um espaço de características de apenas duas dimensões, enquanto que na maioria dos sistemas de classificação se utilizam espaços de dimensões maiores.

2.1.1 Aspectos Construtivos

Na construção de um sistema de classificação, uma série de etapas devem ser executadas, como podemos observar no digrama de blocos da figura 2.

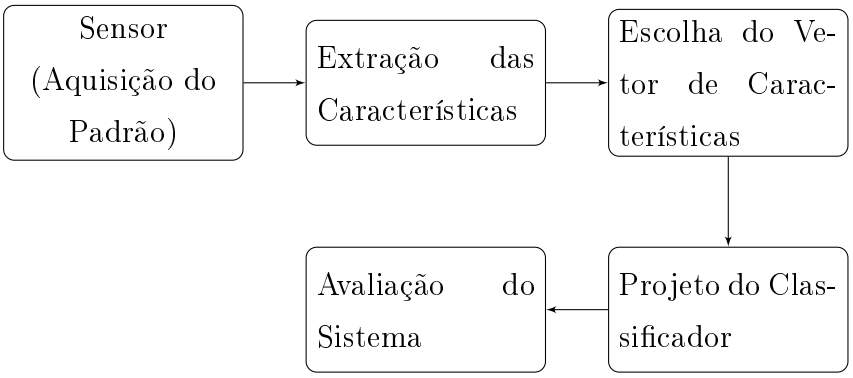

Figura 2 *– Diagrama de Blocos - Projeto de um Sistema de Classificação*

É importante observar que a implementação de cada etapa costuma ser diferente para cada aplicação, e.g. no exemplo 1 seria interessante adquirir as características através de sensores ópticos, enquanto que em um sistema de classificação de texto, poderíamos obter as características através de um tratamenteo estatístico do documento, procedimento adotado nesse trabalho.

2.2 Sistemas de Classificação de Texto

Dado um documento d e um conjunto de classes (ou categorias) C, deseja-se determinar a qual classe $c_i \in C$ o documento d está mais relacionado.

O primeiro passo na solução deste problema trata-se da extração (ou seleção) das características a partir do *corpus* de treinamento.

Uma vez definidas as características, deve-se converter os documentos do *corpus* de treinamento em suas representações em vetores de características, sendo esses os dados alimentados ao classificador para o aprendizado.

Uma vez finalizado o aprendizado, utiliza-se o classificador para classificar documentos de um *corpus* de testes, fazendo-se efetivamente as métricas de qualidade do classificador para posterior análise.

2.2.1 Extração de Características

O processo de extração (ou seleção) de características é um dos passos fundamentais na construção de um sistema de classificação. É nessa etapa que os dados são analisados e, seguindo um determinado critério, se determinam quais características dos dados melhor os distinguem em suas categorias.

Por exemplo, um método bastante conhecido é o Limiar de Frequência (YANG & PEDERSEN 1997) do inglês *Document Frequency*. Esse método parte de um pressuposto um tanto ingênuo: quanto menor a frequência f_i de um termo nos documentos, menos informação ele promove para a classificação.

Inicialmente, os termos do *corpus* de treinamento tem sua frequência computada. Em seguida, são removidos os termos de menor frequência, respeitando um certo limiar escolhido pelo projetista, obtendo-se assim as palavras "mais importantes" do *corpus*, o vetor de características.

Este e inúmeros outros métodos foram comparados por Yang e Pedersen (YANG & PEDERSEN 1997), como por exemplo o Ganho de Informação (**IG**), Informação Mútua (**MI**), Força dos Termos (**TS**) e χ^2 (**CHI**), sendo os três primeiros métodos mais semânticos e consistentes em relação à informação e ao problema da classificação.

Dos resultados obtidos nesse estudo, foi observado que a utilização do método **DF** produziu resultados satisfatórios quando comparado com os métodos **IG** e **CHI**, mesmo sendo o método de menor complexidade computacional dentre os estudados.

2.2.2 Conversão de um Documento em Vetor de Características

Tanto para o treinamento quanto para a classificação, os documentos devem ser convertidos em vetores de características, baseados nas características obtidas na extração. Esse procedimento se dá pelo seguinte tratamento:

1. Remoção de caracteres indesejados (números e símbolos)
2. Remoção das palavras vazias (*stop words*)
3. Marcação do documento (separação das palavras por espaços)

No primeiro passo, são removidos caracteres indesejados como virgulas, pontos, parênteses e etc. Na verdade, estes caracteres são substituídos por espaços, para não perdermos palavras compostas e outros casos especiais.

Em seguida, são removidas as palavras vazias (*stop words*) dos documentos, isto é, palavras que supostamente não adicionam informação à classificação e às diferenças entre os documentos. Por exemplo, podemos citar algumas palavras vazias: "você, ainda, outra, outras, quem, quer, uma, umas". Por ser uma prática comum nos Sistemas de Recuperação da Informação, é comum a existência de dicionários de palavras vazias para as várias linguas. Entretanto, no momento da escrita deste trabalho, não foi possível obter um dicionário padronizado. Sendo assim, foi criado um dicionário utilizando palavras vazias de várias fontes, como por exemplo do projeto *open source* **Apache Lucene**.

Uma vez concluídos os passos acima, é feito o último passo: a marcação do documento. Neste passo, o texto é quebrado em palavras utilizando as vírgulas como pivôs. No algorítmo aqui implementado, foi utilizada uma representação em mapa de frequências para os documentos, isto é, o documento em texto é convertido em um mapa onde as chaves são os palavras e os valores são as frequências das palavras. Por exemplo, se a

palavra "sistema" aparece dez vezes e a palavra "informação" aparece cinco vezes, teríamos o seguinte mapa:

$$M_i = [sistema : 10, informacao : 5, \ldots] \tag{2.2}$$

Poderíamos representar o documento por um vetor de termos em virtude de sua simplicidade mas, entretanto, optamos pela implementação em mapa de frequências pela capacidade de extender o *software* para trabalhos futuros, como na implementação de outros métodos de extração de características e de classificação.

Nas áreas de Processamento de Linguagem Natural e Recuperação da Informação, costuma-se utilizar o modelo **bolsa de palavras**. Nesse modelo, um documento é representado por um conjunto não-ordenado de palavras, desprovido de regras gramaticais e ordem.

2.2.3 Modelo de Espaço Vetorial (MEV)

Proposto por Wong e Yang (SALTON, WONG & YANG 1975), o Modelo de Espaço Vetorial define uma maneira genérica de se representar documentos em um espaço de documentos.

Considere um espaço de documentos D composto por documentos D_i, indexados por um ou mais termos T_i que podem ter pesos atribuídos de acordo com sua importância, ou até pesos binários (0 ou 1). Por exemplo, um documento de texto tem um conjunto de palavras. A cada palavra deste conjunto, podemos associar um peso individual. Desta forma, um documento D_i pode ser representado pelo vetor n-dimensional

$$\mathbf{D_i} = (d_{i1}, d_{i2}, \ldots, d_{in}) \tag{2.3}$$

2.2.4 Combinação da Conversão e do MEV

Baseado no modelo de documento que obtemos na seção 2.2.2, torna-se necessário definir uma transformação $\mathbf{D_i} = T(M_i)$ que aplicada a um mapa de frequências dos termos M_i, obtém o respectivo MEV $\mathbf{D_i}$ para este documento.

No campo dos Sistemas de Recuperação da Informação, tal transformação corresponde ao conceito da **Função de Ranqueamento** (do inglês *Ranking function*), cujo propósito

é determinar a relevância de um documento a uma dada consulta. Tal problema mostra-se bastante abrangente, como observado por Liu (LIU 2009) em sua pesquisa sobre algorítmos de ranqueamento.

Por exemplo, uma função bastante conhecida é a **TF-IDF**, onde a função peso é definida por:

$$d_{ij} = tf_{i,j} \log \frac{|\mathcal{D}|}{|\bar{d} \in \mathcal{D} | t \in \bar{d}|} \tag{2.4}$$

onde $tf_{i,j}$ é a frequência do termo j no documento i e a componente logarítmica é denominada *frequência inversa* (*inverse document frequency*), isto é, frequência de documentos em que o termo está presenet.

Na seção 4.2.3, apresentamos também o Estimador de Máxima Verossimilhança, aplicado como função de ranqueamento sob as classes e documentos utilizados neste trabalho.

2.3 Probabilidade e Estatística

Nessa seção consideramos variáveis aleatórias discretas, cuja notação encontra-se em letras maiúsculas (e.g. X, Y). As suas realizações encontram-se representados por letras minúsculas (e.g. x, y).

2.3.1 Probabilidade Condicional

Quando duas variáveis aleatórias demonstram alguma relação de dependência, torna-se mais fácil estimar o valor de uma ao saber o valor da outra. Matematicamente isso pode ser escrito como:

$$P(X = x_1|Y = y_1) = \frac{P(X = x_1, Y = y_1)}{P(Y = y_1)} \tag{2.5}$$

onde $P(X = x_1|Y = y_1)$ é a probabilidade do evento $X = x_1$ acontecer uma vez que $Y = y_1$ aconteceu.

É importante notar que caso as variáveis sejam independentes, a probabilidade conjunta torna-se o produto das variáveis individuais, tendo portanto

$$P(X = x_1|Y = y_1) = P(X = x_1) \tag{2.6}$$

ou seja, a informação da ocorrência do evento $Y = y_1$ não contribui com o evento $X = x_1$.

2.3.2 Lei da Probabilidade Total e a Regra de Bayes

A Lei da Probabilidade Total afirma que, dado um evento A e m diferentes maneiras de ocorrer este evento $A_1, A_2, \ldots, A_m$, caso esses subeventos A_i sejam mutuamente exclusivos, a probabilidade de A ocorrer é dada pela soma das probabilidades dos subeventos A_i:

$$P(A) = \sum_i P(A_i) \tag{2.7}$$

Considere então uma variável aleatória Y que assume valor y em m diferentes maneiras, em função de outra variável $x \in \mathcal{X} = \{x_1, x_2, \ldots, x_m\}$. Como essas probabilidade são mutuamente exclusivas, tem-se da Lei da Probabilidade Total que a probabilidade $P(Y = y)$ pode ser computada por:

$$P(Y = y) = \sum_{x \in \mathcal{X}} P(y, x) \tag{2.8}$$

Da definição de probabilidade condicional, temos:

$$P(x|y) = \frac{P(x, y)}{P(y)} \tag{2.9}$$

Como $P(x, y) = P(x|y)P(y) = P(y|x)P(x)$, podemos escrever a equação acima como:

$$P(x|y) = \frac{P(y|x)P(x)}{\sum_{x \in \mathcal{X}} P(y|x)P(x)} \tag{2.10}$$

A equação acima é denominada regra de Bayes. Uma interpretação interessante da regra de Bayes é que ela faz uma inversão estatística, tornando $P(y|x)$ em $P(x|y)$.

Por exemplo, supondo que x é uma causa e y um efeito, tem-se que na presença da causa x, é fácil determinar a probabilidade do efeito y ser observado, isto é, a *semelhança* $P(y|x)$.

Como uma série de causas x_i ocasionam o mesmo efeito y, observar o efeito y não ajuda na determinação da causa x. De fato, a utilidade da regra de Bayes aparece na determinação de $P(x|y)$, uma vez sabida a *semelhança* e a probabilidade *a priori* da causa $P(x)$.

Costuma-se chamar $P(x|y)$ de probabilidade *a posteriori*, ou simplesmente *posterior*, visto que esta probabilidade mostra as diferenças causadas pela observação do evento y na probabilidade $P(x)$, ou seja, é uma probabilidade posterior ao evento y.

2.4 Teoria Decisória de Bayes

2.4.1 Conceitos Fundamentais

A teoria decisória de Bayes é uma abordagem estatística para o problema da classificação de padrões. De fato, essa teoria utiliza-se de medições, probabilidades e custos de decisão na execução de um processo decisório.

Suponha um problema decisório onde um classificador deve decidir a qual classe $c_i \in C$ um documento $\mathbf{d}$ pertence. No uso da teoria decisória de Bayes, teríamos pela regra de Bayes:

$$P(c_i|d) = \frac{P(\mathbf{d}|c_i)P(c_i)}{P(d)} \tag{2.11}$$

isto é, a probabilidade $P(c_i|d)$ do evento "classe c_i dado um documento d" é relacionada com a probabilidade de semelhança $P(\mathbf{d}|c_i)$, isto é, a probabilidade de obtermos o documento $\mathbf{d}$ dado que o mesmo é da classe c_i; da probabilidade *a priori* de ocorrer um documento da classe c_i $(P(c_i))$; e da evidência $P(d)$. De fato, a evidência é dada por:

$$P(\mathbf{d}) = \sum_{c \in C} P(\mathbf{d}|c)P(c) \tag{2.12}$$

onde $\mathbf{d}$ é um vetor de características $\mathbf{d} = (d_1, d_2, \ldots, d_n)$. Desta forma, pode-se observar que o cálculo da expressão acima revela um valor $P(\mathbf{d})$ constante. De fato, este valor serve apenas para garantir que as probabilidades somem um. Além disso, a probabilidade $P(c_i|d)$, considerando o vetor de características, pode ser escrita como:

$$P(c_i|d_1, d_2, \ldots, d_n) = \frac{P(d_1, d_2, \ldots, d_n|c_i)P(c_i)}{P(\mathbf{d})} \tag{2.13}$$

Pode-se notar que o numerador do lado direito da equação é equivalente à probabilidade conjunta $P(c_i, d_1, d_2, \ldots, d_n)$. Supondo que as características d_i, d_j são independentes $(i \neq j)$, a probabilidade conjunta pode ser escrita como:

$$P(c_i, d_1, d_2, \ldots, d_n) = P(c_i) \prod_{i=1,\ldots,n} P(d_i|c_i) \tag{2.14}$$

Esta suposição de independência costuma ser taxada como ingênua, caracterizando o popular nome **Bayes Ingênuo** para este classificador.

Substituindo na expressão de $P(c_i|d)$, obtemos:

$$P(c_i|d) = \frac{P(c_i) \prod_{i=1,\ldots,n} P(d_i|c_i)}{P(\mathbf{d})} \tag{2.15}$$

expressão essa que determina a probabilidade do documento $\mathbf{d}$ pertencer a classe c_i, supondo a independencia das características.

2.4.2 Treinamento

O treinamento desse classificador se dá pelo cálculo da:

- Probabilidade *a priori* por classe: $P(c_i)$;
- Probabilidade condicional de ocorrência de uma palavra d_i numa classe c_i: $P(d_i|c_i)$.

Uma maneira simples de estimar esses valores é através do Estimador de Máxima Verossimilhança, que utiliza-se da frequência relativa dos parâmetros. Para estimar a probabilidade *a priori*, fazemos:

$$P(c_i) = \frac{N_{c_i}}{N} \tag{2.16}$$

onde N_{c_i} é o número de documentos da classe c_i e N é o número total de documentos.

Similarmente, pode-se estimar $P(d_i|c_i)$ por:

$$P(d_i|c_i) = \frac{N_{d_i,c_i}}{N_{dco}} \tag{2.17}$$

onde N_{d_i,c_i} é o número de ocorrências da característica d_i em documentos da classe c_i e N_{dco} é o total de ocorrências das características em documentos da classe c_i.

Um dos problemas que ocorrre com essa estimativa simples é a aparição do valor nulo para combinações que não pertencem ao *corpus* de treinamento. Para suprimir este problema, pode-se aplicar a suavização de Laplace:

$$P(d_i|c_i) = \frac{N_{d_i,c_i} + 1}{N_{dco} + |\mathbf{d}|} \tag{2.18}$$

isto é, soma-se o valor 1 no numerador e 1 no denominador para cada característica, o que equivale ao tamanho do vetor de características $|d|$.

2.4.3 Classificação

Uma vez finalizado o treinamento, tem-se em mãos as probabilidades *a priori* $P(c_i)$ e as probabilidades condicionais $P(d_i|c_i)$. Com estes valores, é possível computar as probabilidades $P(c_1|d), P(c_2|d), \ldots, P(c_m|d)$, sendo a maior destas a classe escolhida como melhor representante para o documento d.

Formalmente, a classificação é dada por:

$$c_i = \arg\max_{c \in C} P(c|\mathbf{d}) \tag{2.19}$$

onde

$$P(c|\mathbf{d}) = \frac{P(c)\prod_{i=1,\ldots,n} P(d_i|c)}{P(\mathbf{d})} \tag{2.20}$$

3 Metodologia

Nesse trabalho será adotada uma metodologia tradicional (THEODORIDIS & KOUTROUMBAS 200
(DUDA, HART & STORK 2001) para o projeto do sistema de classificação artificial. Esta metodologia foi descrita brevemente na seção 2.1.1.

No problema em questão, a classificação de textos, devemos aplicar algumas traduções aos nomes da metodologia, são elas:

- **padrão**: chamamos de *documento* o padrão considerado nesse sistema
- **sensor**: *web crawler* é o nome dado à ferramenta utilizada para *download* dos documentos e nesse caso é o próprio sensor

Inicialmente foi feita a implementação e execução da coleta de dados, obtendo um conjunto de teses utilizado nos experimentos. Em seguida, foi implementado o método de extração das características, assim como o treinamento supervisionado, como descrito na seção 2.4.2. Por fim, foi implementada a função de classificação, com base na descrição vista na seção 2.4.3.

Ao fim da implementação do classificador, foram feitos os devidos experimentos como listados nos objetivos deste trabalho. No capítulo a seguir será descrito o projeto e a implementação do sistema, seguindo a metodologia aqui apresentada.

4 Projeto e Implementação do Sistema de Classificação

4.1 Projeto

4.1.1 Introdução

O sistema de classificação tem seu projeto mostrado no diagrama de blocos da figura 3.

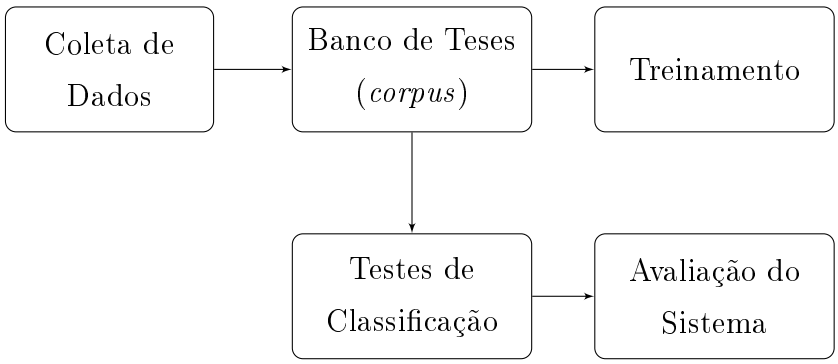

Figura 3 *– Diagrama de Blocos - Sistema de Classificação*

O primeiro bloco refere-se à coleta de dados (seção 4.1.2), etapa na qual são obtidas teses e criado um banco de teses. Em seguida temos o treinamento do classificador, alimentado pelo banco de teses. Após essa fase, são realizados os testes de classificação onde são armazenados resultados quantitativos sobre a qualidade do classificador. Finalmente, tem-se a avaliação do sistema, onde são inferidos comportamentos do sistema baseado nas medidas feitas anteriormente.

4.1.2 Coleta de Dados

Os dados a serem coletados são teses de mestrado e doutorado, como dito anteriormente. Nesse trabalho foi utilizado o banco de teses da Universidade de São Paulo (USP), sendo

esta uma fonte bem estabelecida e com um grande acervo de documentos. O endereço deste banco é: http://www.teses.usp.br/

Para esta coleta, o *web crawler* foi planejado para funcionar de maneira realimentada, continuando sua busca por documentos até não encontrar mais páginas na listagem.

Ao alimentá-lo com a primeira página de uma lista de teses de um dado curso, o *link* para a próxima página já é adquirido, obtendo então uma progressão que varre todas as teses do curso. Tal estrutura é mostrada no diagrama da figura 4.

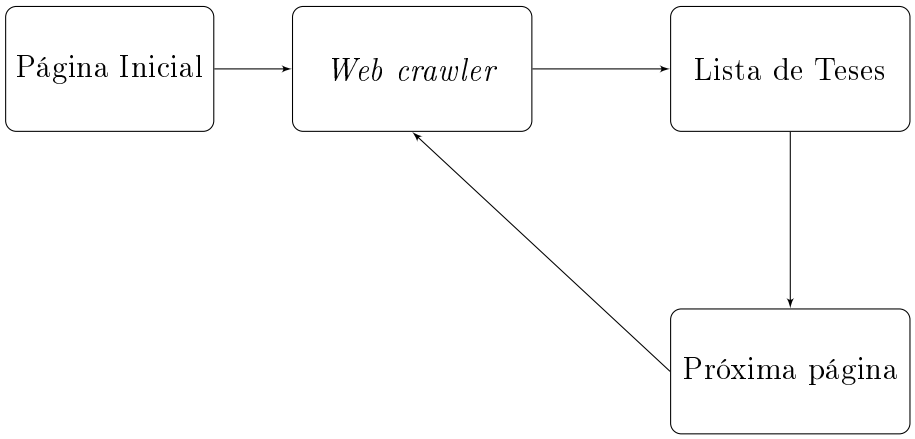

Figura 4 *– Diagrama de Blocos - Coleta de Dados*

4.2 Implementação

Na implementação do projeto descrito na seção 4.1, foi escolhida a linguagem de programação Python, pela sua versatilidade, simplicidade e por possuir bibliotecas nativas de cálculo numérico, varredura *web* (*web scraping*) e analisadores de texto.

Foram escolhidas as seguintes bibliotecas:

- **Scrapy**, *framework* de varredura utilizado na coleta das teses;
- **numpy**, para cálculos matemáticos de alta velocidade;

Como *software* adicional, utilizamos o **MongoDB** como banco de dados para armazenar as teses, por apresentar uma ótima velocidade de leitura e escrita de documentos.

4.2.1 Coleta de Dados

A coleta de dados foi implementada utilizando o *framework* **Scrapy**. Este *framework* fornece facilidades para implementar *web spiders* - robôs que varrem sites de maneira esquemática e obedecendo um conjunto de regras definidas pelo programador.

A coleta foi implementada de acordo com o diagrama visto na figura 4, tendo os documentos armazenados em um banco **MongoDB**.

4.2.2 Extração de Características

No processo de extração de características, utilizamos o critério do Limiar de Frequência **DF** (seção 2.2.1), definido a seguir:

Definição 1 (Limiar de Frequência (DF))**.** *Dado um conjunto de documentos $d \in \mathcal{D}$, as características são definidas por:*

$$X = \{t \in V(\mathcal{D}) | t_f > t_{lim}\} \tag{4.1}$$

onde $V(\mathcal{D})$ denota o vocabulário do espaço de documentos $\mathcal{D}$ e t_{lim} é a frequência limiar em que se deve considerar um termo.

Esse critério considera os termos mais frequentes como mais importantes, efetivamente eliminando os de mais baixa frequência. Na implementação desse método, utilizamos o algorítmo mostrado no Algorítmo 1 a seguir.

Algorítmo 1 Extração das Características

for document in trainingDocs **do**
  for term in Set(Tokenize(document.text)) **do**
    $frequency[term] \leftarrow frequency[term] + 1$
  end for
  $cutFrequency \leftarrow \frac{max(frequency.values) + mean(frequency.values)}{sizedivider}$
  $features \leftarrow frequency(value > cutFrequency)$
end for

O algorítmo 1 inicia-se fazendo um mapeamento do vocabulário presente nos documentos, isto é, uma contagem das palavras e de suas respectivas frequências de documento (em quantos documentos ela aparece).

É importante notar a separação de um documento em termos (função *Tokenize()*) e em seguida em um conjunto (função *Set()*), onde a frequência própria do termo no

documento termina por ser ignorada. Isso se faz necessário pois o método **DF** apenas considera quantos documentos em que aparece o termo, enquanto ignora sua frequência.

Uma vez obtido o vocabulário, o método **DF** requer a definição de um limiar onde as características são importantes. Definimos esse limiar com uma função de corte dada por:

$$f_c = \frac{max(\mathbf{c}) + mean(\mathbf{c})}{sizedivider} \tag{4.2}$$

onde *size divider* é um número inteiro especificado pelo programador, $max(\mathbf{c})$ denota o valor máximo de frequência e $mean(\mathbf{c})$ denota o valor médio de frequência.

Através de alterações no *size divider*, é possível controlar a seletividade do filtro. Por exemplo, ao aumentar *size divider*, temos uma redução na frequência de corte f_c, o que acarreta em mais características sendo extraídas (fato importante para os experimentos realizados na seção 5).

4.2.3 Treinamento

Para o treinamento do classificador, foi implementado um programa (também na linguagem Python) seguindo o procedimento descrito na seção 2.4.2. Nessa implementação foram feitas algumas modificações ao algorítmo original, resultando numa maior velocidade no treinamento. O algorítmo final é mostrado no Algorítmo 2 a seguir.

Algorítmo 2 Treinamento

for class in classes **do**
  $prior[class] \leftarrow log\left(\frac{class.ndocs}{classes.ndocs}\right)$
  for doc in class.docs **do**
    $allText \leftarrow allText + doc.text$
  end for
  for term in Tokenize(allText) **do**
    if term in features **then**
      $termFrequency[term] \leftarrow termFrequency[term] + 1$
    end if
  end for
  $base \leftarrow Sum(termFrequency.values) + termFrequency.size$
  for term in features **do**
    $conditional[term][class] \leftarrow log\left(\frac{termFrequency[term]+1}{base}\right)$
  end for
end for

Inicialmente, o algorítmo trata de calcular as probabilidades *a priori* das classes (eq.

(2.16)), armazenadas um mapa chamado *prior*. Em seguida, para cada classe, são concatenados todos os textos, extraídos seus termos e suas frequências são devidamente contadas. Finalmente, são calculadas as probabilidades condicionais aplicando a frequência dos termos à equação (2.18). Essas probabilidades condicionais são armazenadas num mapa chamado *conditional*.

É importante observar algumas otimizações como o uso da concatenação dos textos dos documentos na contagem de termos, assim como no cálculo prévio do logarítmo das probabilidades (descrito na seção 4.2.4).

4.2.4 Classificação

Para a classificação, foi implementado um programa em Python seguindo as prescrições da seção 2.4.3. Uma pequena modificação foi feita na função de estimação da classe:

$$c_i = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(c) \prod_{i=1,\ldots,n} P(d_i|c)}{P(d)} \tag{4.3}$$

Um problema inerente a essa função é a ocorrência de *overflow*no cálculo do produtório das $P(d_i|c)$. A fim de eliminar este problema, calculamos então o logarítmo do produto, uma vez que $log(xy) = log(x) + log(y)$. Outra otimização comum é utilizar $P(d) = 1$, já que este termo é constante para todas as classes e não afeta a estimativa final.

Feitas essas modificações, obtemos:

$$c_i = \arg\max_{c \in C} \left(log(P(c)) + \sum_i log(P(d_i|c)) \right) \tag{4.4}$$

A partir dessa expressão, implementamos o algorítmo de classificação mostrado no Algorítmo 3 listado a seguir.

Algorítmo 3 Classificação

```
for class in classes do
    score[class] ← prior[class]
end for
for term in Tokenize(document.text) do
    if term in features then
        score[class] ← score[class] + conditional[term][class]
    end if
end for
class = maximum(score)
```

O algorítmo mostrado acima procede computando a probabilidade $P(c|d)$ para cada classe $c \in C$, denotada aqui pelo mapa *score*.

Inicia-se por adicionar ao *score* as probabilidades *a priori*, computadas previamente no treinamento (Algorítmo 2). Em seguida, é feita uma varredura pelos termos do documento em busca das características, efetivamente somando suas probabilidades condicionais $conditional[term][class]$ (correspondentes à equação (2.18)).

Finalmente, toma-se a decisão de escolha baseado na classe cujo *score* foi máximo, isto é, a classe que maior probabilidade $P(c|d)$.

Uma vez concluída a implementação do sistema de classificação, submetemos tal sistema a testes e avaliações de qualidade, conteúdo este abordado no capítulo 5 a seguir.

5 Procedimentos Experimentais

Neste capítulo é descrito um procedimento experimental a fim de validar o classificador construído, observando algumas métricas de qualidade e relacionando variáveis de interesse.

5.1 Objetivos

Os experimentos aqui descritos tiveram como objetivo observar a relação entre o tamanho do vetor de características **c** e a qualidade do classificador (acurácia global). Para tal, nos utilizamos da modificação paramétrica *size divider* (introduzida na seção 4.2.2) para a seleção de características a fim de modificar o tamanho deste vetor.

5.2 Preparação

Uma vez realizada a coleta de teses e montado um banco de teses, foi feito um procedimento experimental de análise sobre o mesmo. O banco utilizado neste experimento tem as seguintes características:

- Total de teses: 647 teses;
- Cursos escolhidos ao acaso, considerando a inclusão de dois cursos com alto volume e os demais com volume moderado a fim de observar as diferenças de performance;
- Utilizados 75% de teses de cada curso para treinamento, escolhidos aleatoriamente dentro do *corpus*;
- Utilizados os restantes 25% de teses para testes e avaliação;

Por curso, utilizou-se as seguintes quantidades:

Curso	Número de Teses
Engenharia Mecânica de Energia de Fluídos	51
Biotecnologia	203
Geotectônica	54
Processamento de Sinais e Instrumentação	55
Genética	52
Enfermagem Psiquiátrica	164
Engenharia de Sistemas	68
Total:	647

Tabela 1 *– Cursos e Teses*

5.3 Resultados

Dando prosseguimento, o procedimento descrito acima foi executado para valores do *size divider* de 2 a 18, com um passo de 2 entre cada execução. Para cada *size divider* foram feitos todos os passos do sistema (extração de características, treinamento e testes). Dos resultados, obtivemos os seguintes:

size divider	Características $\|\mathbf{c}\|$	Acurácia Global (%)
2	730	17,18 %
4	2453	43,56 %
6	4184	61,35 %
8	5724	65,64 %
10	7016	68,71 %
12	8378	74,23 %
14	9892	76,07 %
16	11065	77,30 %
18	12359	84,66 %

Tabela 2 *– size divider, Número de Características e Acurácia Global*

Como esperado, obtivemos uma maior taxa de acertos em virtude do aumento do tamanho do vetor de características. De fato, isso pode ser explicado pela maior densidade de informação absorvida pelo vetor de características, e consequentemente pela representação em MEV dos documentos de entrada. Plotando a acurácia em função do

número de características, obtemos o gráfico apresentado na figura 5 a seguir.

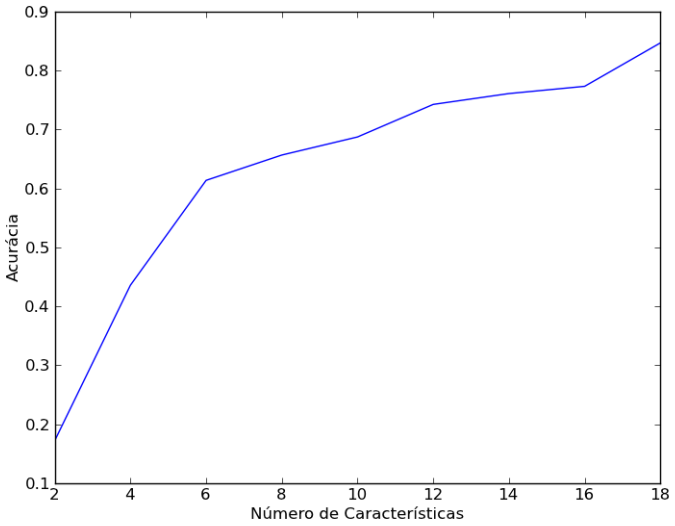

Figura 5 *– Acurácia × Número de características*

Nota-se, entretanto, que o aumento do número de características implica num maior tempo de execução da seleção de características, do treinamento e também do processo de classificação. Na execução desse experimento, medimos os tempos de execução de cada etapa, mostrados a seguir na tabela 3.

size divider	$\|\mathbf{c}\|$	$t_{extracao}$ (s)	$t_{treinamento}$ (s)	$t_{classificacao}$ (s)	t_{total} (s)
2	730	19,005	20,587	6,774	46,366
4	2453	19,224	22,065	16,725	58,014
6	4184	40,791	49,013	17,227	107,031
8	5724	40,947	48,837	18,232	108,016
10	7016	40,130	48,803	19,084	108,017
12	8378	23,279	22,239	9,817	55,335
14	9892	39,339	53,127	23,227	115,693
16	11065	39,954	50,699	20,994	111,617
18	12359	39,530	49,789	20,374	109.693

Tabela 3 *– size divider, Número de Características e Tempos de Execução das Etapas*

Da tabela 3, pode se observar um tempo total aproximadamente crescente com o número de características. Segundo Manning (MANNING, RAGHAVAN & SCHTZE 2008), a complexidade de tempo do treinamento (seção 4.2.3) é $O(|\mathcal{D}|L_{avg} + |C||V|)$, onde $|D|$

é o número de documentos, L_{avg} é o tamanho médio dos documentos, $|C|$ é o número de classes e $|V|$ é o tamanho do vetor de características. De fato, com o aumento de $|V|$, podemos constatar da tabela 3 um tempo predominantemente crescente, como esperado pela de complexidade.

Manning ainda mostrou que a complexidade de tempo da classificação (seção 4.2.4) é $O(|C|M_a)$, onde $|C|$ é o número de classes e M_a é o número de termos no documento a ser classificado. De fato, tal complexidade pode ser observada no algorítmo 3, uma vez que o vetor de características é utilizado para verificar se um termo do documento é de fato uma característica. Sendo assim, pode-se observar que a complexidade é majorada pela quantidade de termos, mesmo que ainda possua operações relativas ao tamanho do vetor de características. Portanto, podemos observar essa relativa independência na coluna $t_{classificacao}$, que varia em proporções bem menores do que das outras colunas.

6 Considerações Finais

Com a realização do trabalho, foi possível obter um grande volume de conhecimento no âmbito dos Sistemas de Recuperação da Informação. Do projeto de classificadores aos métodos de mineração de dados, foi possivel realizar uma breve revisão do estado da arte em extração de características, revisar conhecimentos básicos de Probabilidade & Estatística e Processos Estocásticos, e ainda verificar a construção e funcionamento do mais clássico classificador artificial.

Do sistema de classificação construído, foi possível alcançar uma acurácia global de $84,66\%$, isto é, $84,66\%$ de acertos na classificação de teses dentre sete cursos. Aliado a este alto desempenho, tem-se a desvantagem de um longo tempo de execução, tomando um tempo máximo de $115,69$ segundos.

Inicialmente, o treinamento e a classificação foram implementados de acordo com os algorítmos definidos por Manning (MANNING, RAGHAVAN & SCHTZE 2008). Tal classificador teve resultados de acurácia similares aos obtidos neste trabalho, porém, teve tempos de execução muito elevados. Otimizações foram feitas no treinamento, mais especificamente no processo de contagem de termos, e no cálculo das probabilidades *a posteriori*, como visto na seção 4.2.3. Outra simples otimização aplicada foi pré-computar os logarítmos utilizados na fórmula de classificação, distribuindo melhor o processamento entre as fases de treinamento e de classificação. Ao fim do algorítmo de treinamento, foi implementada também uma otimização chamada *memoização*, em que os valores de logarítmos são mapeados para seus operandos, aproveitando os valores para repetições de seus operandos. Para o processo de classificação, o laço que computa as contribuições *a priori* foi separado do laço de contribuições *a posteriori*, reduzindo em $|d|$ o número de cálculos redundantes.

Ao fim das otimizações, o sistema alcançou uma alta velocidade de classificação, levando entre 10 a 20 segundos para classificar um documento de 32 mil palavras. Similarmente, o treinamento após as otimizações alcançou a marca de 30 a 50 segundos para

um total de 485 documentos, com uma média de 30 mil palavras cada. Os experimentos foram executados em um computador com processador Intel(R) Core(TM) i5 M520 2.40 GHz de quatro núcleos e 4 GB de memória RAM (DDR3).

Essas marcas de desempenho são vantagens conhecidas do classificador *Naive Bayes*, visto que suas complexidades de tempo são lineares com os dados de entrada, como visto na seção 5.3. Pela simplicidade de construção e baixa complexidade de tempo, esses classificadores se mostram extremamente interessantes para tarefas de classificação rotineiras, alcançando desempenhos aceitáveis dentro de certos limites operacionais.

6.1 Trabalhos Futuros

Ao longo desse trabalho, foram observados alguns comportamentos interessantes em relação à modificações experimentais realizadas nos métodos. Uma das modificações se deu na fórmula de classificação, onde a informação de frequência do termo no documento foi incorporada. A expressão experimentada é dada por:

$$c_i = \arg\max_{c \in C} \left(log(P(c)) + \sum_i N_{d,d_i} \times log(P(d_i|c)) \right) \tag{6.1}$$

onde N_{d,d_i} denota a frequência do termo d_i no documento d. Feita essa modificação, foi possível obter um crescimento mais acelerado para a acurácia global em função do número de características. Dos resultados preliminares, podemos citar uma acurácia de $42,00\%$ para 758 características, enquanto que a função de classificação tradicional apresenta $17,80\%$ para um número similar de características.

Baseado nesses resultados, torna-se interessante investigar a causa de tal melhora, assim como seu rigor matemático ou até como formalizar tal modificação.

Por outro lado, também é de interesse a comparação do classificador *Naive Bayes* com outros tipos de classificadores, como por exemplo o *Support Vector Machine* ou as *Decision trees*. Além disso, ainda há um grau de liberdade na extração de características, etapa esta que pode acarretar um melhor desempenho.

Referências

ABBASI, A.; CHEN, H.; SALEM, A. Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums. **ACM Trans. Inf. Syst.**, ACM, New York, NY, USA, v. 26, p. 12:1–12:34, June 2008. ISSN 1046-8188. Disponível em: <http://doi.acm.org/10.1145/1361684.1361685>.

CHANG, W.-Y.; YANG, H.-T. Partial discharge pattern recognition of cast-resin current transformers using fuzzy c-means clustering approach. **WSEAS Trans. Comp. Res.**, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, v. 3, p. 172–181, March 2008. ISSN 1991-8755. Disponível em: <http://dl.acm.org/citation.cfm?id=1466884.1466891>.

DUDA, R.; HART, P.; STORK, D. **Pattern classification**. Wiley, 2001. (Pattern Classification and Scene Analysis: Pattern Classification). ISBN 9780471056690. Disponível em: <http://books.google.com.br/books?id=YoxQAAAAMAAJ>.

GUPTA, C. N. et al. Neural network classification of homomorphic segmented heart sounds. **Appl. Soft Comput.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 7, p. 286–297, January 2007. ISSN 1568-4946. Disponível em: <http://dl.acm.org/citation.cfm?id=1221581.1221762>.

LI, Y. H.; JAIN, A. K. Classification of text documents. **The Computer Journal**, v. 41, p. 537–546, 1998.

LIU, T.-Y. Learning to rank for information retrieval. **Found. Trends Inf. Retr.**, Now Publishers Inc., Hanover, MA, USA, v. 3, p. 225–331, March 2009. ISSN 1554-0669. Disponível em: <http://dl.acm.org/citation.cfm?id=1618303.1618304>.

MANNING, C. D.; RAGHAVAN, P.; SCHTZE, H. **Introduction to Information Retrieval**. New York, NY, USA: Cambridge University Press, 2008. ISBN 0521865719, 9780521865715.

SALTON, G.; WONG, A.; YANG, C. S. A vector space model for automatic indexing. **Commun. ACM**, ACM, New York, NY, USA, v. 18, p. 613–620, November 1975. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/361219.361220>.

THEODORIDIS, S.; KOUTROUMBAS, K. **Pattern Recognition, Third Edition**. Orlando, FL, USA: Academic Press, Inc., 2006. ISBN 0123695317.

WU, Q.; FULLER, E.; ZHANG, C.-Q. Text document classification and pattern recognition. In: **Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining**. Washington, DC, USA: IEEE Computer Society, 2009. p. 405–410. ISBN 978-0-7695-3689-7. Disponível em: <http://dl.acm.org/citation.cfm?id=1602240.1602729>.

ZHAO, W. et al. Face recognition: A literature survey. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 35, p. 399–458, December 2003. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/954339.954342>.

APÊNDICE A – Dicionário de Palavras Vazias

Nessa seção é apresentado o dicionário de palavras vazias utilizado na implementação no tratamento de documentos (seção 2.2.2).

de, a, o, que, e, do, da, em, um, para, é, com, não, uma, os, no, se, na, por, mais, as, dos, como, mas, foi, ao, ele, das, tem, à, á, se, sua, o, ser, quando, muito, há, nos, já, está, e, também, só, pelo, pela, até, isso, ela, entre, era, depois, sem, mesmo, aos, ter, seus, quem, nas, me, esse, eles, estão, você, tinha, foram, essa, num, nem, suas, me, às, minha, têm, numa, pelos, elas, havia, seja, qual, será, nós, tenho, lhe, deles, essas, esses, pelas, este, fosse, dele, t, te, vocês, vos, lhes, meus, minhas, te, tua, teus, tuas, nosso, nossa, nossos, nossas, dela, delas, esta, estes, estas, aquele, aquela, aqueles, aquelas, isto, aquilo, esto, está, estamos, estão, estive, esteve, estivemos, estiveram, estava, estávamos, estavam, estivera, estivéramos, esteja, estejamos, estejam, estivesse, estivéssemos, estivessem, estiver, estivermos, estiverem, hei, há, havemos, hão, houve, houvemos, houveram, houvera, houvéramos, haja, hajamos, hajam, houvesse, houvéssemos, houvessem, houver, houvermos, houverem, houverei, houverá, houveremos, houverão, houveria, houveríamos, houveriam, so, somos, são, era, éramos, eram, fui, foi, fomos, foram, fora, fôramos, seja, sejamos, sejam, fosse, fôssemos, fossem, for, formos, forem, serei, será, seremos, serão, seria, seríamos, seriam, tenho, tem, temos, tém, tinha, tínhamos, tinham, tive, teve, tivemos, tiveram, tivera, tivéramos, tenha, tenhamos, tenham, tivesse, tivéssemos, tivessem, tiver, tivermos, tiverem, terei, terá, teremos, terão, teria, teríamos, teriam, ver

APÊNDICE B – Programas

Nessa seção são apresentados os programas escritos na implementação do projeto (seção 4.2).

B.1 Classificador NB e Treinamento: classifiers.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import time
5  import numpy
6  import operator
7  import logging
8
9  import cStringIO as StringIO
10
11 import database
12
13 from bag import *
14
15 log = logging.getLogger(__file__)
16
17 class NaiveBayesClassifier(object):
18
19     def __init__(self, classes, training_docs, testing_docs):
20         self.classes = classes
21         self.training_docs = training_docs
22         self.testing_docs = testing_docs
```

```
23          self.prior = {}
24          self.cond = {}
25          self._features = []
26
27      @property
28      def features(self):
29          return self._features
30
31      @features.setter
32      def features(self, features):
33          self._features = features
34
35      def train(self):
36          """ Performs a naive-bayes on the given features. Fills
                out
37          dictionaries prior and cond with prior probabilities
38          P(c) of a class 'c' and conditional P(term|c).
39          """
40          docs = self.training_docs
41          n = docs.count()
42          prior = {}
43          cond = {}
44          memo = {}
45
46          log.debug('starting training on %d documents..', n)
47          start_time = time.time()
48
49          for cls in self.classes:
50              # Compute prior probabilities
51              nc = docs.find({'field': cls}).count()
52
53              # Maximum Likelihood Estimate (MLE)
54              prior[cls] = numpy.log(nc/float(n))
55
56              # Join all documents for faster counting
```

```
57              textfile = StringIO.StringIO()
58              for doc in docs.find({'field': cls}):
59                  if not database.doc_has_data(doc):
60                      continue
61                  textfile.write(('␣'.join(tokenize(doc['data']))
                        + '␣').encode('utf-8', 'replace'))
62
63              # Count vocabulary occurences on joined documents
64              nterm = dict.fromkeys(self.features, 0)
65              for term in textfile.getvalue().split():
66                  if term in self.features:
67                      nterm[term] += 1
68
69              # Precompute denominator for conditional prob.
                    estimator
70              base = float(sum(nterm.values()) + len(nterm))
71
72              # Compute conditional probabilities
73              for term in self.features:
74                  if term not in cond:
75                      cond[term] = {}
76                  val = (nterm[term] + 1)/base
77                  if val not in memo:
78                      memo[val] = numpy.log(val)
79
80                  cond[term][cls] = memo[val]
81
82          log.debug('finished␣training␣(took␣%.3f␣secs)', time.
                time() - start_time)
83
84          self.prior = prior
85          self.cond = cond
86
87      def classify(self, doc):
88          """ Classify the input document 'doc'.
```

```
89          """
90          tokens = tokenize_map(doc['data'])
91          score = {}
92
93          for cls in self.classes:
94              score[cls] = self.prior[cls]
95
96          for term, count in tokens.iteritems():
97              count = 1
98              if term not in self.cond:
99                  continue
100             for cls in self.classes:
101                 score[cls] += count * self.cond[term][cls]
102
103         return max(score.iteritems(), key=operator.itemgetter(1)
                )[0]
```

B.2 Conversão de Documentos (MEV): bag.py

```
1  # -*- coding: utf-8 -*-
2
3  from ptbr_dict import STOP_WORDS
4
5  SYMBOLS = list(set(u'\\/.,()[]%:;^~`\'!\"=+-_<>?'))
6
7  def clean(data):
8      """ Remove symbols and numbers """
9      for symbol in SYMBOLS:
10         data = data.replace(symbol, u'␣')
11     for i in xrange(10):
12         data = data.replace(u'%d' % i, u'')
13     return data
14
15 def tokenize(data, threshold=3):
```

```
16      return filter(lambda k: len(k) > threshold, clean(data).
            lower().split())
17
18  def tokenize_map(data, threshold=3):
19      d = {}
20      for k in clean(data).lower().split():
21          d[k] = d.setdefault(k, 0) + 1
22      return d
23
24  def bag_of_words(data):
25      # Lower and convert to set
26      data = set(tokenize(data))
27      # Convert to set and remove stop words
28      data = set(data) - STOP_WORDS
29      return data
30
31  def frequency_bag(data):
32      bag = bag_of_words(data)
33      freq_bag = {}
34      # Lower and convert to set
35      data = clean(data).lower().split()
36      for w in bag:
37          freq_bag[w] = data.count(w)
38      return freq_bag
```

B.3 Utilitários para Banco de Dados: database.py

```
1  # -*- coding: utf-8 -*-
2  from pymongo import Connection
3
4  db = Connection().theses
5
6  def get_classes():
7      """ Return available classes on theses database.
8      """
```

```
9       return list(set([d['field'] for d in db.theses.find({}, {'
            field': 1})]))
10
11  def doc_has_data(doc):
12      """ Returns whether the document has non-empty data.
13      """
14      return True if 'data' in doc and len(doc['data']) > 0 else
            False
```

B.4 *Web spider* para banco de teses (USP): usp.py

```
1   from scrapy.spider import BaseSpider
2   from scrapy.selector import HtmlXPathSelector
3   from scrapy.http import Request
4   from scrapy.conf import settings
5
6   from scrapy import log
7
8   from urlparse import urljoin
9   from theses.items import *
10
11  from pymongo import Connection
12
13  import tempfile
14  import os
15
16
17  class USPListAreasSpider(BaseSpider):
18      """ Lists USP thesis areas (e.g. biology, mechanical
            engineering, architecture, etc)
19      """
20      name = "usp-list-areas"
21      allowed_domains = ['teses.usp.br']
22      start_urls = ['http://www.teses.usp.br/index.php?option=
            com_jumi&fileid=6&Itemid=61&lang=pt-br&pagina=1']
```

```
23      base_url = 'http://www.teses.usp.br'
24
25      def parse(self, response):
26          hxs = HtmlXPathSelector(response)
27          try:
28              next_page = hxs.select('//div[@class="CorpoPaginacao
                    "]/a[last()-1]/@href').extract()[0]
29              yield Request(urljoin(self.base_url, next_page),
                    callback=self.parse)
30          except:
31              pass
32          for f in hxs.select('//div[@class="dadosLinha dadosCor1
                "] | //div[@class="dadosLinha dadosCor2"]'):
33              item = FieldItem()
34              item['name'] = f.select('.//div[@class="
                    dadosAreaNome"]/a/text()').extract()[0]
35              item['size'] = int(f.select('.//div[@class="
                    dadosAreaNome"]/text()').extract()[0].replace('(
                    ', '').replace(')', ''))
36              item['type'] = f.select('.//div[@class="
                    dadosAreaTipo"]/text()').extract()[0]
37              item['url'] = urljoin(self.base_url, f.select('./div
                    [@class="dadosAreaNome"]/a/@href').extract()[0])
38              yield item
39
40  class USPThesisSpider(BaseSpider):
41      """ Retrieve theses info.
42      """
43      name = 'usp-list-theses'
44      allowed_domains = ['teses.usp.br']
45      start_urls = []
46      base_url = 'http://www.teses.usp.br'
47      db = Connection().theses
48
49      def start_requests(self):
```

```
50          for f in settings['FIELDS']:
51              field = self.db.fields.find_one({'name': f})
52              print 'Creating request for ', field['url']
53              yield Request(field['url'], callback=self.
                    parse_theses_list)
54
55      def parse_theses_list(self, response):
56          hxs = HtmlXPathSelector(response)
57          print 'Parsing list'
58          try:
59              next_page = hxs.select('//div[@class="CorpoPaginacao
                    "]/a[last()-1]/@href').extract()[0]
60              yield Request(urljoin(self.base_url, next_page),
                    callback=self.parse_theses_list)
61          except:
62              print 'Finished listing thesis from field'
63          for f in hxs.select('//div[@class="dadosLinha dadosCor1
                "] | //div[@class="dadosLinha dadosCor2"]'):
64              item = ThesesItem()
65              item['author'] = f.select('.//div[@class="
                    dadosDocNome"]/a/text()').extract()[0]
66              item['url'] = f.select('.//div[@class="dadosDocNome
                    "]/a/@href').extract()[0]
67              item['title'] = f.select('.//div[@class="
                    dadosDocTitulo"]/text()').extract()[0]
68              item['field'] = f.select('.//div[@class="
                    dadosDocArea"]/a/text()').extract()[0]
69              item['type'] = f.select('.//div[@class="dadosDocTipo
                    "]/a/text()').extract()[0]
70              item['dept'] = f.select('.//div[@class="
                    dadosDocUnidade"]/a/text()').extract()[0]
71              item['year'] = int(f.select('.//div[@class="
                    dadosDocAno"]/a/text()').extract()[0])
72              yield item
73
```

```
74  class USPDownloadThesisSpider(BaseSpider):
75      """ Retrieve theses docs.
76      """
77      name = 'usp-theses-docs'
78      allowed_domains = ['teses.usp.br']
79      start_urls = []
80      base_url = 'http://www.teses.usp.br'
81      db = Connection().theses
82
83      def start_requests(self):
84          for t in self.db.theses.find():
85              if 'data' in t and len(t['data']) > 0:
86                  continue
87              # Create a request to download the pdf and process
                    it
88              req = Request(urljoin(self.base_url, t['url']),
                    callback=self.parse_thesis_spec)
89              req.meta['thesis'] = t
90              yield req
91
92      def parse_thesis_spec(self, response):
93          hxs = HtmlXPathSelector(response)
94          thesis = response.meta['thesis']
95
96          # Pdf download
97          all_a = hxs.select('//a')
98          urls = []
99          for a in all_a:
100             text = a.select('text()').extract()
101             if len(text) > 0 and text[0].endswith('.pdf'):
102                 urls.append(a.select('@href').extract()[0])
103
104         if len(urls) == 0:
105             # Invalidate damned fragmented theses with many pdfs
```

```
106             log.msg('Invalid thesis, no files', level=log.
                    WARNING, spider=self)
107             return
108         elif len(urls) > 1:
109             log.msg('Many pdfs, considering only first', level=
                    log.WARNING, spider=self)
110 
111         url = urljoin(self.base_url, urls[0])
112         log.msg('Downloading pdf from %s' % url, level=log.INFO,
                spider=self)
113         request = Request(url, callback=self.parse_doc)
114         request.meta['thesis'] = response.meta['thesis']
115         yield request
116 
117     def parse_doc(self, response):
118         thesis = response.meta['thesis']
119         log.msg('Parsing pdf for thesis %s' % thesis['author'],
                level=log.DEBUG, spider=self)
120 
121         t = tempfile.NamedTemporaryFile()
122         t.write(response.body)
123         t.flush()
124         os.system('pdftotext %s' % t.name)
125 
126         thesis['data'] = open(t.name + '.txt').read()
127         log.msg('Downloaded thesis (len=%d)' % (len(thesis['data
                '])), level=log.DEBUG, spider=self)
128         # Save it back
129         self.db.theses.save(thesis)
```