Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

PreTexT: uma ferramenta para pré-processamento de textos utilizando a abordagem *bag-of-words*

Edson Takashi Matsubara
Claudia Aparecida Martins
Maria Carolina Monard

Nº 209

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Agosto/2003

PreTexT: uma ferramenta para pré-processamento de textos utilizando a abordagem *bag-of-words*

Edson Takashi Matsubara[1]
Claudia Aparecida Martins[1,2]
Maria Carolina Monard[1]

[1]Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística
C.P. 668, 13560-970 - São Carlos, SP - Brasil
e-mail: {edsontm, cam, mcmonard}@icmc.usp.br

[2]Universidade Federal de Mato Grosso
Instituto de Ciências Exatas e da Terra
Departamento de Ciência da Computação
Av. Fernando Corrêa da Costa s/n
78060-900, Cuiabá, MT - Brasil

Resumo: A representação de documentos textuais em um formato estruturado para o processo de Mineração de Textos tem uma influência fundamental em quão bem um algoritmo de aprendizado poderá generalizar. A abordagem *bag-of-words* é uma das representações estruturadas mais simples, mais utilizada e que tem obtido um bom desempenho no processo de Mineração de Textos. No entanto, essa abordagem é caracterizada pela alta dimensionalidade e por valores esparsos na representação dos textos, visto que cada palavra é um possível atributo nessa representação. São necessárias, portanto, ferramentas computacionais que possam realizar de forma automática a transformação dos documentos em uma representação estruturada e que, ao mesmo tempo, auxilie na redução da dimensionalidade. Neste trabalho é descrita em detalhes a ferramenta PreTexT que tem essas características, bem como diversas outras funcionalidades que a distingue de outras ferramentas existentes.

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

Agosto 2003

Trabalho realizado com apoio CAPES.

Sumário

1 Introdução **1**

2 Mineração de Textos **2**

3 O Pré-Processamento de Textos **4**

3.1 Representação de Textos 4

3.2 Redução da Dimensionalidade dos Atributos 11

3.2.1 Algoritmos de *Stemming* 12

3.2.2 A Lei de Zipf e os Cortes de Luhn 13

4 A Ferramenta Computacional PreTexT **14**

4.1 O Arquivo de Parâmetros 17

4.2 O Módulo **stem.pl** 21

4.2.1 Arquivos de Entrada 21

4.2.2 Arquivos de Saída 23

4.2.3 Execução do Módulo **stem.pl** 25

4.3 O Módulo **report.pl** 26

4.3.1 Arquivos de Entrada 27

4.3.2 Arquivos de Saída 28

4.3.3 Execução do Módulo **report.pl** 29

4.4 Scripts Auxiliares 31

5 A Biblioteca de Classes do PreTexT **33**

5.1 Representação da Tabela Atributo-Valor 33

5.2 Descrição das Classes 34

6 Considerações Finais **38**

A Instalação **39**

B Execução Rápida **40**

C Algoritmos de *Stemming* **40**

C.1 Regras do Algoritmo de *Stemming* para o Inglês 41

C.2 Regras do Algoritmo de *Stemming* para o Português 44

C.3 Regras do Algoritmo de *Stemming* para o Espanhol 48

Referências **51**

Lista de Figuras

1 Fator de ponderação *idf* × *linear* 7
2 Documento `texto5.txt` representado por diferentes medidas 9
3 Documento `texto6.txt` representado por diferentes medidas 10
4 Documento `texto7.txt` representado por diferentes medidas 11
5 A curva de Zipf e os cortes de Luhn 14
6 A ferramenta PreTexT 15
7 Exemplo de arquivo de parâmetros 18
8 *Rank* de *stems* construído a partir da freqüência 21
9 Exemplo de arquivo `oneGram.all` 24
10 Exemplo de arquivo `oneGram.txt` 24
11 Exemplo de arquivo `stemWdTF.all` 25
12 Exemplo de execução do módulo **stem.pl** 26
13 Exemplo dos arquivos `.names` (a) e `.data` (b) na sintaxe padrão do Discover 29
14 Exemplo de gráfico gerado utilizando o *rank* de *stems* 29
15 Exemplo de execução do módulo **report.pl** 30
16 Exemplo de execução do *script* `checkWord.pl` 31
17 Exemplo de execução do *script* `predict.pl` 32
18 Diagrama de classes em UML do DiscoverTable, DiscoverNames e DiscoverData 34
19 Diagrama de classes em UML da StemBase 36
20 Diagrama de classes em UML do Report 37
21 Medidas de comprimento mínimo da palavra 45

Lista de Tabelas

1 Representação de documentos 4
2 Representação de documentos utilizando *bag-of-words* 8
3 Exemplo de tabela gerada com o uso de taxonomias 28
4 Regras de eliminação de sufixos para o inglês 44
5 Regras de eliminação de sufixos para o português 46
6 Terminações verbais do português 47
7 Regras de eliminação de sufixos para o espanhol 49
8 Terminações verbais do espanhol 50

1 Introdução

Uma grande parte das informações disponíveis nas organizações está em formato digital dispersa em documentos textuais. Assim, há um grande interesse em extrair conhecimento desse grande volume de informação. Esse é o objetivo do processo de Mineração de Textos utilizando, entre outros, algoritmos de Aprendizado de Máquina. Entretanto, sistemas de Aprendizado de Máquina raramente conseguem trabalhar diretamente com esse tipo de informação devido ao formato não estruturado desses textos. Desse modo, torna-se evidente a necessidade de desenvolver sistemas que possam tratar essa informação não estruturada e transformá-las em informação estruturada para possibilitar o uso de sistemas de aprendizado já existentes.

O PreTexT, apresentado neste trabalho, é uma ferramenta computacional desenvolvida na linguagem de programação Perl (Wall, Christiansen, & Schwartz, 1996), que tem como objetivo realizar pré-processamento de textos e transformar esses textos em um formato estruturado utilizado pela maioria dos algoritmos de Aprendizado de Máquina (AM). Deve ser observado que existem outras ferramentas disponíveis para esse fim (McCallum, 1996; Banerjee & Pedersen, 2003). Entretanto, foi decidido implementar o PreTexT pois essas ferramentas não possuem algumas funcionalidades que consideramos necessárias. Um outro motivo, é a facilidade de integração do PreTexT no projeto Discover, já que a sua implementação contempla os requisitos necessários para integrá-lo facilmente nesse ambiente, o que não acontece com as ferramentas disponíveis.

O projeto Discover (Baranauskas & Batista, 2000) consiste de um ambiente no qual estão integrados algoritmos de aprendizado implementados pela comunidade, bem como ferramentas específicas desenvolvidas por pesquisadores do Laboratório de Inteligência Computacional (LABIC)[1] relacionados ao projeto, para descoberta de conhecimento em bases de dados. Um dos principais objetivos do Discover é tentar diminuir o esforço, por parte dos membros do projeto, necessário para realizar experimentos para extrair conhecimento de dados. De um modo geral, o projeto Discover pode ser entendido como um conjunto de métodos que são aplicados sobre os dados ou sobre o conhecimento extraído a partir dos dados.

Dessa forma, é muito importante que o projeto Discover ofereça uma base sólida para a manipulação de dados e conhecimento. Essa base é composta por sintaxes padrão para a representação de dados e conhecimento, e por bibliotecas que oferecem um conjunto de funcionalidades básicas de manipulação de arquivos nessas sintaxes. Atualmente, existem

[1] http://labic.icmc.usp.br.

definidas sintaxes padrão para a representação de dados e para a representação de conhecimento extraído de diversos indutores simbólicos, bem como bibliotecas que oferecem funcionalidades sobre essas sintaxes padrão (Batista & Monard, 2003; Prati, Baranauskas, & Monard, 2001a,b). Novas sintaxes estão sendo especificadas, principalmente para a representação de regras de regressão (Dosualdo, 2003), regras de associação (Melanda, 2002) e clusters (Martins, Monard, & Halembeck, 2002).

O pré-processamento de dados (documentos) textuais, realizado pelo PreTexT, utiliza a abordagem *bag-of-words* que transforma esses dados não estruturados em um formato estruturado, especificamente uma tabela atributo-valor, na sintaxe padrão do Discover. Várias funcionalidades estão implementadas no PreTexT, com o objetivo de auxiliar o usuário a reduzir a dimensão (número de atributos) dessa tabela considerando os modelos induzidos pelos algoritmos de aprendizado que fazem parte do Discover, bem como agrupar atributos em um novo atributo definido pelo usuário.

Também, foram implementadas no PreTexT diversas medidas para atribuição de valores aos atributos na representação dos textos. Além das medidas tradicionais geralmente citadas na literatura, novas medidas foram propostas neste trabalho. Essas novas medidas visam tentar suprir algumas características presentes nas medidas tradicionais em relação ao comportamento para diversos valores dos atributos. Dessa forma, uma ampla gama de medidas que foram implementadas podem ser utilizadas no PreTexT.

Este trabalho está organizado da seguinte forma: na Seção 2 é descrito brevemente o processo de Mineração de Textos; na Seção 3 é apresentada a etapa de pré-processamento, usando a abordagem *bag-of-words*. Na Seção 4 são apresentadas as características, funcionalidades e modos de execução da ferramenta PreTexT. A biblioteca de classes que compõem o PreTexT é descrita na Seção 5 e as considerações finais são apresentadas na Seção 6. Nos Apêndices A e B são mostrados, respectivamente, o processo de instalação e um modo de execução rápida do PreTexT e, por fim, no Apêndice C são mostrados os algoritmos de *stemming* para o idioma inglês, português e espanhol utilizados na implementação do PreTexT.

2 Mineração de Textos

O processo de descoberta de padrões úteis e interessantes em uma coleção de documentos (textos) é conhecido como Mineração de Textos (MT). Devido a natureza textual não estruturada, os documentos necessitam de um pré-processamento para serem submetidos

a algoritmos de aprendizado. A transformação dos documentos em uma representação mais adequada, como em uma tabela atributo-valor, é uma etapa de suma importância, visto que a representação desses documentos tem uma influência fundamental em quão bem um algoritmo de aprendizado poderá generalizar a partir dos exemplos (Sebastiani, 2002).

A representação de documentos pode ser feita usando diversas abordagens, entre elas, a abordagem *bag-of-words* utilizada no PreTexT. Na abordagem *bag-of-words*, cada documento é representado como um vetor das palavras que ocorrem no documento, ou em representações mais sofisticadas como frases ou sentenças. Entretanto, resultados experimentais mostraram que representações mais sofisticadas, às vezes, perdem em desempenho com relação a abordagem *bag-of-words* (Apté, Damerau, & Weiss, 1994; Dumais, Platt, Heckerman, & Sahami, 1998; Lewis, 1992). De acordo com Lewis (1992), a razão mais provável para explicar esses resultados é que, embora termos mais sofisticados tenham qualidade semântica superior, a qualidade estatística é inferior em relação a termos baseados em palavras simples. No entanto, independente da tarefa de MT, algumas etapas são essenciais e são definidas como:

1. coleta de documentos;
2. pré-processamento;
3. extração de conhecimento;
4. avaliação e interpretação dos resultados.

A coleta de documentos é responsável pela recuperação de documentos relevantes ao domínio de aplicação do conhecimento a ser extraído. A etapa de pré-processamento é responsável por transformar os documentos em um formato adequado para serem submetidos a algoritmos de extração automática de conhecimento. A extração de conhecimento tem a finalidade de descobrir padrões úteis e desconhecidos presentes nos documentos utilizando, entre outros, sistemas de aprendizado. Por fim, a etapa de avaliação é necessária para verificar se o objetivo foi alcançado ou se algumas das etapas necessitam ser refeitas.

O pré-processamento dos documentos é uma etapa não trivial e bastante custosa, devido à forma não estruturada dos documentos. Como mencionado, os documentos necessitam ser transformados em um formato adequado, tais como uma tabela atributo-valor, para serem submetidos a algoritmos de aprendizado. No entanto, a representação de documentos no

formato atributo-valor é caracterizada pela alta dimensão e por dados bastante esparsos, sendo fundamental que essa etapa seja realizada com devido cuidado para obter um bom desempenho do processo de MT. A seguir, é descrita em maiores detalhes, a etapa de pré-processamento de documentos.

3 O Pré-Processamento de Textos

Considerando que a primeira etapa do processo tenha sido cumprida, ou seja, os documentos estejam disponíveis, é necessário realizar o pré-processamento desses documentos. Um dos procedimentos geralmente adotado, e utilizado neste trabalho, é a representação usando a abordagem *bag-of-words*, na qual cada documento é representado como um vetor de palavras que ocorrem no documento. Nas próximas seções são apresentados a representação de documentos e algumas técnicas para reduzir a dimensão do conjunto de atributos.

3.1 Representação de Textos

Dada uma coleção de N documentos $D = \{d_1, d_2, ..., d_N\}$ e um conjunto de Ncl categorias $C = \{c_1, c_2, ..., c_{Ncl}\}$ associadas à coleção de documentos D, é possível realizar tarefas relacionadas ao processo de MT, tais como categorização (ou classificação), sumarização e agrupamento (*clustering*) de documentos. A tarefa de categorização de documentos consiste em induzir um classificador que possa determinar se o documento d_i pertence (ou não) a categoria c_j, $i \in 1, 2, ..., N$ e $j \in 1, 2, ..., Ncl$.

A representação de documentos usando a abordagem *bag-of-words* pode utilizar o formato de uma tabela atributo-valor, como representada na Tabela 1. Cada documento d_i é um exemplo da tabela e cada termo (palavra) t_j é um elemento do conjunto de atributos.

	t_1	t_2	...	t_M	C
d_1	a_{11}	a_{12}	...	a_{1M}	c_1
d_2	a_{21}	a_{22}	...	a_{2M}	c_2
...	...	...	...	...	...
d_N	a_{N1}	a_{N2}	...	a_{NM}	c_N

Tabela 1: Representação de documentos

Mais especificamente, na Tabela 1 estão representados N documentos (exemplos) compostos por M termos (atributos). Cada documento d_i é um vetor $d_i = (a_{i1}, a_{i2}, ...a_{iM})$,

no qual o valor a_{ij} refere-se ao valor associado ao j-ésimo termo do documento i. O valor a_{ij}, do termo t_j no documento d_i, pode ser calculado utilizando diferentes medidas, tais como:

- *boolean*;
- *tf*;
- *tflinear*;
- *tfidf*.

A medida *boolean* usa a representação binária para os termos. Quando o termo está presente no documento o valor de a_{ij} é 1, caso contrário 0. Essa medida é muito simples e, geralmente, medidas estatísticas são empregadas levando em consideração a freqüência que os termos são encontrados nos documentos (Salton & Buckley, 1988). A medida *tf* (*term frequency*) considera o valor de a_{ij} como a freqüência que o termo aparece no documento. Essa medida é definida pela Equação 1, na qual $freq(t_j, d_i)$ é a freqüência do termo t_j no documento d_i.

$$tf(t_j, d_i) = freq(t_j, d_i) \tag{1}$$

Porém, quando alguns termos aparecem na maioria dos documentos, esses termos raramente fornecem informações úteis que possam diferenciá-los em uma tarefa de MT. Assim, para discriminar melhor os documentos, é possível utilizar também informações que indiquem a freqüência que o 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 tenha uma "força" de representação menor. A medida *tfidf* faz justamente isso. Essa medida, definida pela Equação 2, é bastante citada na literatura. A freqüência do termo no documento é multiplicada por um fator de ponderação. Esse fator varia entre 0 e $\log N$ e é calculado pela Equação 3, na qual N é o número de documentos da coleção e $d(t_j)$ é o número de documentos nos quais o termo t_j ocorre pelo menos uma vez. Assim, quando o termo aparece em todos os documentos o fator de ponderação é igual a 0, quando aparece em apenas 1 documento ele é $\log N$.

$$tfidf(t_j, d_i) = freq(t_i, d_j) \; . \; idf(t_j) \tag{2}$$

$$idf(t_j) = \log \frac{N}{d(t_j)} \tag{3}$$

Neste trabalho é proposta uma nova medida, semelhante à *tfidf*, denominada *tflinear*. Nessa medida, definida pela Equação 4, o fator de ponderação definida pela Equação 5 é linear, diferente do logarítmico usado na medida *tfidf*, Equação 3. Desse modo, o fator de ponderação de *tflinear* varia entre 0 e 1.

$$tflinear(t_j, d_i) = freq(t_i, d_j) \; . \; linear(t_j) \tag{4}$$

$$linear(t_j) = 1 - \frac{d(t_j)}{N} \tag{5}$$

Por exemplo, suponha que em uma coleção de 100 documentos ($N = 100$) o termo `estudar` apareça em 90 documentos ($d(t_j = \texttt{estudar}) = 90$), ou seja, o termo aparece em quase todos os documentos. O fator de ponderação linear para esse termo é

$$linear(t_j = \texttt{estudar}) = 1 - \tfrac{90}{100} = 0.1$$

que é um valor muito próximo de zero. Os valores de a_{ij}, para $t_j = \texttt{estudar}$, serão sempre multiplicados por 0.1, ou seja, todos os valores que compõem a coluna do termo `estudar` na tabela atributo-valor — Tabela 1 na página 4 — são multiplicados por 0.1. Assim, o termo `estudar` fica com menos "força" do que os termos que apareçam em apenas um documento, para os quais tem-se

$$linear(t_j) = 1 - \tfrac{1}{100} = 0.99$$

O fator linear para uma coleção com 1000 documentos e um termo com freqüência 50 aparecendo em apenas 1 documento, nesse caso o fator linear é dado por $(1 - \frac{1}{1000}) = 0.999$ que é muito próximo a 1. O fator logarítmico enfatiza, principalmente, termos que aparecem em poucos documentos em uma grande coleção de documentos. O valor do fator de ponderação logarítmico, para essa mesma coleção de documentos é $\log \frac{1000}{1} = 3$. Assim, o valor $tfidf$ de a_{ij} para esse termo corresponde ao valor da sua freqüência triplicada (50 vai para 150). Portanto, para esses casos, o fator linear não enfatiza o termo tanto quanto o fator logarítmico.

A Figura 1 ilustra o comportamento desses dois fatores de ponderação para uma coleção com 100 documentos. O fator *idf* para termos que apareçam em apenas um documento

é log $\frac{100}{1} = 2$. Como pode ser observado, conforme a termo está presente em uma quantidade maior de documentos, sua "força" diminui. Assim, o fator *idf* para termos que aparece em todos os documentos é log $\frac{100}{100} = 0$.

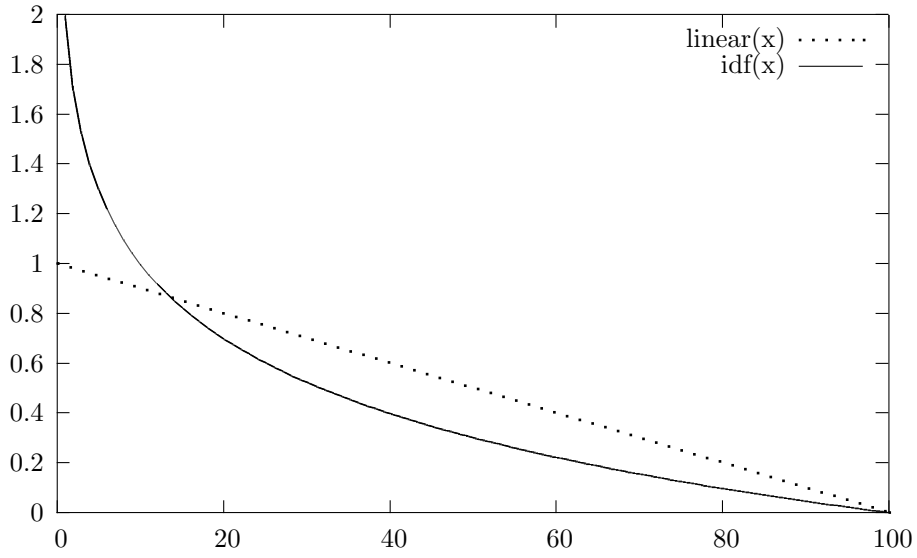

Figura 1: Fator de ponderação *idf* × *linear*

No entanto, para coleções de documentos com temas semelhantes, é possível que alguns termos apareçam em todos os documentos da coleção. Nesse caso, tanto os fatores de ponderação *idf* quanto *linear* são nulos. Desse modo, os valores desses termos são zerados para todos esses documentos, ou seja, todas as colunas correspondentes a esses termos. Uma possível solução é não permitir que o fator de ponderação seja zero. Assim, neste trabalho é proposto um outro fator denominado *smooth*.

O *smooth* é ativado quando o fator de ponderação é igual a zero, ou seja, quando um termo aparece em todos os documentos. Quando ativado, ele aumenta temporariamente em 10% a variável N que contém o número de documentos da coleção. Desse modo o fator não pode ser igual a 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.

Um outro aspecto que também deve ser levado em conta é o tamanho dos documentos na coleção. Freqüentemente, o tamanho desses documentos é muito diferente e essa diferença de tamanho deveria estar melhor refletida nas medidas utilizadas. Por exemplo, considere dois documentos que pertencem a mesma categoria com tamanhos de 1 Kbyte

e 100 Kbytes respectivamente. Nesse caso, existirá uma grande diferença na freqüência dos termos em ambos os documentos. Uma possível solução para esse problema é normalizar os valores da tabela atributo-valor — Tabela 1 na página 4. Essa normalização é freqüentemente realizada usando a normalização linear, definida pela Equação 6, ou a normalização quadrática, definida pela Equação 7.

$$NormLinear(t_j, d_i) = \frac{a_{ij}}{MAX_{i=1..N}(a_{ij})} \tag{6}$$

$$NormQuadratic(t_j, d_i) = \frac{a_{ij}}{\sqrt{\sum_{i=1}^{N}(a_{ij})^2}} \tag{7}$$

Para ilustrar os valores dessas medidas em uma coleção de documentos, foram criados 7 (sete) documentos — Tabela 2 — para mostrar como as medidas se comportam. Assim, o primeiro documento (`texto1.txt`) é composto somente pelo termo `cas` com freqüência 1, o segundo documento (`texto2.txt`) é composto por dois termos, `cas` e `filh`, também ambos com freqüência 1 e assim por diante, até que os três últimos documentos são compostos por 5 termos com freqüências variadas.

documento	amig	cidad	trabalh	filh	cas
texto1.txt	0	0	0	0	1
texto2.txt	0	0	0	1	1
texto3.txt	0	0	1	1	1
texto4.txt	0	1	1	1	1
texto5.txt	5	5	5	5	5
texto6.txt	5	8	13	28	28
texto7.txt	5	6	10	8	8

Tabela 2: Representação de documentos utilizando *bag-of-words*

O objetivo dos quatro primeiros documentos é apenas mostrar a variação das medidas em função da quantidade de documentos nos quais um mesmo termo aparece. Dessa forma, verifica-se que o termo `amig` aparece em 3 documentos, o termo `cidad` aparece em 4 documentos, o termo `trabalh` aparece em 5 documentos, e assim por diante até que o último termo `cas` aparece em todos os documentos. O fator de ponderação é calculado de acordo com o número de documentos em que o termo aparece. Portanto, os quatro primeiros documentos, `texto1.txt`, `texto2.txt` `texto3.txt` e o `texto4.txt` que contém um, dois, três e quatro termos com valores não nulos, respectivamente, foram criados estrategicamente para mostrar as variações das medidas. Em outras palavras, esses valores foram criados para melhor ilustrar a "força" que o fator de ponderação exerce sobre as medidas.

Para verificar o comportamento das medidas, observe os diferentes valores dos termos para os documentos `texto5.txt`, `texto6.txt` e `texto7.txt`. Todos os cinco termos que aparecem na coleção aparecem no `texto5.txt` com freqüência 5. Desse modo, o fator de ponderação altera a "força" de cada termo de acordo com o número de documentos em que o termo aparece, ou seja, quanto maior o número de documentos nos quais o termo aparece, menor é a sua força de representação, ilustrado pelos valores decrescentes na Figura 2. A medida tf, que não usa fator de ponderação, considera todos os termos com freqüência constante 5.

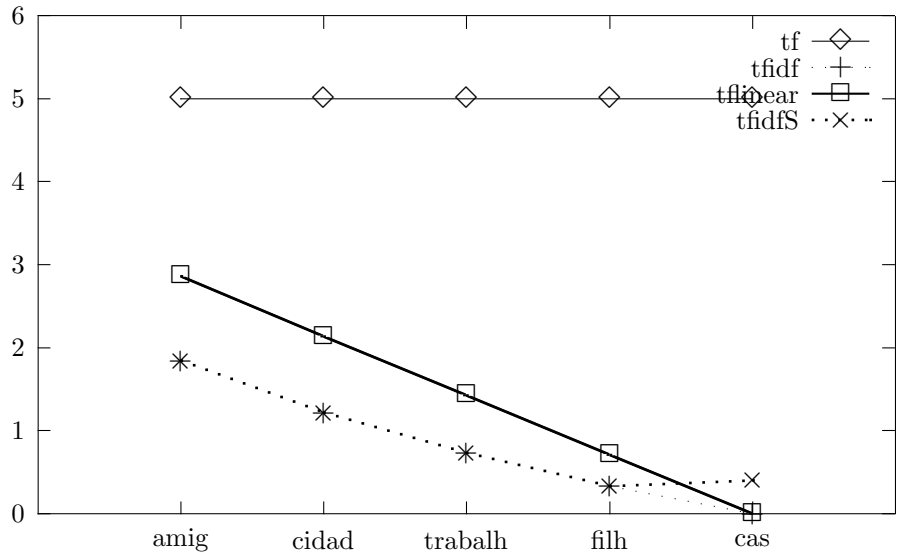

Figura 2: Documento `texto5.txt` representado por diferentes medidas

Os valores das medidas $tfidf$ e $tfidfS$ são os mesmos para os primeiros quatro termos, as quais estão sobrepostas na figura, diferindo apenas para o termo `cas`, que possui o fator de ponderação dado por

$$idf(\texttt{cas}) = \log \frac{7}{d(7)} = 0$$

Quando o valor do fator de ponderação é zero, como nesse caso, o *smooth* altera temporariamente a variável que contém o número de documentos da coleção, não permitindo que o fator de ponderação seja zero, como no caso dos valores do termo `cas` para a medida $tfidfS$. Como pode ser observado, para as medidas $tfidf$ e $tflinear$ que não utilizam o *smooth*, o termo `cas` que aparece em todos os documentos tem seu valor zerado.

O `texto6.txt` é um documento criado especialmente para verificar o comportamento das

medidas usando valores distintos para os termos, mostrando a relação entre os fatores de ponderação, como ilustrado na Figura 3.

Os valores dos termos para o documento `texto6.txt` foram criados para ilustrar o comportamento da medida $tfidf$ quando atinge valores praticamente constantes para os termos, como ilustrado na Figura 3. Observe que nessa medida os valores da freqüência (tf) são fortemente influenciados pelo fator de ponderação e que o último termo, `cas`, tem valores bem diferentes para todas as medidas.

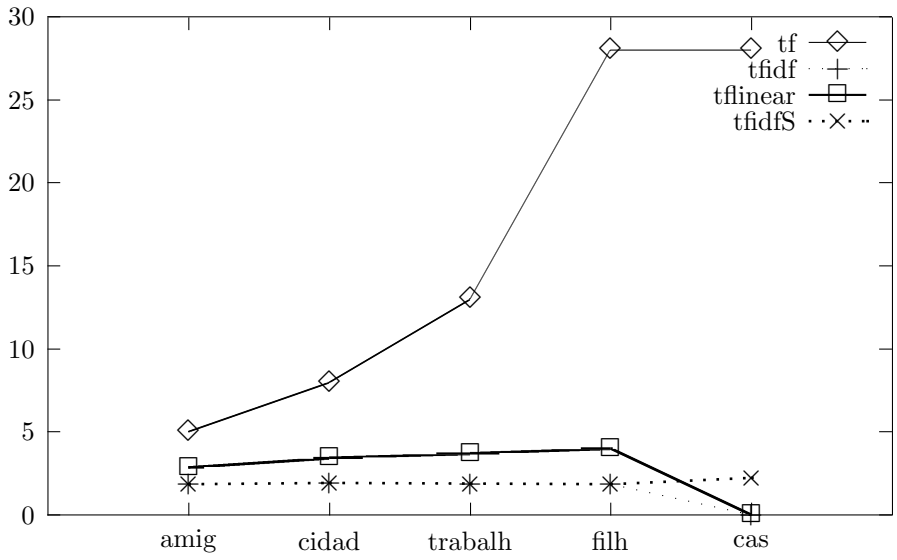

Figura 3: Documento `texto6.txt` representado por diferentes medidas

Ainda, como pode ser observado, os valores dos termos `amig` e `cas`, cujas freqüências são bem diferentes (5 e 28 respectivamente), quando os fatores de ponderação *idf* e *linear* são utilizados nas medidas $tfidf$ e $tflinear$, os valores desses dois termos ficam semelhantes, pois, a freqüência do termo `amig` é 5, porém esse termo aparece em apenas três dos sete documentos da coleção, enquanto que a freqüência do termo `cas` é 28, porém esse termo aparece em todos documentos da coleção. Finalmente, a Figura 4 ilustra essas medidas para o documento `texto7.txt`.

Portanto, o fator de ponderação é fortemente dependente do número de documentos em que o termo aparece, podendo mudar significativamente os valores dos termos na representação dos documentos. Em geral, o fator de ponderação diminui a freqüência dos termos. De fato, para o fator de ponderação *linear*, o valor da freqüência será sempre menor ou igual a própria freqüência. Para o fator de ponderação *idf*, o valor será menor caso o termo apareça em muitos documentos. O *smooth*, quando utilizado, atua apenas nos termos que aparecem em todos os documentos da coleção para que os valores desses

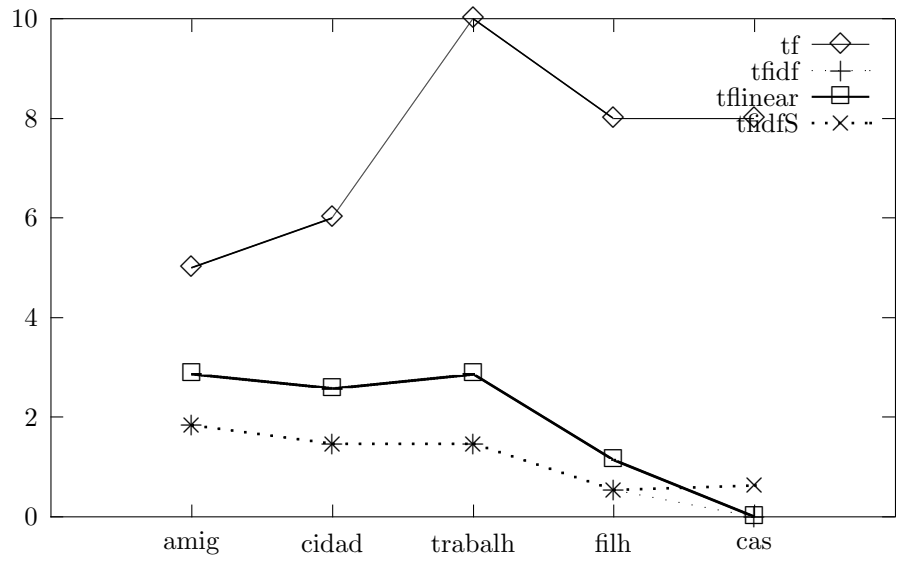

Figura 4: Documento `texto7.txt` representado por diferentes medidas

termos não se tornem nulos.

3.2 Redução da Dimensionalidade dos Atributos

Representar uma coleção de documentos utilizando a abordagem *bag-of-words* pode ser muito custoso pois o vetor que representa cada documento deve conter todos os termos de toda a coleção de documentos. Facilmente um conjunto de somente 10 documentos de tamanho médio pode necessitar de mais de 2000 atributos (termos) para ser representado, ou seja, cada documento é representado por um vetor de alta dimensão. Mais ainda, considere uma coleção de 100 documentos, na qual em muitos desses documentos somente 3 termos aparecem. No entanto, cada documento deve ser representado por um vetor no qual encontram-se representados todos os termos que aparecem na coleção de documentos. Assim, quanto maior o número de documentos na coleção, o tamanho desses documentos bem como os termos tratados, provavelmente, menor será a porcentagem de valores não nulos a_{ij} dos termos utilizados na representação da tabela atributo-valor — Tabela 1 na página 4. Em outras palavras, naturalmente, essa tabela é uma tabela esparsa.

Vários métodos podem ser utilizados a fim de reduzir a quantidade de termos (atributos) necessário para representar uma coleção de documentos, visando uma melhor representação e melhor desempenho do processo de MT. Entre outros, a transformação de cada termo para o radical que o originou, por meio de algoritmos de *stemming*, é um método amplamente utilizado e difundido. Uma outra forma para reduzir a dimensionalidade

dos atributos é buscando os termos que são mais representativos na discriminação dos documentos usando a Lei de Zipf e os cortes de Luhn. A seguir é descrito em detalhes cada uma dessas técnicas.

3.2.1 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 conseqüência da aplicação de algoritmos de *stemming* consiste na remoção de prefixos ou sufixos de um termo, ou mesmo na transformação de um verbo para sua forma no infinitivo. Por exemplo, as palavras `trabalham`, `trabalhando`, `trabalhar`, `trabalho` e trabalhos podem ser transformados para um mesmo *stem* `trabalh`. Desse modo, algoritmos de *stemming* podem ser utilizados para reduzir a dimensão da tabela atributo-valor.

Percebe-se que 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). O algoritmo tem sido amplamente usado, referenciado e adaptado nos últimos 20 anos. Diversas implementações do algoritmo estão disponibilizadas na *Web*, entre elas a página oficial escrita e mantida pelo autor para distribuição do seu algoritmo (`http://www.tartarus.org/~martin/PorterStemmer`).

Para a língua portuguesa, o algoritmo de *stemming* foi adaptado e implementado baseado no algoritmo do Porter. Nesse algoritmo, os sufixos que estão ao final de um *stem* com um comprimento mínimo estabelecido são eliminados considerando 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 palavras em português ou espanhol. Ou seja, o fato das linguagens provenientes do latim terem formas verbais altamente conjugadas em sete tempos, cada uma com seis terminações diferentes, é necessário ter um tratamento diferenciado para essas terminações.

É pouco provável que o algoritmo de *stemming* retorne o mesmo *stem* para todas as palavras que tenham a mesma origem ou radical morfológico, devido a própria natureza intrínseca dos sistemas relacionados à Recuperação de Informações (RI) e Processamento de Linguagem Natural (PLN). Já foi observado que a medida que o algoritmo se torna mais específico na tentativa de minimizar a quantidade de *stems* diferentes para palavras

com um mesmo radical, a eficiência do algoritmo degrada (Porter, 1980; Imamura, 2001).

3.2.2 A Lei de Zipf e os Cortes de Luhn

Um outro modo para reduzir a dimensionalidade dos atributos é buscar os termos que são mais representativos na discriminação dos documentos. A Lei de Zipf descreve uma maneira de encontrar termos considerados pouco representativos em uma determinada coleção de documentos. A lei, formulada por George Kingsley Zipf, professor de lingüística de Harvard (1902-1950), declara que a freqüência de ocorrência de algum evento está relacionada a uma função de ordenação. Zipf mostrou que uma das características das linguagens humanas, populações das cidades e muitos outros fenômenos humanos e naturais, seguem uma distribuição similar, a qual denominou de "*Principle of Least Effort*" (Zipf, 1949).

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, então, a forma da curva é a "curva de Zipf" para aquela coleção de documentos. Se a curva de Zipf for plotada em uma escala logarítmica, ela aparece como uma reta com inclinação -1, embora Gelbukh & Sidorov (2001) mostram que a curva de Zipf é dependente da linguagem e do estilo. A Lei de Zipf em documentos de linguagem natural pode ser aplicada não apenas aos termos mas, também, a frases e sentenças da linguagem. Na realidade, a lei de Zipf é uma observação empírica que se aplica em diversos domínios, e segue a seguinte distribuição:

$$p_1 = \frac{c}{1}, p_2 = \frac{c}{2}, ..., p_n = \frac{c}{n} \tag{8}$$

na qual $c = \frac{1}{H_n}$ e $H_n = \sum_{j=1}^{n} \frac{1}{j}$ (Knuth, 1973). Ou seja, considerando uma coleção de documentos escritos em linguagem natural, pode ser observado que o j-ésimo termo mais comum ocorre com freqüência inversamente proporcional a j.

Enquanto Zipf verificou sua lei utilizando jornais escritos em inglês, Luhn usou a lei como uma hipótese nula para especificar dois pontos de corte, os quais denominou de superior e inferior, para excluir termos não relevantes (Luhn, 1958). 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. A Figura 5 mostra 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 as palavras correspondentes ordenadas segundo essa freqüência.

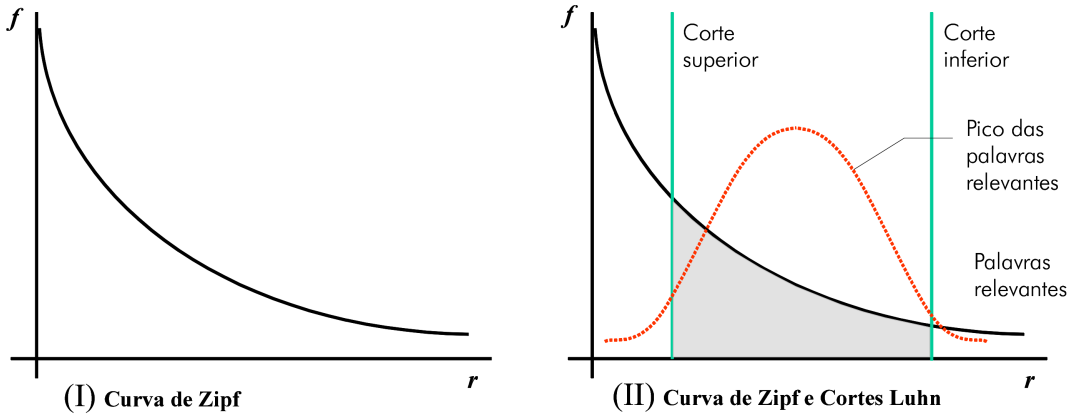

Figura 5: A curva de Zipf e os cortes de Luhn

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 são estabelecidos por tentativa e erro (Van Rijsbergen, 1979). A Lei de Zipf não é restrita apenas aos termos mas também a *stem* ou sentenças dos documentos para a maiorias dos idiomas. A seguir, é apresentada a ferramenta computacional PreTexT que utiliza os conceitos apresentados.

4 A Ferramenta Computacional PreTexT

PreTexT é uma ferramenta computacional desenvolvida com o objetivo de realizar o pré-processamento de uma coleção de documentos utilizando a abordagem *bag-of-words*. Uma de suas principais funcionalidades é transformar palavras escritas em português, espanhol ou inglês, em *stems*. O algoritmo de *stemming* implementado na ferramenta é baseado no algoritmo do Porter para a língua inglesa e adaptada para o português e o espanhol. A implementação foi realizada utilizando o paradigma de orientação a objetos em Perl (Wall, Christiansen, & Schwartz, 1996).

Como mencionado, o pré-processamento de documentos é uma etapa árdua e composta por uma seqüência de passos, os quais muitas vezes precisam ser refeitos. Considerando es-

sas características, PreTexT, ilustrado na Figura 6, foi implementado em dois módulos: **stem.pl** e **report.pl**. Desse modo, é possível tornar esse processo menos dispendioso criando arquivos intermediários para que, caso necessário, o processo seja refeito apenas a partir do passo necessário.

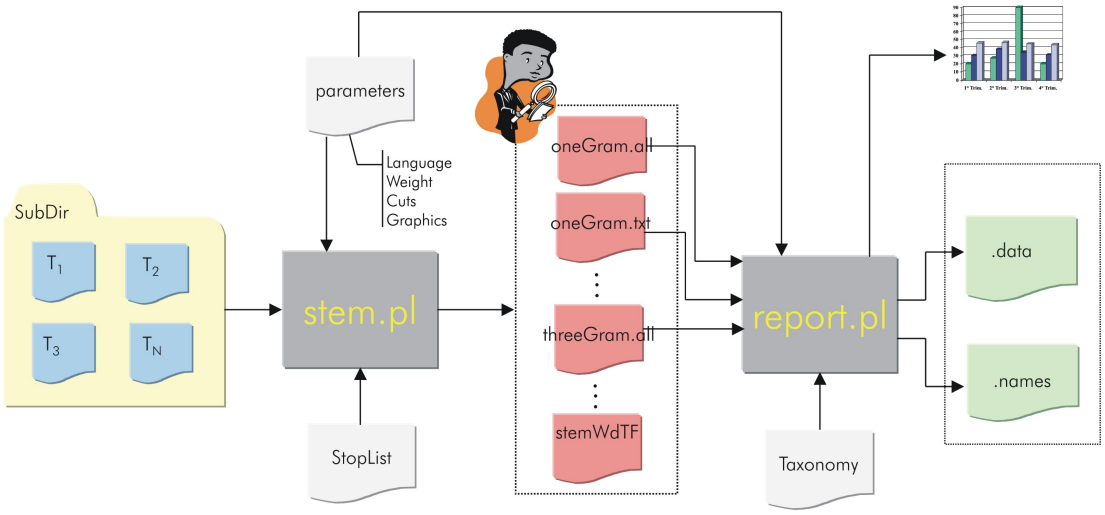

Figura 6: A ferramenta PreTexT

O módulo **stem.pl** da ferramenta é responsável pela transformação das palavras em *stems*. Os documentos a serem processados devem estar em um diretório, como ilustrado na Figura 6, ou em uma hierarquia de subdiretórios. Para execução do módulo **stem.pl** é necessário um arquivo contendo algumas especificações dos parâmetros de execução, bem como um diretório contendo uma ou mais listas de *stopwords*, que são palavras pouco significativas como artigos, preposições e conjunções que pouco caracterizam os documentos. A saída desse módulo consiste em diversos arquivos intermediários, denominados de **stembase**, que contém as *stems* correspondentes às palavras contidas nos documentos e informações úteis relacionadas a cada um dos *stems*.

É importante ressaltar que, para uma mesma coleção de documentos e uma mesma lista de *stopwords*, o módulo **stem.pl** gera a mesma **stembase**. Assim, uma vez transformado os documentos em um conjunto de *stems*, esse módulo não precisa ser executado novamente, e o usuário pode gerar diferentes tabelas atributo-valor executando apenas o módulo **report.pl** com diferentes parâmetros.

O módulo **report.pl**, a partir de alguns arquivos gerados pelo módulo **stem.pl** e os parâmetros especificados pelo usuário, retorna informações para geração de gráficos e criação da tabela atributo-valor. A tabela é gerada no formato padrão do Discover[2]

[2]A ferramenta PreTexT será integrada futuramente ao ambiente Discover, um projeto de pesquisa em desenvolvimento no Laboratório de Inteligência Computacional, LABIC - http://labic.icmc.usp.

representados pelo arquivos de dados (`.data`) e pelo arquivo de atributos (`.names`) — Figura 13 na página 29. Para calcular os valores dos atributos na tabela, a ferramenta utiliza qualquer uma das medidas, descritas na Seção 3.1 na página 4, de acordo com o parâmetro especificado pelo usuário. Além disso, a ferramenta apresenta facilidades para reduzir a dimensionalidade do conjunto de atributos usando a lei de Zipf e os cortes de Luhn.

Uma das características da ferramenta é a construção de *stems* usando mais de um *gram*. No PreTexT[3], 1-*gram* se refere a um *stem* simples, enquanto 2 e 3-*gram* referem-se a 2 ou 3 *stems*, cujas palavras ocorrem seqüencialmente no documento. A ferramenta permite a concatenação de até 3 *stems*, ou seja, 3-*gram*. Porém, os *gram* são formados a partir dos *stems* gerados e, portanto, termos que são *stopwords*, como por exemplo o termo 'de', não comparecem na concatenação de *stems*. Desse modo, é possível trabalhar com a combinação de *gram* a fim de obter uma melhor representação dos documentos. A utilização de mais de um *gram* permite que palavras que aparecem seqüencialmente no documento como "inteligência artificial", "aprendizado de máquina" e "mineração de texto", que são mais representativas conceitualmente quando utilizadas juntas, possam ser utilizadas no PreTexT.

Uma outra característica da ferramenta é a facilidade de usar indução construtiva na criação de novos atributos na tabela atributo-valor. A indução construtiva é realizada utilizando um arquivo contendo um conjunto de taxonomias definidas pelo usuário, o qual, quando disponível é interpretado pelo módulo **report.pl**.

A ferramenta PreTexT está implementada usando idéias aceitas e difundidas na comunidade científica. No entanto, PreTexT possui diversas facilidades implementadas e, quando executada usando uma coleção de documentos, retorna uma grande quantidade de informações relacionadas a esses documentos. O grande diferencial da ferramenta consiste nessa diversidade de informações geradas, uso de taxonomias e o uso de várias medidas que podem ser utilizadas para auxiliar no processo de Mineração de Textos.

Nas próximas seções são apresentados em maiores detalhes o arquivo de parâmetros e os arquivos de entrada e saída de cada um dos módulos **stem.pl** e **report.pl**.

`br`, para planejamento e execução de experimentos relacionados com o uso de sistemas de aprendizado no processo de Mineração de Dados e Textos (Batista, 2003; Prati, 2003).

[3]É importante ressaltar que existem outras formas para representação de *gram*. Neste trabalho, *n-gram* significa que *n stems* são concatenados formando um único *stem*. Ainda a palavra *gram* deve ser usada sempre no singular.

4.1 O Arquivo de Parâmetros

A principal interação entre o usuário e a ferramenta é realizada pela definição de parâmetros no arquivo de configurações — Figura 6 na página 15, denominado `parameters.cfg`. Nesse arquivo são definidas todas as informações necessárias à execução dos dois módulos de PreTexT. O arquivo de parâmetros contém a definição dos parâmetros divididos em quatro grupos, denominados respectivamente, `%global%`, `%one%`, `%two%` e `%three%`, como mostrado a seguir.

```
%global%
%one%
%two%
%three%
```

Caso o arquivo de parâmetros consiste somente dessas quatro linhas, então PreTexT é executado usando os valores *default*, os quais são mostrados na Figura 7. Comentários, um por linha, devem conter o caracter # no início da linha.

No primeiro grupo desse arquivo — `%global%` — são especificados parâmetros globais, tais como a linguagem na qual os textos estão escritos e os diretórios que contém os dados, entre outros. Todos os parâmetros que começam com `dir` são considerados definições de diretórios.

Nas divisões `%one%`, `%two%` e `%three%` são definidos os parâmetros para 1, 2 e/ou 3-*gram*. É possível habilitar um, dois ou três *gram*, ou qualquer combinação dessas. Desse modo podem-se construir tabelas com 1 e 2 *gram*, 1 e 3 *gram*, 2 e 3 *gram*, ou com 1, 2 e 3 *gram*.

Todos os parâmetros de execução do PreTexT são descritos em detalhes a seguir.

- `language`: especifica a linguagem que os textos estão escritos.
  Valores possíveis: `port`, `espn` ou `ingl`.
  Valor *default* `port`.
  Consultado por **stem.pl**.

- `logFile`: especifica o nome do arquivo com o `log` de execuções;
  Valor *default* `pretex.log`.
  Consultado por **stem.pl** e **report.pl**.

- `dirStemBase`: especifica o nome do diretório que deve conter os arquivos intermediários (**stembase**).

```
# Esta é uma linha de comentário
%global%
language      = port
logFile       = pretex.log
dirStemBase   = stembase
dirTextBase   = textbase
dirStopList   = stoplist
dirDiscover   = discover
dirGraphics   = graphics
dirPredict    = predict
%one%
measure       = freq
normalize     = quadratic
smooth        = disabled
min           = 0
max           = 0
std_dev       = 0
%two%
measure       = freq
normalize     = quadratic
smooth        = disabled
min           = 0
max           = 0
std_dev       = 0
%three%
measure       = freq
normalize     = quadratic
smooth        = disabled
min           = 0
max           = 0
std_dev       = 0
```

Figura 7: Exemplo de arquivo de parâmetros

Valor *default* `stembase`.
Consultado por **stem.pl** e **report.pl**.

- `dirTextBase`: especifica o diretório que contém a coleção de textos.
  Valor *default* `textbase`.
  Consultado por **stem.pl** e **report.pl**.

- `dirStopList`: especifica o diretório que contém a(s) lista(s) de *stopwords*.
  Valor *default* `stoplist`.
  Consultado por **stem.pl** e **report.pl**.

- `dirDiscover`: especifica o diretório no qual serão armazenados os arquivos `.data` e `.names`.

Valor *default* `discover`.
Consultado por **stem.pl** e **report.pl**.

- `dirGraphics`: especifica o diretório no qual serão armazenados os arquivos para criação de gráficos.
  Valor *default* `graphics`.
  Consultado por **stem.pl** e **report.pl**.

- `gram`: se `disabled` o *gram* é desabilitado para o módulo em execução. Por exemplo, pode-se habilitar os três *grams* para executar o **stem.pl** e habilitar apenas um dos *grams* para o **report.pl**.
  Valores possíveis: `enabled` e `disabled`
  Valor *default* `enabled`.
  Consultado por **stem.pl** e **report.pl**.

- `measure`: define a medida que é utilizada para construir a tabela atributo-valor.
  Valores possíveis `bool`, `freq`, `tfidf` e `tflinear`.
  Valor *default* `freq`.
  Consultado por **report.pl**.

- `normalize`: define o método de normalização a ser aplicado aos valores dos atributos da tabela atributo-valor.
  Valores possíveis `linear`, `quadratic` e `disabled`
  Valor *default* `quadratic`.
  Consultado por **report.pl**.

- `smooth`: se `disabled` o *smooth* é desabilitado. Caso contrário *smooth* é habilitado.
  Valores possíveis: `enabled` e `disabled`
  Valor *default* `disabled`.
  Consultado por **report.pl**.

- `min`: se definido, são carregados apenas os *stems* que tiverem freqüência maior ou igual ao valor definido.
  Valor *default* `0` (desabilitado).
  Consultado por **report.pl**.

- `max`: se definido, são carregados apenas os *stems* que tiverem freqüência menor ou igual ao valor definido.
  Valor *default* `0` (desabilitado).
  Consultado por **report.pl**.

- `std_dev`: se definido, é calculado o *rank* de *stems*. Com o desvio padrão desse *rank* é definido um intervalo $(\overline{x} - ks; \overline{x} + ks)$, no qual $\overline{x}$ é a média do *rank*, s é o desvio padrão desse *rank* e k é o valor definido pelo usuário. Os *stems* que estão fora desse intervalo são descartados.
  Valor *default* `0` (desabilitado).
  Consultado por **report.pl**.

Os três últimos parâmetros são usados na redução da dimensão da tabela atributo-valor, representando os pontos de corte de Luhn. Os parâmetros `max` e `min` definem manualmente os pontos de corte de Luhn. Todos os *stems* cujas freqüências, considerando toda a coleção de documentos, encontram-se fora do intervalo definido por esses parâmetros são ignorados na construção da tabela atributo-valor. Como mencionado, os cortes são efetuados somente quando definidos no arquivo de parâmetros, no qual o usuário pode realizar qualquer combinação com os cortes `min` e `max`. Por exemplo:

- para `min` = 3, todos os *stems* com freqüência 1 e 2 são ignorados e os *stems* com freqüência maiores ou igual a 3 são mantidos na tabela atributo-valor;

- para `max` = 10, todos os *stems* com freqüência maior que 10 são ignorados e os *stems* com freqüência igual ou menor a 10 são mantidos;

- para `min` = 3 e `max` = 10, somente os *stems* com freqüência maior ou igual a 3 e menores ou igual 10 sao mantidos.

O parâmetro `std_dev` faz cortes dos *stems* de acordo com o desvio padrão sobre o *rank* de *stems*. O *rank* é construído da seguinte maneira: suponha uma competição no qual é possível ter empate entre as colocações, ou seja, vários primeiros, segundos e terceiros colocados. O importante no *rank* são quantas colocações existem, neste caso do primeiro até o terceiro.

Imagine uma corrida de *stems*, como os ilustrados na Figura 8 (a), no qual a velocidade é determinada pela freqüência de cada *stem* e todos largam na mesma posição. Assim o primeiro colocado é o *stem* que tiver a maior freqüência, o segundo colocado o *stem* que tiver a segunda maior freqüência, e assim por diante, podendo haver sempre empate em todas as colocações. O *rank* de *stems* é exatamente isso, o *stem* de maior freqüência fica com a primeira colocação, o de segunda maior freqüência fica na segunda colocação e assim por diante é construído o *ranking*.

stem	freq
amig	5
linux	5
gnu	5
cidad	7
trabalh	12
filh	13
idad	13
cas	17

(a)

rank	freq
1	17
2	13
3	12
4	7
5	5

(b)

Figura 8: A freqüência dos *stems* (a) gera o *Rank* de *stems* (b)

Com o *rank* construído, é calculado o desvio padrão e a média dos valores de 1 a 5, obtendo-se o valor 1.58 e 2.5, respectivamente[4]. Supondo que seja definido no arquivo de parâmetros `std_dev = 0.8`, o intervalo do *rank* é [2.5 - 1.58×0.8 , 2.5 + 1.58×0.8] = [1.73,4.26]. Esse intervalo é sobre o *rank* e, portanto, ele é convertido para [7,13] que é um intervalo sobre a freqüência. Assim, os *stems* com freqüências menor que 7 e maior que 13 são ignorados.

4.2 O Módulo stem.pl

O módulo **stem.pl** é responsável por transformar as palavras dos documentos em *stems* e armazenar essas informações no diretório **stembase**. Para a execução desse módulo são necessários alguns arquivos e diretórios que são apresentados a seguir.

4.2.1 Arquivos de Entrada

Os arquivos de entrada para o módulo **stem.pl** consistem de:

- arquivo de parâmetros (`parameters.cfg`): o módulo **stem.pl** consulta principalmente os parâmetros `language`, `dirStemBase`, `dirTextBase`, `dirStopList` e os *grams* habilitados (`gram = enabled`).
- diretório de *stopwords* (**stoplist**): a **stoplist** é um diretório contendo um ou mais arquivos com as *stopwords*. Cada um desses arquivos são denominados *stopfiles*. Essas *stopfiles* contém apenas palavras como artigos, preposições, conjunções, entre outras, que são pouco significativas para caracterizar o documento. O PreTexT aloca em memória apenas as *stopfiles* que tiverem a extensão `.enabled`. Desse modo

[4]Para os exemplos do PreTexT, sempre é utilizado o ponto para separar as casas decimais, ao invés da vírgula, pois na execução do PreTexT, só é possível utilizar o ponto

é possível habilitar e desabilitar várias *stopfiles* apenas alterando a sua extensão. Nas *stopfiles*, cada *stopword* deve ser separada por quebra de linha como ilustrado a seguir:

```
de
porem
afinal
ou
cujo
```

Durante a execução, todas as *stopfiles* ativas (com extensão `.enabled`) são concatenadas formando uma única lista. Assim, é aconselhável que o usuário crie listas específicas para cada domínio, como por exemplo, uma lista específica para economia, outra para informática e assim por diante. Desse modo, quando a base de texto tiver documentos nesses domínios, o usuário pode ativar essas listas renomeando esses arquivos com a extensão `.enabled`.

- diretório de base de textos (**textbase**): a base de textos, ou **textbase**, pode ser construída de duas formas, que determinam se os dados devem ser rotulados ou não:

  1. Dados não rotulados: criar o diretório **textbase** contendo os arquivos textos da coleção sem nenhuma divisão de subdiretórios.
  2. Dados rotulados: criar o diretório **textbase** com as classes separadas por subdiretórios dentro da **textbase**.

     Por exemplo, suponha que sejam fornecidos os arquivos `debian.txt`, `oracle.txt`, `economiafolha.txt` e `economiaestadao.txt`, sendo que os dois primeiros são da classe `tecnologia` e os dois últimos da classe `economia`. Para que no final de todo o processo seja construída a tabela com os dados rotulados, deve-se construir o diretório **textbase** com a seguinte estrutura.

     ```
     textbase/
              tecnologia/
                         debian.txt
                         oracle.txt
              economia/
                       economiafolha.txt
                       economiaestadao.txt
     ```

     Desse modo, com o arquivos separados por subdiretórios, no final do processo, o PreTexT rotula automaticamente os documentos.

4.2.2 Arquivos de Saída

A saída do módulo **stem.pl** consiste de 8 (oito) arquivos intermediários que compõem a **stembase**, alguns com informações necessárias ao módulo **report.pl** e outros com informações úteis para o usuário.

Para cada *gram* definido pelo usuário são gerados dois arquivos: um com extensão `.all` e outro com extensão `.txt`. O arquivo com extensão `.all` contém todos os *stems* da coleção, com as freqüências respectivas, e o arquivo com extensão `.txt` contém a freqüência dos *stems* para cada um dos documentos da coleção. Assim, são gerados até seis arquivos, dois para cada *gram* e outros dois arquivos restantes contém as palavras que formaram cada um dos *stems*.

A **stembase** fornece informações no nível da palavra, o que permite saber em qual *stem* cada palavra foi transformada e em quais documentos se encontram cada um desses *stems*. Nos arquivos com extensão `.all` é possível observar, entre outras coisas, quais são os *stems* que podem ser *stopwords* em potencial, observando a freqüência e em quantos documentos o *stem* aparece. Além das *stopwords*, é possível verificar quais são os pontos de cortes máximo e mínimo mais razoáveis verificando até que freqüência as palavras mais relevantes aparecem.

Os oito arquivos gerados pelo módulo **stem.pl** são descritos a seguir.

- `oneGram.all`: este arquivo contém todos os *stems* para 1-*gram* de todos os documentos da coleção e em quantos arquivos o *stem* aparece, como ilustrado na Figura 9 na página seguinte. Por exemplo, na linha 3 dessa figura, `12:(3/5):trabalh`, informa que 12 é a freqüência do *stem* `trabalh` e (`3/5`) significa que esse *stem* apareceu em 3 dos 5 arquivos da coleção.

- `oneGram.txt`: este arquivo contém os *stems* separados por documentos, como ilustrado na Figura 10 na próxima página. Nessa figura, `texto1.txt` (linha 1) é composto apenas pelo *stem* `cas` com freqüência 6, `texto2.txt` (linha 3) é composto por dois *stems*, dos quais o *stem* `cas` aparece com freqüência 1 e o *stem* `filh` com freqüência 3.

- `twoGram.all`: semelhante a `oneGram.all`, porém com *stems* de 2-*grams*.

- `twoGram.txt`: semelhante a `oneGram.txt`, porém usando 2-*grams*.

- `threeGram.all`: semelhante a `oneGram.all`, porém usando 3-*grams*.

- `threeGram.txt`: semelhante a `oneGram.txt`, porém usando 3-*grams*.

- `stemWdTF.all`: este arquivo contém os *stems* e as palavras que as formaram ordenados pela freqüência, como mostrado na Figura 11 na página oposta. Nessa figura as palavras `filha filhas filho` e `filhos` foram transformadas no *stem* `filh`. Os valores na frente dos *stems* mostram a freqüência do *stem* na coleção de documentos e em quantos documentos o *stem* apareceu. Por exemplo, na linha 16 é informado que o *stem* `filh` aparece 13 vezes em 4 dos 5 documentos da coleção. Os valores na frente das palavras mostram a freqüência das palavras na coleção de documentos. Na linha 17 a palavra `filha` aparece 4 vezes na coleção de documentos.

- `stemWdST.all`: semelhante ao `stemWdTF.all`, mas ordenado alfabeticamente.

```
1  5:(1/5):amig
2  7:(2/5):cidad
3  12:(3/5):trabalh
4  13:(4/5):filh
5  17:(5/5):cas
```

Figura 9: Exemplo de arquivo `oneGram.all`

```
1  textbase/texto1.txt
2          cas:6
3  textbase/texto2.txt
4          cas:1
5          filh:3
6  textbase/texto3.txt
7          cas:1
8          filh:1
9          trabalh:1
10 textbase/texto4.txt
11         cas:1
12         cidad:1
13         filh:1
14         trabalh:1
15 textbase/texto5.txt
16         amig:5
17         cas:8
18         cidad:6
19         filh:8
20         trabalh:10
```

Figura 10: Exemplo de arquivo `oneGram.txt`

```
amig : 5 (1/5)
      amiga : 2
      amigas : 1
      amigo : 1
      amigos : 1
cidad : 7 (2/5)
      cidade : 4
      cidades : 2
      cidadoes : 1
trabalh : 12 (3/5)
      trabalham : 1
      trabalhando : 1
      trabalhar : 2
      trabalho : 6
      trabalhos : 2
filh : 13 (4/5)
      filha : 4
      filhas : 3
      filho : 3
      filhos : 3
cas : 17 (5/5)
      casada : 1
      casado : 2
      casara : 2
      casaram : 5
      casas : 2
      casos : 2
      casou : 3
```

Figura 11: Exemplo de arquivo `stemWdTF.all`

4.2.3 Execução do Módulo stem.pl

Com os parâmetros, arquivos e diretórios definidos corretamente, a execução do módulo **stem.pl**, para transformar os documentos em uma base de *stems*, é realizada digitando na linha de comando **stem.pl**.

Um exemplo de execução é ilustrado na Figura 12 na próxima página. Nesse exemplo, a partir da linha 9 até a 20 são mostrados todos os parâmetros lidos pelo **stem.pl** definidos no arquivo `parameters.cfg`. Como pode-se notar, apenas o parâmetro para um *gram* está habilitado. Portanto, são criados *stems* formados por apenas uma palavra. Nas linhas 24 e 25 é informado que foram lidas 634 *stopwords* divididas em dois arquivos. Todas essas informações são mostradas durante a execução do módulo **stem.pl** e são armazenadas no arquivo de log (`pretex.log`).

É importante salientar que na execução desse módulo, é recomendável que sejam ativado todos os *gram*, já que computacionalmente o tempo de execução é praticamente o mesmo.

```
Execution started at Mon Jul  7 20:58:56 2003

        PreTex: stem.pl

        Implemented by LABIC


             === PARAMETERS READ ===
             language      = port
             dirStemBase   = stembase
             dirTextBase   = textbase
             dirStopList   = stoplist
             dirDiscover   = discover
             dirGraphics   = graphics
             dirPredict    = predict
             logFile       = pretex.log

             ENABLED         one gram
             DISABLED        two gram
             DISABLED        three gram


             ### STOP LIST ###
              Total StopWords 634
              Total StopFiles 2

.....
Creating StemBase
=== one ===
Total Stems : 5
Total Texts : 5
stembase/oneGram.all created!
stembase/oneGram.txt created!
stembase/stemWdST.all created!
stembase/stemWdTF.all created!


Stemize Success

Execution finished at Mon Jul  7 20:58:56 2003
Execution Time: 0 seconds
```

Figura 12: Exemplo de execução do módulo **stem.pl**

4.3 O Módulo report.pl

A base de stems (**stembase**) gerada pelo módulo **stem.pl** contém, entre outros, os arquivos com informações sobre as freqüências dos *stems*, como o ilustrado na Figura 10 na página 24 para 1-*gram*. A partir desses arquivos, o PreTexT cria a tabela atributo-valor, na qual cada documento é um exemplo, e os *stems* são os atributos dessa tabela, no formato da Tabela 2 na página 8 no formato padrão do sistema Discover.

O valor dos atributos da tabela atributo-valor, pode ser calculada utilizando qualquer uma das medidas descritas na Seção 3.1 na página 4. O módulo **report.pl** é também responsável, se requerido pelo usuário, pela aplicação dos cortes de Luhn. Para a execução

desse módulo **report.pl** são necessários alguns arquivos apresentados a seguir.

4.3.1 Arquivos de Entrada

Os arquivos de entrada para o módulo **stem.pl** consistem de:

- arquivo de parâmetros (`parameters.cfg`): é na execução do módulo **report.pl** que são consultados os parâmetros: `max`, `min`, `measure`, `smooth` e `normalize`.
- diretório **stembase**: a **stembase** gerada pelo módulo **stem.pl**. São consultados os arquivos `.all` e `.txt` para cada *gram* definido pelo usuário.
- arquivo de taxonomias (`taxonomy.txt`): este arquivo é opcional, ou seja, caso esteja disponível no mesmo diretório no qual foi invocado **report.pl**, o PreTexT realiza indução construtiva. A indução construtiva é caracterizada pela criação de novos atributos a partir de alguns atributos já existentes. Esses novos atributos, geralmente, são generalizações de dois ou mais atributos.

  Por exemplo, considere os cinco *stems* `amig`, `cidad`, `trabalh`, `filh` e `cas` na Figura 10 na página 24, que são os *stems* correspondentes a coleção de documentos. Caso o usuário considere que os *stems* `cas` e `filh` podem ser agrupados em um único *stem*, no qual o usuário decide nomear `familia`, e os outros três *stems* `trabalh` `cidad` e `amig` podem ser agrupados em um *stem* nomeado `outros` pelo usuário, então, o arquivo `taxonomia.txt` deve conter essa informação, como ilustrado abaixo.

  ```
  familia: cas, filh
  outros: trabalh, cidad, amig
  ```

  Nesse caso, a tabela com as freqüências dos cinco *stems* originais, — Tabela 3 (a) — é transformada pra a tabela ilustrada na Tabela 3 (b) com apenas dois atributos e cuja freqüência é a soma das freqüências dos atributos originais, como definido no arquivo `taxonomia.txt` correspondente.

  É importante salientar que essa taxonomia é geralmente definida pelo usuário/especialista após a análise dos arquivos de *stem* gerado por **stem.pl**. As palavras que representam a junção de *stems*, como no exemplo, `familia` e `outros`, aparecem no arquivo de saída `.names` sempre com letras maiúsculas, para que possam ser diferenciadas dos *stems*.

documento	amig	cidad	trabalh	filh	cas
texto1.txt	0	0	0	0	6
texto2.txt	0	0	0	3	1
texto3.txt	0	0	1	1	1
texto4.txt	0	1	1	1	1
texto5.txt	5	6	10	8	8

(a)

FAMILIA	OUTROS
6	0
4	0
2	1
2	2
16	21

(b)

Tabela 3: Exemplo de tabela gerada com o uso de taxonomias

4.3.2 Arquivos de Saída

A saída do módulo **report.pl** consiste de 2 (dois) arquivos que representam a tabela atributo-valor na sintaxe padrão do DISCOVER[5] e arquivos com dados para geração de gráficos, descritos a seguir:

- arquivo `discover.names`: neste arquivo são declarados os atributos (*stems*) como ilustrado na Figura 13 (a);
- arquivo `discover.data`: este arquivo contém os valores dos atributos para cada exemplo e representa a tabela atributo-valor propriamente dita, como ilustrado na Figura 13 na próxima página (b);

  A primeira linha do arquivo `discover.names` representa o atributo da primeira coluna no `discover.data`, a segunda linha do `discover.names` representa o atributo da segunda coluna no `discover.data`, e assim por diante.

  O primeiro atributo, `filename`, é sempre igual para qualquer que seja a coleção de documentos, pois ele representa o nome do arquivo no qual está armazenado o documento e, normalmente, não é utilizado pelos algoritmos de Aprendizado de Máquina;

- arquivos para geração de gráficos: são gerados arquivos com dados sobre o *rank* de *stems* os quais podem ser utilizados para gerar os gráficos no software que o usuário desejar.

  Além desses arquivos, o PRETEXT gera 3 *scripts* para **GnuPlot**^{TM} (Crawford, 1998), `oneGnuPlot.script`, `twoGnuPlot.script` e `threeGnuPlot.script`, para cada conjunto de *grams* respectivos, os quais geram os gráficos no formato LaTeX (Kopla & Daly, 1999). Esses *scripts* podem ser facilmente alterados para gerar outros formatos, como `png` e `ps` alterando a primeira linha dos *scripts*.

[5]Detalhes da sintaxe Padrão do sistema DISCOVER são encontradas em http://www.icmc.usp.br/~gbatista/Discover/SintaxePadraoFinal.html

```
filename:string:ignore.
amig:integer.
cidad:integer.
trabalh:integer.
filh:integer.
cas:integer.
```

(a)

```
textbase/texto1.txt,0,0,0,0,6
textbase/texto2.txt,0,0,0,3,1
textbase/texto3.txt,0,0,1,1,1
textbase/texto4.txt,0,1,1,1,1
textbase/texto5.txt,5,6,10,8,8
```

(b)

Figura 13: Exemplo dos arquivos `.names` (a) e `.data` (b) na sintaxe padrão do DISCOVER

Por exemplo, a Figura 14, gerada pelo *script* `oneGnuPlot.script` , representa o *rank* de *stems* apresentado na Figura 8 na página 21. Apesar de simples, quando aplicado em conjunto de documentos maiores pode ser muito útil para auxiliar a determinar os pontos de cortes de Luhn — Figura 5 na página 14.

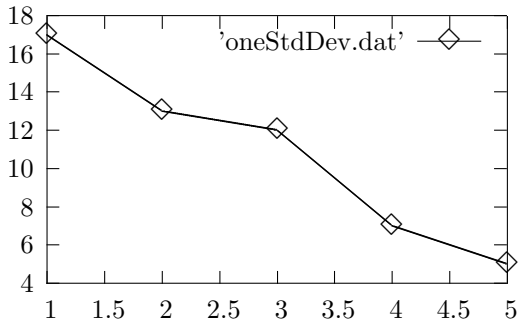

Figura 14: Exemplo de gráfico gerado utilizando o *rank* de *stems*

4.3.3 Execução do Módulo report.pl

Com os parâmetros, arquivos e diretórios definidos corretamente, a execução de **report.pl** é feita invocando o módulo no mesmo diretório do arquivo de parâmetros. Assim, para o PRETEXT criar os arquivos de dados (`.data`) e atributos (`.names`) no formato do DISCOVER, basta digitar na linha de comando **report.pl**.

Na Figura 15 está ilustrado um exemplo de arquivo de log de execução do **report.pl**. A partir da linha 7 até 19 são apresentados os parâmetros lidos pelo módulo. Como já mencionado, uma característica presente quando uma coleção de documentos em formato não estruturado é transformada em formato estruturado usando a abordagem *bag-of-words*, é que a tabela atributo-valor gerada é muito esparsa, ou seja, a porcentagem de valores não nulos é muito pequena. PRETEXT calcula essa porcentagem que é mostrada na linha 37 dessa figura.

```
Execution started at Mon Jul  7 20:52:41 2003

PreTex: report.pl
Implemented by LABIC


            === PARAMETERS READ ===
            language      = port
            dirStemBase   = stembase
            dirTextBase   = textbase
            dirStopList   = stoplist
            dirDiscover   = discover
            dirGraphics   = graphics
            dirPredict    = predict
            logFile       = pretex.log
            ############ one ############
             measure      = freq
             normalize    = disabled
             smooth       = enabled
            ########################
            Taxonomy was NOT Loaded

              ======one======

Loading stembase/oneGram.all

Number of stems loaded from stembase/oneGram.all   5

Loading stembase/oneGram.txt
.....
graphics/oneStdDev.dat created!
graphics/oneStdDevFF.dat created!
graphics/oneGnuPlot.script created!
=== one ===
Total Stems : 5
Total Texts : 5
Matrix Density                                    60%
=== CREATING DISCOVER NAMES ===
Number of Stems                                    5
Discover Names Created
=== CREATING DISCOVER DATA ===
TotalPoints (point = text)                         5
.....
Number of Texts                                    5
Discover Data Created


report Success

Execution finished at Mon Jul  7 20:52:41 2003
Execution time: 0 seconds
```

Figura 15: Exemplo de execução do módulo **report.pl**

```
c:\workdir>checkWord.pl algumas
algumas is a STOPWORD!!
word: algumas
stem: algum
```

Figura 16: Exemplo de execução do *script* `checkWord.pl`

4.4 Scripts Auxiliares

Foram implementados no PreTexT vários *scripts* auxiliares, descritos a seguir:

- `checkWord.pl`: transforma uma palavra, fornecida como parâmetro, no *stem* correspondente. Esse *script* é útil para verificar o *stem* de uma palavra e para verificar se palavra é ou não uma *stopword* definida na **stoplist**. A execução desse *script* é feita invocando o commando `checkWord.pl <palavra>` no qual `<palavra>` é a palavra que se deseja verificar, como exemplificado na Figura 16:

  Nesse exemplo, foi realizado uma consulta referente a palavra `algumas`. O *script* informa que essa palavra está atualmente declarada como *stopword*[6] e também, informa ao usuário que o `algum` é o *stem* dessa palavra.

- `predict.pl`: após gerados os arquivos `discover.names` e `discover.data`, eles devem ser submetido a algum algoritmo de Aprendizado de Máquina para induzir um modelo. Ou seja, esses arquivos definem o conjunto de treinamento usado pelo algoritmo. Entretanto, o modelo induzido deve ser avaliado em um conjunto de dados, independente, denominado conjunto de teste. Em outras palavras, o conjunto de teste é constituída por uma nova coleção de documentos. No entanto, não é possível executar novamente os módulos **stem.pl** e **report.pl** para realizar a transformação desse novos documentos, pois, mesmo utilizando parâmetros idênticos aos realizado para realizar a transformação da coleção de documentos usados para treinamento, a nova coleção de documentos deve ser utilizado para testar o modelo induzido pelo algoritmo de AM, pode ter *stems* diferentes. Assim, o *script* `predict.pl` é responsável pela transformação da nova coleção de documentos utilizando os mesmos atributos (*stems*) que a coleção de documentos usada para induzir o modelo.

  Para a sua execução basta criar uma **textbase**, porém com o nome "`predict`" ao invés de "`textbase`" como explicado na Seção 4.2 na página 21 item **textbase**, e executar na linha de comando `predict.pl`. A Figura 17 na próxima página ilustra um exemplo de execução desse *script*.

[6] Alguns arquivos no diretório de *stopword* contém a palavra `algumas`.

```
Execution started at Thu Jul 17 20:26:56 2003

PreTex: predict.pl
Implemented by LABIC


            === PARAMETERS READ ===
            language      = port
            dirStemBase   = stembase
            dirTextBase   = textbase
            dirStopList   = stoplist
            dirDiscover   = discover
            dirGraphics   = graphics
            dirPredict    = predict
            logFile       = pretex.log
            ############ one ############
             measure      = tfidf
             normalize    = disabled
             smooth       = enabled
            ##########################
            Taxonomy was NOT Loaded
...
              ======one======

Loading stembase/oneGram.all

Number of stems loaded from stembase/oneGram.all   5

Loading stembase/oneGram.txt
.....
Loading TFIDF
Loading SMOOTH
=== CREATING DISCOVER TEST ===
TotalPoints (point = text)                         3
...
Number of Texts                                    3
Discover Test Created


report Success

Execution finished at Thu Jul 17 20:26:56 2003
Execution time: 0 seconds
```

Figura 17: Exemplo de execução do *script* `predict.pl`

5 A Biblioteca de Classes do PreTexT

O PreTexT é um pacote de *scripts* em Perl que utilizam um conjunto de classes para realizar pré-processamento de documentos. No PreTexT, a tabela atributo-valor é a principal estrutura de armazenamento de dados. Essa estrutura foi projetada para ser manipulada por outras classes.

Nas seções seguintes são descritos em maiores detalhes a estrutura que armazena a tabela atributo valor e as classes que são utilizadas para manipular essa tabela.

5.1 Representação da Tabela Atributo-Valor

Normalmente, uma matriz é utilizada para representar uma estrutura de tabela atributo-valor. Para Mineração de Texto, muitas vezes essa não é a melhor estrutura de armazenamento. Isso ocorre devido a grande dimensão dos vetores na representação *bag-of-words*. Deve ser observado que a dimensão dos vetores que representam a coleção de documentos, é a mesma para qualquer documento na coleção, é dada pelo número de termos em toda a coleção de documentos. Desse modo, é gerada uma matriz de alta dimensão com poucos valores não nulos.

Suponha, por exemplo, uma coleção de 100 documentos com 1000 termos cada, no qual, todos os termos são disjuntos, *i.e.*, não há termos em comum. Com a representação de tabela atributo-valor como uma matriz, cada documento é representado com um vetor de tamanho 100000, mas tendo apenas 10% de seus valores não nulos. Essa representação, na maioria das vezes, quando não tratada, desperdiça muita memória. No PreTexT, ao invés dessa representação, pelo menos durante a execução, é utilizado uma tabela *hash* para cada documento. Desse modo, no exemplo apresentado, é utilizado uma estrutura que equivaleria aproximadamente a um vetor de tamanho 2000 (1000 para os valores dos atributos e 1000 para os atributos), ao invés de 100000.

A Figura 18 mostra o diagrama de classes em UML que representa a tabela atributo-valor do PreTexT. Os atributos ficam armazenados no DiscoverNames e os nomes dos arquivos texto que contém os documentos ficam armazenados no DiscoverData. Como já descrito, cada exemplo é uma tabela *hash* que fica armazenada na classe DiscoverTable. O acesso aos valores da tabela é feito com o método `get`, que faz com que a tabela se comporte como uma matriz, quando na verdade, ela está acessando uma tabela *hash*. O conjunto dessas três classes representam a tabela atributo-valor utilizada no PreTexT.

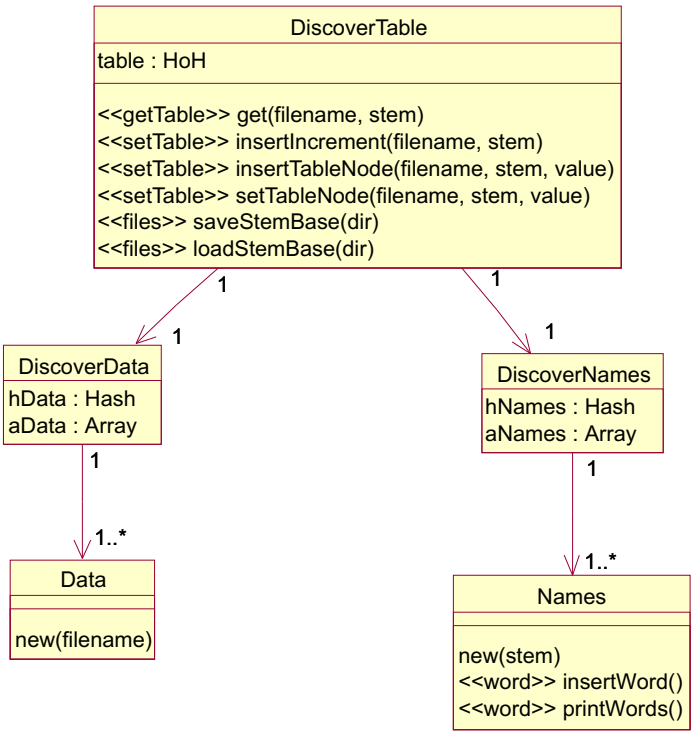

Figura 18: Diagrama de classes em UML do DiscoverTable, DiscoverNames e DiscoverData

Para cada *gram* é criado uma instância do DiscoverTable, assim, pode-se ter até três instâncias, sendo uma para cada *gram*. No entanto, criando um DiscoverData para cada DiscoverTable, as tabelas estão replicando dados pois para 1, 2 e 3-*gram*, os arquivos documentos são os mesmos. A programação orientada a objeto permite a utilização da instância da classe DiscoverData em várias DiscoverTable. Portanto, após a criação do primeiro DiscoverTable, todos os demais utilizam a mesma instância do primeiro.

5.2 Descrição das Classes

O PreText é composto por várias classes que são responsáveis por manipular e acessar a classe DiscoverTable, bem como classes auxiliares. Todas elas são descritas a seguir.

Constants

No pacote Constants são definidas as principais constantes utilizadas pelo sistema. Desse modo, a alteração de qualquer constante nesse pacote implica na alteração da constante em todas as classes.

Message

O pacote Message é responsável pelas saídas de execução do sistema. Todas as mensagens que são mostradas para o usuário em tempo de execução são procedimentos desse pacote.

Parameters

A classe Parameters é responsável pela leitura e verificação de erros no arquivo de parâmetros (`parameters.cfg`). Quando a classe é carregada com êxito, todos os parâmetros podem ser acessado por métodos dessa classe.

ParGram

A classe ParGram contém os parâmetros, `max`, `min`, `measure`, `smooth` e `normalize`, para cada *gram* habilitado.

TextBase

A classe TextBase contém e manipula os nomes dos arquivos documentos que compõem a coleção. Desse modo, quando algum método precisa dos arquivos documentos disponíveis, essa classe acessa o diretório da base de documentos e retorna a informação requisitada.

DiscoverNames

A classe DiscoverNames, como já descrito, contém os atributos (*stems*) de toda a coleção de documentos.

DiscoverData

A classe DiscoverData, como já descrito, contém os nomes dos arquivos documentos.

DiscoverTable

A classe DiscoverTable é encarregada, entre outras coisas, de armazenar e manipular os dados da tabela atributo-valor.

Names

A classe Names representa um atributo (*stem*). Essa classe contém, além do *stem*, outros dados como o número de arquivos nos quais o *stem* apareceu, as palavras que se transformaram nesse *stem* e, quando necessário, qual o fator de ponderação associado ao *stem*.

Data

A classe Data representa o nome de um arquivo documento.

GraphicsTxt

Essa classe gera os dados necessário para a geração do gráfico do *rank* de *stems* a partir de uma DiscoverTable.

StemBase

A classe StemBase é responsável pela transformação da base de documentos em base de *stems*. Durante a sua execução são invocados alguns métodos para essa transformação, descritos a seguir:

StemText

Essa classe é responsável por transformar a TextBase em DiscoverTable.

StopList

A classe StopList é encarregada de carregar em memória todas as *stopfiles* que estiverem com extensão `.enabled`. Depois de carregadas, os métodos de acesso podem verificar se uma palavra é ou não uma *stopword*.

StemAlg

A classe StemAlg transforma uma palavra em um *stem*.

A Figura 19 mostra o diagrama de classes em UML da StemBase. Note que a cardinalidade entre a StemBase e o DiscoverTable é 1..3, que representa os três *gram* que podem ser utilizados pelo sistema.

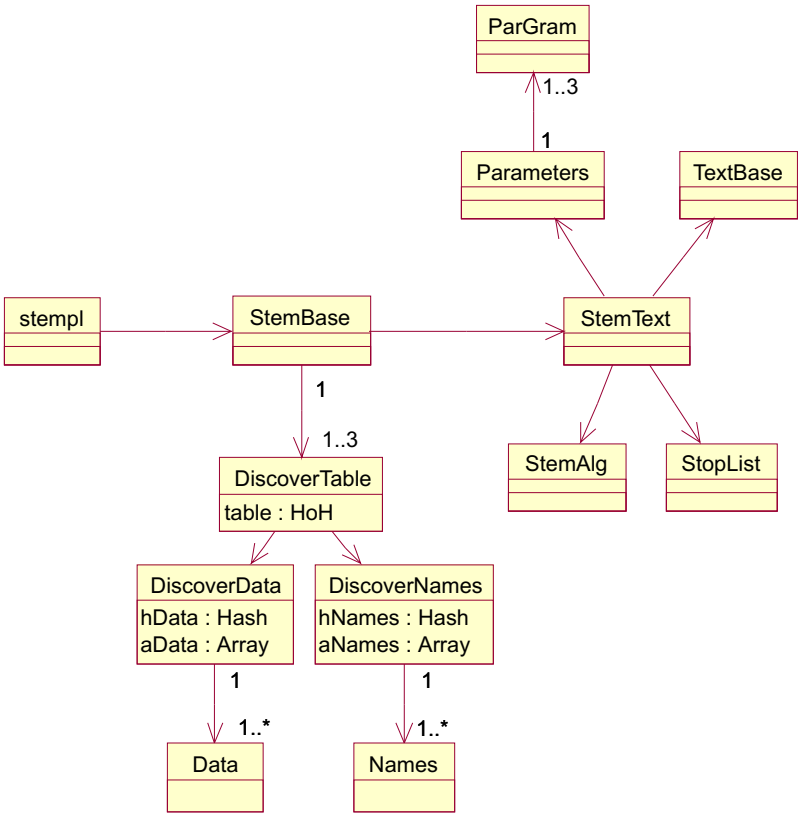

Figura 19: Diagrama de classes em UML da StemBase

Report

A classe Report é responsável pela transformação de uma **stembase** em uma tabela na sintaxe padrão do sistema DISCOVER. Para essa transformação são utilizadas as seguintes classes:

Taxonomy

A classe Taxonomy é responsável pela indução construtiva.

Cuts

A classe Cuts é responsável pelo corte automático calculado a partir do desvio padrão do *rank* de *stems*.

Measure

A classe Measure transforma a DiscoverTable na medida definida pelo arquivo de parâmetros. Tanto as medidas quanto as normalizações e o *smooth* são feitas por essa classe.

A Figura 20 mostra o diagrama de classes em UML do Report. Com esse modelo, depois de devidamente alocada em memória, a DiscoverTable passa por transformações, de acordo com os parâmetros definidos. Desse modo, a DiscoverTable não é replicada mas transformada. Assim, os parâmetros habilitam ou desabilitam as transformações que podem ser os cortes, medidas, *smooth* ou normalizações, possibilitando combinar todas elas.

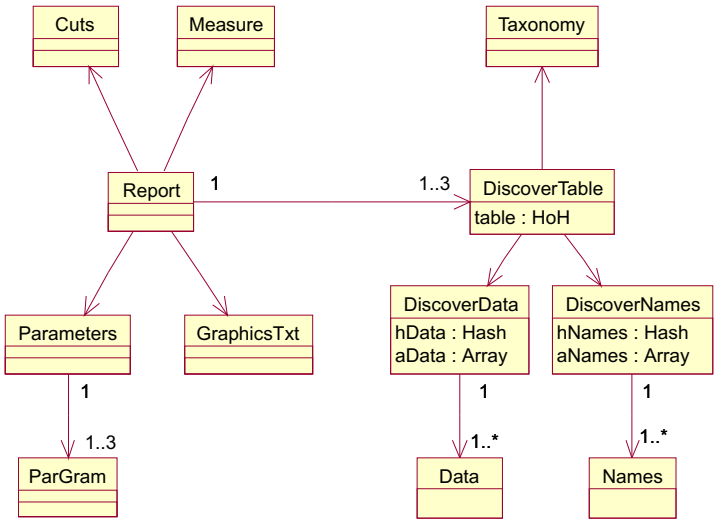

Figura 20: Diagrama de classes em UML do Report

Predict

A classe Predict transforma uma base de documento em uma tabela na sintaxe padrão do sistema DISCOVER.

6 Considerações Finais

O bom resultado em um processo de mineração usando algoritmos de aprendizado está diretamente relacionado não somente a quantidade mas, principalmente, a qualidade dos dados que descrevem uma situação do mundo real. Em um processo de Mineração de Textos, no qual os dados estão em um formato não estruturado textual, é necessário que esses dados sejam pré-processados de tal forma que possam ser submetidos a algoritmos de aprendizado. Neste trabalho foi descrita a ferramenta computacional PreTexT que realiza o pré-processamento de dados textuais de forma interativa e iterativa.

PreTexT é uma ferramenta desenvolvida usando o paradigma de orientação a objetos. Dessa forma, apesar das diversas facilidades e funcionalidades já implementadas, a ferramenta está preparada para que outras funcionalidades possam facilmente ser incorporadas no futuro. As características atuais da ferramenta incluem: a transformação de documentos, usando a abordagem *bag-of-words*, em um formato estruturado de uma tabela atributo-valor, para documentos escritos em português, espanhol e inglês; a atribuição de valores aos atributos usando diversas medidas propostas na literatura e outras medidas propostas neste trabalho; a redução da dimensionalidade dos atributos; o uso de taxonomias por meio de indução construtiva; a geração de vários arquivos auxiliares com informações úteis ao usuário; a geração de arquivos para criar gráficos relacionados a freqüência dos *stems*; entre outros. Essas facilidades, bem como a grande quantidade de informações geradas por PreTexT, são o diferencial dessa ferramenta em relação a outras ferramentas existentes.

A saída de PreTexT é um arquivo contendo os documentos representados na forma de uma tabela atributo-valor e um arquivo contendo informações sobre os atributos. Esses arquivos estão na sintaxe padrão utilizada pelo sistema Discover e podem ser convertidos automaticamente para a sintaxe de dados de mais de dezesseis algoritmos de aprendizado usando o ambiente computacional discover learning environment - dle (Batista & Monard, 2003), no qual PreTexT será futuramente integrado. Como mencionado, outros recursos podem ser facilmente implementados na ferramenta PreTexT, os quais deverão surgir naturalmente mediante o uso e a interação com pesquisadores de áreas correlatas à Tecnologia da Informação.

A ferramenta foi utilizada para realizar diversos experimentos utilizando diferentes coleções de textos. Os resultados desses trabalhos podem ser encontrados em (Martins, Monard, & Matsubara, 2003a,c,b; Martins, Godoy, Monard, Matsubara, & Amandi, 2003).

A Instalação

O PreTexT foi implementado para ser usado tanto em ambiente Unix quanto Windows. O requisito para a instalação e execução do PreTexT é a linguagem de programação Perl instalado na máquina. Normalmente não é necessário a instalação de pacotes adicionais para a execução dos dois principais módulos.

Com o **Perl** instalado na máquina, os passos para instalação do PreTexT são:

Unix

1. perl Makefile.pl
2. make
3. make install (precisa ser root)

Windows

1. perl Makefile.pl
2. nmake[7]
3. nmake install (precisa ser Administrador)

Instalação sem Administrador ou root

Para essa instalação não é preciso seguir os passos descritos anteriormente. Basta descompactar o PreTexT e alterar a linha "`use lib  '/caminho/do/pretex';`" dos arquivos **stem.pl** e o **report.pl** que estão dentro do subdiretório `pretex/local` como exemplificado a seguir:

```
use lib "C:/Documents and Settings/edsontm.LABIC/Desktop/pretex";

require pretex::StemBase;
...
```

[7] nmake está disponível em http://download.microsoft.com/download/vc15/Patch/1.52/W95/EN-US/Nmake15.exe.

É importante observar que mesmo para o ambiente Windows a barra "/" segue o padrão de sistemas Unix, ou seja, nunca se deve utilizar o "\", mas sempre o "/". O caminho definido deve ser absoluto, ou seja, no Windows sempre começando "`C:/`" e no Unix sempre começando com " `/`".

Após alteração dos arquivos basta inserir o caminho do `pretex/local` na variável *path* do sistema.

B Execução Rápida

Para que o usuário tenha o primeiro contato com o PreTexT é recomendável que ele execute os passos a seguir:

1. criar um arquivo `parameters.cfg` com o seguinte conteúdo:

```
%global%
%one%
%two%
%three%
```

2. criar uma pasta com o nome `stoplist` e inserir os arquivos `stopWdPt.enabled` e `irregulares.enabled`, que contém algumas *stopwords* para a lingua portuguesa e os verbos irregulares, respectivamente, que acompanham o pacote de instalação;
3. criar uma pasta com o nome **textbase** e inserir os textos que o usuário desejar;
4. executar o **stem.pl**;
5. executar o **report.pl**.

Após a execução são criados dois arquivos, o arquivo de dados (`.data`) e o arquivo de atributos (`.names`) na sintaxe padrão do sistema Discover. Note que esses dados não estão rotulados. Para gerar dados rotulados ver o item **textbase** na Seção 4.2.1 na página 21.

C Algoritmos de *Stemming*

Algoritmos de *stemming* consistem em uma normalização lingüística, na qual as formas variantes de uma palavra são reduzidas a uma forma comum tendo como conseqüência a

diminuição do número de atributos. Muitas vezes, o algoritmo remove prefixos ou sufixos de uma palavra, ou mesmo transforma um verbo para sua forma no infinitivo. O algoritmo de *stemming* é, portanto, fortemente dependente da língua na qual os textos estão escritos. Para determinadas línguas, como as orientais, pode ser impossível ou extremamente complexa a aplicação de um algoritmo de *stemming*. Na ferramenta PreTexT foram implementados os algoritmos de *stemming* para textos escritos na língua inglesa, portuguesa, espanhola e usando o algoritmo do Porter, cujas regras são descritas a seguir.

C.1 Regras do Algoritmo de *Stemming* para o Inglês

O algoritmo de *stemming* implementado neste trabalho para a língua inglesa é basicamente o algoritmo e a implementação original do Porter que se encontra em http://www.tartarus.org/~martin/PorterStemmer. Para um melhor entendimento do algoritmo, é descrito seu funcionamento como mostrado em (Porter, 1980). As regras foram definidas usando uma medida que determina quando uma palavra é suficientemente grande para poder eliminar o sufixo. Essa medida, denotada por m, é uma função das seqüências de vogais (a, e, i, o, u ou y se for precedida por uma consoante) seguidas por seqüências de consoantes. Se V é uma seqüência de vogais e C é uma seqüência de consoantes, então m é:

$$C(VC)^m V$$

onde C e V nos extremos são opcionais e m é o número de vezes que uma seqüência VC é repetida dentro da palavra. Por exemplo:

$m = 0$ free, why
$m = 1$ frees, whose
$m = 2$ prologue, compute

As regras para eliminar os sufixos são descritos da seguinte forma:

$$[condição]\ S_1 \rightarrow S_2$$

o que significa que se uma palavra termina com o sufixo S_1 e o *stem* que se encontra antes de S_1 satisfaz a condição estipulada, S_1 será trocada por S_2. A [*condição*], que é opcional, usualmente é dada por m, por exemplo:

$$(m > 1)EMENT \rightarrow$$

indica que se for verdade que $m > 1$ o sufixo EMENT é eliminado.

A [*condição*] pode ser formada por outras condições concatenadas com os operadores lógicos AND, OR ou NOT, e pode incluir algum dos seguintes elementos:

$*<X>$ o *stem* termina com a letra indicada por X
$*v*$ o *stem* contém pelo menos uma vogal
$*d$ o *stem* termina com consoante dupla
$*o$ o *stem* termina com consoante-vogal-consoante e a última consoante não é w, x ou y.

A Tabela 4 apresenta as regras de eliminação de sufixos previstas pelo algoritmo na notação descrita.

Nº	Condição	Ação			Exemplo		
1a		sses	→	es	caresses	→	caress
		ies	→	i	ponies	→	poni
					ties	→	ti
		ss	→	ss	caress	→	caress
		s	→		cats	→	cat
1b	$m > 0$	eed	→	ee	feed	→	feed
					agreed	→	agree
1b(a)	$*v*$	ed	→		plastered	→	plaster
					bled	→	bled
1b(b)	$*v*$	ing	→		motoring	→	motor
					sing	→	sing
	se(1b(a) OR 1b(b))	at	→	ate	conflat(ed)	→	conflate
		bl	→	ble	troubl(ed)	→	trouble
		iz	→	ize	siz(ed)	→	size
	se(1b(a) OR 1b(b))	at	→	single	hopp(ing)	→	hop
	AND ($*d$ AND NOT		→	letter	tann(ed)	→	tan
	($*l$ OR $*s$ OR $*Z$))				fall(ing)	→	fall
					hiss(ing)	→	hiss
					fizz(ed)	→	fizz
	se(1b(b) AND		→	e	fail(ing)	→	fail
	($m = 1$ AND $*O$))				fil(ed)	→	file
1c	$*v*$	y	→	i	happy	→	happi
					sky	→	sky
2	$m > 0$	ational	→	ate	relational	→	relate
	$m > 0$	tional	→	tion	conditional	→	condition
					rational	→	rational

continua na próxima página

continuação da página anterior

Nº	Condição	Ação			Exemplo		
	$m > 0$	enci	→	ence	valenci	→	valence
	$m > 0$	anci	→	ance	hesitanci	→	hesitance
	$m > 0$	izer	→	ize	digitizer	→	digitize
	$m > 0$	abli	→	able	conformabli	→	conformable
	$m > 0$	alli	→	al	radicalli	→	radical
	$m > 0$	entli	→	ent	differentli	→	diferent
	$m > 0$	eli	→	e	vileli	→	vile
	$m > 0$	ousli	→	ous	analogousli	→	analogous
	$m > 0$	ization	→	ize	vietnamization	→	vietnamize
	$m > 0$	ation	→	ate	predication	→	predicate
	$m > 0$	ator	→	ate	operator	→	operate
	$m > 0$	alism	→	al	feudalism	→	feudal
	$m > 0$	iveness	→	ive	decisiveness	→	decisive
	$m > 0$	fulness	→	ful	hopefulness	→	hopeful
	$m > 0$	ousness	→	ous	callousness	→	callous
	$m > 0$	aliti	→	al	formaliti	→	formal
	$m > 0$	iviti	→	ive	sensitiviti	→	sensitive
	$m > 0$	biliti	→	ble	sensibiliti	→	sensible
3	$m > 0$	icate	→	ic	triplicate	→	triplic
	$m > 0$	ative	→		formative	→	form
	$m > 0$	alize	→	al	formalize	→	formal
	$m > 0$	iciti	→	ic	electriciti	→	electric
	$m > 0$	ical	→	ic	electrical	→	electric
	$m > 0$	ful	→		hopeful	→	hope
	$m > 0$	ness	→		goodness	→	good
4	$m > 1$	al	→		revival	→	reviv
	$m > 1$	ance	→		allowance	→	allow
	$m > 1$	ence	→		inference	→	infer
	$m > 1$	er	→		airliner	→	airlin
	$m > 1$	ic	→		gyroscopic	→	gyroscop
	$m > 1$	able	→		adjustable	→	adjust
	$m > 1$	ible	→		defensible	→	defens
	$m > 1$	ant	→		irritant	→	irrit
	$m > 1$	ement	→		replacement	→	replac
	$m > 1$	ment	→		adjustament	→	adjust
	$m > 1$	ent	→		dependent	→	depend
	$m > 1$ AND ($*s$ OR $*T$)	ion	→		adoption	→	adopt
	$m > 1$	ou	→		homologou	→	homolog
	$m > 1$	ism	→		comunism	→	comun
	$m > 1$	ate	→		activate	→	activ
	$m > 1$	iti	→		angulariti	→	angular

continua na próxima página

continuação da página anterior

Nº	Condição	Ação			Exemplo		
	$m > 1$	ous	→		homologous	→	homolog
	$m > 1$	ive	→		effective	→	effect
	$m > 1$	ize	→		bowdlerize	→	bowdler
5a	$m > 1$	e	→		probate	→	probat
			→		rate	→	rate
	$m = 1$ AND NOT $*o$	e	→		cease	→	ceas
5b	$m = 1$ AND $*d$ AND $*l$		→	single	controll	→	control
				letter	roll	→	rol

Tabela 4: Regras de eliminação de sufixos para o inglês

Como mencionado, no PreTexT foram implementadas essas regras no algoritmo de *stemming* para o inglês. O algoritmo consiste basicamente em eliminar as palavras que são *stopwords* e aplicar as regras descritas em cada palavra que aparece nos documentos.

C.2 Regras do Algoritmo de *Stemming* para o Português

O algoritmo de *stemming* implementado para a língua portuguesa está baseado no algoritmo do Porter. Também, nesse algoritmo, os sufixos que estão ao final de uma palavra são eliminados, de acordo com um comprimento mínimo estabelecido e considerando algumas regras pré-estabelecidas.

Para estabelecer o comprimento mínimo são utilizadas duas variáveis, posV e pos2, que indicam duas posições críticas dentro da palavra. Ambas variáveis são complementares e possuem a mesma função de m, no algoritmo do Porter para inglês, isto é, determinar se a palavra é suficientemente grande para eliminar um sufixo. posV é utilizado para eliminar terminações verbais, enquanto que pos2 para eliminar outras formas de sufixos. Exemplos do uso dessas variáveis são mostrados na Figura 21, onde V e 2, são respectivamente, posV e pos2 e são determinadas da seguinte forma:

- pos2 está no fim de uma seqüência de consoantes que segue a segunda seqüência de vogais;
- se a palavra começa com duas ou mais vogais, posV está na primeira consoante;
- se começa com vogal-consoante, posV está na segunda vogal;

- se começa com consoante, posV está na primeira vogal ou na terceira letra que está mais a direita.

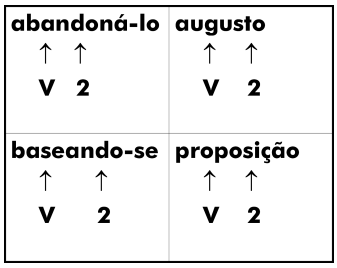

Figura 21: Medidas de comprimento mínimo da palavra

Após determinar essas duas posições, posV e pos2, aplicam-se as regras descritas na Tabela 5. As funções depois(variável) presente em algumas das regras denota que a regra só pode ser aplicada se o sufixo encontrar-se depois da posição indicada em variável.

Uma das grandes diferenças entre o algoritmo de *stemming* para palavras em inglês e palavras em português ou espanhol está no fato de que caso não seja possível eliminar nenhum sufixo de acordo com estas regras, são analisadas as terminações verbais da palavra. O fato das linguagens provenientes do latim terem formas verbais altamente conjugadas com sete tempos, cada uma com seis finais diferentes, é necessário um tratamento diferenciado para essas terminações. Para a língua portuguesa as terminações verbais são mostradas na Tabela 6.

O algoritmo implementado no PreTexT para o português manipula essas características e cada palavra, após ser submetida ao algoritmo de *stemming*, será denominada *stem*. Os passos utilizados no algoritmo podem ser descritos da seguinte forma:

Passo 1: Eliminam-se as palavras que são *stopwords* e alguns verbos irregulares, tais como **dar, diz, est, faz, hav, ir, pod, sab, ser, ter, ver, vir**. Para cada palavra verifica se a mesma não pertence a uma lista de *stopwords* da língua portuguesa. Caso a palavra pertencer à lista de *stopwords*, a mesma é ignorada e não será considerada como *stem*, ou seja, não será um atributo relevante. Além das *stopwords*, os verbos irregulares mais comuns também são eliminados os quais de certa forma, podem ser considerados como *stopwords*.

Passo 2: Estabelecem-se os valores de das variáveis posV e pos2 indicando duas posições críticas dentro da palavra. Como mencionado, é importante manter um tamanho mínimo que um *stem* deve ter. Caso contrário, palavras muito pequenas podem ser reduzidas a um *stem* que não seja tão interessante quanto a palavra tal como aparece no texto. Por exemplo, a palavra **ação** poderia ser reduzida ao *stem* **a**;

Nº	Condição	Ação			Exemplo		
1		[ae]is	→	[ae]l	casais	→	casal
		ns	→	m	álbuns	→	álbum
		res	→	r	fatores	→	fator
		s	→		casas	→	casa
2	depois(pos2)	idade	→		adversidade	→	advers
3	se(2) e depois(pos2)	abil	→		amigabil(idade)	→	amig
		iv	→		exclusiv(idade)	→	exclus
		ic	→		infelic(idade)	→	infel
4	depois(pos2)	ic[ao]	→		irônic[ao]	→	irôn
5	depois(pos2)	ável	→		lamentável	→	lament
		ível	→		acessível	→	acess
6	depois(pos2)	ismo	→		alcoolismo	→	alcool
7	se(1) ou depois(pos2)	ções	→		acomodação(s)	→	acomoda
		ção	→		resolução	→	resolu
8	se(1) ou depois(pos2)	ador	→		colonizador(es)	→	coloniz
		adora	→		colonizador(as)	→	coloniz
9	depois(pos2)	os[ao]	→		corajos[ao]	→	coraj
10	depois(pos2)	ista	→		dermatologista	→	dermatolog
11	depois(pos2)	amento	→		pensamento	→	pens
		imento	→		discernimento	→	discern
		amente	→		discretamente	→	discret
12	se(11) ou depois(pos2)	os	→		generos(amente)	→	gener
		ativ	→		comparativ(amente)	→	compar
		ad	→		deliberad(amente)	→	deliber
		iv	→		compulsiv(amente)	→	compuls
		ic	→		demagogic(amente)	→	demagog
13	depois(pos2)	mente	→		dificilmente	→	dificil
14	se(13) ou depois(pos2)	avel	→		favoravel(mente)	→	favor
		ivel	→		possivel(mente)	→	poss
15	depois(pos2)	iv[ao]	→		primitiv[ao]	→	primit
16	se(15) ou depois(pos2)	at	→		recreat(ivo)	→	recre
17	depois(pos2)	eza	→		sutileza	→	sutil

Tabela 5: Regras de eliminação de sufixos para o português

Passo 3: Eliminam-se as terminações: -los, -las, -se e depois cada a palavra é submetida às regras estabelecidas;

Passo 4: Aplicam-se as regras de eliminação de sufixos da Tabela 5, desde que se cumpra o **Passo 2**;

Passo 5: Se não foi eliminado nenhum sufixo no passo anterior, são consideradas as terminações verbais dos verbos regulares que são apresentadas na Tabela 6. Para eliminar quaisquer desses sufixos, deve ser verdade depois(posV).

Com relação à implementação do algoritmo do Porter em particular e os seus resultados, vale ressaltar algumas observações, tais como o *stem* retornado para a palavra **pensamento**, por exemplo, será **pensam** e não **pens**, por causa do pos2. Assim como para a

1a Conjugação						
Infinitivo	**-ar**					
	1a PS	2a PS	3a PS	1a PP	2a PP	3a PP
Presente	-o	-as	-a	-amos	-ais	-am
Subjuntivo	-e	-es	-e	-emos	-eis	-em
Futuro	-arei	-arás	-ará	-aremos	-areis	-arão
Condicional	-aria	-arias	-aria	-aríamos	-aríeis	-ariam
Imperfeito	-ava	-avas	-ava	-ávamos	-áveis	-avam
Passado	-ei	-aste	-ou	-ávamos	-astes	-aram
Imp. Subjuntivo	-asse	-asses	-asse	-ássemos	-ásseis	-assem
Mais Perfeito	-ara	-aras	-ara	-áramos	-áreis	-aram
Presente Partic.	-ando					
Passado Partic.	-ada -ado -adas -ados					

2a Conjugação						
Infinitivo	**-er**					
	1a PS	2a PS	3a PS	1a PP	2a PP	3a PP
Presente	-o	-es	-e	-emos	-eis	-em
Subjuntivo	-a	-as	-a	-amos	-ais	-am
Futuro	-erei	-erás	-erá	-eremos	-ereis	-erão
Condicional	-eria	-erias	-eria	-eríamos	-eríeis	-eriam
Imperfeito	-ia	-ias	-ia	-íamos	-íeis	-iam
Passado	-i	-este	-eu	-emos	-estes	-eram
Imp. Subjuntivo	-esse	-esses	-esse	-éssemos	-ésseis	-essem
Mais Perfeito	-era	-eras	-era	-éramos	-éreis	-eram
Presente Partic.	-endo					
Passado Partic.	-ida -ido -idas -idos					

3a Conjugação						
Infinitivo	**-ir**					
	1a PS	2a PS	3a PS	1a PP	2a PP	3a PP
Presente	-o	-es	-e	-imos	-is	-em
Subjuntivo	-a	-as	-a	-amos	-ais	-am
Futuro	-irei	-irás	-irá	-iremos	-ireis	-irão
Condicional	-iria	-irias	-iria	-iríamos	-iríeis	-iriam
Imperfeito	-ia	-ias	-ia	-íamos	-íeis	-iam
Passado	-i	-iste	-iu	-imos	-istes	-eram
Imp. Subjuntivo	-isse	-isses	-isse	-íssemos	-ísseis	-issem
Mais Perfeito	-ira	-iras	-ira	-íramos	-íreis	-iram
Presente Partic.	-indo					
Passado Partic.	-ida -ido -idas -idos					

Tabela 6: Terminações verbais do português

palavra **beleza** o *stem* será **beleza** e não **bel**. Alguns verbos de tamanho muito pequeno, não são reduzidos, por exemplo, **abrir** fica **abrir**, **abrem** fica **abrem**, **abriu** fica **abriu**, devido à restrição de posV. Também, o algoritmo trabalha com palavras acentuadas normalmente, porém, quando uma palavra é reduzida a seu *stem*, todas as letras acentuadas foram transformadas em letras sem acentuação. Isto porque se pode gerar *stems* diferentes para palavras que deveriam ter o mesmo *stem*, como por exemplo, o *stem* da palavra **academia** ficaria **acad** e da palavra **acadêmicos** ficaria **acadêm**. As letras maiúsculas

são transformadas em minúsculas. Os números foram eliminados bem como palavras de apenas um dígito.

C.3 Regras do Algoritmo de *Stemming* para o Espanhol

O algoritmo de *stemming* para o espanhol é semelhante ao algoritmo empregado para o idioma em português, ou seja, os sufixos que estão no final de um *stem* com um comprimento mínimo estabelecido são eliminados considerando algumas regras pré-estabelecidas (Godoy, 2001). As regras utilizadas para a língua espanhola são mostradas nas Tabelas 7 na próxima página e 8 na página 50 e os passos utilizados pelo algoritmo para textos em espanhol implementado no PreTexT são os mesmos utilizados para os textos em português, Seção C.2 na página 44.

No	Condição	Ação			Exemplo		
1		s	→		absurdas	→	absurda
2	depois(pos2)	idad	→		actividad	→	activ
3	depois(pos2)	idade	→		actividade(s)	→	activ
4	Se (2 ou 3) e	abil	→		irresponsabil(idad)	→	irrespons
	depois(pos2)	iv	→		radioactiv(idad)	→	radioact
		ic	→		complic(idad)	→	compl
5	depois(pos2)	ic[ao]	→		ecológic[ao]	→	ecológ
6	depois(pos2)	logía	→	log	analogía	→	analog
7	depois(pos2)	able	→		aceptable	→	acept
8	depois(pos2)	ible	→		apacible	→	apac
9	depois(pos2)	ismo	→		atletismo	→	atlet
10	depois(pos2)	ación	→		coronación	→	coron
11	depois(pos2)	ución	→	u	atribución	→	atribuc
12	depois(pos2)	acione	→		alteracione(s)	→	alter
13	depois(pos2)	ucione	→	u	atribucione(s)	→	atrib
14	depois(pos2)	ador	→		acusador	→	acus
15	depois(pos2)	ador[ae]	→		innovador	→	innov
16	depois(pos2)	os[ao]	→		ingenioso	→	ingeni
17	depois(pos2)	ista	→		maquinista	→	maquin
18	depois(pos2)	amiento	→		mejoramiento	→	mejor
		imiento	→		ofrecimiento	→	ofrec
19	depois(pos2)	amente	→		oportunamente	→	oportun
20	se(19) e depois(posV)	os	→		orgullos(amente)	→	orgull
		ativ	→		significativ(amente)	→	signific
		ad	→		acertad(amente)	→	acert
		ic	→		demagogic(amente)	→	demagog
		iv	→		figurativ(amente)	→	figurat
21	depois(pos2)	mente	→		finalmente	→	final
22	se(21) e depois(pos2)	able	→		inevitable(mente)	→	inevit
		ible	→		irremisible(mente)	→	irremis
23	depois(pos2)	iv[ao]	→		obsesiva	→	obses
24	se(23) e depois(pos2)	at	→		legislat(ivo)	→	legisl
25	depois(pos2)	anza	→		confianza	→	confi
26	se(10 ou 12 ou 14) e depois(pos2)	ic	→		calific(adora)	→	calif

Tabela 7: Regras de eliminação de sufixos para o espanhol

1a Conjugação						
Infinitivo	**-ar**					
	1a PS	2a PS	3a PS	1a PP	2a PP	3a PP
Presente	-o	-as	-a	-amos	-áis	-an
Subjuntivo	-e	-es	-e	-emos	-éis	-en
Futuro	-aré	-arás	-ará	-aremos	-aréis	-arán
Condicional	-aría	-arías	-aría	-aríamos	-aríais	-arían
Imperfeito	-aba	-abas	-aba	-ábamos	-abais	-aban
Passado	-é	-aste	-ó	-amos	-asteis	-aron
Imp. Subjuntivo	-ase	-ases	-ase	-ásemos	-aseis	-asen
	-ara	-aras	-ara	-áramos	-árais	-aran
Presente Partic.	-ando					
Passado Partic.	-ada -ado -adas -ados					
Imperativo	-ad					

2a Conjugação						
Infinitivo	**-er**					
	1a PS	2a PS	3a PS	1a PP	2a PP	3a PP
Presente	-o	-es	-e	-emos	-éis	-enm
Subjuntivo	-a	-as	-a	-amos	-áis	-an
Futuro	-eré	-erás	-erá	-eremos	-eréis	-erán
Condicional	-ería	-erías	-ería	-eríamos	-eríais	-erían
Imperfeito	-ía	-ías	-ía	-íamos	-íais	-ían
Passado	-í	-iste	-ió	-imos	-isteis	-ieron
Imp. Subjuntivo	-iese	-ieses	-iese	-iésemos	-ieseis	-iesen
	-iera	-ieras	-iera	-iéramos	-ierais	-ieran
Presente Partic.	-iendo					
Passado Partic.	-ida -ido -idas -idos					
Imperativo	-ed					

3a Conjugação						
Infinitivo	**-ir**					
	1a PS	2a PS	3a PS	1a PP	2a PP	3a PP
Presente	-o	-es	-e	-imo	-ís	-en
Subjuntivo	-a	-as	-a	-amos	-áis	-an
Futuro	-iré	-irás	-irá	-iremos	-iréis	-irán
Condicional	-iría	-irías	-iría	-iríamos	-iríais	-irían
Imperfeito	-ía	-ías	-ía	-íamos	-íais	-ían
Passado	-í	-iste	-ió	-imos	-isteis	-ieron
Imp. Subjuntivo	-iese	-ieses	-iese	-iésemos	-ieseis	-iesen
	-iera	-ieras	-iera	-iéramos	-ierais	-ieran
Presente Partic.	-iendo					
Passado Partic.	-ida -ido -idas -idos					
Imperativo	-id					

Tabela 8: Terminações verbais do espanhol

Referências

Apté, C., F. Damerau, & S. M. Weiss (1994). Automated learning of decision rules for text categorization. *Information Systems 12*(3), 233–251. http://citeseer.nj.nec.com/apte94automated.html. 3

Banerjee, S. & T. Pedersen (2003, February). The design, implementation and use of the ngram statistics package. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 370–381. 1

Baranauskas, J. A. & G. E. A. P. A. Batista (2000). O projeto DISCOVER: Idéias iniciais (comunicação pessoal). 1

Batista, G. E. A. P. A. & M. C. Monard (2003). Descrição da arquitetura e do projeto do ambiente computacional DISCOVER LERNING ENVIRONMENT - DLE. Relatório Técnico 187, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_187.zip. 2, 38

Batista, G. E. A. P. A. (2003). Pré-processamento de dados em aprendizado de máquina supervisionado. Tese de Doutorado, ICMC-USP. http://www.icmc.usp.br/~gbatista/pdfs/TeseDoutorado.pdf. 16

Crawford, D. (1998). Gnuplot: An interactive plotting program. http://www.ucc.ie/gnuplot/gnuplot.html. 28

Dosualdo, D. G. (2003). Investigação de regressão no processo de mineração de dados. Dissertação de Mestrado, ICMC-USP. 2

Dumais, S., J. Platt, D. Heckerman, & M. Sahami (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings of the 7th International Conference on Information and Knowledge Management (CIKM 98)*. http://research.microsoft.com/~sdumais/cikm98.pdf. 3

Gelbukh, A. & G. Sidorov (2001). Zipf and heaps laws coefficients depend on language. In *Proceedings CICLing2001, Conference on Intelligent Text Processing and Computational Linguistics, Lecture Notes in Computer Science*, Number 2004. http://www.gelbukh.com/CV/Publications/2001/CICLing-2001-Zipf.htm. 13

Godoy, D. L. (2001). Personalsearcher: Un agente inteligente para búsqueda de información en la www. Master Thesis, ISISTAN, Argentina. 48

Imamura, C. Y. (2001, setembro). Pré-processamento para extração de conhecimento de bases textuais. Dissertação de Mestrado, ICMC-USP. 13

Knuth, D. E. (1973). *The Art of Computer Programming*, Volume 3. Addison-Wesley. 13

Kopla, H. & P. Daly (1999). *A Guide to LaTeX: Document Preparation for Beginners and Advanced Users* (3 ed.). Addison-Wesley. 28

Lewis, D. D. (1992, June). An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 37–50. 3

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research and Development 2*(2), 159–165. 13

Martins, C. A., D. Godoy, M. C. Monard, E. T. Matsubara, & A. Amandi (2003, julho). Uma experiência em mineração de textos utilizando clustering probabilístico e clustering conceitual. Relatório Técnico 205, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_205.zip. 38

Martins, C. A., M. C. Monard, & G. C. Halembeck (2002). A computational framework for interpreting clusters through inductive learning. Relatório Técnico 173, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_173.zip. 2

Martins, C. A., M. C. Monard, & E. T. Matsubara (2003a). Reducing the dimensionality of bag-of-words text representation used by learning algorithms. In *Proceedings of The Third IASTED International Conference on Artificial Intelligence and Applications (AIA 2003)*, Benalmádena, Espanha. (to be published). 38

Martins, C. A., M. C. Monard, & E. T. Matsubara (2003b). Uma ferramenta computacional para auxiliar no pré pré-processamento de textos. In *IV Encontro Nacional de Inteligência Artificial - SBC*. CD-ROM, 6 pgs. (short-paper). 38

Martins, C. A., M. C. Monard, & E. T. Matsubara (2003c). Uma metodologia para auxiliar na seleção de atributos relevantes usados por algoritmos de aprendizado no processo de classificação de textos. In *XXIX Conferencia LatinoAmericana de Informatica - CLEI*, La Paz, Bolívia. (to be published). 38

McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/~mccallum/bow. 1

Melanda, E. A. (2002). Pós-processamento de conhecimento de regras de associação. Qualificação de Doutorado, ICMC-USP. 2

Porter, M. (1980). An algorithm for suffix stripping. *Program 14*(3), 130–137. 12, 13, 41

Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001a). Extração de informações padronizadas para a avaliação de regras induzidas por algoritmos de aprendizado de máquina simbólico. Relatório Técnico 145, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip. 2

Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001b). Uma proposta de unificação da linguagem de representação de conceitos de algoritmos de aprendizado de máquina simbólicos. Technical Report 137, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip. 2

Prati, R. C. (2003, abril). O *framework* de integração do sistema DISCOVER. Dissertação de Mestrado, ICMC-USP. 16

Salton, G. & C. Buckley (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management 24*(5), 513–523. 5

Sebastiani, F. (2002, March). Machine learning in automated text categorisation. *ACM Computing Surveys 34*(1), 1–47. http://faure.iei.pi.cnr.it/~fabrizio/Publications/ACMCS02.pdf. 3

Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow. http://citeseer.nj.nec.com/vanrijsbergen79information.html. 14

Wall, L., T. Christiansen, & R. L. Schwartz (1996). *Programming in PERL*. O'Reilly,Inc. 1, 14

Zipf, G. (1949). *Human Behaviour and the Principle of Least Effort*. Addison-Wesley. 13