

Instituto de Ciências Matemáticas e de Computação

ISSN - XXXX-XXXX

PreTeXT: A Reestruturação da Ferramenta de Pré-Processamento de Textos

Matheus Victor Brum Soares

Ronaldo C. Prati

Maria Carolina Monard

Nº XXX

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos

Agosto/2008

# PreTexT: A Reestruturação da Ferramenta de Pré-Processamento de Textos

Matheus Victor Brum Soares\*

Ronaldo C. Prati\*

Maria Carolina Monard\*

\*Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Laboratório de Inteligência Computacional e-mail: {caneca, prati, mcmonard}@icmc.usp.br

---

**Resumo:** A quantidade de informação textual armazenada digitalmente vem crescendo a cada dia. No entanto, a nossa capacidade de processar e analisar essa informação não vem acompanhando este crescimento. Dessa maneira, é importante desenvolver processos semi-automáticos para extrair conhecimento relevante dessa informação, tais como o processo de mineração de textos. O pré-processamento de textos é uma das principais etapas da mineração de textos, e também uma das mais custosas. Essa etapa visa transformar texto não estruturado em um formato estruturado, como uma tabela atributo-valor. O PRETEXT é uma ferramenta computacional que realiza esse tipo de pré-processamento utilizando funcionalidades como n-grama, *stemming*, *stoplists*, cortes por frequência, taxonomias, normalizações, gráficos, medidas *tf*, *tf-idf*, *tf-linear*, *boolean*, entre outras. Esta ferramenta passou por uma reestruturação e reimplementação recentemente e este trabalho consiste em apresentar a fundo as funcionalidades e o modo de uso da nova versão da ferramenta PRETEXT.

**Palavras Chaves:** Mineração de Textos, Pré-Processamento, *Stemming*.

---

Agosto 2008

---

‡Trabalho realizado com apoio institucional da USP.

# Sumário

Sumário	ii
Lista de Figuras	iii
Lista de Tabelas	iii
<b>1 Introdução</b>	<b>1</b>
<b>2 Pré-Processamento de Textos</b>	<b>1</b>
2.1 Construção de Atributos a Partir de Bases Textuais . . . . .	3
2.1.1 Tokenização . . . . .	3
2.1.2 <i>Stemming</i> . . . . .	4
2.1.3 Taxonomias . . . . .	5
2.1.4 Remoção de Stopwords . . . . .	5
2.1.5 Cortes de Palavras Baseado em Frequência . . . . .	6
2.1.6 <i>N</i> -grama . . . . .	7
2.2 Valores de Atributos . . . . .	7
2.2.1 <i>Boolean</i> . . . . .	8
2.2.2 <i>Term Frequency</i> . . . . .	8
2.2.3 <i>Term Frequency Linear</i> . . . . .	9
2.2.4 <i>Term Frequency - Inverse Document Frequency</i> . . . . .	10
2.2.5 Suavização . . . . .	11
2.2.6 Normalizações . . . . .	11
<b>3 Remodelagem e Reimplementação do PreText</b>	<b>13</b>
3.1 O Arquivo de Configuração <i><b><code>&lt;code&gt;XML&lt;/code&gt;</code></b></i> . . . . .	15
3.2 O módulo <i><b><code>&lt;code&gt;Maid.pm&lt;/code&gt;</code></b></i> . . . . .	21
3.2.1 Arquivos de Entrada . . . . .	22
3.2.2 Arquivos de Saída . . . . .	24
3.2.3 Execução do Módulo <i><b><code>&lt;code&gt;Maid.pm&lt;/code&gt;</code></b></i> . . . . .	25
3.2.4 Advertências . . . . .	27
3.3 O módulo <i><b><code>&lt;code&gt;NGram.pm&lt;/code&gt;</code></b></i> . . . . .	27
3.3.1 Arquivos de Entrada . . . . .	27
3.3.2 Arquivos de Saída . . . . .	27
3.3.3 Execução do Módulo <i><b><code>&lt;code&gt;NGram.pm&lt;/code&gt;</code></b></i> . . . . .	28
3.3.4 Advertências . . . . .	29
3.4 O módulo <i><b><code>&lt;code&gt;Report.pm&lt;/code&gt;</code></b></i> . . . . .	29

3.4.1	Arquivos de Entrada . . . . .	30
3.4.2	Arquivos de Saída . . . . .	31
3.4.3	Execução do Módulo <code>Report.pm</code> . . . . .	33
3.4.4	Advertências . . . . .	34
3.5	<i>Script</i> Auxiliar de Configuração . . . . .	34
3.6	Instalação . . . . .	34
3.6.1	Windows . . . . .	34
3.6.2	Linux . . . . .	35
<b>4</b>	<b>Arquitetura de Classes</b>	<b>35</b>
4.1	<code>Start.pl</code> . . . . .	35
4.2	<code>Maid.pm</code> . . . . .	36
4.3	<code>NGram.pm</code> . . . . .	39
4.4	<code>Report.pm</code> . . . . .	39
<b>5</b>	<b>Considerações Finais</b>	<b>42</b>
	<b>Referências</b>	<b>42</b>

## Lista de Figuras

1	A curva de Zipf e os cortes de Luhn . . . . .	7
2	O novo funcionamento do PRETEXT . . . . .	14
3	Exemplo básico do arquivo de configuração <code>config.xml</code> . . . . .	16
4	Exemplo detalhado do arquivo de configuração <code>config.xml</code> . . . . .	17
5	Exemplo do arquivo de símbolos <code>simbols.xml</code> . . . . .	22
6	Exemplo de um <i>stopfile</i> . . . . .	23
7	Exemplo da organização do diretório de documentos rotulados. . . . .	24
8	Exemplo do arquivo <code>stemWdST.all</code> . . . . .	25
9	Exemplo de execução do módulo <code>Maid.pm</code> . . . . .	26
10	Exemplo do arquivo <code>1Gram.all</code> . . . . .	28
11	Exemplo do arquivo <code>1Gram.txt</code> . . . . .	28
12	Exemplo de execução do módulo <code>NGram.pm</code> . . . . .	29
13	Exemplo de um arquivo de taxonomia. . . . .	30
14	Exemplo dos arquivos <code>.names</code> (a) e <code>.data</code> (b). . . . .	31
15	Exemplo do gráfico 1-StdDev. . . . .	32
16	Exemplo do gráfico 1-StdDevFF. . . . .	32
17	Exemplo de execução do módulo <code>Report.pm</code> . . . . .	33

18	Diagrama de classes do módulo <code>Start.pl</code> . . . . .	36
19	Diagrama de classes do módulo <code>Maid.pm</code> . . . . .	37
20	Símbolos transformados em divisores de frase por <i>default</i> . . . . .	37
21	Diagrama de classes do módulo <code>NGram.pm</code> . . . . .	39
22	Diagrama de classes do módulo <code>Report.pm</code> . . . . .	40

## Lista de Tabelas

1	Representação de documentos no formato atributo-valor . . . . .	2
2	Tabela atributo-valor utilizando medida <i>boolean</i> . . . . .	8
3	Tabela atributo-valor utilizando medida <i>tf</i> . . . . .	9
4	Tabela atributo-valor utilizando medida <i>tf-linear</i> . . . . .	10
5	Tabela atributo-valor utilizando medida <i>tf-idf</i> . . . . .	10
6	Tabela atributo-valor utilizando medida <i>tf</i> e a normalização linear por linha. . . . .	12
7	Tabela atributo-valor utilizando medida <i>tf</i> e a normalização linear por coluna. . . . .	12
8	Tabela atributo-valor utilizando medida <i>tf</i> e a normalização quadrática por linha. . . . .	13
9	Tabela atributo-valor utilizando medida <i>tf</i> e a normalização quadrática por coluna. . . . .	13
10	Funcionalidades da nova versão do PRETEXT . . . . .	15
11	<i>Rank</i> de <i>stems</i> construído a partir da frequência . . . . .	21
12	Codificação <b><i>XML</i></b> de alguns símbolos. . . . .	23
13	Transformação da tabela original para a tabela com taxonomias. . . . .	30
14	Transformações de entidades HTML realizadas pelo PRETEXT. . . . .	38

# 1 Introdução

O uso freqüente de computadores gera, como consequência, uma grande quantidade de dados digitais que precisam ser analisados. A mineração de textos (MT) (Weiss et al., 2004) é um processo que pode ser utilizado para extrair informações úteis dessa grande quantidade de textos digitais gerados no dia a dia. A mineração de textos tem diversas aplicações tais como: classificação de documentos, recuperação de informação, organização de documentos e extração de informação.

Dentre as etapas da MT está a etapa de pré-processamento de textos que consiste em transformar documentos textuais em um formato estruturado, tal como uma tabela atributo-valor, para que possam ser aplicados algoritmos de aprendizado de máquina (Mitchell, 1997; Monard & Baranauskas, 2003) para extrair conhecimento dessa informação textual. Porém essa transformação é um processo custoso e demorado que deve ser feito com cuidado para que o conhecimento adquirido posteriormente seja útil para o usuário final.

Existe no LABIC<sup>1</sup> uma ferramenta para pré-processamentos de textos chamada PRETEXT (Matsubara et al., 2003) que recentemente foi reestruturada e reimplementada de forma a atender um número maior de necessidades dos usuários que necessitam realizar o pré-processamento de textos em grandes conjuntos de documentos textuais de forma mais rápida e com mais liberdade para a escolha de suas funções. A nova versão da ferramenta PRETEXT será explicada em detalhes neste trabalho.

Este trabalho está dividido da seguinte maneira: no Capítulo 2 está uma revisão bibliográfica dos conceitos e métodos utilizados na ferramenta PRETEXT, o Capítulo 3 está a descrição da ferramenta, seus módulos principais e modos de utilização, o Capítulo 4 é dado uma breve descrição de todas as classes da nova arquitetura da ferramenta, assim como a forma de interação entre elas, o Capítulo 5 estão as considerações finais deste trabalho.

## 2 Pré-Processamento de Textos

Como mencionado, uma das grandes dificuldades da MT é que dados textuais geralmente não estão em formato estruturado. Esse fato implica em uma limitação à utilização de algoritmos de aprendizado de máquina, pois esses algoritmos geralmente necessitam que

---

<sup>1</sup><http://labic.icmc.usp.br/>

os dados estejam representados de uma maneira estruturada. A transformação de textos não estruturados em dados estruturados requer o pré-processamento dos textos.

Uma das maneiras de transformar textos em dados estruturados é transformá-los em uma representação atributo-valor utilizando a abordagem *bag of words*, na qual a frequência das palavras (termos), independentes de seu contexto ou significado, são contadas. A partir dessa contagem gerada uma tabela cujas entradas contém informações relacionadas à frequência de cada palavra. Uma representação de documentos usando a abordagem *bag of words* no formato de uma tabela atributo-valor é mostrada na Tabela 1.

	$t_1$	$t_2$	$t_3$	$\dots$	$t_M$	Classe( $C$ )
$d_1$	$a_{11}$	$a_{12}$	$a_{13}$	$\dots$	$a_{1M}$	$c_1$
$d_2$	$a_{21}$	$a_{22}$	$a_{23}$	$\dots$	$a_{2M}$	$c_2$
$d_3$	$a_{31}$	$a_{32}$	$a_{33}$	$\dots$	$a_{3M}$	$c_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$d_N$	$a_{N1}$	$a_{N2}$	$a_{N3}$	$\dots$	$a_{NM}$	$c_N$

Tabela 1: Representação de documentos no formato atributo-valor

Nessa tabela, cada documento  $d_i$ , para  $i$  variando de 1 até o número total  $N$  de documentos considerados, é um exemplo da tabela e cada palavra  $t_j$ , para  $j$  variando de 1 até o tamanho  $M$  do vocabulário utilizado, é um atributo. Diversos métodos para a construção desses atributos são descritos na Seção 2.1. O atributo  $C$  é um atributo especial, geralmente denominado atributo classe, que designa a classe à qual pertence cada documento. Cada  $c_i$  pode assumir um valor do conjunto  $C = \{C_1, \dots, C_{N_{cl}}\}$ , no qual  $N_{cl}$  é o número de classes. Cada entrada  $a_{ij}$  refere-se ao valor correspondente ao documento (exemplo)  $i$  associado ao termo (atributo)  $j$ . Existem várias maneiras de associar valores a cada um dos termos em um documento, sendo as mais comuns a presença ou ausência da palavra, número absoluto de aparições das palavras (frequência absoluta), ou a frequência relativa dessas palavras em relação ao número de documentos. Essas representações são discutidas na Seção 2.2.

Uma tabela atributo-valor gerada utilizando a abordagem *bag of words* tem algumas peculiaridades. Geralmente a utilização dessa tabela leva a um problema chamado maldição da dimensionalidade, na qual a tabela apresenta um grande número de atributos (devido ao grande número de palavras – vocabulário – utilizados no conjunto de documentos). Entretanto, cada documento utiliza relativamente poucos atributos para a sua descrição (o texto de cada documento geralmente consiste em um subconjunto pequeno de todas as palavras existentes do conjunto de documentos), gerando assim uma tabela muito esparsa.

Muitos algoritmos de aprendizado não estão preparados para esse tipo de tabelas esparsas, deixando o processamento ineficiente, ou até inviável. Dessa maneira, alguns métodos de redução de atributos devem ser utilizados a fim de condensar a informação pertinente para a etapa de extração de conhecimento do processo de MT. Outra característica especial desses dados é a presença de somente valores positivos. Essa característica também deve ser levada em consideração com o objetivo de encontrar os melhores atributos para descrever o conjunto de documentos.

## 2.1 Construção de Atributos a Partir de Bases Textuais

Na transformação de documentos textuais em tabelas atributo-valor, existem alguns métodos para auxiliar na redução do número de atributos visando melhorar a relevância da informação para a classificação do texto. Esses métodos são brevemente descritos a seguir.

### 2.1.1 Tokenização

O primeiro passo para examinar um texto não estruturado é identificar suas características importantes. Uma das maneiras de fazer isso é quebrar o fluxo contínuo de caracteres em palavras ou, também chamado de *tokens*<sup>2</sup>. Esse processo é trivial para uma pessoa que tenha conhecimento da estrutura da linguagem. Porém, para um programa de computador, isso pode ser mais complicado. Para realizar esse processo, é necessário a remoção de alguns caracteres indesejados, tais como sinais de pontuação, separação silábica, marcações especiais e números, os quais, isoladamente, fornecem pouca informação.

O processo de tokenização é uma tarefa não trivial, devido à forma como os *tokens* devem ser extraídos do texto (dos Santos, 2002; Manning & Schütze, 1999). Em alguns casos um espaço em branco não é suficiente para auxiliar no reconhecimento de um *token*, visto que em um texto usualmente existem sinais de pontuação, como vírgula, ou ponto final, que não fazem parte de um determinado *token*. Desta maneira, se for considerado somente um espaço em branco para a divisão de *tokens*, podem existir *tokens* que deveriam ser semelhantes, porém, eles são agrupados de maneiras distintas. Existe também a ambigüidade do ponto final, que não permite determinar quando esse ponto representa uma divisão entre sentenças e quando representa uma abreviação. Dentre os símbolos não alfanuméricos, existem aqueles que têm uma relevância para o aprendizado, e aqueles que simplesmente podem ser ignorados. A existência de letras maiúsculas e minúsculas

---

<sup>2</sup>Neste trabalho palavras, *tokens* e termos são utilizados indistintamente como sinônimos



também pode dificultar o processo de tokenização, pois uma mesma palavra ocorrendo no início de uma frase (usualmente escrita com maiúscula) pode ser agrupada em um *token* diferente da mesma palavra ocorrendo no meio da frase. Esses aspectos devem ser cuidadosamente considerados, pois uma extração eficiente de *tokens* gera melhores resultados, *i.e.* diminuição do número de atributos finais na abordagem *bag of words*.

### 2.1.2 *Stemming*

Um dos métodos amplamente utilizado e difundido que pode ser utilizado a fim de reduzir a quantidade de *tokens* necessários para representar uma coleção de documentos é a transformação de cada termo para o radical que o originou, por meio de algoritmos de *stemming*. Basicamente, algoritmos de *stemming* consistem em uma normalização lingüística, na qual as formas variantes de um termo são reduzidas a uma forma comum denominada *stem*. A consequência da aplicação de algoritmos de *stemming* consiste na remoção de prefixos ou sufixos de um termo. Por exemplo, os *tokens* **observar**, **observadores**, **observasse**, **observou** e **observe** podem ser transformados para um mesmo *stem* **observ**.

Os algoritmos de *stemming* são fortemente dependentes do idioma no qual os documentos estão escritos. Um dos algoritmos de *stemming* mais conhecidos é o algoritmo do Porter, que remove sufixos de termos em inglês (Porter, 1980, 2006). Esse algoritmo tem sido amplamente usado, referenciado e adaptado nas últimas três décadas. Diversas implementações do algoritmo estão disponibilizadas na WEB, inclusive na sua página oficial<sup>3</sup>, escrita e mantida pelo autor para a distribuição do seu algoritmo. Existe também uma linguagem de programação especialmente criada para geração de *stem* para vários idiomas, chamada *Snowball* (Porter, 2001).

Para a língua portuguesa, existem algoritmos de *stemming* que foram adaptados do algoritmo de Porter. Nesses algoritmos, os sufixos dos *tokens*, com um comprimento mínimo estabelecido, são eliminados considerando se algumas regras pré-estabelecidas. Caso não seja possível eliminar nenhum sufixo de acordo com essas regras, são analisadas as terminações verbais da palavra. Essa é a principal diferença entre o algoritmo de *stemming* para palavras em inglês e para palavras em português ou espanhol, por exemplo. Enquanto na língua inglesa a conjugação dos verbos é quase inexistente para verbos regulares, pois usualmente acrescenta-se a letra **s** no final do verbo no presente na terceira pessoa do singular, as linguagens provenientes do latim apresentam formas verbais altamente conjugadas em sete tempos, que contêm seis terminações diferentes cada tempo.

---

<sup>3</sup><http://www.tartarus.org/~martin/PorterStemmer>

Portanto, é necessário ter um tratamento diferenciado para essas terminações verbais.

É pouco provável que o algoritmo de *stemming* retorne o mesmo *stem* para todos os *tokens* que tenham a mesma origem ou radical morfológico, pois a maioria dos algoritmos de *stemming* ignoram o significado dos termos, introduzindo alguns erros. *Tokens* com significados diferentes podem ser reduzidos a um mesmo *stem*, nesse caso ocorre um erro de *over-stemming*. Por exemplo, os *tokens* **barato** e **barata** podem ser reduzidos a um mesmo *stem* **barat**. Já *tokens* com significados similares quando reduzidos a *stem* diferentes conduzem ao erro de *under-stemming*. Por exemplo, os *tokens* **viagem** e **viajar** podem ser reduzidos aos *stems* **viag** e **viaj**, respectivamente. Outro tipo de erro, denominado de *mis-stemming*, consiste em retirar o sufixo de um *token* quando na verdade essa redução não é necessária. Por exemplo, o *token* **lápis** poderia ser reduzido ao *stem* **lapi**, dependendo de como o plural das palavras é tratado pelo algoritmo de *stemming*. Já foi observado que a medida que o algoritmo se torna mais específico na tentativa de minimizar a quantidade de *tokens* diferentes para palavras com um mesmo radical, a eficiência do algoritmo degrada (Porter, 1980).

### 2.1.3 Taxonomias

Para certos domínios da aplicação, alguns *tokens* diferentes podem fazer mais sentido se analisados em um nível de abstração mais elevado. Por exemplo, **maçã**, **manga**, e **uva** podem ter um sentido mais geral se forem todos considerados como **fruta**. Esse tipo de generalização de termos pode auxiliar o processo de aprendizado. A taxonomia é a classificação de objetos baseado em similaridades entre eles. Essa classificação no pré-processamento de textos é realizada levando-se em consideração a semântica dos *token* presentes no conjunto de documentos. Entretanto, para muitos domínios, não existe uma taxonomia de termos, o que dificulta a utilização de métodos baseados em taxonomias. Nesse caso, essa etapa, caso utilizada, deve ser assistida por um especialista no domínio.

### 2.1.4 Remoção de Stopwords

Em qualquer língua, várias palavras são muito comuns e não são significativas para o algoritmo de aprendizado quando consideradas isoladamente. Essas palavras incluem pronomes, artigos, preposições, advérbios, conjunções, entre outras. Essas palavras são geralmente chamadas de *stopwords* em MT. Para essas palavras, pode ser gerada uma *stoplist*, na qual inúmeras *stopwords* são armazenadas para que sejam desconsideradas ao se processar o texto. Dessa forma, a remoção de *stopwords* minimiza consideravelmente

a quantidade total de *tokens* usada para representar documentos. Portanto, o uso das *stoplists* auxilia no processo de mineração de textos, removendo palavras quais se sabe, a priori, que não são relevantes para caracterizar os textos.

### 2.1.5 Cortes de Palavras Baseado em Frequência

Outra forma de reduzir o número de atributos é encontrar os *tokens* mais representativos dentre os existentes. A Lei de Zipf (Zipf, 1949) pode ser usada para encontrar termos considerados pouco representativos em uma determinada coleção de documentos. Existem diversas maneiras de enunciar a Lei de Zipf para uma coleção de documentos. A mais simples é procedimental: pegar todos os termos na coleção e contar o número de vezes que cada termo aparece. Se o histograma resultante for ordenado de forma decrescente, ou seja, o termo que ocorre mais freqüentemente aparece primeiro, e assim por diante. Então, a forma da curva é a “curva de Zipf” para aquela coleção de documentos. Para a maioria dos idiomas, se a curva de Zipf for plotada em uma escala logarítmica, ela corresponde a como uma reta com inclinação de aproximadamente  $-1$ .

Enquanto Zipf verificou sua lei utilizando jornais escritos em inglês, Luhn usou a lei como uma hipótese para especificar dois pontos de corte para excluir *tokens* não relevantes (Luhn, 1958) em uma coleção de documentos. Os termos que excedem o corte superior são os mais freqüentes e são considerados comuns por aparecer em qualquer tipo de documento, como as preposições, conjunções e artigos. Já os termos abaixo do corte inferior são considerados raros e, portanto, não contribuem significativamente na discriminação dos documentos. Na Figura 1 é mostrada a curva da Lei de Zipf (I) e os cortes de Luhn aplicados a Lei de Zipf (II), no qual o eixo cartesiano  $f$  representa a freqüência das palavras e o eixo cartesiano  $r$ ,  $r = 1, 2, 3, \dots$ , as palavras correspondentes ordenadas segundo essa freqüência. Por exemplo, para  $r = 1$ ,  $f_1$  representa a o *token* de maior freqüência; para  $r = 2$ ,  $f_2$  representa o *token* com segunda maior freqüência, e assim por diante.

Assim, Luhn propôs uma técnica para encontrar termos relevantes, assumindo que os termos mais significativos para discriminar o conteúdo do documento estão em um pico imaginário posicionado no meio dos dois pontos de corte. Porém, uma certa arbitrariedade está envolvida na determinação dos pontos de corte, bem como na curva imaginária, os quais devem ser estabelecidos por tentativa e erro (Van Rijsbergen, 1979). A Lei de Zipf não é restrita apenas aos *tokens*, mas também a *stems* ou *n*-grama dos documentos.

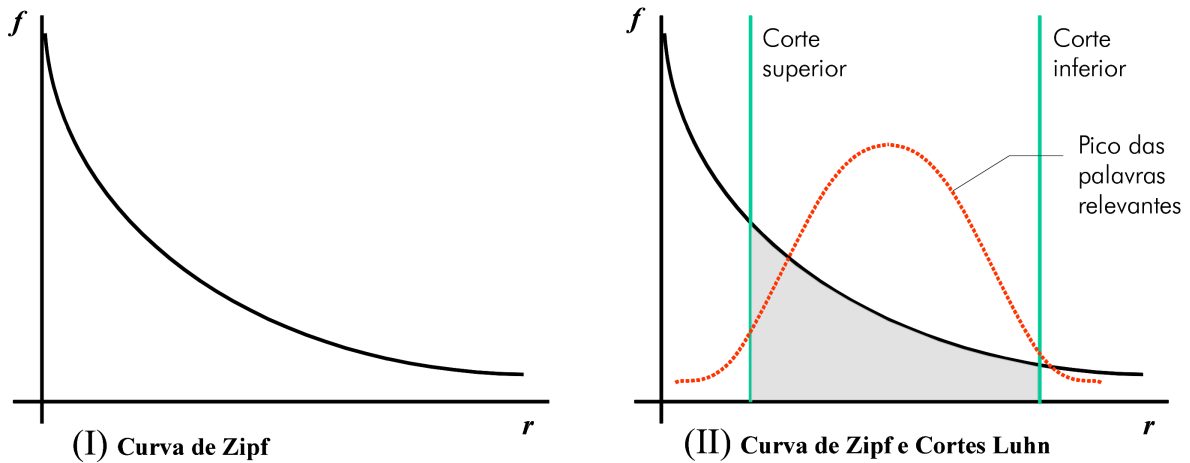


Figura 1: A curva de Zipf e os cortes de Luhn

### 2.1.6 *N*-grama

A ocorrência de palavras em sequência pode conter mais informação do que palavras isoladas. Desse modo, criando-se atributos pela união de duas ou mais palavras consecutivas, pode-se gerar atributos com um maior poder de predição. O *n*-grama<sup>4</sup> é exatamente essa junção de palavras, onde *n* representa o número de palavras que foram unidas para a geração de um atributo. Existe também a abordagem de *n*-grama com utilização de janelas. Nesse caso, os *tokens* são unidos não só com seu vizinho diretamente adjacente, mas também com *tokens* mais distantes, respeitando o valor especificado na janela. Existem vários *n*-grama que são gerados por simples acaso, porém, aqueles que apresentam uma frequência maior podem ser muito úteis para o aprendizado. Por exemplo, considerar as palavras São e Paulo individualmente pode agregar pouco conhecimento, pois São pode referir-se ao verbo *ser* e Paulo é um nome próprio relativamente comum no Brasil. Entretanto, o termo composto São Paulo pode agregar muito mais informação se o texto se refere à cidade ou estado de São Paulo.

## 2.2 Valores de Atributos

Como mencionado, na Tabela 1 cada documento  $d_i$  é um vetor  $d_i = (a_{i1}, a_{i2}, \dots, a_{iM})$ , no qual o valor  $a_{ij}$  refere-se ao valor associado ao  $j$ -ésimo termo (atributo) do documento  $i$ . O valor  $a_{ij}$  do termo  $t_j$  no documento  $d_i$  pode ser calculado utilizando diferentes medidas, levando em consideração a frequência que os termos aparecem nos documentos (Salton & Buckley, 1988). Existem várias medidas que podem ser utilizadas para calcular o

<sup>4</sup>É importante ressaltar que a palavra grama deve ser usada sempre no singular.

valor dos atributos na tabela atributo-valor. Para exemplificar o uso de algumas dessas medidas, brevemente descritas a seguir, serão utilizados os *tokens* **divert**, **cas**, **futebol**, **amig**, **jant** e **famil** e sete documentos.

### 2.2.1 Boolean

Esta medida atribui o valor um (verdadeiro) ao atributo se ele existe no documento e zero (falso) caso contrário, como mostrada na Equação 1.

$$a_{ij} = \begin{cases} 1, & \text{se } t_j \text{ ocorre em } d_i \\ 0, & \text{caso contrário} \end{cases} \quad (1)$$

Na Tabela 2, o Documento 1 recebe o valor 1 no atributo **famil** pois ele contém uma ou mais palavras que foram transformadas no *token* **famil**, em contra partida, recebe o valor 0 em todos os outros atributos por não possuir palavras que se transformaram nos outros diversos *tokens*. Os valores dos atributos dos outros documentos de 2 a 7, são calculados da mesma maneira.

	<b>divert</b>	<b>cas</b>	<b>futebol</b>	<b>amig</b>	<b>jant</b>	<b>famil</b>
Documento 1	0	0	0	0	0	1
Documento 2	0	0	0	0	1	1
Documento 3	0	0	0	1	1	1
Documento 4	0	0	1	1	1	1
Documento 5	0	1	1	1	1	1
Documento 6	1	1	1	1	1	1
Documento 7	1	1	1	1	1	1

Tabela 2: Tabela atributo-valor utilizando medida *boolean*.

### 2.2.2 Term Frequency

A representação binária nem sempre é adequada, pois em muitos casos deve ser utilizada uma medida levando em consideração a frequência que um termo aparece no documento. O *tf* — *Term Frequency* — consiste na contagem de aparições de um determinado atributo (termo) em um documento, atribuindo-se essa contagem ao valor do atributo (frequência absoluta). Essa medida é definida pela Equação 2, na qual  $freq(t_j, d_i)$  é a frequência do termo  $t_j$  no documento  $d_i$ .

$$a_{ij} = tf(t_j, d_i) = freq(t_j, d_i) \quad (2)$$

Na Tabela 3 é exemplificada a utilização da medida *tf*. Por exemplo, no Documento 3 o atributo **jant** recebe o valor 3 pois nesse documento existem exatamente 3 palavras que foram transformadas no *token* **jant**, e de forma análoga o atributo **famil** recebe o valor 2. Os valores dos atributos dos outros documentos são calculados da mesma maneira.

	<b>divert</b>	<b>cas</b>	<b>futebol</b>	<b>amig</b>	<b>jant</b>	<b>famil</b>
Documento 1	0	0	0	0	0	1
Documento 2	0	0	0	0	1	1
Documento 3	0	0	0	1	3	2
Documento 4	0	0	1	2	6	2
Documento 5	0	1	3	6	2	3
Documento 6	1	4	2	7	4	2
Documento 7	7	4	6	9	3	4

Tabela 3: Tabela atributo-valor utilizando medida *tf*.

### 2.2.3 *Term Frequency Linear*

Alguns termos aparecem na maioria dos documentos e raramente fornecem informações úteis que possam diferenciar os documentos em uma tarefa de mineração de textos. É possível utilizar também informações que indiquem a frequência com que um termo aparece na coleção de documentos. Nesse caso, um fator de ponderação pode ser utilizado para que os termos que aparecem na maioria dos documentos tenham um peso de representação menor. A *tf-linear* — *Term Frequency Linear* — (Matsubara et al., 2003) definida pelas Equações 3 e 4, utiliza um fator linear de ponderação. Esse fator é dado por um menos a frequência relativa do número de documentos em que o termo aparece no número total de documentos.

$$a_{ij} = tflinear(t_j, d_i) = freq(t_j, d_i) \times linear(t_j) \quad (3)$$

$$linear(t_j) = 1 - \frac{d(t_j)}{N} \quad (4)$$

Na Tabela 4 é mostrado o valor do *tf-linear* levando-se em consideração a frequência dos atributos mostrados na Tabela 3. Dessa maneira, o atributo **divert** que aparece em dois dos sete documentos existentes recebe o fator de ponderação  $linear(\mathbf{divert}) = 1 - \frac{2}{7} \cong 0,714$ , e é multiplicado pela sua frequência absoluta. Por exemplo no Documento 7, isso gera o valor  $tflinear(\mathbf{divert}, \text{Documento 7}) = 7 \times 0,714 \cong 5,0$ . Os valores dos atributos dos outros documentos são calculados da mesma maneira.

	divert	cas	futebol	amig	jant	famil
Documento 1	0,0	0,0	0,0	0,0	0,0	0,0
Documento 2	0,0	0,0	0,0	0,0	0,1	0,0
Documento 3	0,0	0,0	0,0	0,3	0,4	0,0
Documento 4	0,0	0,0	0,4	0,6	0,9	0,0
Documento 5	0,0	0,6	1,3	1,7	0,3	0,0
Documento 6	0,7	2,3	0,9	2,0	0,6	0,0
Documento 7	5,0	2,3	2,6	2,6	0,4	0,0

Tabela 4: Tabela atributo-valor utilizando medida *tf-linear*.

#### 2.2.4 Term Frequency - Inverse Document Frequency

A *tf-idf* — *Term Frequency - Inverse Document Frequency* — também é uma medida que pondera a frequência dos termos, de tal maneira que termos que aparecem na maioria dos documentos tenham um peso de representação menor (Jones, 1972; Robertson, 2004). Nesse caso, o fator de ponderação *idf* é inversamente proporcional ao logaritmo do número de documentos em que o termo aparece no número total  $N$  de documentos — Equação 5 e 6.

$$a_{ij} = tfidf(t_j, d_i) = freq(t_j, d_i) \times idf(t_j) \quad (5)$$

$$idf(t_j) = \log \frac{N}{d(t_j)} \quad (6)$$

Um exemplo da medida *tf-idf* é apresentado na Tabela 5. O atributo **futebol** recebe o fator de ponderação  $idf(\text{futebol}) = \log \frac{7}{4} \cong 0,243$  por aparecer em quatro dos sete documentos existentes (Tabela 3). No Documento 5, o atributo **futebol** recebe o valor de  $tfidf(\text{futebol}, \text{Documento 5}) = 3 \times 0,243 \cong 0,7$ . Os valores dos atributos dos outros documentos são calculados da mesma maneira.

	divert	cas	futebol	amig	jant	famil
Documento 1	0,0	0,0	0,0	0,0	0,0	0,0
Documento 2	0,0	0,0	0,0	0,0	0,1	0,0
Documento 3	0,0	0,0	0,0	0,1	0,2	0,0
Documento 4	0,0	0,0	0,2	0,3	0,4	0,0
Documento 5	0,0	0,4	0,7	0,9	0,1	0,0
Documento 6	0,5	1,5	0,5	1,0	0,3	0,0
Documento 7	3,8	1,5	1,5	1,3	0,2	0,0

Tabela 5: Tabela atributo-valor utilizando medida *tf-idf*.

### 2.2.5 Suavização

Não raramente, em algumas coleções de documentos é possível que alguns *tokens* apareçam em todos documentos da coleção. Dessa maneira, os fatores de ponderação *idf* e *linear* se tornam nulos. Assim sendo, os valores desses *tokens* são zerados para todos os documentos, e o *tokens* é inutilizado nesta coleção de documentos. Uma solução para este problema é fazer com que os fatores de ponderação nunca sejam nulos, a partir de um critério de suavização, *i.e.* *smooth*. Esse critério de suavização somente é ativado quando o fator de ponderação é igual a zero. Quando ativado, ele altera o fator de ponderação de forma a não permitir que seja nulo.

Uma abordagem que pode ser utilizada para a suavização, é aumentar temporariamente em 10% a variável  $N$  que contém o número de documentos da coleção. Sendo assim, o fator de ponderação não pode ser considerado zero.

Suponha que o valor de *idf* para um termo que apareça em todos os documentos é  $\log \frac{100}{100} = 0$ . Com o fator de ponderação igual a zero, o *smooth* é ativado. O valor de  $N$  é aumentado temporariamente para 110 obtendo assim  $\log \frac{110}{100} = 0.04$ . Desse modo, os valores da coluna correspondente a esse termo serão multiplicados por 0.04 ao invés de 0.

### 2.2.6 Normalizações

Um aspecto importante que também deve ser levado em consideração é o tamanho dos documentos na coleção. Frequentemente, o tamanho desses documentos é muito diferente e essa diferença de tamanho poderia estar melhor refletida nas medidas utilizadas. Por exemplo, considere dois documentos que pertencem à mesma categoria com tamanhos de 1 Kbyte e 100 Kbytes respectivamente. Nesse caso, existirá uma grande diferença na frequência dos termos em ambos os documentos. Uma possível solução para esse problema é normalizar os valores da tabela atributo-valor. Essa normalização pode ter seu foco nas colunas (ou atributos), ou nas linhas (ou documentos).

Frequentemente essa normalização é realizada utilizando a normalização linear, ou normalização quadrática. Em ambos casos, o máximo valor de um atributo é 1. A normalização linear, é definida pela Equação 7 quando utilizado a normalização por linhas, e é definida pela Equação 8 quando utilizado a normalização por colunas.

$$NormLinear(t_j, d_i) = \frac{a_{ij}}{MAX_{k=1..N}(a_{kj})} \quad (7)$$



$$NormLinear(t_j, d_i) = \frac{a_{ij}}{MAX_{k=1..N}(a_{ik})} \quad (8)$$

A normalização quadrática, é definida pela Equação 9 quando utilizado a normalização por linhas, e é definida pela Equação 10 quando utilizado a normalização por colunas.

$$NormQuadratic(t_j, d_i) = \frac{a_{ij}}{\sqrt{\sum_{k=1}^N (a_{kj})^2}} \quad (9)$$

$$NormQuadratic(t_j, d_i) = \frac{a_{ij}}{\sqrt{\sum_{k=1}^N (a_{ik})^2}} \quad (10)$$

Nas Tabelas 6 e 7 é mostrado o resultado da aplicação da normalização linear por linha e coluna respectivamente, para o conjunto de exemplos com a medida *tf* da Tabela 3. Na qual, o valor máximo atribuído a um atributo passa a ser 1, e os outros valores são proporcionalmente reduzidos com relação ao maior valor desse atributo. Já nas Tabelas 8 e 9 observa-se um exemplo da normalização quadrática por linha e coluna respectivamente, na qual a proporção de redução dos valores é realizada utilizando todos os valores do atributo.

	divert	cas	futebol	amig	jant	famil
Documento 1	0,0	0,0	0,0	0,0	0,0	0,3
Documento 2	0,0	0,0	0,0	0,0	0,2	0,3
Documento 3	0,0	0,0	0,0	0,1	0,5	0,5
Documento 4	0,0	0,0	0,2	0,2	1,0	0,5
Documento 5	0,0	0,3	0,5	0,7	0,3	0,8
Documento 6	0,1	1,0	0,3	0,8	0,7	0,5
Documento 7	1,0	1,0	1,0	1,0	0,5	1,0

Tabela 6: Tabela atributo-valor utilizando medida *tf* e a normalização linear por linha.

	divert	cas	futebol	amig	jant	famil
Documento 1	0,0	0,0	0,0	0,0	0,0	1,0
Documento 2	0,0	0,0	0,0	0,0	1,0	1,0
Documento 3	0,0	0,0	0,0	0,3	1,0	0,7
Documento 4	0,0	0,0	0,2	0,3	1,0	0,3
Documento 5	0,0	0,2	0,5	1,0	0,3	0,5
Documento 6	0,1	0,6	0,3	1,0	0,6	0,3
Documento 7	0,8	0,4	0,7	1,0	0,3	0,4

Tabela 7: Tabela atributo-valor utilizando medida *tf* e a normalização linear por coluna.

	divert	cas	futebol	amig	jant	famil
Documento 1	0,0	0,0	0,0	0,0	0,0	0,2
Documento 2	0,0	0,0	0,0	0,0	0,1	0,2
Documento 3	0,0	0,0	0,0	0,1	0,3	0,3
Documento 4	0,0	0,0	0,1	0,2	0,7	0,3
Documento 5	0,0	0,2	0,4	0,5	0,2	0,5
Documento 6	0,1	0,7	0,3	0,5	0,5	0,3
Documento 7	1,0*	0,7	0,8	0,7	0,3	0,6

Tabela 8: Tabela atributo-valor utilizando medida  $tf$  e a normalização quadrática por linha.

\*O valor real deste atributo é 0,9899, porém arredondando para uma casa decimal temos o valor 1,0. É valido ressaltar que o valor 1,0 só se aplica quando o *token* só aparece em um documento.

	divert	cas	futebol	amig	jant	famil
Documento 1	0,0	0,0	0,0	0,0	0,0	1,0
Documento 2	0,0	0,0	0,0	0,0	0,7	0,7
Documento 3	0,0	0,0	0,0	0,3	0,8	0,5
Documento 4	0,0	0,0	0,1	0,3	0,9	0,3
Documento 5	0,0	0,1	0,4	0,8	0,3	0,4
Documento 6	0,1	0,4	0,2	0,7	0,4	0,2
Documento 7	0,5	0,3	0,4	0,6	0,2	0,3

Tabela 9: Tabela atributo-valor utilizando medida  $tf$  e a normalização quadrática por coluna.

### 3 Remodelagem e Reimplementação do PreText

O PRETEXT (Matsubara et al., 2003) é uma ferramenta computacional que realiza o pré-processamento do texto utilizando a abordagem *bag of words* e implementa todos os métodos citados no Capítulo 2. A ferramenta foi desenvolvida utilizando o paradigma de orientação a objetos, na linguagem de programação **Perl**. Desde sua criação, o PRETEXT vem sendo utilizado em diversos trabalhos envolvendo mineração de textos dentro e fora do ICMC. Após solicitações de diversos usuários, a ferramenta PRETEXT passou por um processo de remodelagem e reimplementação com o objetivo de suprir as necessidades adicionais requeridas pelos usuários, as quais são descritas neste trabalho.

Atualmente, a ferramenta possui as características que estão ilustradas na Figura 2 e explicadas brevemente a seguir.

- O módulo **Start.pl** lê o arquivo de configuração **config.xml** e, com base nos parâmetros especificados neste arquivo, gerencia os demais módulos.
- O módulo **Maid.pm** é responsável pela limpeza do conjunto de documentos iniciais  $\{T_1, T_2, \dots, T_N\}$ , remoção das *stopwords* contidas no arquivo **stoplist.xml** e remoção de símbolos não relevantes a partir do arquivo **simbol.xml**. A geração dos *stems* é

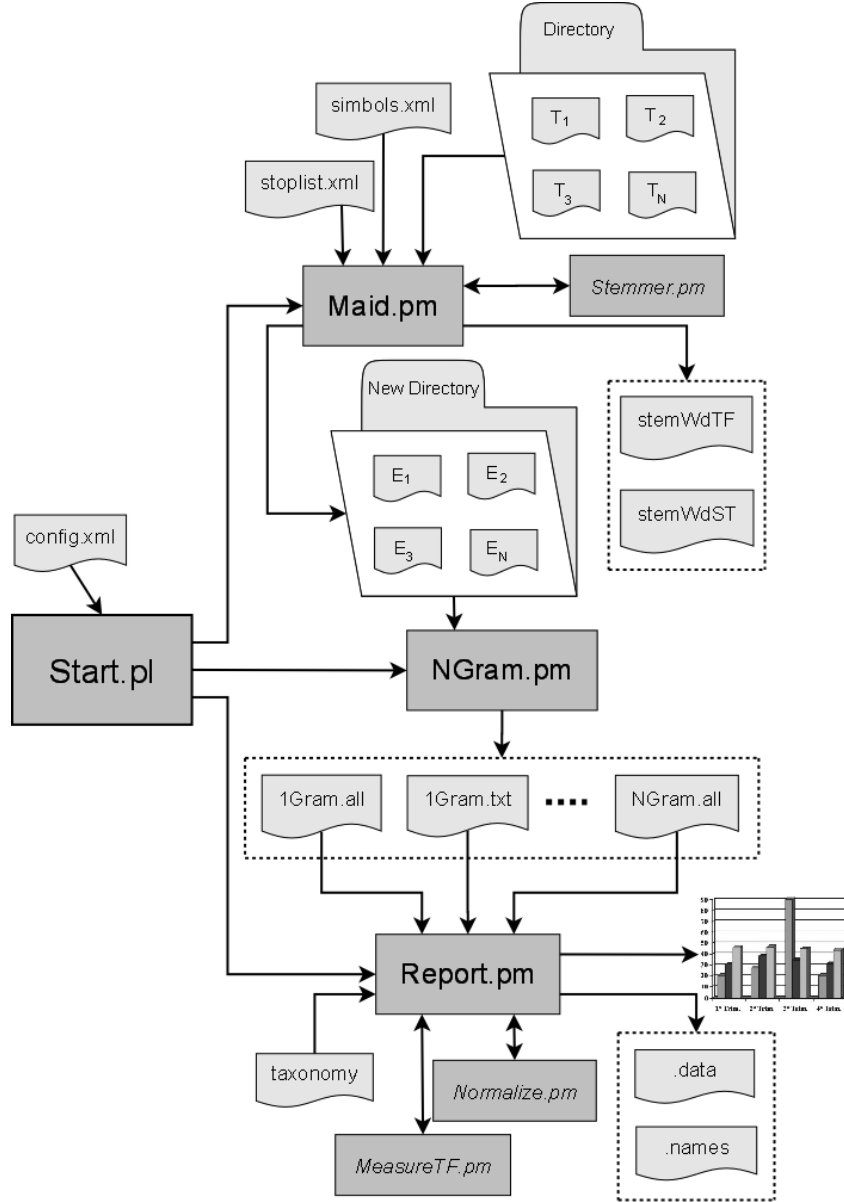


Figura 2: O novo funcionamento do PRETEXT

realizada por uma das classes que herdam a classe abstrata **Stemmer.pm** e contém o algoritmo de *stemming* para a linguagem solicitada. No PRETEXT, estão disponíveis algoritmos de *stemming* para as línguas portuguesa, espanhola e inglesa. Como resultado são gerados os arquivos **stemWdTF.all** (*stems* ordenados por frequência) e **stemWdST.all** (*stems* ordenados por ordem alfabética) que guardam a contagem de *stems* e os *tokens* que foram transformados em cada *stem*, além de um conjunto de arquivos “limpos”  $\{E_1, E_2, \dots, E_N\}$ , que são utilizados como entrada para o módulo **NGram.pm**.

- O módulo **NGram.pm** é responsável pela geração dos *n*-grama e tem como saída os

arquivos **NGram.txt** (contagem de  $n$ -grama em cada documento) e **NGram.all** (contagem de  $n$ -grama total do conjunto de documentos).

- O módulo **Report.pm** recebe como entrada os arquivos **.txt** e **.all**, faz o processamento das taxonomias contidas no arquivo **taxonomy** e calcula as medidas e normalizações solicitadas pelo usuário. O cálculo dessas medidas e normalizações é realizado pelas classes que herdam as abstratas **MeasureTF.pm** e **Normalize.pm**, produzindo como resultado uma tabela atributo-valor no formato do DSX do DISCOVER (Prati, 2003) (arquivos **.data** e **.names**) e alguns gráficos.

Realizando o pré-processamento de textos utilizando a nova versão da ferramenta PRETEXT é possível calcular uma grande quantidade de informação referente aos documentos processados, como os arquivos limpos, e todos arquivos de saída com informações sobre *tokens*, *stems* e  $n$ -grama, assim como a tabela atributo-valor nos formatos requisitados pelo usuário. A Tabela 10 mostra uma idéia geral das principais funcionalidades divididas por cada grande módulo (**Maid.pm**, **NGram.pm** e **Report.pm**).

<b>Maid.pm</b>	<b>NGram.pm</b>	<b>Report.pm</b>
limpeza dos documentos	$n$ -grama para qualquer valor de $n$	gráficos
remoção de tags HTML		cortes por frequência
tratamento de símbolos		cortes por documentos
<i>stoplist</i> XML		taxonomia
<i>stemming</i> para português, inglês e espanhol*		normalizações por linha e coluna: quadrática e linear*
criação dos arquivos “limpos”		medidas: <i>tf-idf</i> , <i>tf</i> , <i>tf-linear</i> e <i>boolean</i> *
		tabela atributo-valor transposta

Tabela 10: Funcionalidades da nova versão do PRETEXT

---

\*Essas funcionalidades podem ser facilmente extendidas.

Segue uma descrição detalhada dos arquivos de entrada e saída da ferramenta, bem como a forma de execução de cada módulo separadamente.

### 3.1 O Arquivo de Configuração ****X***ML*

Uma das principais formas de interagir com a ferramenta PRETEXT é por meio de seu arquivo de configuração **config.xml**. O arquivo no formato ****X***ML* foi escolhido para

que seja possível que outras aplicações possam interagir de maneira simples com o PRETEXT. Esse arquivo constitui de quatro partes principais, como mostrado na Figura 3, denominadas `pretext`, `maid`, `ngram` e `report`, na qual são definidas as configurações gerais da ferramenta, e as configurações dos três módulos principais `Maid.pm`, `NGram.pm`, `Report.pm`, respectivamente.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <pretext lang="pt" dir="textos">
3     <maid>
4     </maid>
5     <ngram>
6         <gram n="1"/>
7     </ngram>
8     <report>
9         <gram n="1"/>
10    </report>
11 </pretext>
```

Figura 3: Exemplo básico do arquivo de configuração `config.xml`

O PRETEXT contém um conjunto de configurações *default*, porém existem algumas opções mandatórias que devem obrigatoriamente serem definidas. Um exemplo de um arquivo com uma configuração básica é mostrado na Figura 3. Todas as opções de configuração são explicadas a seguir utilizando como exemplo o arquivo de configuração descrito na Figura 4.

Opções gerais do PRETEXT – Figura 4, linhas 2 a 6, e 50 são:

- **lang**: especifica a linguagem em que os textos estão escritos. (Linha 3)  
Valores iniciais possíveis: **pt** para português, **sp** para espanhol ou **en** para inglês.  
Valor *default* **pt**.
- **dir**: especifica o diretório que contém a coleção de textos. (Linha 4)  
Opção mandatória.
- **log**: especifica o nome do arquivo com o **log** de execuções. (Linha 5)  
Valor *default* **pretex.log**.
- **silence**: especifica se o PRETEXT irá ou não exibir mensagens na tela durante a execução. (Linha 6)  
Valores possíveis: **on** para modo silencioso, **off** para modo de exibição de mensagens.  
Valor *default* **off**.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <pretext
3   lang="pt"
4   dir="textos"
5   log="pretext.log"
6   silence="off">
7
8   <maid>
9     <number/>
10    <html/>
11    <simbols/>
12    <stoplist dir="stoplist">
13      <stopfile>port.xml</stopfile>
14      <stopfile>ingl.xml</stopfile>
15    </stoplist>
16    <stemming dir="steminfo"/>
17  </maid>
18
19  <ngram dir="ngraminfo">
20    <gram n="1"/>
21    <gram n="4"/>
22    <gram n="9"/>
23  </ngram>
24
25  <report
26    ngramdir="ngraminfo"
27    discover="discover"
28    graphics="graphics"
29    taxonomy="taxonomia.txt"
30    transpose="disabled">
31
32    <gram n="1"
33      max="500"
34      min="10"
35      measure="tf"
36      smooth="disabled"/>
37    <gram n="4"
38      maxfiles="100"
39      minfiles="5"
40      measure="tfidf"
41      normalize="qua"
42      normalizetype="c"/>
43    <gram n="9"
44      std_dev="0.5"
45      measure="tflinear"
46      smooth="enabled"
47      normalize="lin"
48      normalizetype="l"/>
49  </report>
50 </pretext>

```

Figura 4: Exemplo detalhado do arquivo de configuração config.xml

Opções do módulo `Maid.pm` – Figura 4, linhas 8 a 17:

- **number**: habilita a limpeza de números. (Linha 9)
- **html**: habilita a limpeza de tags html. (Linha 10)
- **simbols**: habilita a limpeza de caracteres não alfa-numéricos. (Linha 11)
- **stoplist**: habilita limpeza de *stopwords*. (Linhas 12 a 15)
  - **dir**: especifica o diretório que contém os *stopfiles*, arquivos que contém as *stoplists*. (Linha 12)  
Valor *default* **stoplist**.
  - **stopfile**: especifica nome do *stopfile*. Essa opção pode ser repetida quantas vezes for necessária. (Linhas 13 e 14)  
Opção mandatória caso **stoplist** esteja habilitado.
- **stemming**: habilita geração de *stems* para a língua especificada anteriormente. (Linha 16)
  - **dir**: especifica o diretório no qual serão armazenados os arquivos contendo informações sobre os *stems*. (Linha 16)  
Valor *default* **steminfo**.

Opções do módulo `NGram.pm` – Figura 4, linhas 19 a 23:

- **dir**: especifica o diretório no qual serão armazenados os arquivos contendo as informações sobre os *n*-grama. (Linha 19)  
Valor *default* **ngraminfo**.
- **gram**: habilita um novo *n*-grama. Essa opção pode ser repetida quantas vezes for necessária. (Linhas 20 a 22)  
Opção mandatória caso o módulo `NGram.pm` esteja habilitado.
  - **n**: especifica o valor de *n*. (Linhas 20 a 22)  
Opção mandatória para cada **gram** habilitado.

Opções do módulo `Report.pm` – Figura 4, linhas 25 a 49:

- **ngramdir**: especifica o diretório no qual estão armazenados os arquivos contendo informações sobre os  $n$ -grama. (Linha 26)  
Valor *default* caso o módulo `NGram.pm` esteja habilitado é o mesmo diretório determinado em seus parametros. Caso o módulo não esteja habilitado, será `ngraminfo`.
- **discover**: especifica o diretório no qual serão armazenados os arquivos `.data` e `.names`. (Linha 27)  
Valor *default* `discover`.
- **graphics**: especifica o diretório no qual serão armazenados os arquivos para criação de gráficos. (Linha 28)  
Valor *default* `graphics`.
- **taxonomy**: especifica nome do arquivo de taxonomias. (Linha 29)  
Valor *default* desabilitado.
- **transpose**: habilita a criação da tabela atributo-valor transposta. (Linha 30)  
Valores possíveis: `disabled` e `enabled`.  
Valor *default* `disabled`.
- **gram**: habilita um novo  $n$ -grama. Opção pode ser repetida quantas vezes for necessária. (Linhas 32 a 48)  
Opção mandatória caso o módulo `Report.pm` esteja habilitado.
  - **n**: especifica o valor de  $n$ . (Linhas 32, 37 e 42)  
Opção mandatória para cada **gram** habilitado.
  - **max**: se definido, na tabela atributo-valor são carregados apenas os *tokens* que tiverem frequência absoluta menor ou igual ao valor definido. (Linha 33)  
Valor *default* desabilitado.
  - **min**: se definido, na tabela atributo-valor são carregados apenas os *tokens* que tiverem frequência absoluta maior ou igual ao valor definido. (Linha 34)  
Valor *default* desabilitado.
  - **maxfiles**: se definido, na tabela atributo-valor são carregados apenas os *tokens* que estão contidos em um número de documentos menor ou igual ao valor definido. (Linha 38)  
Valor *default* desabilitado.
  - **minfiles**: se definido, na tabela atributo-valor são carregados apenas os *tokens* que estão contidos em um número de documentos maior ou igual ao valor definido. (Linha 39)  
Valor *default* desabilitado.



- **std\_dev**: se definido, é calculado o *rank* de *stems*. Com o desvio padrão desse *rank* é definido um intervalo  $(\bar{x} - ks; \bar{x} + ks)$ , no qual  $\bar{x}$  é a média do *rank*,  $s$  é o desvio padrão desse *rank* e  $k$  é o valor definido pelo usuário. Os *tokens* que estão fora desse intervalo são descartados. (Linha 44)  
Valor *default* desabilitado.
- **measure**: define a medida que é utilizada para construir a tabela atributo-valor. (Linhas 35, 40 e 45)  
Valores iniciais: **tf**, **boolean**, **tfidf** e **tflinear**.  
Valor *default* **tf**.
- **smooth**: habilita o critério de suavização da medida. (Linhas 36 e 46)  
Valores possíveis: **disabled** e **enabled**.  
Valor *default* **disabled**.
- **normalize**: habilita o método de normalização a ser aplicado aos valores dos atributos da tabela atributo-valor. (Linhas 41 e 47)  
Valores iniciais: **lin** e **qua**  
Valor *default* desabilitado.
- **normalizetype**: caso habilitado **normalize**, define o tipo de normalização por linha ou coluna. (Linhas 42 e 48)  
Valores possíveis: **c** e **l**.  
Valor *default* **c**.

A opção **n** contida nas configurações do **NGram.pm** e do **Report.pm** se refere ao valor  $n = 1, 2, 3, \dots$  do  $n$ -grama e podem assumir a priori qualquer valor inteiro positivo. Com esse recurso, pode-se gerar  $n$ -grama de ordem alta, como 10 ou 15, que representam frases que se repetem em alguns textos (por exemplo, textos técnicos) e podem até representar um atributo significativo para a aplicação dos algoritmos de aprendizado. Cada  $n$ -grama é processado individualmente, com suas medidas, cortes e normalizações individuais especificadas pelo usuário, e após o processo, unidas na tabela atributo-valor final.

As opções **max**, **min**, **maxfiles**, **minfiles** e **std\_dev** contida nas configurações do **Report.pm** se referem à opções de redução de dimensionalidade por cortes de *tokens*. As opções **max** e **min** definem manualmente os pontos superior e inferior dos cortes de Luhn. Todos os *tokens* cujas frequências absolutas se encontram fora do intervalo definido pelo usuário, levando em conta toda a coleção de documentos, serão ignorados no processo de criação da tabela atributo-valor. Já as opções **maxfiles** e **minfiles** consideram a frequência de aparição dos *tokens* nos documentos do conjunto de documentos, independente do número de vezes que cada *token* aparece dentro dos documentos. Dessa maneira *tokens*

que estão fora do intervalo manualmente definido serão também ignorados na criação da tabela atributo-valor.

A opção `std_dev`, por sua vez, faz cortes nos *tokens* de acordo com o desvio padrão sobre o *rank* dos *tokens*. Esse *rank* é construído de maneira que *tokens* com a mesma frequência pertencem ao mesmo *ranking*. O importante deste *rank* é o número de colocações diferentes. Os *rankings* são definidas pela frequência absoluta do *token*, ou seja, os *tokens* com maior frequência absoluta receberão os primeiras *rankings*, e os com menor receberão os últimos *rankings*. Na Tabela 11 (a) é exemplificada as frequências dos *tokens* de um conjunto de documentos. A partir dessas frequências, na Tabela 11 (b) é mostrado os *ranks* gerados.

<i>stem</i>	freq
amig	5
trabalh	5
divert	5
cas	7
futebol	12
amig	13
jant	13
famil	17

(a)

<i>rank</i>	freq
1	17
2	13
3	12
4	7
5	5

(b)

Tabela 11: A frequência dos *tokens* (a) gera o *rank* de *tokens* (b)

A partir dos *ranks* exemplificados na Figura 11 (b), é calculado o desvio padrão e a média dos valores, obtendo-se 1.58 e 2.5, respectivamente<sup>11</sup>. Supondo que seja definido no arquivo de configuração a opção `std_dev="0.8"`, o intervalo do *rank* é:

$$[2.5 - 1.58 \times 0.8, 2.5 + 1.58 \times 0.8] = [1.73, 4.26]$$

Por somente existir *ranks* inteiros, o intervalo é arredondado para  $[2, 4]$  e convertido novamente para a frequência do *token* para que possa ser realizado o corte por frequência. O intervalo final seria  $[7, 13]$ , ou seja, *tokens* com frequência menor que 7 e maior que 13 são ignorados.

## 3.2 O módulo Maid.pm

Esse módulo é responsável pela limpeza dos documentos iniciais, remoção das *stopwords*, remoção de símbolos não relevantes e geração dos *stems* com o algoritmo de *stemming* da linguagem solicitada. Como resultado são gerados os arquivos **stemWdTF.all** (*stems*

<sup>11</sup>Por convenção no PRETEXT, sempre é utilizado o ponto para separar as casas decimais.

ordenados por frequência) e **stemWdST.all** (*stems* ordenados por ordem alfabética) que guardam a contagem de *stems* e os *tokens* que foram transformados em cada *stem*, e um conjunto de arquivos “limpos”. Para sua execução, são necessários alguns arquivos que serão explicados a seguir.

### 3.2.1 Arquivos de Entrada

Os arquivos de entrada do módulo **Maid.pm** consistem de:

- arquivo de configuração **config.xml**: o módulo **Maid.pm** consulta as configurações gerais do PRETEXT, e as informações específicas do seu módulo.
- arquivo de símbolos **symbols.xml**: o PRETEXT possui a facilidade de permitir ao usuário a escolha de símbolos que não acrescentam informação relevante ao conjunto de textos, por meio do arquivo **symbols.xml**. Nesse arquivo, o usuário irá definir quais símbolos não alfa-numéricos serão eliminados dos textos, e quais dentre eles representam divisores de frases, para que não sejam gerados *n*-grama com *tokens* de frases diferentes.

A forma de construção do **symbols.xml** é apresentada a seguir, utilizando o exemplo na Figura 5.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <symbols clearall="false">
3   <symbol break="true">.</symbol>
4   <symbol break="true">,</symbol>
5   <symbol break="true">!</symbol>
6   <symbol break="true">?</symbol>
7   <symbol break="false"><</symbol>
8   <symbol break="false">$</symbol>
9   <symbol break="false"><=</symbol>
10  <symbol break="false">~</symbol>
11  <symbol>@</symbol>
12  <symbol>#</symbol>
13 </symbols>
```

Figura 5: Exemplo do arquivo de símbolos **symbols.xml**

- **clearall**: caso não seja necessária a escolha de símbolos específicos, o PRETEXT tem a opção de limpar todos símbolos não alfa-numéricos do conjunto de documentos. Para isso, são utilizados símbolos *default* para serem considerados como divisores de frases. Caso o usuário deseje especificar os símbolos, não deve habilitar essa opção. (Linha 2)

Valores possíveis: **true** e **false**.

Valor *default* **true**.

- **symbol**: especifica um novo símbolo a ser removido. Opção pode ser repetida quantas vezes for necessária. (Linhas 3 a 12)

- \* **break**: habilite essa opção caso o símbolo a ser removido seja um divisor de frases. (Linhas 3 a 10)

Valores possíveis: **true** e **false**.

Valor *default* **false**.

Devido as regras de construção de ***XML***, alguns símbolos não podem ser escritos por extenso. Tais símbolos necessitam de uma codificação especial para que possam ser removidos do conjunto de textos. Essas codificações são mostradas na Tabela 12.

Codificação	Símbolo
&amp;	&
&gt;	>
&lt;	<

Tabela 12: Codificação ***XML*** de alguns símbolos.

- arquivos de *stoplists*: para definir as *stopwords* a serem ignoradas no processo de limpeza do texto, deve-se utilizar também arquivos ***XML*** contendo todas as palavras e variações de palavras que serão ignoradas. O arquivo deve ser construído como mostra o exemplo na Figura 6.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <stopfile>
3   <stopword>abaixo</stopword>
4   <stopword>acaso</stopword>
5   <stopword>acerca</stopword>
6   <stopword>acima</stopword>
7   <stopword>ademaís</stopword>
8 </stopfile>
```

Figura 6: Exemplo de um *stopfile*

A opção **stopword** pode ser repetida quantas vezes for necessária para que sejam inseridos tantas *stopwords* quanto o usuário desejar. Não é necessário que todas as *stopwords* estejam em um único arquivo *stopfile*. O PRETEXT pode carregar vários *stopfiles*, porém, para facilitar o entendimento de cada *stopfile*, é aconselhável que sejam agrupados *stopwords* por temas de interesse: por exemplo *stopfile* geral em português; *stopfile* de esportes; *stopfile* geral em inglês.

- conjunto de documentos: os documentos texto a serem processados podem estar organizados de duas maneiras, uma para documentos rotulados, e outra para documentos sem rótulo, como explicado a seguir.

1. Documentos não rotulados: cria-se um diretório contendo todos os documentos sem divisão de subdiretórios.
2. Documentos rotulados: cria-se um diretório contendo subdiretórios, nos quais cada um deles contém os documentos de uma mesma classe.

Para exemplificar, considere os arquivos `basquete.txt`, `futebol.txt`, `eleições.txt` e `ministério.txt`, nos quais os dois primeiros são da classe `esporte` e os dois últimos da classe `política`. Para que o PRETEXT gere automaticamente as classes dos documentos, deve ser construído um diretório como ilustrado na Figura 7.

```

diretorio\
    esporte\
        basquete.txt
        futebol.txt
    política\
        eleições.txt
        ministério.txt

```

Figura 7: Exemplo da organização do diretório de documentos rotulados.

### 3.2.2 Arquivos de Saída

Como saída do módulo `Maid.pm` são gerados arquivos, alguns dos quais farão parte dos arquivos de entrada do próximo módulo `NGram.pm`, e outros que fornecem diversas informações para o usuário. Estes arquivos são explicados a seguir.

- conjunto de documentos “limpos”: para oferecer maior liberdade ao usuário, o módulo `Maid.pm` tem como saída o mesmo número  $N$  de arquivos que recebeu como entrada. Porém, esses arquivos já estão com seus *tokens* reduzidos pelas diversas funções de redução de atributos selecionadas pelo usuário e executadas pelo PRETEXT. Esses arquivos contém os dados de entrada para o módulo `NGram.pm`.
- arquivos `stemWdTF.all` e `stemWdST.all`: esses arquivos contém a contagem de *stems* e os *tokens* que foram transformados em cada *stem*. No arquivo `stemWdTF.all` eles estão ordenados por frequência, e no arquivo `stemWdST.all` estão ordenados alfabeticamente. Na Figura 8 é ilustrado o arquivo `stemWdTF.all`, no qual os

*tokens* politico, politicamente, politicos, politicas e politica foram transformados no *stem* polit. Os valores que seguem os *stem* representam a frequência com que eles aparecem na coleção de documentos e a quantidade de documentos que contêm esse *stem*. Por exemplo, a linha 1 da Figura 8 mostra que o *stem* polit aparece 15 vezes em 5 dos 5 documentos dessa coleção de documentos. De modo similar, os valores que seguem um *token* representam a frequência deste *token* na coleção de documentos. Por exemplo, na linha 6, o *token* politica aparece 4 vezes nessa coleção de documentos.

```

1 polit : 14(5/5)
2     politico:2
3     politicamente:2
4     politicos:3
5     politicas:3
6     politica:4
7 proced : 9(3/5)
8     proceder:1
9     procederemos:2
10    procedimentos:2
11    procedimento:4
12 regul : 13(4/5)
13    reguladores:1
14    regulador:1
15    regulada:1
16    regulamentos:2
17    regulam:2
18    regulamento:2
19    regular:4
20 tranquil : 6(2/5)
21    tranquila:1
22    tranquilidade:2
23    tranquilo:3
24 valoriz : 9(1/5)
25    valorizou:1
26    valorizam:1
27    valoriza:1
28    valorizada:1
29    valorizar:1
30    valorizadas:1
31    valorizados:1
32    valorizado:2

```

Figura 8: Exemplo do arquivo **stemWdST.all**

### 3.2.3 Execução do Módulo Maid.pm

Uma vez que todos as opções estão configuradas de forma correta, e todos arquivos de entrada que serão utilizados foram indicados, o modulo Maid.pm pode ser executado,

ativando o PRETEXT por meio do comando `perl Start.pl`.

Na Figura 9 é mostrado um exemplo da execução do módulo `Maid.pm`. Nesse exemplo, as linhas 7 a 10 mostram as configurações gerais do PRETEXT, e as linhas 16 a 25 as configurações do módulo `Maid.pm`, ambas definidas no arquivo `config.xml`. Pode-se notar que todas as opções de limpeza e redução de atributos foram habilitadas para essa execução. Nas linhas 27 a 29 é informado que foram carregados 1010 *stopwords* que estão em 2 *stopfiles*.

```
1 #-----#
2 #           PreText           #
3 #                             #
4 #   Implemented by LABIC      #
5 #-----#
6
7 #===== PARAMETERS =====#
8 language      = pt
9 directory     = exemplo
10 log file      = pretext.log
11
12 #-----#
13 #           Maid.pm           #
14 #-----#
15
16 #===== PARAMETERS =====#
17 html clear    = ENABLED
18 number clear  = ENABLED
19 simbol clear  = ENABLED
20 stoplist      = ENABLED
21 > directory   = stoplist
22 > stopfile    = stoplist/port.xml
23 > stopfile    = stoplist/ingl.xml
24 stemming     = ENABLED
25 > directory   = steminfo
26
27     ### STOP LIST ###
28     Total StopWords 1010
29     Total StopFiles 2
30
31 Maid          :.....:.....:.....:.....: OK
32
33 #-----#
34 Total Time: 0
35 #-----#
```

Figura 9: Exemplo de execução do módulo `Maid.pm`.

A linha 31 mostra o progresso de execução do módulo. Nessa linha o usuário pode acompanhar o andamento da limpeza de seus arquivos. A velocidade de execução deste módulo depende da quantidade e tamanho dos documentos no conjunto de documentos, bem como da quantidade de opções de limpeza habilitadas.

### 3.2.4 Advertências

O PRETEXT pode não funcionar corretamente caso os documentos no conjunto de documentos estejam codificado no formato UTF-8. É aconselhável a utilização de somente arquivos em formato ISO 8859-1.

## 3.3 O módulo NGram.pm

Esse módulo é responsável pela criação dos  $n$ -grama. O usuário tem completa liberdade para determinar o valor  $n$  do  $n$ -grama. Pode-se gerar  $n$ -grama usuais, como o um-grama, bi-grama e tri-grama, mas também é possível a criação de  $n$ -grama de ordem maior, como dito anteriormente. Para cada grama definido no arquivo de configuração são gerados dois arquivos: `nGram.all` e `nGram.txt`. O arquivo com extensão `.all` contém todos os *tokens* da coleção e suas respectivas frequências, enquanto que o arquivo com extensão `.txt` contém a frequência dos *tokens* para cada documento da coleção.

### 3.3.1 Arquivos de Entrada

Para a execução do módulo NGram.pm somente é necessário o conjunto de documentos “limpos” gerado pelo módulo Maid.pm, e o arquivo de configuração `config.xml` para execução deste módulo. O módulo consulta no arquivo de configuração as informações gerais do PRETEXT e as informações específicas para este módulo.

### 3.3.2 Arquivos de Saída

A saída deste módulo é utilizada como entrada do módulo Report.pm. Esses arquivos de saída são explicados a seguir.

- `nGram.all`: este arquivo contém todos os *token* para o  $n$ -grama definido. Ele contém também a frequência absoluta de cada *token*, e o número de vezes que aparece em cada documento da coleção. Na Figura 10 é ilustrado o arquivo `1Gram.all` gerado pelo módulo NGram.pm contendo informações sobre o um-grama. Na linha 3 pode ser observado que o *token* `proced` aparece 9 vezes em 3 dos 5 documentos da coleção.
- `nGram.txt`: este arquivo contém os *tokens* separados por documento, e a frequência com que esses *tokens* aparecem em cada documento. Na Figura 11 é ilustrado o



```
1 6:(2/5):tranquil
2 9:(1/5):valoriz
3 9:(3/5):proced
4 13:(4/5):regul
5 14:(5/5):polit
```

Figura 10: Exemplo do arquivo 1Gram.all.

arquivo 1Gram.txt que contém informações sobre o um-grama. Na linha 2 pode ser observado que o *token* `polit` aparece 6 vezes no documento `exemplo1.txt`.

```
1 exemplo_Maid/exemplo1.txt
2     polit:6
3 exemplo_Maid/exemplo2.txt
4     polit:1
5     regul:3
6 exemplo_Maid/exemplo3.txt
7     polit:1
8     proced:1
9     regul:1
10 exemplo_Maid/exemplo4.txt
11     polit:1
12     proced:1
13     regul:1
14     tranquil:1
15 exemplo_Maid/exemplo5.txt
16     polit:5
17     proced:7
18     regul:8
19     tranquil:5
20     valoriz:9
```

Figura 11: Exemplo do arquivo 1Gram.txt.

### 3.3.3 Execução do Módulo NGram.pm

O módulo `NGram.pm` deve ser executado da mesma maneira que o módulo `Maid.pm`, digitando `perl Start.pl` após verificar que todas as configurações e arquivos necessários estão disponíveis.

Na Figura 12 é ilustrado um exemplo da execução do módulo `NGram.pm` para a geração de um-grama, bi-grama, tri-grama e nove-grama. Nesse exemplo, as linhas 7 a 10 mostram as configurações gerais do PRETEXT, assim como no módulo anterior, e as linhas 16 a 21 as configurações do módulo `NGram.pm`.

As linhas 23 a 26 mostram o progresso de criação de cada  $n$ -grama, por serem processados individualmente. Deve ser observado que a execução deste módulo requer uma grande

```

1 #-----#
2 #           PreText          #
3 #                           #
4 #   Implemented by LABIC     #
5 #-----#

7 #===== PARAMETERS =====#
8 language      = pt
9 directory     = exemplo
10 log file      = pretext.log

12 #-----#
13 #           NGram.pm         #
14 #-----#

16 #===== PARAMETERS =====#
17 directory     = ngraminfo
18 1-Gram        = ENABLED
19 2-Gram        = ENABLED
20 3-Gram        = ENABLED
21 9-Gram        = ENABLED

23 Criando 1Gram :.....: OK
24 Criando 2Gram :.....: OK
25 Criando 3Gram :.....: OK
26 Criando 9Gram :.....: OK

28 #-----#
29 Total Time: 0
30 #-----#

```

Figura 12: Exemplo de execução do módulo `NGram.pm`.

quantidade de memória RAM.

### 3.3.4 Advertências

É recomendado que antes da execução desse módulo seja executado o módulo `Maid.pm` para que seja criado o diretório de documento “limpos” onde os documentos já se encontram no formato ideal para a utilização do módulo `NGram.pm`. Caso o usuário não queira uma limpeza de dados muito apurada, o módulo `Maid.pm` pode ser executado somente com a limpeza básica, sem nenhuma opção extra de limpeza. Deve ser observado que a não execução do módulo `Maid.pm` pode ocasionar em um processo de geração de  $n$ -grama falho.

## 3.4 O módulo `Report.pm`

Esse é o módulo do PRETEXT responsável principalmente pela criação da tabela atributo-valor no formato do DISCOVER, na qual cada documento é um exemplo e os *tokens* são os atributos, a partir dos arquivos `nGram.all` e `nGram.txt`, e das configurações especificadas pelo usuário. O valor que cada *token* recebe é calculado utilizando algumas

das medidas implementadas no PRETEXT escolhida pelo usuário. O módulo `Report.pm` realiza também os cortes por frequência, se requeridos pelo usuário.

### 3.4.1 Arquivos de Entrada

Como entrada de dados para esse módulo temos:

- arquivo de configuração `config.xml`: o módulo `Report.pm` também consulta as informações gerais da ferramenta e suas informações específicas.
- arquivos de  $n$ -grama `.all` e `.txt`: estes arquivos gerados pelo módulo `NGram.pm` são essenciais para a execução do módulo `Report.pm`.
- arquivo de taxonomias: caso esteja habilitado a opção de utilização de taxonomia, um arquivo de taxonomia deverá ser criado pelo usuário, para o PRETEXT realizar indução construtiva. A indução construtiva é realizada por meio de criação de novos atributos a partir da junção de um ou mais *tokens* já existentes, representando a generalização de um atributo.

Considere por exemplo que os cinco *tokens* citados anteriormente `tranquil`, `valoriz`, `proced`, `regul` e `polit` são organizados da seguinte maneira, `regul` e `polit` são agrupados no *token* `governo` e os outros *tokens* `tranquil`, `valoriz` e `proced` são agrupados em um *token* chamado `outros`. Dessa maneira, o arquivo de taxonomia deve conter a informação ilustrada na Figura 13.

```
governo: regul, polit
outros: tranquil, valoriz, proced
```

Figura 13: Exemplo de um arquivo de taxonomia.

Assim, a tabela atributo-valor original com cinco *tokens* é transformada em uma tabela com apenas dois atributos, na qual o valor dos atributos representa a soma das frequências dos *tokens* que a geraram, como é mostrado na Tabela 13.

documento	polit	regul	proced	tranquil	valoriz	documento	GOVERNO	OUTROS
exemplo1.txt	6	0	0	0	0	exemplo1.txt	6	0
exemplo2.txt	1	3	0	0	0	exemplo2.txt	4	0
exemplo3.txt	1	1	1	0	0	exemplo3.txt	2	1
exemplo4.txt	1	1	1	1	0	exemplo4.txt	2	2
exemplo5.txt	5	8	7	5	9	exemplo5.txt	13	21

(a)

(b)

Tabela 13: Transformação da tabela original para a tabela com taxonomias.

O arquivo de taxonomias deve ser construindo pelo usuário, ou um especialista do domínio, a partir de uma análise dos arquivos de *stem* e *n*-grama. Para diferenciar *tokens* gerados automaticamente pela ferramenta e aqueles que foram construídos por meio do arquivo de taxonomias, no arquivo `.names` os atributos gerados a partir das taxonomias estão representados com letras maiúsculas.

### 3.4.2 Arquivos de Saída

Os arquivos de saída do módulo `Report.pm` consistem de dois arquivos que representam a tabela atributo-valor no formato do DISCOVER, e alguns arquivos para a construção de gráficos. Esses arquivos são descritos a seguir.

- arquivo `discover.names`: este arquivo contém a declaração de todos os atributos da tabela atributo-valor gerados pelo PRETEXT, como é ilustrada na Figura 14 (a).

```
filename:string:ignore.
"polit":integer.
"regul":integer.
"proced":integer.
"tranquil":integer.
"valoriz":integer.
```

(a)

```
"exemplo_Maid/exemplo1.txt",6,0,0,0,0
"exemplo_Maid/exemplo2.txt",1,3,0,0,0
"exemplo_Maid/exemplo3.txt",1,1,1,0,0
"exemplo_Maid/exemplo4.txt",1,1,1,1,0
"exemplo_Maid/exemplo5.txt",5,8,7,5,9
```

(b)

Figura 14: Exemplo dos arquivos `.names` (a) e `.data` (b).

- arquivo `discover.data`: neste arquivo estão os valores dos atributos para todos os documentos da coleção, e representa a tabela atributo-valor como mostrado na Figura 14 (b).

A primeira linha do arquivo `discover.names` corresponde ao atributo cujo o valor está representado na primeira coluna do arquivo `discover.data`, a segunda linha corresponde ao segundo atributo, e assim por diante. O primeiro atributo de todas as tabelas geradas pelo PRETEXT é sempre o `filename` que representa o documento que originou esses valores para seus atributos. Geralmente, este atributo não é utilizado no aprendizado de máquina, portanto é ignorado.

- arquivos para a geração de gráficos: o módulo `Report.pm` gera dois arquivos de dados referentes ao *rank* dos *tokens* para cada *n*-grama solicitado pelo usuário. Junto com esses arquivos, são gerados também *scripts* para **GnuPlot**<sup>TM</sup> (Crawford, 1998) os quais são utilizados para a geração de gráficos no formato L<sup>A</sup>T<sub>E</sub>X (Lamport,

1986; Kopka & Daly, 2004). Esses *scripts* podem ser alterados para a geração de gráficos em outros formatos alterando somente a primeira linha do arquivo.

As Figuras 15 e 16 são exemplos de gráficos gerados pelo `Report.pm` a partir dos dados processados no um-grama. Na primeira figura é mostrada o *rank* dos *tokens*, começando do mais freqüente ao menos freqüente sem, levar em conta *tokens* com a mesma freqüência. Na segunda figura já é levado em conta quantos *tokens* diferentes possuem determinada freqüência. Esses gráficos podem ser muito úteis para a determinação dos pontos de corte por freqüência.

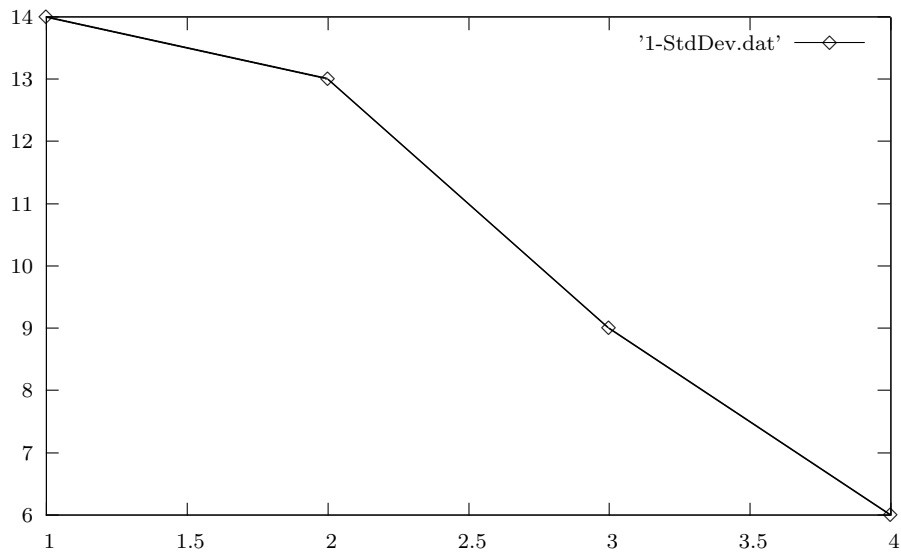


Figura 15: Exemplo do gráfico 1-StdDev.

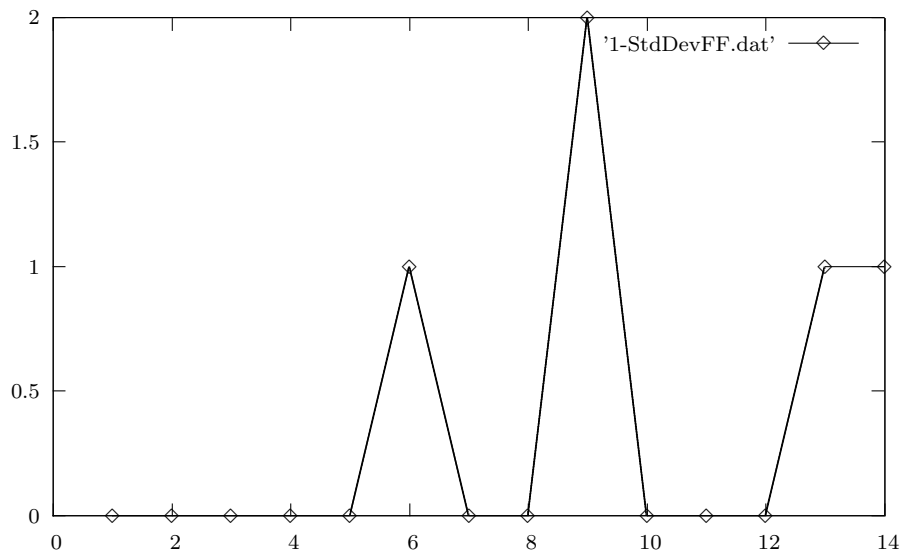


Figura 16: Exemplo do gráfico 1-StdDevFF.

### 3.4.3 Execução do Módulo Report.pm

O modulo Report.pm também é executado da mesma maneira que os outros módulos principais, digitando perl Start.pl e com todas as configurações e arquivos de entrada presentes.

```
1 #-----#
2 #           PreText          #
3 #                           #
4 #   Implemented by LABIC     #
5 #-----#
6
7 #===== PARAMETERS =====#
8 language      = pt
9 directory     = exemplo
10 log file      = pretext.log
11
12 #-----#
13 #           Report.pm        #
14 #-----#
15
16 #===== PARAMETERS =====#
17 NGram Dir     = ngraminfo
18 Discover Dir   = discover
19 Graphics Dir   = graphics
20 1-Gram        = ENABLED
21 > Measure     = tf
22
23     ### Taxonomy was NOT Loaded ###
24
25 #----- 1-Gram -----#
26 1Gram.all      :.....:.....:.....:.....: OK
27 Number of 1-gram loaded from ngraminfo/1Gram.all      5
28
29 1Gram.txt      :.....:.....:.....:.....: OK
30 graphics/1-StdDev.dat created!
31 graphics/1-StdDevFF.dat created!
32 graphics/1-GnuPlot.script created!
33
34 Loading Term Frequency
35 Writing Measure :.....:.....:.....:.....: OK
36
37 ===== Summary =====
38 N-Gram       : 1
39 Total Stems  : 5
40 Total Texts  : 5
41 -----
42
43 Matrix Density                                150.00
44
45 #--- Discovery Table ---#
46 Discover Data :.....:.....:.....:.....: OK
47 Number of Texts                                5
48 Discover Names :.....:.....:.....:.....: OK
49 Number of Stems                                5
50
51 #-----#
52 Total Time: 0
53 #-----#
```

Figura 17: Exemplo de execução do módulo Report.pm.

Na Figura 17 é mostra um exemplo de execução do módulo Report.pm configurado so-

mente para um-grama. Pode ser observado que, como na execução dos outros módulos, as linhas 7 a 10 mostram as configurações gerais do PRETEXT, e as linhas 16 a 21 as configurações do módulo NGram.pm. Na linha 23 tem-se a informação de que não foi utilizado arquivo de taxonomias para essa execução.

A linha 26 mostra o progresso da leitura do arquivo `.all` e, logo após, na linha 27, a informação de que foram carregados na memória 5 *tokens*, e na linha 29 está o progresso da leitura do arquivo `.txt`. As linhas 30 a 32 indicam a criação dos arquivos para geração de gráficos, e logo após, nas linhas 34 e 35, é calculada a medida solicitada pelo usuário.

Nas linhas 35 a 41 tem-se um breve sumário, contendo informações sobre os dados que serão gravados na tabela atributo-valor. A linha 43 contém a informação da densidade da matriz gerada e, por fim, as linhas 45 a 49 está a escrita dos arquivos `.data` e `.names`.

#### 3.4.4 Advertências

A utilização de conjuntos de documentos muito volumosos pode comprometer a execução do módulo Report.pm. Contudo, se forem feitos bons cortes de frequência de *tokens* nesses arquivos, a velocidade de execução será muito superior. Portanto, é recomendado sempre utilizar cortes quando o volume de dados a ser processado é demasiadamente grande.

### 3.5 *Script* Auxiliar de Configuração

Para facilitar a criação do arquivo de configuração `config.xml`, existe no PRETEXT um *script* auxiliar somente para a criação do arquivo de configuração chamado `CreateConfig.pl` que pode ser executado digitando `perl CreateConfig.pl` e este *script* irá te auxiliar na criação do arquivo de configuração.

### 3.6 Instalação

#### 3.6.1 Windows

Para instalar o PRETEXT no Windows, tenha o ActivePerl instalado, e descompacte o PRETEXT em alguma pasta de sua preferência para utilizá-lo.

### 3.6.2 Linux

Para instalar o PRETEXT no Linux, tenha o perl instalado, e é necessário, também, ter os pacotes `build-essential` e `libc6` instalados no seu sistema. Para ter certeza que estes pacotes estão presentes digite:

```
1 sudo apt-get install build-essential libc6
```

Logo após, descompacte o arquivo `IO-Dirent-0.02.tar.gz` e instale com os comandos:

```
1 perl Makefile.PL
2 sudo make
3 sudo make install
```

E faça o mesmo com o arquivo `XML-Parser-2.34.tar.gz`. Descompacte-o e execute os comandos:

```
1 perl Makefile.PL
2 sudo make
3 sudo make install
```

Após realizado esse procedimento, apenas descompacte o PRETEXT em um diretório de sua preferência e você pode utiliza-lo de dentro deste diretório.

## 4 Arquitetura de Classes

O PRETEXT é uma ferramenta computacional que utiliza o paradigma de programação orientada a objetos, escrita na linguagem de programação **Perl**. Todas classes do PRETEXT serão explicadas e ilustradas brevemente nesta seção.

### 4.1 Start.pl

A implementação atual do PRETEXT consiste de um módulo único que gerencia os outros módulos. O `Start.pl` é responsável pela chamada do módulo de configuração



`ReadConfig.pm` que acessa o arquivo de configuração `config.xml` e, a partir dessas configurações, seleciona quais outros módulos serão utilizados no pré-processamento especificado pelo usuário. Este módulo e os módulos que relacionam diretamente com ele são ilustrados na Figura 18.

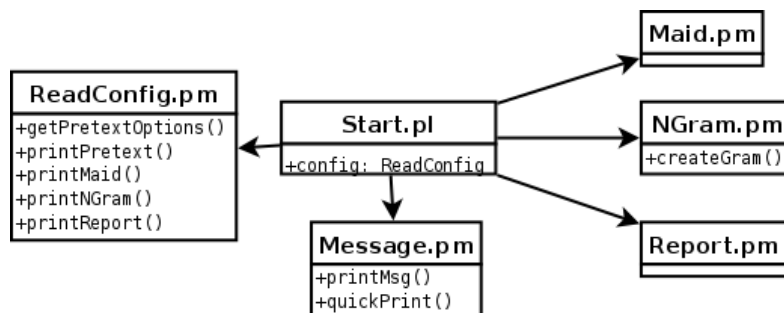


Figura 18: Diagrama de classes do módulo `Start.pl`

**ReadConfig.pm** Classe que realiza a leitura do arquivo `config.xml`. Cada configuração específica pode ser acessada a partir de um método *get*. Este módulo também imprime na tela as configurações que estão sendo utilizadas em cada módulo. Esta classe é acessada pelo módulo `Start.pl` para a leitura e impressão das configurações, o objeto criado, é então passado para os outros módulos que acessam seus métodos *get*.

**Message.pm** Todas as impressões na tela durante a execução do PRETEXT são gerenciadas por esta classe. Nela também é realizado o *log* da ferramenta. Esta classe é utilizada por praticamente todas as outras classes do PRETEXT.

## 4.2 Maid.pm

Este módulo é chamado pelo módulo `Start.pl` para realizar a limpeza dos arquivos. Este módulo e aquelas que se relacionam com ele estão ilustrados na Figura 19.

As limpezas realizadas são a limpeza básica `basicClear()` que todos transforma os caracteres em letras minúsculas, retira acentos e cedilhas, elimina quebras de linha, elimina alguns símbolos que são necessários para que o PRETEXT execute suas funções sem interferência, tais como `:`, `|`, `_` e `-`, e caso esteja habilitada a opção correspondente, elimina os números.

A limpeza de símbolos `clearSymbols()` verifica o arquivo `symbols.xml` para tratar os símbolos ou excluindo-os ou transformando em divisores de frases. Caso esteja habilitada

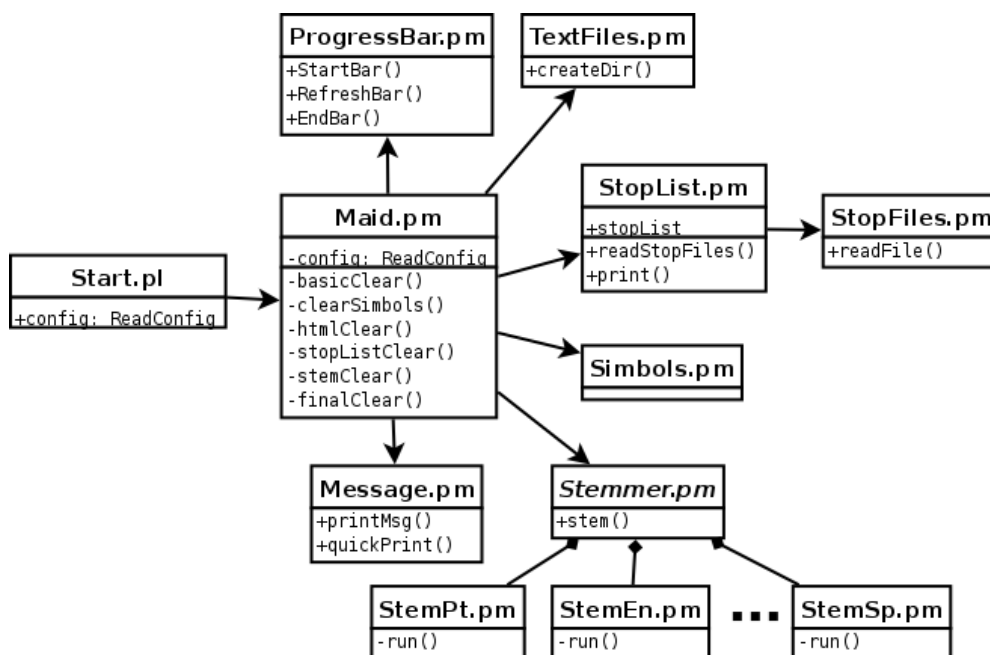


Figura 19: Diagrama de classes do módulo Maid.pm

a opção `clearall`, o PRETEXT elimina todos os símbolos com exceção de 21 deles que se transformam em divisores de frases. Estes 21 símbolos são mostrados na Figura 20.

. , ; : " ' ‘ ! ? / \ | % + { } [ ] ( ) \*

Figura 20: Símbolos transformados em divisores de frase por *default*.

Limpeza de tags HTML `htmlClear()` retira as tags HTML para que somente o texto puro seja utilizado para a geração da tabela atributo-valor. Algumas entidades HTML são transformadas em seus correspondentes em ASCII. Por exemplo `&acute` que representa o caractere á é transformado no caractere a sem o acento, pois o PRETEXT trabalha sem distinção de acentuação. Todas as transformações realizadas pelo PRETEXT são descritas na Tabela 14.

A limpeza final `finalClear()` elimina espaços e divisores de frases múltiplos. A limpeza de *stopwords* `stopListClear()` e o algoritmo de *stemming* `stemClear()` utilizam de outras classes especializadas que são explicadas a seguir.

**ProgressBar.pm** Esta classe gera e gerencia a barra de progresso. É acessada sempre que alguma ação deve ser realizada inúmeras vezes, para que o usuário seja informado sobre o andamento do processo.

Codificação	Símbolo
&aacute; &agrave; &acirc; &atilde; &auml;	a
&ccedil;	ç
&eacute; &egrave; &ecirc; &euml;	e
&iacute; &igrave; &icirc; &iuml;	i
&oacute; &ograve; &ocirc; &otilde; &ouml;	o
&uacute; &ugrave; &ucirc; &uuml;	u

Tabela 14: Transformações de entidades HTML realizadas pelo PRETEXT.

**TextFiles.pm** Esta classe faz a leitura de quantos arquivos existem dentro do diretório de trabalho, e armazena os nomes e caminhos de cada arquivo para que possam ser acessados para serem pré-processados. Esta classe também cria uma cópia dos diretórios e subdiretórios para o armazenamento dos arquivos “limpos”.

**StopList.pm** Esta classe é responsável pelo gerenciamento da *stoplist*. Como mencionado, uma *stoplist* pode conter um ou mais *stopfiles*. Todas as *stopwords* são carregadas e armazenadas por esta classe para que o módulo **Maid.pm** possa ignorar tais palavras.

**StopFiles.pm** Cada *stopfile* é lido separadamente por esta classe e o resultado é enviado para a classe **StopList.pm**.

**Simbols.pm** Classe responsável pela leitura do arquivo `simbols.xml`. Os símbolos armazenados por esta classe são utilizados pela classe **Maid.pm** para o tratamento de símbolos por meio de métodos *get*.

**Stemmer.pm** Utilizando as vantagens provenientes do paradigma de orientação a objeto, foi criado uma classe abstrata **Stemmer.pm** que serve como base para os algoritmos de *stemming* para diversas línguas. A partir dessa classe, podem ser criadas outras classes que deverão herdar e implementar as funções específicas de geração de *stem* da classe abstrata **Stemmer.pm**. Dessa maneira, qualquer usuário com conhecimento da linguagem de programação **Perl** pode implementar um novo algoritmo de *stemming* para outra língua e especificar no arquivo de configuração a língua que deseja utilizar para o *stemming*. Essa facilidade permite que a ferramenta seja facilmente ampliada futuramente, sem que seja necessário alterar nenhum dos módulos já existentes. Os módulos de *stemming* devem ter o nome **StemXx.pm** onde **Xx** é a identificação da língua para qual o algoritmo de *stemming* gera seus *stems*.

**StemPt.pm, StemEn.pm, StemSp.pm** Já estão implementados a partir da classe abstrata **Stemmer.pm** os módulos **StemPt.pm**, **StemEn.pm** e **StemSp.pm** para *stemming* na língua portuguesa, inglesa e espanhola respectivamente.

### 4.3 NGram.pm

Este módulo também é chamado pelo módulo **Start.pl** e é responsável pela criação de *n*-grama. A Figura 19 ilustra as relações deste módulo dentro do PRETEXT. Por possuir uma função muito específica, o **NGram.pm** utiliza-se somente de poucas classes de suporte. Ele faz uso de arquivos temporários para a criação dos *n*-grama, os quais são apagados após a sua utilização.

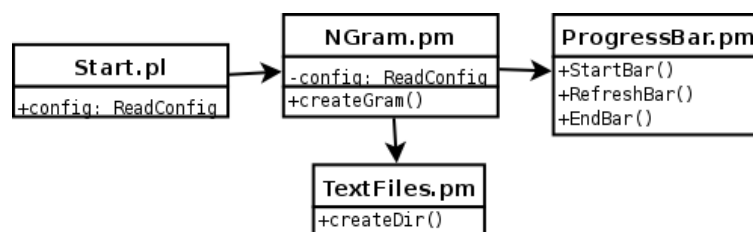


Figura 21: Diagrama de classes do módulo **NGram.pm**

### 4.4 Report.pm

Este é o último módulo principal do PRETEXT que é chamado pelo módulo **Start.pl**. Devido a sua complexidade, este módulo utiliza várias classes de suporte. Na Figura 22 é mostrada a relação deste módulo com as classes de suporte.

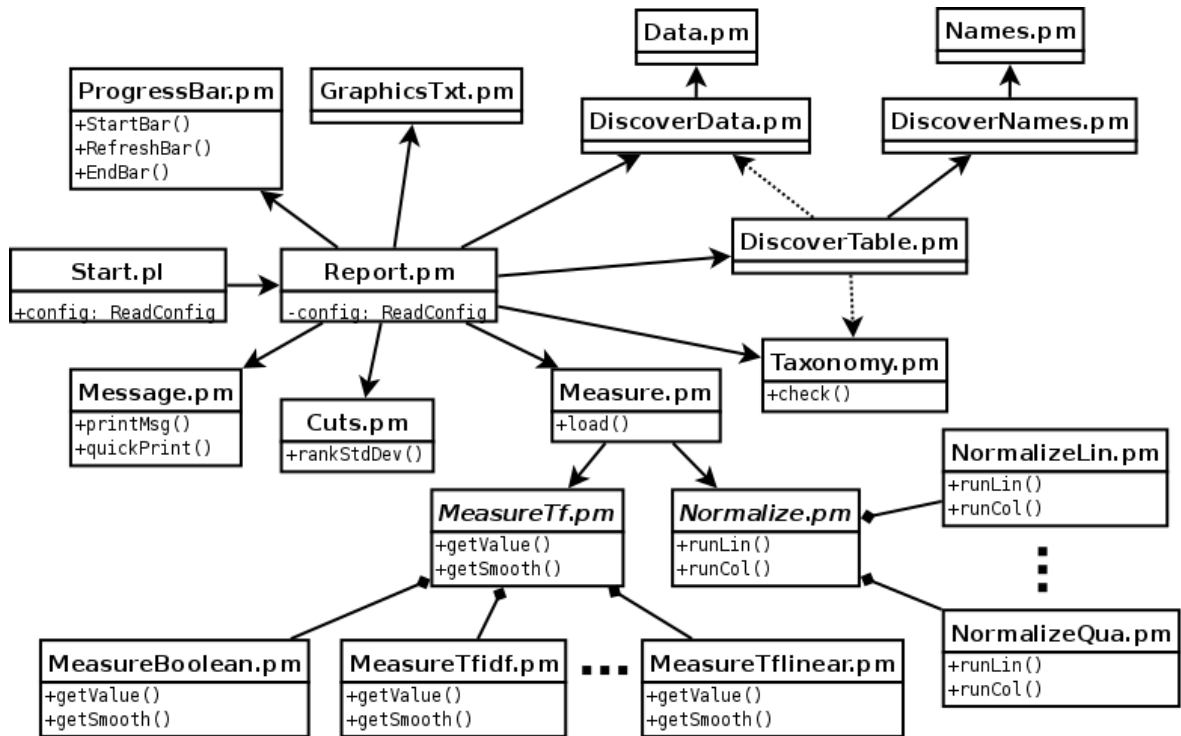


Figura 22: Diagrama de classes do módulo `Report.pm`

**DiscoverTable.pm** Esta classe gerencia a tabela atributo-valor para cada  $n$ -grama separadamente. Ao final do processo, todas as tabelas são unidas em uma só tabela atributo-valor. Ela carrega na memória os *tokens* a partir dos arquivos de  $n$ -grama `.all` e `.txt` e realiza os cortes por frequência de *token* e de documentos.

**DiscoverData.pm** Esta classe instancia inúmeros objetos da classe `Data.pm`, para armazenar todos os arquivos textos que estão sendo utilizados no pré-processamento. Ela é instanciada pela classe `Report.pm` para ser utilizada por todas as instâncias da classe `DiscoverTable.pm`, pois os documentos processados são os mesmos para todos os  $n$ -grama e não há necessidade de duplicação desta classe.

**Data.pm** Esta classe guarda o nome de um documento texto.

**DiscoverName.pm** Nesta classe ficam armazenados os atributos da tabela atributo-valor. Esta classe instancia objetos da classe `Name.pm`.

**Name.pm** Esta classe representa um *token*, *i.e.* um atributo da tabela atributo-valor. Esta classe armazena várias informações a respeito do *token*, assim como número de

arquivos nos quais o *token* apareceu, as palavras que foram transformadas neste *token*, e, caso necessário, o fator de ponderação do *token*.

**Taxonomy.pm** Esta classe é responsável por carregar na memória as taxonomias contidas no arquivo de taxonomias. Ele é carregado pelo módulo **Report.pm**. Porém, o objeto gerado por esta classe é utilizado somente na classe **DiscoverTable.pm**, onde, por meio do método **check()**, os *tokens* contidos no arquivo de taxonomias são transformados em suas generalizações.

**GraphicsTxt.pm** Classe responsável pela criação dos gráficos de *rank* dos *tokens*. É executada uma vez para cada *n*-grama diferente habilitado no **Report.pm**.

**Cuts.pm** Classe que realiza o corte por desvio padrão, explicado na Seção 3.1.

**Measure.pm** Esta classe é responsável por aplicar as medidas, suavização e normalizações a partir da frequência absoluta de cada *token* no conjunto de documentos, para a geração da tabela atributo-valor.

**MeasureTf.pm** Esta classe abstrata implementa a medida *tf*, usada como padrão caso não seja especificado nenhuma medida para o *n*-grama correspondente. Havendo o interesse de criação de uma nova medida e/ou um novo método de suavização das medidas para quando a medida resultar no valor 0 (zero), o usuário pode fazê-lo sem a necessidade de alteração de nenhum outro módulo.

**MeasureBoolean.pm**, **MeasureTfidf.pm** e **MeasureTflinear.pm** Essas classes implementam as medidas *booleana*, *tf-idf* e *tf-linear*, respectivamente. Elas herdam a classe abstrata **MeasureTf.pm** e implementam as funções de medida, e, caso necessário, as funções de suavização.

**Normalize.pm** Esta classe abstrata é utilizada para a criação da normalização dos dados. Essa normalização pode ser realizada por coluna ou por linha.

**NormalizeLin.pm** e **NormalizeQua.pm** Estas classes realizam respectivamente, as normalizações linear e quadrática, herdando a classe abstrata **Normalize.pm** e implementam

as funções de normalização.

## 5 Considerações Finais

Após a reestruturação e remodelagem da ferramenta PRETEXT, várias novas funcionalidades foram implementadas para uma melhor limpeza dos dados e a geração de tabelas atributo-valor de qualidade em menor tempo computacional utilizando conjuntos de documentos maiores que a ferramenta anterior podia processar.

A nova versão da ferramenta conta com as novas funcionalidades de remoção de tags HTML, remoção de números, escolha de símbolos relevantes, geração de  $n$ -grama com qualquer valor de  $n$ , corte de *tokens* por frequência de documentos, tabela atributo valor transposta, normalização por linha e por coluna, entre outros. Além disso todas as funcionalidades da ferramenta antiga foram revisadas e, quando possível, otimizadas para uma execução mais veloz (Soares et al., 2007a,b).

A ferramenta PRETEXT está sendo utilizada no trabalho de mestrado do aluno, e em vários outros trabalhos de mestrado e doutorado dentro e fora do instituto.

## Referências

- Crawford, D. (1998). GnuPlot: An interactive plotting program. <http://www.ucc.ie/gnuplot/gnuplot.html>. Citado na página 31.
- dos Santos, M. A. M. R. (2002). Extraíndo regras de associação a partir de textos. Dissertação de Mestrado, PUC-PR, Curitiba. <http://www.ppgia.pucpr.br/teses/DissertacaoPPGIa-MariaRoveredo-062002.pdf>. Citado na página 3.
- Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28, 11–21. Citado na página 10.
- Kopka, H. & P. W. Daly (2004). *Guide to L<sup>A</sup>T<sub>E</sub>X* (4 ed.). Addison-Wesley. Citado na página 32.
- Lamport, L. (1986). *L<sup>A</sup>T<sub>E</sub>X: A document preparation system*. Addison-Wesley. Citado na página 31.
- Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research and Development* 2(2), 159–165. Citado na página 6.

- Manning, C. D. & H. Schütze (Eds.) (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT Press. Citado na página 3.
- Matsubara, E. T., C. A. Martins, & M. C. Monard (2003). Pretext: Uma ferramenta para pré-processamento de textos utilizando a abordagem *bag-of-words*. Technical Report 209, ICMC-USP. [ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel\\_tec/RT\\_209.pdf](ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_209.pdf). Citado nas páginas 1, 9, e 13.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. Citado na página 1.
- Monard, M. C. & J. A. Baranauskas (2003). Conceitos sobre aprendizado de máquina. In S. O. Rezende (Ed.), *Sistemas Inteligentes - Fundamentos e Aplicações*, Chapter 4, pp. 89–114. Manole. Citado na página 1.
- Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137. <http://tartarus.org/~martin/PorterStemmer/def.txt>. Citado nas páginas 4 e 5.
- Porter, M. F. (2001). Snowball: A language for stemming algorithms. <http://snowball.tartarus.org/texts/introduction.html>. Citado na página 4.
- Porter, M. F. (2006). An algorithm for suffix stripping. *Program: electronic library and information systems* 40(3), 211–218. Citado na página 4.
- Prati, R. C. (2003, Abril). O framework de integração do sistema discover. Dissertação de Mestrado, ICMC-USP. <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-20082003-152116/>. Citado na página 15.
- Robertson, S. (2004). Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation* 60(5), 503–520. Citado na página 10.
- Salton, G. & C. Buckley (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management* 24(5), 513–523. Citado na página 7.
- Soares, M. V. B., R. C. Prati, & M. C. Monard (2007a). Melhorando o desempenho computacional e a geração de atributos na ferramenta de pré-processamento de textos PRETEXT. In *Revista de Iniciação Científica do XXI Congresso de Iniciação Científica e Tecnológica em Engenharia*, São Carlos. Citado na página 42.
- Soares, M. V. B., R. C. Prati, & M. C. Monard (2007b). Readaptação e extensão do ambiente de pré-processamento de textos PRETEXT. In *XV Simpósio Internacional de Iniciação Científica da Universidade de São Paulo*, São Carlos. Citado na página 42.



- Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow. Citado na página [6](#).
- Weiss, S. M., N. Indurkha, T. Zhang, & F. J. Damerau (2004). *Text Mining*. Springer. Citado na página [1](#).
- Zipf, G. (1949). *Human Behaviour and the Principle of Least Effort*. Addison-Wesley. Citado na página [6](#).