5
Implementação e Estudo de Caso

5.1.
Arquitetura Geral do Sistema

Com o objetivo de se obter enriquecimento prático acerca da metodologia de MT estudada, foi implementado um sistema de investigação textual contemplando todas as cinco etapas descritas no capítulo 3. Todas as implementações foram feitas utilizando-se a plataforma .NET™ da Microsoft com a linguagem de programação C#, a qual possui características de orientação a objetos.

O sistema foi desenvolvido de forma clássica, isto é, com acesso às funcionalidades principais através de barra de *menus* que levam a formulários de preenchimentos e visualização de resultados. A Figura 16 ilustra o Diagrama Hierárquico de Funções (DHF) do sistema. Todas as cinco funcionalidades no maior nível hierárquico foram implementadas como **módulos**, melhorando de forma significativa a organização do projeto.

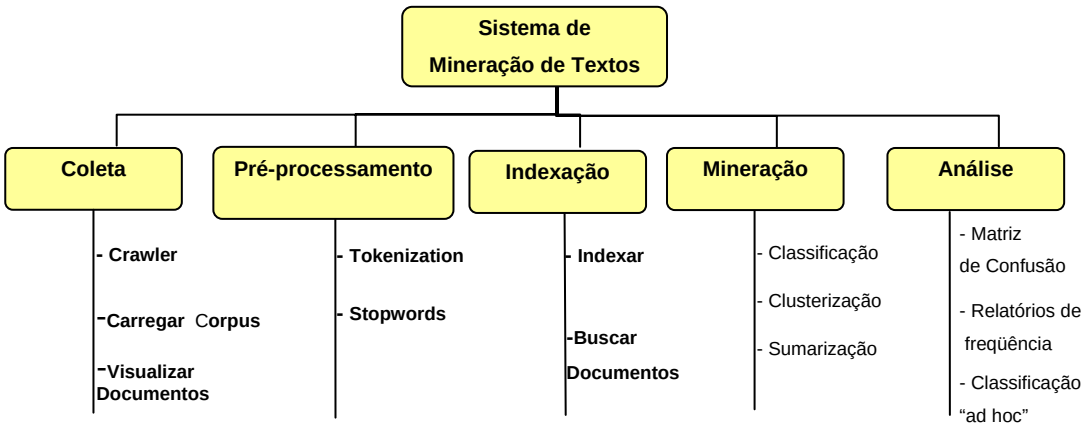

Figura 16 – Diagrama Hierárquico de Funções do sistema implementado.

A arquitetura do sistema segue o padrão de projeto denominado MVC [62] (do inglês, *Model-View-Controller*), conhecido por prover uma solução útil a sistemas complexos a qual separa a camada de apresentação (interface gráfica) da camada de negócio (modelo), através de uma camada intermediária (*Controller*). Assim, alterações na interface gráfica não alteram a lógica do sistema.

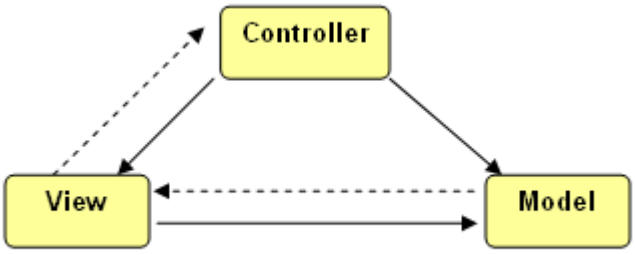

Figura 17 – Modelo MVC utilizado no desenvolvimento do sistema. As setas sólidas indicam associações diretas e as tracejadas indicam associações indiretas.

5.1.1. Módulo de Coleta

O módulo de coleta permite que documentos sejam buscados através de duas das três principais fontes de dados descritas na seção 3.1, que são: (a) coleta em discos rígidos através de pastas de usuários; (b) coleta na Internet através de *web crawlers*.

Embora a coleta na Internet pudesse ter sido implementada de forma independente da coleta em pastas de usuários, optou-se por armazenar todo o conteúdo coletado da *Web* (e.g. documentos HTML, arquivos texto) **primeiramente** no disco rígido local. Desta forma, o recurso de coleta na Internet poderia ser até mesmo uma ferramenta separada. Logo, a única porta de entrada do sistema é através da coleta manual, aonde o usuário informa quais pastas do disco local contêm documentos (sejam estes coletados da Internet ou não) que serão incorporados ao sistema de MT. Para que um novo *corpus* seja formado, o usuário deve acessar o sistema através do *menu* **Coleta->Carregar Corpus.** Em seguida deverá informar a partir de onde os documentos deverão ser obtidos, juntamente com um nome, o qual identificará internamente o novo *corpus.* A

Figura 18 ilustra o diagrama de classes parcial do sistema, com as classes do modelo envolvidos neste momento. Cada documento coletado é fisicamente mapeado em um objeto da classe **Documento**, os quais estão relacionados a um e somente um objeto da classe **Corpus**, através de uma associação do tipo **agregação**[7]**.**

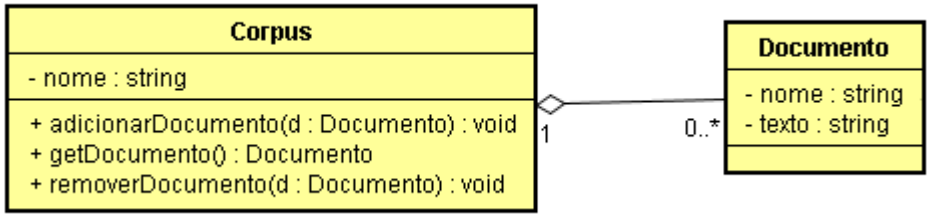

Figura 18 – Diagrama de classes parcial do sistema com as classes Corpus e Documento.

Passando à implementação do *crawler*, esta foi feita utilizando-se o recurso de *multithreading*, isto é, a divisão de um processo em diversas unidades menores de execução. Deste modo, vários *web crawlers* estão fazendo o trabalho de visitar URLs ao mesmo tempo, paralelizando o sistema.

Inicialmente, todas as *threads* do sistema (cada *thread* pode ser entendida com um *web crawler*) encontram-se no estado de "espera". O usuário do sistema fornece então uma URL inicial, chamada de "semente", que é o ponto de partida do processo de coleta. Esta URL recebe alguns tratamentos e é posta em uma estrutura de dados do tipo fila, aonde o primeiro elemento a entrar é o primeiro a sair. A entrada da "semente" na fila, faz com que se inicie o processo contínuo da Figura .

Os *web crawlers* ficam verificando de tempos em tempos se há alguma URL na fila para que estes possam sair do estado de espera. Caso exista, a tarefa do robô é retirá-la da fila, para que outro não a utilize, e faça então a "visitação", através de um requisição. Todo o conteúdo recebido em resposta à requisição é armazenado localmente, registrando uma cópia fiel da página *web* visitada.

[7]Agregação é um caso particular de associação que indica que uma das classes, chamada de "todo", está relacionada com suas partes. É representado por um lonsango vazado ao lado da classe "todo".

A página é então analisada em seu conteúdo. O usuário pode definir filtros sobre as palavras encontradas na página, por exemplo, se esta contiver palavras como “sex” e “free” ela deveria ser ignorada e excluída do disco rígido, pois provavelmente são anúncios sem muita relevância. O contrário também pode ser realizado, como só manter as páginas que possuam todas as palavras de uma lista definida, resultando em uma coleta segmentada. O próximo passo é submeter a página ao *parsing*, responsável por identificar novas URLs referenciadas na forma de *hiperlinks*. O *parsing* não só identifica novas URLs como também aplica filtros definidos pelo usuário, como verificação se um *hiperlink* leva a um site que está na lista de ignorados pelo sistema (a ser definida pelo usuário), ou se leva a um site que já foi visitado, o que não é desejado. Apenas as URLs que passam por todos os filtros são postas na fila, conforme visto no diagrama da Figura 19.

Por último, encontra-se a figura do destilador, introduzido em [31] com a finalidade de exercer alguma política de prioridade sobre as novas URLs na fila, reorganizando-as. Isto permite, por exemplo, que páginas que contenham determinadas palavras possuam prioridade de visitação sobre as outras. A política padrão do destilador implementado é manter a característica inicial de fila à estrutura, mantendo a ordenação segundo a ordem de chegada.

O ciclo é então encerrado e os *web crawlers* já possuem novas páginas para visitação. O processo de coleta só termina quando da interrupção manual pelo usuário ao acionar o botão específico de encerramento na interface. Isso faz com que cada uma das *threads* sejam “encerradas”, liberando os recursos já alocados.

Alguns detalhes de implementação são necessários para garantir que o ciclo seja executado com o sucesso. O primeiro deles é necessário devido ao problema inerente ao uso de *threads* em programação, que é o acesso compartilhado a recursos e estruturas de dados. Como a fila de URLs pode ser acessada de forma quase que simultânea, inconsistências poderiam trazer resultados inesperados e indesejados. Para garantir acesso único, cada *thread* que for acessar a fila precisa requisitar acesso isolado, via comando da linguagem, manipular a estrutura e, logo que possível, liberá-la para uso por outra *thread.*

Outro detalhe é quanto à estrutura que mantém as URLs já visitas e que precisa ser consultada a fim de que não haja visitações duplicadas. Como o número de URLs tende a crescer de forma exponencial, é necessário que a estrutura forneça rápido acesso aos elementos, verificando se a URL já foi ou não

visitada. A estrutura de dados mais adequada para isso é a **Árvore B** [45]**,** bastante utilizada em sistemas de banco de dados e que tem como característica ser balanceada, facilitando a busca por determinado elemento ao evitar que toda a lista de URLs seja percorrida por completo.

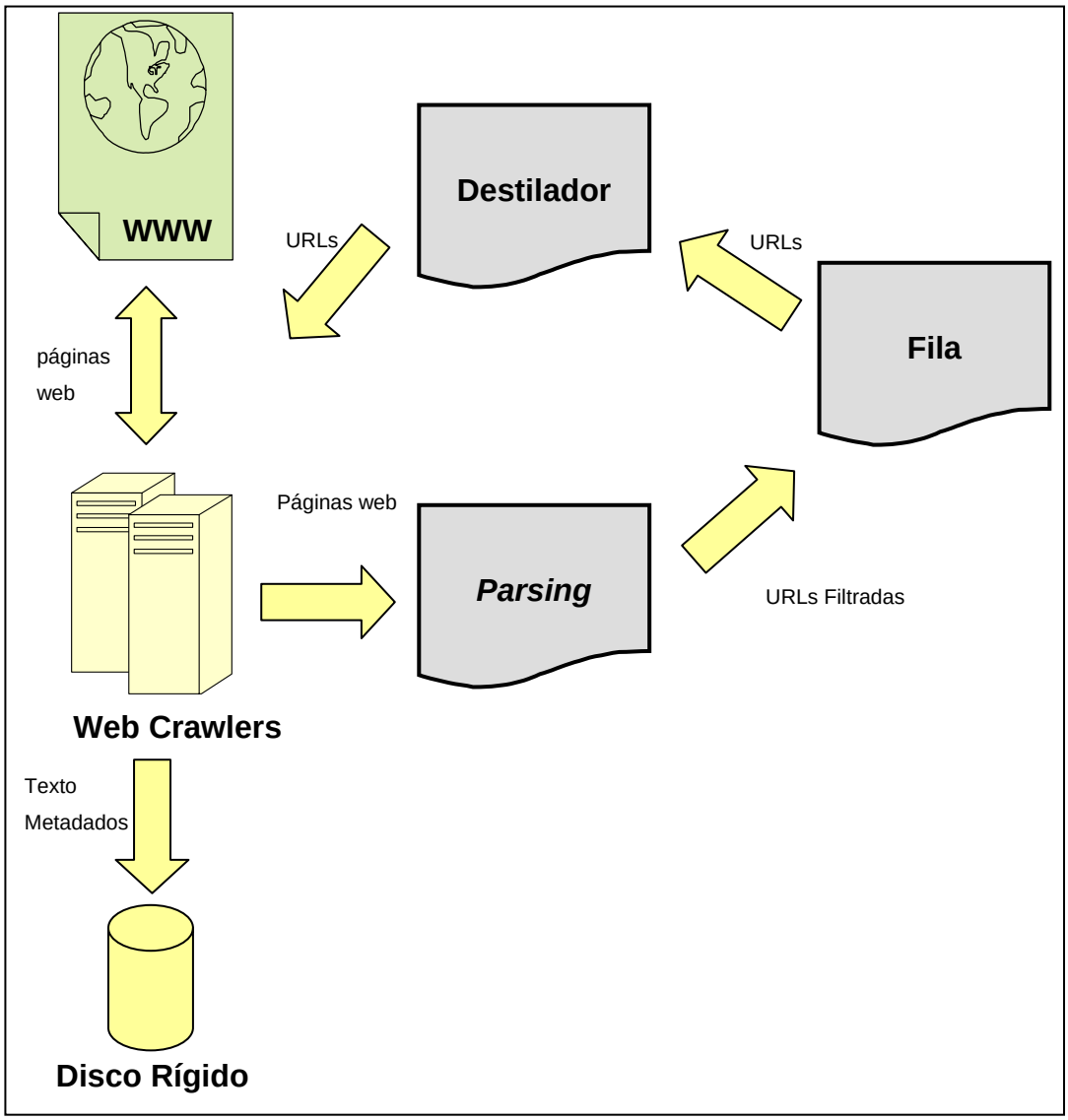

Figura 19 – Ciclo contínuo de execução da coleta na Internet através de *web crawlers.*

5.1.2.
Módulo de Pré-processamento

O módulo de pré-processamento possui implementações de algumas das principais funcionalidades existentes nesta etapa e que foram explicadas na seção 3.2 desta dissertação. A primeira das implementações diz respeito ao processo de *Tokenization* (seção 3.2.1), a qual acaba por introduzir novas classes ao diagrama da Figura 18, conforme ilustrado na Figura 20.

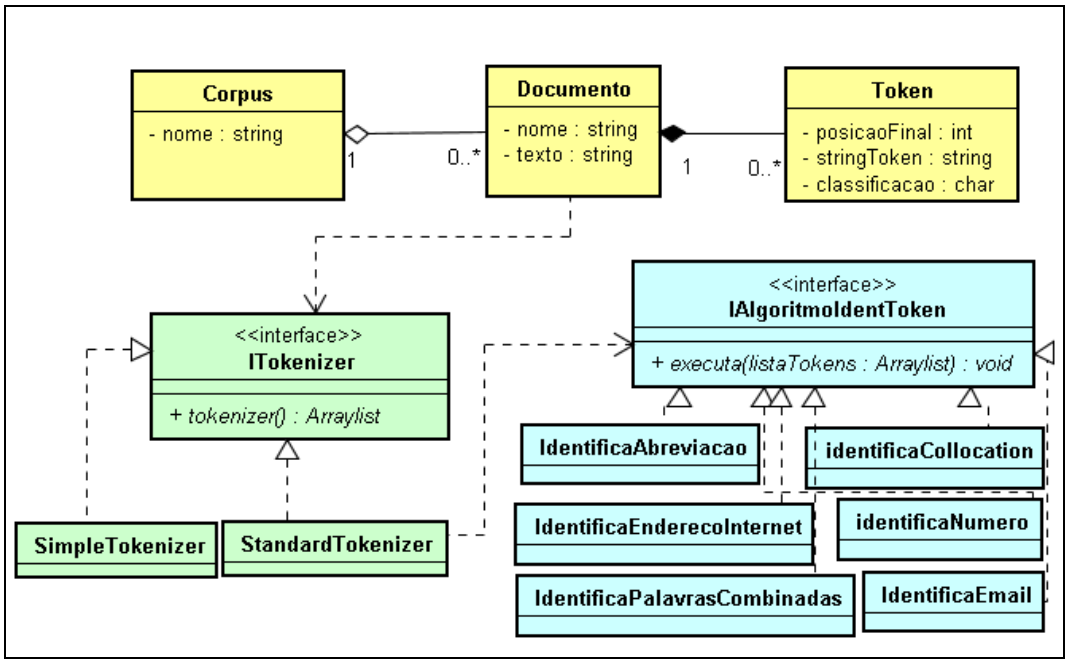

Figura 20 – Diagrama de classes contendo as novas classes introduzidas pela etapa de *tokenization* do módulo de pré-processamento.

A classe **Documento** está associada à classe **Token**, numa relação de "um-para-muitos", indicando que determinado documento possui de zero a vários *tokens*. A classe **Token,** por sua vez, possui os atributos "posicaoFinal" (determina em que posição dentro do texto termina o *token*), "classificação" (indica de que tipo é o *token*, como número ou abreviação) e "stringToken" que é o literal extraído do processo, ou seja, o próprio *token.*

A estratégia de identificar *tokens* em um texto é flexibilizada no sistema, cabendo ao usuário decidir qual usar, indicando na interface acessada através do *menu* **Pré-processamento->Tokenizar.** Internamente, o algoritmo que faz tal identificação encontra-se em uma classe separada das classes **Documento** e

Token, numa implementação baseada no padrão de projeto conhecido por ***strategy*** [62]**.** A aplicação deste padrão introduz diversos benefícios como aumento do reuso da solução, melhor legibilidade do código-fonte, facilidade de realização de testes e promoção do desacoplamento, garantindo maior liberdade aos objetos.

Para garantir que todo algoritmo de *tokenization* possua as mesmas entradas e saídas, variando apenas a forma de manipular as estruturas, foi definida uma *interface*, que em orientação a objetos pode ser entendida como um "contrato" assinado por todos as classes que se propõem a conter algoritmos de *tokenization*. Esta interface é definida no diagrama da Figura como **ITokenizer.** Duas classes implementam tal *interface*: **SimpleTokenizer** e **StandardTokenizer.** A primeira define um algoritmo bastante simples de "quebra" de texto em *tokens*, nas posições aonde se encontram determinados caracteres como espaços e pontuação. **StandardTokenizer** é uma implementação mais elaborada, que envolve o uso de expressões regulares em várias camadas, resultando na identificação de unidades mais complexas, como endereços de *e-mail*, números e *sites*. A implementação dos algoritmos de detecção destas estruturas também segue o padrão ***strategy*****,** com o mapeamento de cada algoritmo de detecção em uma classe distinta. A interface neste caso é a **IAlgoritmoIdentToken**, conforme também visto no diagrama da Figura 20.

Outra implementação realizada no módulo de pré-processamento é a manutenção de *stoplists*, aonde o usuário pode gerenciar diversas listas distintas de palavras, conhecidas como *stopwords* (seção 3.2.3.2), que serão ignoradas do contexto geral do sistema se assim configurado pelo usuário. O usuário tem a opção de utilizar uma ou mais *stoplists*, assim como excluir e incluir novas *stopwords.*

Por último, foi definida uma classe chamada **preProcessing** com alguns algoritmos gerais de pré-processamento, como detecção de erros ortográficos (seção 3.2.2) e identificação do início e fim de frases (seção 3.2.4).

5.1.3.
Módulo de Indexação

A implementação do módulo de indexação acrescenta uma **interface** denominada **IIndex** que define as funções básicas de toda implementação de um novo indexador no sistema. Foi desenvolvido um indexador como exemplo, baseado no **Lucene**[8], uma biblioteca de alto desempenho e escalável, para construção de ferramentas de recuperação de informação. Lucene é madura, livre, sendo um projeto de código aberto implementado em JAVA, mas que já possui versões para várias outras linguagens como PERL, PYTHON, C++, RUBY, DELPHI e a linguagem utilizada no desenvolvimento do sistema desta dissertação, C# do *framework* .NET. A Figura 21 ilustra parte do diagrama de classes que vem sido debatido ao longo deste capítulo, com a adição das partes envolvidas neste módulo.

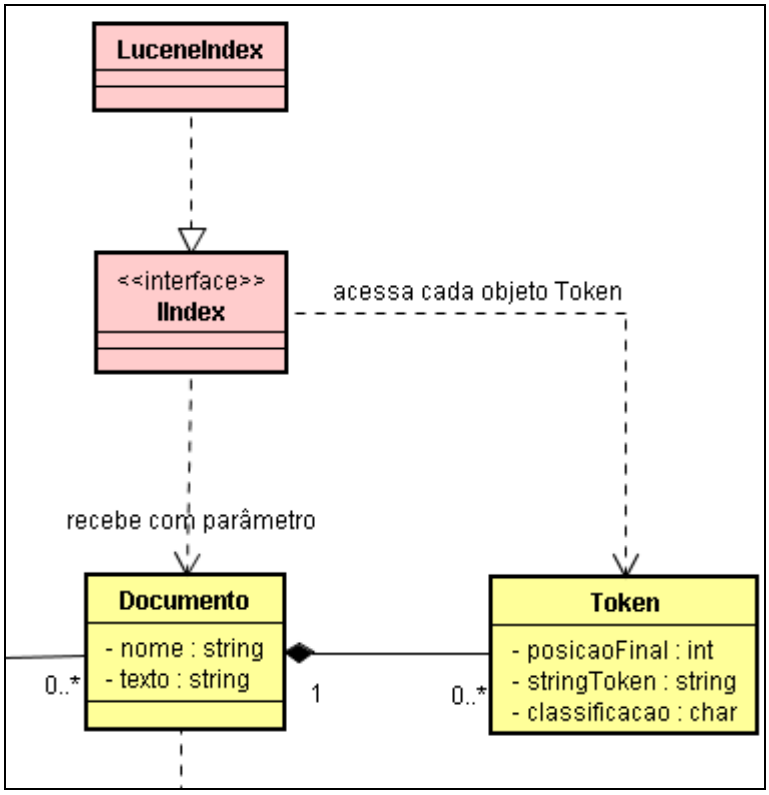

Figura 21 – Parte do diagrama de classes do sistema com a adição das partes envolvidas na indexação

[8]Disponível em http://lucene.apache.org

Desta forma, para que o usuário possa realizar a recuperação, é pré-requisito que todos os documentos envolvidos na indexação já tenham passado pela etapa de *tokenization*. Em seguida, é necessário que seja definido quais *corpus* deverão ser indexados, a partir do *menu* **Indexação->Indexar**. O usuário também deve definir um caminho físico no disco para que a estrutura de índices seja salva.

Para a recuperação, é necessário que seja acessado o *menu* **Indexação->Buscar Documentos**. Uma interface é apresentada aonde devem ser indicados a localização do índice e os parâmetros da busca, as palavras-chave. O usuário deve informar também se deseja encontrar documentos com todas as palavras informadas ao sistema, com pelo menos uma das palavras informadas, ou com todas as palavras digitadas e respeitando a ordem entre elas, recuperando documentos que contenham exatamente a mesma expressão digitada (“*Phrase Query”*). Há ainda a possibilidade de se definir filtros sobre os documentos recuperados, caso estes contenham alguma palavra definida em uma lista de filtro a parte.

Finalmente, todos os documentos recuperados são apresentados em lista única, contendo o nome do arquivo físico, a que *corpus* ele pertence e os cem primeiros caracteres para que se possa ter uma idéia do conteúdo sem que seja necessário efetivamente acessá-lo.

5.1.4. Módulo de Mineração

O módulo de mineração contém implementações de algoritmos que executam tarefas desta etapa, conforme visto no capítulo anterior. A tarefa desenvolvida no sistema de mineração foi a de **categorização automática de textos** (seção 4.1), com a implementação do algoritmo ***Naive Bayes***, explicado na seção (seção 4.1.3) desta dissertação. A implementação do algoritmo introduziu três classes ao diagrama de classes da solução, visto parcialmente na Figura 22.

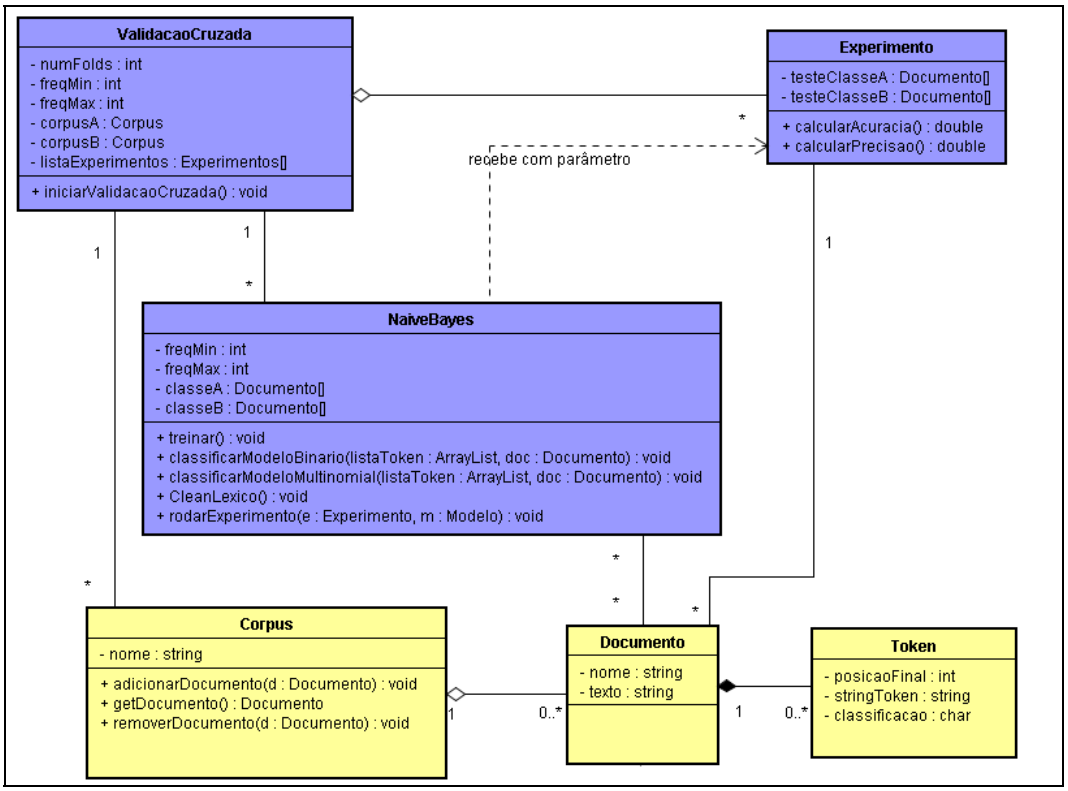

Figura 22 – Diagrama de classes parcial do sistema com a inclusão das classes que compõem o módulo de mineração.

A classe **NaiveBayes** é a principal, contendo o algoritmo propriamente dito. O classificador foi implementando de forma binária[9], isto é, apenas duas classes diferentes são possíveis para categorizar um documento. O algoritmo deve receber pelo menos um documento representante de cada classe de classificação, através dos atributos **classeA** e **classeB**, ambos vetores do tipo **Documento.** Em seguida, deve-se fazer uma chamada ao método **treinar**, dando início à fase de treinamento do algoritmo. Após ter sido treinado, o classificador já está "calibrado" para classificar documentos de forma automática, através de chamadas aos métodos **classificarModeloBinario** e **classificarModeloMultinomial** (explicados nas seções 4.1.3.2 e 4.1.3.3, respectivamente). A classe **NaiveBayes** não possui nenhum método responsável por realizar testes e validação sobre o treinamento, estando tais funcionalidades em classes diferentes, uma decisão que facilitou a codificação e aumentou a coesão entre os componentes.

[9]Não confundir com modelo Binário, explicado na seção 4.1.3.2. Implementação binária significa dizer que apenas duas classes de categorização são permitidas pela ferramenta, por exemplo, determinado documento é ou não é uma receita culinária.

A classe **Experimento** contém métodos para a realização de testes que indicam o poder discriminatório do classificador. Para tanto, um objeto desta classe deve receber também dois conjuntos de documentos, com cada um destes representando uma classe. Em seguida, deve se fazer uma chamada ao método **rodarExperimento**, presente na instância de **NaiveBayes** (classificador) previamente treinado, conforme descrito no parágrafo anterior. Nesta chamada, um **Experimento** deve ser enviado como parâmetro, juntamente com a indicação do modelo a ser usado na classificação (binário ou multinomial). O resultado do teste fica então retido no próprio objeto **Experimento**, sendo acessado através de métodos que realizam cálculos segundo determinadas métricas, como Precisão e Acurácia.

Finalmente, a classe **ValidacaoCruzada** é responsável por realizar a validação sugerida em seu próprio nome, conforme explicada na seção 4.1.1. A classe provê um método que, dado dois conjuntos distintos de documentos, realiza a divisão destes no número de *folds* desejados. Conseqüentemente, são criados o mesmo número de objetos do tipo **NaiveBayes** e **Experimento**, um par por *fold*. O resultado da validação cruzada, que é a média das métricas obtidas em cada experimento, deve ser calculado dentro do objeto que possui as instâncias de **ValidacaoCruzada**. O método único que realiza todos estes passos na classe **ValidacaoCruzada** é o **iniciarValidacaoCruzada**.

A utilização do algoritmo para treinamento é realizada através do menu **Mineração-> Classificação -> Naive Bayes** da ferramenta desenvolvida. O usuário deve preencher todos os campos solicitados, como quais *corpus* serão utilizados no treinamento, modelo (binário ou multinomial) e se deseja realizar validação cruzada antes de efetivamente registrar o classificador no sistema para posterior utilização.

5.1.5. Módulo de Análise de Resultados

Finalmente, no módulo de análise de resultados foram implementados relatórios e gráficos que auxiliam o usuário a compreender melhor os resultados obtidos ao longo das quatro etapas anteriores. A primeira implementação diz respeito a uma perspectiva gráfica acerca do treinamento do classificador, quando

da utilização da validação cruzada como forma de verificar o poder discriminatório do mesmo. Mesmo já sendo possível verificar este poder através de métricas, é interessante que o usuário perceba através da **matriz de confusão,** que tem como finalidade mostrar o número de classificações corretas em oposição às classificações preditas para cada classe, conforme verificado na Tabela 8.

Tabela 8 – Matriz de Confusão

	Documentos classificados como sendo da classe X	**Documentos classificados como sendo da classe Y**
Documentos da Classe X	**A**	**B**
Documentos da Classe Y	**C**	**D**

A partir da matriz de confusão fica fácil deduzir várias das métricas existentes a respeito da performance de classificadores. Por exemplo, a **acurácia** (seção 4.1.2) do classificador, que é o quanto ele classifica de forma correta os exemplos apresentados, é verificada através da soma das corretas classificações (quadrantes A e D) divido pelo total de exemplos apresentados (soma de todos os quadrantes, isto é, A, B, C e D). Já a **precisão**, que é o total de exemplo de determinada classe classificados corretamente, é verificada através do total de classificações corretas (quadrante D para a classe Y) divididos sobre o total de exemplos classificados como sendo da classe Y, corretos ou não (soma dos quadrantes B + D).

Em seguida, foi implementado também um relatório aonde se pode verificar a freqüência de determinado *token* dentro dos *corpus* envolvidos no treinamento do classificador, conforme visto na Tabela 9. Este relatório é essencialmente importante, principalmente no caso de um classificador probabilístico, possibilitando que sejam melhor compreendidas as predições, quando da utilização efetiva do sistema.

Tabela 9 – Relatório de freqüência dos tokens nos corpus envolvidos no treinamento do classificador

Token	**Freqüência *Corpus* X**	**Freqüência *Corpus* Y**	***StopWord***
ventilador	45	188	Não
ele	540	570	Sim
futebol	1040	20	Não

Finalmente, a última implementação deste módulo diz respeito, novamente, à freqüência de *tokens* dentro de *corpus*. Foi implementado um gráfico de barras que mostra o total de *tokens* agrupados por faixas de freqüência. Desta forma, é possível verificar, por exemplo, que no gráfico ilustrado na Figura 23 existem exatamente **dez** diferentes *tokens* que aparecem, no máximo, **quarenta** vezes em todo o *corpus*. Dentre as diversas utilidades deste tipo de gráfico, podemos destacar a possibilidade de se identificar *tokens* raros e *tokens* com grau de freqüência muito alto, para que possam ser removidos durante o treinamento do classificador, evitando-se, assim, possíveis "ruídos" [10].

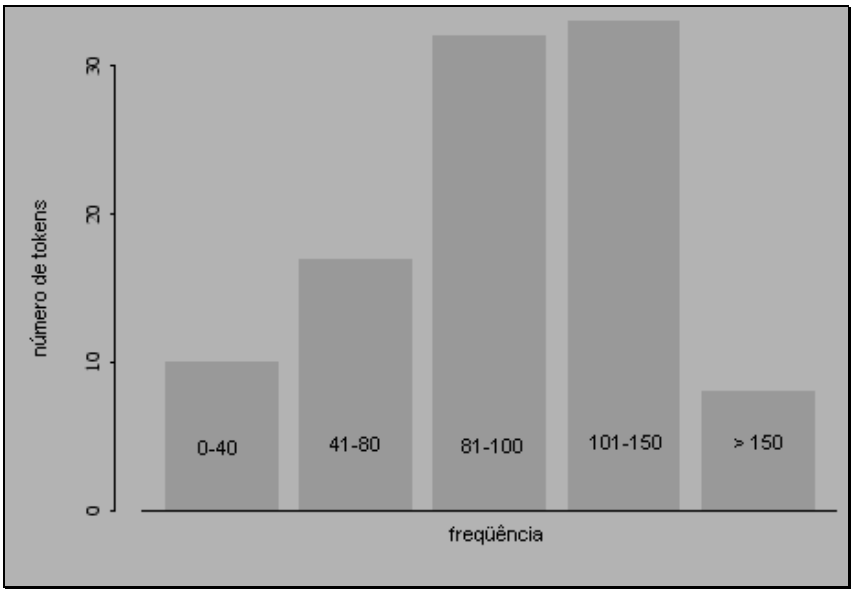

Figura 23 – Gráfico de barras que mostra a relação entre faixas de freqüência e o número total de *tokens* presentes.

[10] Ruídos podem ser considerados dados que, ao contrário do que se deseja, não agregam valor à descoberta de conhecimento, dificultando-a.

5.2. Estudo de Caso: Identificação de Subjetividade em Pesquisas de Opinião

O estudo de caso proposto nesta dissertação utilizou o sistema desenvolvido para a identificação de **subjetividade** em pesquisas de opinião. A subjetividade pode ser entendida como sendo um relato de cunho pessoal do comentarista, ao contrário da objetividade, a qual pode ser considerada como algo de consenso geral.

Pesquisas recentes identificaram que, em geral, a **subjetividade** e a **objetividade** aparecem em pesquisas de opinião de forma pontual, representadas por frases ou parágrafos. Isto é algo discutível, pois há quem defenda o contrário, afirmando que esta é uma constatação simplificadora, e que uma mesma frase ou parágrafo podem conter elementos subjetivos e objetivos.

Trabalhos relacionados à **Análise de Sentimento** (seção 2.2.1.1) ou, ***Sentiment Analisys*** em inglês, identificaram que a aplicação do chamado "filtro de subjetividade" melhora a identificação da polaridade de uma opinião, isto é, se esta é favorável ou desfavorável ao que se está sendo indagado ao comentarista [63][64]. Para um estudo mais específico sobre identificação de polaridade em opiniões com a utilização do filtro de subjetividade, consultar [22].

As aplicações de Análise de Sentimento, em geral, utilizam a tarefa de Categorização de Textos (seção 4.1) e seus algoritmos como forma de resolução do problema. Assim, um classificador deve ser submetido a treinamento, com amostras de textos previamente classificados como sendo subjetivos ou objetivos.

O presente estudo de caso tem como objetivo aplicar tal categorização em comentários de pessoas sobre seus filmes assistidos. Normalmente, neste tipo de texto há frases consideradas objetivas e que normalmente transcrevem parte do filme, numa presunção do comentarista de que aquele que está interessado em sua opinião ainda não o assistiu. Por nada agregar à análise de polaridade, a objetividade é então eliminada dos textos de comentários.

Inicialmente, precisou-se fazer uma **coleta** que fosse representativa. Foram coletados 5.000 (cinco mil) exemplos de comentários objetivos do site *The Internet Movie Database* (http://www.imdb.com), um repositório on-line de informações sobre filmes. Utilizou-se a própria sinopse dos filmes, que por via de regra apresentam uma síntese da obra.

Em contrapartida, a **coleta** de amostras subjetivas foi feita a parir do site *Rotten Tomatoes (*http://www.rottentomatoes.com), que também concentra um grande acervo de filmes e críticas de usuários. Ambos os *corpus* encontram-se na língua inglesa.

Em seguida, foi dado início à etapa de **pré-processamento**, com a atomização (*tokenization*) dos exemplos. Optou-se por se fazer uma atomização simples, o que empiricamente levou-se a concluir que eventuais perdas de informações pudessem ser atenuadas devido ao grande número de amostras coletadas. Como conseqüência ganhou-se em velocidade de processamento, pois a versão simples exige menor esforço computacional que aquela que utiliza regras e dicionários (seção 3.2.1).

Como dito anteriormente, todos os exemplos coletados são provenientes do inglês. Conseqüentemente, utilizou-se a lista padrão do Google de *stopwords* para definir que palavras seriam ignoradas no sistema, listadas na tabela abaixo:

Tabela 10 – Lista de stopwords padrão do Google.

I	in	who
a	is	will
about	it	with
an	la	und
are	of	the
as	on	www
at	or	
be	that	
by	the	
com	this	
de	to	
en	was	
for	what	
from	when	
how	where	

Utilizou-se a etapa de **indexação** para recuperação e manipulação de documentos, e ainda para que fosse possível descobrir *stopwords* inerentes ao domínio, com a verificação do total de documentos recuperados em resposta a buscas por palavras na qual já se tinha alguma intuição, como *movie*, *actor*, *cine*, dentre outras.

Já na fase de **mineração**, foi utilizado o classificador *Naive Bayes* implementado. Foram realizados diversos experimentos, com o objetivo de se

obter exatamente a melhor configuração de treinamento. A Tabela 11 apresenta a relação de experimentos realizados.

Tabela 11 – Resultados dos experimentos realizados com o classificador Naive Bayes no problema de subjetividade versus objetividade.

Experimento	Modelo	Excluir Stopwords	Validação Cruzada	Acurácia Média
1	Naive Bayes Binário	Não	4-Fold	**0.8927**
2	Naive Bayes Multinomial	Não	4-Fold	**0.8869**
3	Naive Bayes Binário	Sim	10-Fold	**0.9060**
4	Naive Bayes Multinomial	Sim	10-Fold	**0.8907**
5	Naive Bayes Binário	Sim	5-Fold	**0.8810**
6	Naive Bayes Multinominal	Sim	5-Fold	**0.8899**

A **acurácia** (seção 4.1.2) foi a medida escolhida para verificar a perfomance de cada uma das configurações. Todos os experimentos obtiveram valores bem próximos, conforme verificado, o que se levou à conclusão de que a performance foi satisfatória. Com o objetivo de se verificar o comportamento do conjunto de treinamento e teste com outro classificador, realizou-se experimentos também com outro classificador, o SVM, na versão implementada por Thorsten Joachims chamada de SVM^{light}, disponível em seu próprio endereço na Internet[11]. Os experimentos com o SVM estão na Tabela 12.

Tabela 12 – Resultados dos experimentos realizados com o classificador SVM no problema de subjetividade versus objetividade.

Experimento	Modelo	Excluir Stopwords	Validação Cruzada	Acurácia Média
1	SVM Linear -Binário	Sim	4-Fold	**0.8845**
2	SVM Linear – Multinomial	Sim	4-Fold	**0.8799**
3	SVM Linear – Binário	Sim	10-Fold	**0.8990**
4	SVM Linear - Multinomial	Sim	10-Fold	**0.89311**

Conforme era esperado, os resultados obtidos com o SVM foram praticamente os mesmos encontrados com o *Naive Bayes.* Finalmente, foi

[11]http://svmlight.joachims.org

realizada uma **análise** final com o melhor modelo verificado (*Naive Bayes*, experimento 4), através da categorização de novos exemplos, conforme sugerido na Tabela 13. Assim, a satisfatoriedade obtida através da alta **acurácia** apresentada pelos experimentos da fase de treinamento foi verificada e confirmada com a **correta** classificação dos novos exemplos.

Tabela 13 – Categorização obtida para novos exemplos utilizado o classificador *Naive Bayes.*

Exemplo	**Classificação**
television made him famous, but his biggest hits happened off screen .	Sentença Objetiva
this documentary by stacy peralta makes a convincing case on behalf of skateboarding , of all things , as a catalyst for this change , and for some delinquent youths in a derelict los angeles neighborhood as its agents .	Sentença Subjetiva
a team of professional thieves are hired to rob a jewelry exchange .	Sentença Objetiva
meticulously mounted , exasperatingly well-behaved film , which ticks off kahlo's lifetime milestones with the dutiful precision of a tax accountant .	Sentença Subjetiva