Aprendizado de regras de substituição para normatização de textos históricos

Alexandre Hirohashi

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do grau
de
Mestre em Ciência da Computação

Área de Concentração : **Ciência da Computação**
Orientador : Prof. Dr. Marcelo Finger
São Paulo - Agosto - 2005

Aprendizado de regras de substituição para normatização de textos históricos

Este exemplar corresponde à redação
final da dissertação devidamente
corrigida e defendida por
Alexandre Hirohashi
e aprovada pela comissão julgadora.

São Paulo, Agosto de 2005.

Banca Examinadora:

- Prof. Dr. Marcelo Finger (Orientador) - IME-USP
- Profa. Dra. Maria das Graças Volpe Nunes - ICMC-USP
- Prof. Dr. Flávio Soares Corrêa da Silva - IME-USP

Agradecimentos

Antes de todos, quero agradecer aos meus pais que, além de me inspirarem, são os responsáveis por eu ser tudo que sou.

Ao meu irmão que sempre esteve presente (principalmente em casa), quando eu não pude.

À Carolina, minha namorada, pelo seu carinho e amor, além de me ajudar a ver oportunidades ao invés de dificuldades.

Ao Prof. Dr. Marcelo Finger pela sua paciência, compreensão, e por acreditar em mim.

À Prof. Dra. "Tia" Anilce Maria Simões que, apesar de muito ocupada, gentilmente revisou meu texto.

À família Dubeux-Kawamura, onde sempre tive apoio e carinho, lembrando-me sempre o que realmente é importante.

Finalmente, aos meus Amigos do IME e da Accenture que me ajudaram chegar até aqui.

Resumo

As ferramentas de lingüística computacional atuais geralmente são baseadas num léxico que contém apenas palavras normatizadas, ou seja, dentro da norma ortográfica vigente. No entanto, textos históricos possuem, muitas vezes, ortografias não-normatizadas.

Para viabilizar o uso de ferramentas de lingüística computacional em textos históricos, almejamos construir um normatizador automático do português.

Com esse fim, propomos neste estudo um mecanismo de transformação de palavras não normatizadas baseado em regras de substituição. Além disso, apresentamos um mecanismo que mede, indiretamente, a eficácia de regras para a sua seleção nem uma fase de treinamento.

Abstract

The current Computation Linguistic tools are usually based on a lexicon without spelling errors, which is also generally inside the present time orthographic standards. However, historical documents often use words of an older ortographic norm.

Therefore, our goal is to build an automatic spelling correction tool for Portuguese historical texts, so that current Computation Linguistic tools could be used. We present an approach that uses substituition-rules to transform words with spelling errors. In addition, we also present a method to indirectly measure the efficacy of these rules.

Conteúdo

Capítulo 1

Introdução

Neste estudo mostramos a implementação de uma ferramenta que aumenta o grau de normatização de textos automaticamente.

Veremos neste capítulo, um breve histórico sobre da lingüística computacional e a definição do problema de detecção/correção de desvios de ortografia.

1.1 Histórico

1.1.1 Lingüística Moderna

No início da década de 50, Noam Chomsky publicou uma série de livros e artigos em que criticou seriamente o estudo da linguagem baseado em *corpora*, ou seja, em amostras de textos naturais extraídos do uso cotidiano.

Chomsky argumentava que o "conhecimento interno" (consciência) da linguagem é mais significativo que a "manifestação externa" (uso), pois qualquer amostra de tal manifestação poderia ser afetada por fatores externos[1]. Portanto, para ele, os aspectos morfológicos, sintáticos e/ou léxicos de determinada língua eram menos importantes do que, por exemplo, a chamada *gramática universal* que se constitui das características comuns entre as línguas.

Outro grande argumento utilizado por Chomsky era sua alegação de que não seria possível estudar as características de uma língua a partir de uma amostra finita de textos, ou seja, de um conjunto finito de construções, uma vez que o número de construções possíveis é infinito e, se não fosse, grande parte de linguagem não poderia ser aprendida, por nos ser inata.

[1] Por exemplo: ignorância, esquecimento temporário, bebida, etc.

Finalmente, Chomsky também afirmava que muitos desses textos naturais seriam compostos em grande parte, por sentenças gramaticalmente incorretas, impossibilitando a sua utilização para estudos de sintaxe, pois isso poderia incorporar construções não-gramaticais nos estudos envolvidos.

Além disso, Abercromble (conforme [17]) citou a falta de recursos (na época) para manipulação de grande quantidade de dados, tornando-o manual. Isso consumia muito tempo e era pouco confiável.

Desse modo, durante duas décadas, os estudos lingüísticos passaram a ser baseados em conjuntos de textos artificiais criados a partir de julgamentos introspectivos de pesquisadores. Esse tipo de abordagem denomina-se *lingüística racional*, em oposição à *lingüística empírica*, que se baseia em corpora.

1.1.2 Lingüística Computacional Empírica

Com a evolução da lingüística, no entanto, alguns estudos se opuseram aos argumentos de Chomsky. Podemos citar, principalmente, os estudos nas áreas da fonética e da aquisição de linguagem, nos quais o uso de textos naturais tornou-se a principal fonte de evidências.

Comprovou-se que a utilização de amostras finitas de textos não é um problema, dado que, normalmente, as construções mais usadas são encontradas nesses textos, que são além disso a única fonte de freqüência/proporcionalidade de uso dessas construções (principalmente quando se quer estudar fenômenos lingüísticos do passado mais remoto, uma vez que, geralmente não encontrarmos "falantes" vivos).

Ao final da década de 60, William Labov [15] mostrou que a maioria das sentenças de textos naturais são gramaticalmente corretas, o que possibilita seu uso em estudos de sintaxe. Vale ressaltar, também, o fato de que, ao contrário de estudos introspectivos, a utilização de corpora possibilita a observação dos dados para comprovação desses estudos.

Nos últimos anos, com a evolução da capacidade de processamento e armazenamento dos computadores, os estudos baseados em corpora tornaram-se cada vez mais viáveis, dos quais podemos citar:

- Correção de erros de grafia, com base em um léxico [16];
- Tradução automática [18];
- Resolução de ambigüidade de sentenças [4];
- Realização de análise sintática [1] [2] [3] [20] [19];
- Reconhecimento de fala [18];

- Etiquetagem morfossintática de palavras de um texto [5] [6] [7] [24].

Recentemente surgiram ferramentas específicas para o processamento da Língua Portuguesa.

Um exemplo de interesse especial para esta dissertação são os estudos relizados pelo Núcleo Interinstitucional de Lingüística Computacional (NILC) que resultaram, por exemplo, em um revisor gramatical denominado *ReGra* [19], fruto de uma pareceria entre USP e Itautec-Philco, que utiliza regras baseadas em combinações lexicais e regras baseadas na análise sintática automática de textos. Tais regras são primeiramente modeladas manualmente, em seguida são verificadas/aperfeiçoadas utilizando um corpus em testes exaustivos para finalmente serem incorporados à ferramenta.

Atualmente esse revisor gramatical é parte integrante de processadores de textos como o Microsoft Word e o Redator da Itautec.

Vimos nesta seção um breve resumo da história da lingüística computacional, considerando a importância da utilização de corpora para o estudo de fenômenos lingüísticos.

1.2 Desvios de Ortografia

Dentre os principais problemas da lingüística computacional, estudaremos, em particular, a detecção e correção de desvios de ortografia. Mais especificamente, esse problema é chamado de normatização quando se trata exclusivamente de textos históricos.

Basicamente, um erro de ortografia consiste em uma instância de uma palavra escrita de maneira incorreta, ou seja, não adequada a uma determinada ortografia. Isso pode ocorrer por diversas razões. Vejam algumas:

- Erros de manuscrita;
- Problemas de aprendizado;
- Diferenças na ortografia vigente à época do texto, em relação à atual;
- Falta de norma ortográfica para determinadas palavras.

Essas razões motivaram o uso de algoritmos de detecção e correção de desvios de ortografia nos modernos editores de texto.

Tais algoritmos são usados, também, em aplicações em que a identificação individual de letras não é confiável, tais como :

- Reconhecimento Ótico de Caracteres (OCR): sistemas onde letras são reconhecidas automaticamente, a partir de uma imagem gerada por um *scanner* ótico;

- Reconhecimento em Tempo Real de Caracteres Manuscritos: sistemas onde letras são reconhecidas automaticamente durante a escrita, utilizando informações colhidas dinamicamente sobre a ordem, número, velocidade e direção de cada traço[2].

Além disto, a maioria das ferramentas de lingüística computacional (etiquetadores, *parsers*, "thesaurus", hifenadores etc.) só trabalham com textos normatizados, ou seja, com textos sem desvios de ortografia.

Um dos corretores ortográficos mais utilizados atualmente denomina-se *Ispell* [13] que procura encontrar palavras corretas usando o conceito de "*near miss*", onde são selecionadas palavras que podem se tornar idênticas à palavra incorreta através de uma das seguintes transformações :

- Inserção de um espaço em branco (`pormente` → `por mente`);
- Troca de duas letras adjacentes (`proque` → `porque`);
- Troca de uma letra (`minte` → `mente`);
- Remoção de uma letra (`trocra` → `troca`);
- Inclusão de uma letra (`csa` → `casa`).

Apesar de significativo, este conceito é insuficiente para corrigir textos não normatizados. Por exemplo as palavras :

- `leuaua`;
- `testauaõ`;
- `sospeytoso`;

Não são corrigidas por nenhuma das transformações propostas.

1.2.1 Padrões de Desvios de Ortografia

Em 1964, Demerau [9] concluiu que aproximadamente 80% dos desvios de ortografia que resultam em não-palavras[3] são causados pelo que ele denominou *erros simples de soletração*, dividindo-os em três classes:

- Inserção: trocar `são` por `sãos`;
- Remoção: trocar `são` por `sã`;
- Substituição: trocar `são` por `sãu`;
- Transposição: trocar `são` por `soã`.

[2] Esse tipo de sistema é muito utilizado atualmente nos ambientes portáteis de computação, tais como handhelds.

[3] Palavras desconhecidas ou não existentes.

Tal conclusão fez com que muitos dos estudos posteriores se concentrassem na correção de erros simples de soletração.

Mais tarde, Kukich [14] dividiu os desvios de ortografia em duas classes: erros relacionados ao teclado, denominados **erros tipográficos**, e erros causados por ignorância dos autores, denominados **erros cognitivos**.

Enquanto os erros tipográficos, geralmente, são erros de substituição, em que uma tecla adjacente, ou, ainda, a tecla homóloga[4] à correta é utilizada, os erros cognitivos correspondem, em sua maioria, a erros fonéticos, em que pedaços da palavra são substituídos por outros, equivalentes foneticamente.

Assim, sabemos, agora, o que é um erro de ortografia, suas causas e como classificá-lo. Como esse problema está bem definido, propomos, a seguir, uma ferramenta para a automatização de sua correção.

[4]Tecla correspondente ao lado oposto do teclado

Capítulo 2

Normatização de Textos

Conforme visto anteriormente, por normatizar um texto entende-se o processo de adequar cada uma de suas palavras a uma ortografia.

Neste capítulo, apresentaremos a proposta de construção de uma ferramenta (*normatizador*) que, por meio de um método de aprendizado automático, consiga aumentar o grau de normatização de textos.

2.1 Modelo Normatizador

O modelo a ser utilizado pelo normatizador é composto de duas fases distintas: a de aprendizado e a que constitui a normatização propriamente dita. Essa normatização consiste na aplicação de um conjunto ordenado de *regras de substituição de cadeia de caracteres* (Ver seção 2.1.1), que representa o "conhecimento" adquirido na fase de aprendizado.

Normalmente, sistemas baseados em aprendizado utilizam *coleções de referência*, isto é, soluções de uma amostra significativa do tipo de problema que se quer resolver.

Seria natural selecionar um conjunto finito de textos não-normatizados, proceder manualmente à sua normatização, e treinar o sistema com esses textos. Assim, por meio de uma análise comparativa, o treinamento criaria um "conhecimento" para a normatização de outros textos.

Estamos propondo, no entanto, um método *indireto* de medir a eficácia (Ver seção 2.1.2) de regras, para incorporá-las ou não ao "conhecimento" durante o treinamento, sem a necessidade de utilizar textos normatizados manualmente. Em primeiro lugar, conceituaremos o que é uma *regra de substituição de cadeia de caracteres*, e finalmente mostraremos o método propriamente dito que utiliza textos etiquetados morfossintaticamente por um etiquetador automático.

2.1.1 Regras de Substituição de Cadeia de Caracteres

Para corrigir a forma de uma palavra (ortografia), utilizaremos a heurística de *substituição de cadeia de caracteres*[1], ou seja, para normatizar uma palavra, iremos substituir seqüências de letras que compõem completamente ou parcialmente a palavra.

Essas regras realizam transformações ortográficas, visando a um aumento do grau de normatização de palavras não normatizadas.

Forma

As *Regras de Substituição de cadeia de caracteres* devem definir, de maneira completa, o *escopo no texto*, o *escopo na palavra* e a *string de substituição.*

Assim, propomos a seguinte estrutura para as *Regras de Substituição de Cadeia de Caracteres*:

[*Expressão regular*$_1$] [*Expressão regular*$_2$] [*String de Substituição*]

A primeira parte (*Expressão regular*$_1$) define quais palavras do texto deverão sofrer alguma manipulação, ou seja, o *escopo no texto.*

A segunda parte (*Expressão regular*$_2$) define qual o *fragmento* da palavra que deverá ser substituído, ou seja, o *escopo na palavra.*

Finalmente, a terceira parte (*String de Substituição*) define o novo fragmento a ser utilizado, com a substituição.

Uso

A *ordem* em que as regras de substituição são aplicadas reflete-se diretamente no resultado final.

Por exemplo, considere a palavra

`hauiaõ`

e as regras

a. `aõ aõ ão`

b. `hauião ão am`

c. `hauiam u v`

[1]Também denominadas *strings* : Seqüência de caracteres que são processados como uma unidade de informação

Se essas regras forem aplicadas em uma determinada ordem (`[a,b,c]`), o resultado será 'haviam', que é a forma mais moderna da palavra. De outro modo, o resultado final poderia ser 'hauiam' (`[c,a,b]`), ou, ainda, 'hauião' (`[c,b,a]`) que não correspondem à forma esperada.

Existem outras heurísticas para a normatização de palavras, dentre as quais podemos citar a aproximação fonética - a conversão das palavras não normatizadas em fonemas permite que se encontre a normatização, pela comparação da construção fonética das palavras de um léxico normatizado.

2.1.2 Eficácia de Uma Regra de Substituição

Veremos, adiante (Ver seção 2.2.1), que o *etiquetador* Tycho Brahe apresenta um *léxico de treinamento* composto, na sua maioria, por palavras na forma moderna, ou seja, por palavras normatizadas.

Com base nesse fato, propomos a utilização da variação da *precisão da etiquetagem* desse etiquetador em um *corpus referência*, para medir indiretamente a eficácia de uma *Regra de Substituição* na normatização de palavras.

Ou seja, como uma *regra de substituição* limita-se a alterar a forma das palavras, sem alterar a construção do texto (ordem/número das palavras), podemos medir a eficácia dessa regra em normatizar palavras por meio da diferença entre:

- a *precisão* da etiquetagem do *corpus referência* original, e
- a *precisão* da etiquetagem do *corpus referência* transformado pela *regra de substituição.*

Como veremos adiante (Ver seção 2.2.3), se essa diferença indicar que a *precisão* da etiquetagem aumentou após a aplicação de uma *regra de substituição*, então essa regra indiretamente *aumentou o grau de normatização* das palavras desconhecidas.

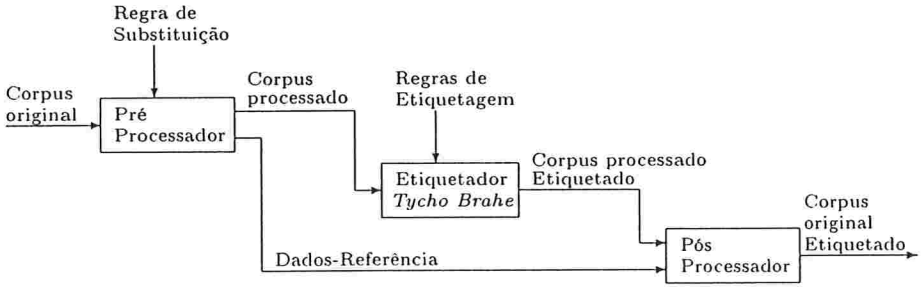

Figura 2.1: Processo de medição da eficácia de uma regra

Desse modo, o processo de medição da eficácia de uma regra consiste em (Veja Figura 2.1) :

1. um *pré-processamento*, que aplique a *regra de substituição*;

2. a etiquetagem propriamente dita;

3. um *pós-processamento*, que reverta as substituições, mantendo a etiquetagem gerada pelo etiquetador, recuperando o texto original, possibilitando uma comparação automática com a versão etiquetada manualmente.

Vimos que esse processo mede *indiretamente* a eficácia de uma regra, utilizando o etiquetador Tycho Brahe. Explicaremos, a seguir, a razão de existir esse etiquetador, e, resumidamente como ele foi implementado.

2.2 *Corpus* Tycho Brahe

Por *corpus* entende-se um conjunto de textos naturais utilizado como material para estudos de lingüística. Esse conjunto deve ser uma amostra estatisticamente válida do tipo de texto que se deseja estudar, isto é, deve apresentar o maior número de construções/estruturas a serem analisadas, na proporção em que são utilizadas.

O *Corpus* Tycho Brahe está sendo construído como parte de um projeto [23] que estuda a evolução da Língua Portuguesa, com o objetivo de investigar as variações do uso de próclises, mesóclises e ênclises durante os séculos XVI, XVII, XVIII e XIX.

Esse *corpus* é composto por textos originais ortograficamente transcritos que foram escolhidos com os seguintes critérios filológicos :

- Edições revistas pelos próprios autores ou baseadas em manuscritos;
- Textos de autores cuja biografia indica poucos e/ou curtos períodos fora de Portugal, preferencialmente;
- Textos com aproximadamente 50.000 palavras.

Cada um dos textos selecionados originou (ou originará) uma versão correspondente integralmente *etiquetada morfossintaticamente.*

Etiquetar um texto significa classificar cada uma de suas palavras de acordo com o contexto. Para o *corpus* Tycho Brahe, a classificação relevante é a morfossintática, ou seja, a classe gramatical (verbo, substantivo etc.) com algum detalhamento (tempo, gênero, número etc). Um conjunto finito de *etiquetas* foi definido para esse *corpus* (especificado em [23]), de acordo com o objetivo do projeto.

2.2.1 *Etiquetador* Tycho Brahe

O grande número de palavras a serem etiquetadas para a construção do *Corpus* Tycho Brahe motivou a construção de ferramentas para a realização de etiquetagem automática.

Conforme Brill [6] propôs para a Língua Inglesa, Carlos Chacur [8] implementou um *etiquetador* automático para a Língua Portuguesa. Esse *etiquetador* recebeu mesmo o nome do *corpus* e utiliza regras de transformação, ou seja, regras que transformam uma etiquetagem inicial do texto (veja Figura 2.2). Tais regras são geradas em uma *fase de treinamento*, que utiliza textos etiquetados manualmente (*corpus de treinamento*).

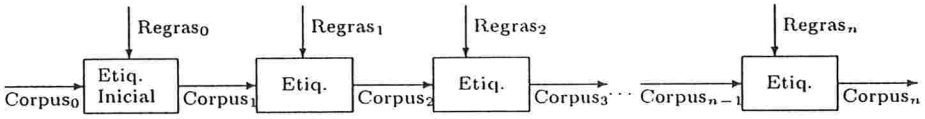

Figura 2.2: Etiquetagem baseada em Regras de Transformações Sucessivas

2.2.2 Etiquetas do *Corpus* Tycho Brahe

De acordo com [11], o número de etiquetas necessárias para todas as construções morfossintáticas da Língua Portuguesa é da ordem de 300. Como foi demonstrado em [8], a complexidade do treinamento para o método de Brill é $O(N^4)$, sendo N o número de etiquetas utilizadas.

Por esses motivos, acredita-se que é impraticável o uso direto do método Brill para a Língua Portuguesa (de fato, um experimento realizado prolongou-se por mais de 62 dias).

Para minimizar essa complexidade, propôs-se uma reestruturação [23] do formato das etiquetas, que, agora, constituem-se de uma base e de sufixos. Por exemplo, a etiqueta

`D-UM-F-P`

apresenta uma base, que nesse caso, indica um artigo (ou determinante: `D`) e sufixos de detalhamento da base: natureza (artigo indefinido: `UM`), gênero (artigo feminino: `F`) e número (plural: `P`).

Após essa reestruturação, o número de etiquetas básicas presentes no *Corpus* Tycho Brahe baixou para 33, na mesma ordem do número de etiquetas da Língua Inglesa, tornando viável o treinamento para etiquetagem básica (de acordo com [11], um treinamento realizado durou 42 horas).

2.2.3 Erros de Etiquetagem

Para medir a precisão do etiquetador, é necessária a utilização de um *corpus-referência* diferente do *corpus de treinamento*, também etiquetado manualmente. Aplica-se o etiquetador a esse *corpus-referência* com as etiquetas previamente retiradas, e então a precisão é dada por:

$$\text{Precisão} = \frac{\text{N}^o \text{ de etiquetas coincidentes}}{\text{N}^o \text{ total de etiquetas}}$$

Atualmente, o etiquetador apresenta uma precisão de 95,43%, utilizando um *corpus de treinamento* de 390.000 palavras.

Analisando o conjunto de etiquetas não coincidentes, podem-se separar os erros de etiquetagem de palavras em dois grandes grupos :

- Palavras Conhecidas, ou seja, aquelas que estão no *léxico de treinamento* do etiquetador;
- Palavras Desconhecidas, obviamente aquelas que não estão no *léxico de treinamento* do etiquetador.

Entende-se por *léxico* o conjunto de tipos de palavras e suas propriedades (categorias morfossintáticas, gênero, número, etc). Em última análise, esse conjunto pode ser definido por todas as palavras de todos os textos de um *Corpus*. Se o *Corpus* tem um novo texto incluído, as palavras desse texto passam a pertencer ao *léxico* desse *Corpus*.

Pode-se afirmar que a maioria dos erros do primeiro grupo (*Palavras Conhecidas*) ocorrem por causa do tamanho insuficiente do *corpus de treinamento*, ou seja, este corpus não tem textos suficientes para conter todas as formas de utilização dessas palavras na Língua.

Vale a pena ressaltar que o acerto na etiquetagem de uma palavra não depende *somente* do fato de essa estar ou não no *léxico de treinamento*, uma vez que, durante a fase de treinamento, o etiquetador pode inferir, por exemplo, uma regra, tendo como base somente a combinação de etiquetas de palavras próximas.

Analisando os atuais erros de etiquetagem, verificou-se que um grande número deles pertence ao segundo grupo, ou seja, ao grupo de *Palavras Desconhecidas*.

Pode-se tentar diminuir o número desse tipo de erro de duas maneiras:

- Aumentar o tamanho do *léxico de treinamento*;
- Transformar o máximo de *palavras desconhecidas* em *palavras conhecidas*.

Mais uma vez, o tamanho do *corpus de treinamento* provavelmente é uma causa significativa da quantidade desse tipo de erro, pois quanto maior o tamanho do corpus, maior vai ser (em uma escala relativamente menor) o tamanho do léxico e, portanto menor será o número de *palavras desconhecidas*.

Como a taxa de crescimento do *léxico de treinamento* diminui à medida que o *corpus de treinamento* aumenta (já que a possibilidade de um texto novo apresentar uma palavra desconhecida é cada vez menor), existe um momento em que não é mais *eficiente* aumentar o *corpus de treinamento* visando diminuir o número de erros de etiquetagem de *palavras desconhecidas*.

Outra razão para uma palavra ser desconhecida, é o fato de que ela não pertence ao léxico de treinamento, ou seja, ela é uma ocorrência de um erro de ortografia em relação à norma do léxico de treinamento.

Portanto, corrigir desvios de ortografia de um texto diminui a ocorrência de palavras desconhecidas (i.e. palavras não pertencentes ao léxico de treinamento), em outras palavras diminui a ocorrência de erros de etiquetagem, *aumentando a precisão de etiquetagem*.

Esta é a razão pela qual estamos utilizando o aumento da precisão de etiquetagem para descobrir se uma *regra de substituição* deve ser ou não incorporada no "conhecimento" do normatizador, ou seja, se essa regra aumenta o grau de normatização de palavras.

A partir dessa conclusão, propomos um método indireto de aprendizado que seleciona regras de transformação ortográfica com base na variação no desempenho da etiquetagem a partir de um conjunto de regras-candidatas gerados automáticamente a partir de um pequeno dicionário normatizador.

Capítulo 3

Implementação

Neste capítulo apresentamos a implementação do normatizador, a partir do modelo apresentado no capítulo anterior.

O sistema se divide em três grandes módulos (Ver figura 3.1):

- Gerador de regras-candidatas;
- "Treinador";
- Normatizador.

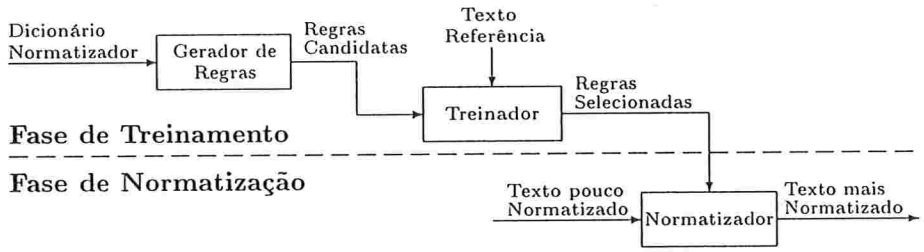

Figura 3.1: Processo de Treinamento/Normatização

Os dois primeiros módulos (*Gerador* e *Treinador*) são utilizados somente na fase de treinamento, enquanto o *Normatizador* é utilizado na fase de normatização.

Usando um pequeno Dicionário de palavras, o primeiro módulo gera um grande conjunto de regras-candidatas de onde o segundo módulo seleciona (filtra) um subconjunto de regras usando um Texto-Referência, dando preferência a regras genéricas que podem normatizar palavras não pertencentes a este Dicionário. A normatização de textos realizada pelo último módulo utiliza esse

novo conjunto de regras selecionadas.

Iniciaremos a exposição com a descrição da geração de regra-candidatas, para, em seguida, mostrarmos como é feita a seleção de regras (Treinador). Finalizaremos apresentando a normatização propriamente dita.

Os algoritmos serão descritos um a um, considerando sua finalidade e complexidade. A implementação desses é rigorosamente aderente a este texto, e foi realizada na linguagem de programação PERL [22].

3.1 Geração de Regras Candidatas

O módulo de geração de regras consome um *Dicionário Normatizador* gerando um conjunto de regras-candidatas com diferentes níveis de generalização.

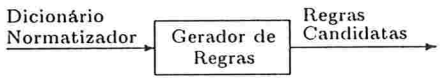

Figura 3.2: Geração de Regras Candidatas

Esse Dicionário Normatizador é composto de um conjunto de pares de palavras, em que a primeira corresponde à versão normatizada da segunda.

Veremos a seguir como é o processo de geração de regras, sua eficácia e ainda, a proporção de regras geradas por par de palavras do Dicionário.

3.1.1 Algoritmo

Para cada par de palavras do Dicionário Normatizador, o algoritmo realiza três grandes etapas:

1. Identificar as diferenças entre as palavras;
2. Para cada diferença, gerar regras de escopo no texto;
3. Para cada diferença, gerar regras de escopo na palavra.

Ao final dessas 3 etapas, gera-se um conjunto de regras de substituição, a partir da combinação de todas as regras de escopo de cada uma das diferenças encontradas entre as palavras.

Identificar Diferenças

A etapa de identificação de diferenças consiste na comparação progressiva de cada caractere das duas palavras, identificando os seguintes aspectos:

- O fragmento coincidente anterior à diferença, que chamaremos de *prefixo coincidente*;
- A diferença propriamente dita, ou seja, o fragmento indesejado e seu substituto;
- O fragmento coincidente posterior à diferença, que chamaremos de *sufixo coincidente*.

Ao final dessa etapa, já é possível identificar a terceira parte da regra de substituição, a que define o fragmento substituto.

Por exemplo, usando a palavra 'avisos' que era escrita antigamente como 'avizos', temos:

- prefixo coincidente : 'avi';
- diferença propriamente dita:
  - fragmento indesejado : 'z';
  - fragmento desejado : 's';
- sufixo coincidente : 'os'.

Mas os fragmentos coincidentes não são simultaneamente mandatórios, isto é, não é obrigatório que **ambos** os fragmentos existam. Por exemplo, a palavra 'não' era escrita antigamente como 'naõ', e, nesse caso, temos:

- prefixo coincidente : 'n';
- diferença propriamente dita:
  - fragmento indesejado : 'aõ';
  - fragmento desejado : 'ão';
- sufixo coincidente : ''.

Além disso, pode haver mais de uma diferença por par de palavras. No caso da palavra 'Relação', que era escrita como 'Rellaçaõ', temos:

- diferença 1 :
  - prefixo coincidente : 'Re';
  - diferença propriamente dita:
    - fragmento indesejado : 'll';
    - fragmento desejado : 'l';
  - sufixo coincidente : 'aç'.
- diferença 2 :

- prefixo coincidente : 'aç';
- diferença propriamente dita:
  * fragmento indesejado : 'aõ';
  * fragmento desejado : 'ão';
- sufixo coincidente : ''.

Como vimos no início desta seção, utilizaremos um Dicionário Normatizador para obter pares de palavras, onde a primeira corresponde à versão normatizada da segunda.

Desse modo, implementamos o seguinte algoritmo, para encontrar as diferenças de cada par de palavras:

Nome: FindDifferences

Entrada:

- `correctWord` (Palavra Normatizada)
- `wrongWord` (Palavra Não Normatizada)

Descrição:

1. prefixo coincidente ← fragmento coincidente, a partir do início
2. faça enquanto não acabar a palavra
3. encontre o fragmento indesejado
4. encontre o fragmento de substituição
5. sufixo coincidente ← próximo fragmento coincidente
6. crie regras de escopo no texto, usando:
   - prefixo coincidente;
   - fragmento indesejado;
   - fragmento desejado;
   - sufixo coincidente.
7. prefixo coincidente ← sufixo coincidente
8. fim-faça

A complexidade desse algoritmo é $O(\mathtt{n})$, sendo:

$$\mathtt{n} = max(length(\mathtt{correctWord}), length(\mathtt{wrongWord}))$$

isto é, n é o maior número de letras entre o par de palavras.

Vale a pena ressaltar que é possível encontrar no máximo n/2 diferenças por par de palavras.

Cria Regra de Escopo no Texto

Nessa fase, para cada diferença identificada gera-se uma série de expressões regulares que selecionam palavras de um texto.

Cada uma destas expressões regulares define a primeira parte de uma série de regras-candidatas, que serão geradas.

Inicialmente, o algoritmo cria regras de escopo no texto considerando o prefixo e o sufixo simultaneamente (linhas 1 a 22).

A seguir, o algoritmo cria regras de escopo no texto considerando o prefixo (linhas 23 a 39) e o sufixo (linhas 40 a 56) isoladamente.

Finalmente, cria-se uma regra de escopo no texto considerando somente o fragmento indesejado (linhas 57 a 60).

Nome: CreateSearchRules

Entrada:

- `prefix` (Prefixo Coincidente)
- `diffWrong` (Fragmento Indesejado)
- `diffCorrect` (Fragmento Desejado)
- `suffix` (Sufixo Coincidente)

Descrição:

1. se `prefix` não está vazio e `suffix` não está vazio
2. `searchRule` ← `prefix`+`diffWrong`+`suffix`
3. cria regra de escopo na palavra, usando:
   - `prefix`
   - `diffWrong`;
   - `diffCorrect`;
   - `suffix`;
   - `searchRule`.
4. `preaux` ← última letra de `prefix`
5. `posaux` ← primeira letra de `suffix`
6. `searchRule` ← `preaux`+`diffWrong`+`posaux`
7. cria regra de escopo na palavra, usando:
   - `prefix`
   - `diffWrong`;
   - `diffCorrect`;
   - `suffix`;
   - `searchRule`.
8. se `prefix` termina com vogal

9. se `suffix` inicia-se com vogal
10. `searchRule` ← "`[aeiou]`"+`diffWrong`+"`[aeiou]`"
11. senão
12. `searchRule` ← "`[aeiou]`"+`diffWrong`+"`[^aeiou]`"
13. fim-se
14. senão
15. se `suffix` inicia com vogal
16. `searchRule` ← "`[^aeiou]`"+`diffWrong`+"`[aeiou]`"
17. senão
18. `searchRule` ← "`[^aeiou]`"+`diffWrong`+"`[^aeiou]`"
19. fim-se
20. fim-se
21. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
22. fim-se
23. se `prefix` não está vazio
24. `searchRule` ← `prefix`+`diffWrong`
25. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
26. `searchRule` ← `prefix`
27. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
28. `preaux` ← última letra de `prefix`
29. `searchRule` ← `preaux`+`diffWrong`
30. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
31. se `diffWrong` não está vazio

32. se `prefix` termina com vogal
33. `searchRule` ← "`[aeiou]`"+`diffWrong`
34. senão
35. `searchRule` ← "`[^aeiou]`"+`diffWrong`
36. fim-se
37. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
38. fim-se
39. fim-se
40. se `suffix` não está vazio
41. `searchRule` ← `diffWrong`+`suffix`
42. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
43. `searchRule` ← `suffix`
44. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
45. `posaux` ← primeira letra de `suffix`
46. `searchRule` ← `diffWrong`+`posaux`
47. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
48. se `diffWrong` não está vazio
49. se `suffix` inicia com vogal
50. `searchRule` ← `diffWrong`+"`[aeiou]`"
51. senão
52. `searchRule` ← `diffWrong`+"`[^aeiou]`"
53. fim-se
54. cria regra de escopo na palavra, usando:

- `prefix`
- `diffWrong`;
- `diffCorrect`;
- `suffix`;
- `searchRule`.

55. fim-se
56. fim-se
57. se `diffWrong` não está vazio
58. `searchRule` ← `diffWrong`
59. cria regra de escopo na palavra, usando:
    - `prefix`
    - `diffWrong`;
    - `diffCorrect`;
    - `suffix`;
    - `searchRule`.
60. fim-se

A complexidade desse algoritmo é $O(1)$, sendo, ao final, geradas, dependendo da posição e tamanho da diferença encontrada, entre 1 e 12 regras de escopo no texto, isto é, gera-se a primeira parte da regra se substituição.

Cria Regra de Escopo na Palavra

Finalmente, nessa última fase, gera-se uma série de expressões regulares que definem o fragmento a ser substituído pelo fragmento correto.

Ou seja, cada uma dessas expressões regulares geradas define a segunda parte da regra de substituição, permitindo a montagem de regras-candidatas.

Inicialmente, o algoritmo cria uma regra de escopo na palavra considerando somente o fragmento indesejado (linhas 1 a 5).

A seguir, o algoritmo cria regras de escopo na palavra considerando o prefixo e o sufixo simultaneamente (linhas 6 a 15).

Finalmente, criam-se regras de escopo considerando o prefixo (linhas 16 a 24) e o sufixo (linhas 25 a 33) isoladamente.

Nome: CreateSubstRules

Entrada:

- `prefix` (Prefixo Coincidente)
- `suffix` (Sufixo Coincidente)
- `diffWrong` (Fragmento Indesejado)
- `diffCorrect` (Fragmento Desejado)
- `searchRule` (Regra de Escopo no Texto)

Descrição:

1. se `diffWrong` não está vazio
2. `substRule` ← `diffWrong`
3. `subst` ← `diffCorrect`
4. cria regra-candidata, usando:
   - `searchRule`
   - `substRule`
   - `subst`
5. fim-se
6. se `prefix` não está vazio e `suffix` não está vazio
7. `substRule` ← `prefix+diffWrong+suffix`
8. `subst` ← `prefix+diffCorrect+suffix`
9. cria regra-candidata, usando:
   - `searchRule`
   - `substRule`
   - `subst`
10. `preaux` ← última letra de `prefix`
11. `posaux` ← primeira letra de `suffix`
12. `substRule` ← `preaux+diffWrong+posaux`

13. `subst` ← `preaux+diffCorrect+posaux`
14. cria regra-candidata, usando:
    - `searchRule`
    - `substRule`
    - `subst`
15. fim-se
16. se `prefix` não está vazio
17. `substRule` ← `prefix+diffWrong`
18. `subst` ← `prefix+diffCorrect`
19. cria regra-candidata, usando:
    - `searchRule`
    - `substRule`
    - `subst`
20. `preaux` ← última letra de `prefix`
21. `substRule` ← `preaux+diffWrong`
22. `subst` ← `preaux+diffCorrect`
23. cria regra-candidata, usando:
    - `searchRule`
    - `substRule`
    - `subst`
24. fim-se
25. se `suffix` não está vazio
26. `substRule` ← `diffWrong+suffix`
27. `subst` ← `diffCorrect+suffix`
28. cria regra-candidata, usando:
    - `searchRule`
    - `substRule`
    - `subst`
29. `posaux` ← primeira letra de `suffix`
30. `substRule` ← `diffWrong+posaux`
31. `subst` ← `diffCorrect+posaux`
32. cria regra-candidata, usando:
    - `searchRule`
    - `substRule`
    - `subst`
33. fim-se

A complexidade desse algoritmo é $O(1)$, sendo, ao final, geradas, dependendo da posição e tamanho da diferença encontrada, entre 1 e 7 regras de escopo na palavra, isto é, gera-se a segunda parte da regra se substituição. Assim, finalmente, podem-se gerar as regras-candidatas, uma vez que temos todas as partes geradas.

Ao final das três etapas, temos entre 1 e 84 regras-candidatas geradas por diferença encontrada no par de palavras. A complexidade total desse processo é $O((n/2) * 12 * 7) = O(n)$.

Veja no Apêndice B, um exemplo de geração de regras a partir de um par de palavras.

3.2 Processo de Aprendizado

Este processo consiste em filtrar um conjunto ordenado de regras que será utilizado para a normatização de textos.

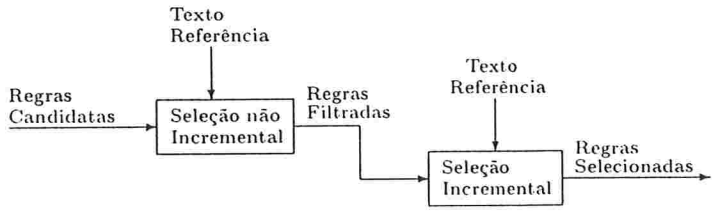

Figura 3.3: Processo de Aprendizado

Como já dito anteriormente, essa seleção consiste em comparar a precisão de etiquetagem automática de um texto-referência (texto etiquetado manualmente por um lingüista), antes e depois da aplicação de uma regra-candidata (Ver seção 2.2.3).

Temos aqui, duas etapas (Ver figura 3.3):

- seleção não-incremental de regras;
- seleção incremental de regras.

A primeira etapa realiza uma seleção não-incremental, ou seja, selecionam-se regras que aumentam a precisão de etiquetagem em relação à versão original do texto-referência.

Na seleção incremental, as regras são escolhidas de acordo com a variação da precisão de etiquetagem, considerando as versões decorrentes de transformações de regras-candidatas anteriormente selecionadas.

3.2.1 Seleção Não-Incremental

Esta etapa consiste em filtrar as regras que aumentam a precisão de etiquetagem da versão inicial do texto-referência.

Em outras palavras, cada regra tem sua precisão de etiquetagem comparada à versão original do texto-referência. Independentemente do fato de esta regra aumentar ou não a precisão de etiquetagem, a próxima regra também utilizará a versão original do texto-referência em seu processo de seleção.

Figura 3.4: Seleção Não Incremental

Segue o algoritmo para a seleção não incremental de regras:

Entrada:

- `reference` (Texto referência etiquetado manualmente)
- `untagged` (Texto referência sem etiquetas)
- `rules` (Conjunto de regras-candidatas)

Saída:

- `tmpRules` (Conjunto de regras filtradas)

Descrição:

1. realiza etiquetagem em `untagged`
2. `originalScore` ← *score*(`reference`, `untagged`)
3. faça enquanto existem regras-candidatas
4. tmp ← *applyRule*(regra-candidata, `untagged`)
5. `changes` ← # palavras alteradas pela regra-candidata
6. se `changes` > 0
7. realiza etiquetagem em `tmp`
8. `atualScore` ← *score*(`reference`, `untagged`)
9. se `originalScore` > `atualScore`
10. seleciona regra-candidata
11. senão
12. ignora regra-candidata
13. fim-se
14. fim-se
15. fim-faça

Este algoritmo aplica cada uma das regras (linha 3) na versão inicial do texto-referência, e seleciona somente as regras que aumentam a precisao de etiquetagem automática do texto (linhas 6 a 11).

Como veremos a seguir, a contagem de erros de etiquetagem apresenta uma complexidade $O(\mathtt{n})$, sendo que n corresponde ao número de palavras do texto-referência. Posteriormente, veremos que a aplicação de uma regra-candidata também tem uma complexidade $O(\mathtt{n})$.

Portanto este algoritmo tem uma complexidade $O(\mathtt{n} * \mathtt{m})$, em que n corresponde ao número de palavras do texto-referência e m corresponde ao número total de regras-candidatas.

Contagem de Erros de Etiquetagem

O algoritmo a seguir consiste na contagem de etiquetas erradas decorrentes de uma etiquetagem automática realizada pelo etiquetador Tycho Brahe (Veja seção 2.2.1), em relação a uma etiquetagem manual.

Nome: score

Entrada:

- `refTagged` (Texto-referência etiquetado manualmente)
- `tagged` (Versão do Texto-referência etiquetado automaticamente)

Saída:

- `errors` (Número de erros de etiquetagem)

Descrição:

1. `errors` $\leftarrow 0$
2. faça enquanto houver palavras
3. `tag1` $\leftarrow$ próxima etiqueta em `refTagged`
4. `tag2` $\leftarrow$ próxima etiqueta em `tagged`
5. se `tag1` $\neq$ `tag2`
6. `errors` $\leftarrow$ `errors` $+1$
7. fim-se
8. fim-faça

Pode-se constatar que a complexidade deste algoritmo é $O(\mathtt{n})$, sendo que n corresponde ao número de palavras do texto-referência.

Aplicação de Uma Regra

O próximo algoritmo realiza a aplicação de uma Regra de Substituição de Cadeia de Caracteres (Ver seção 2.1.1) no Texto-Referência, criando uma nova versão.

Para cada palavra (linha 5), testa-se se o resultado da Expressão regular$_1$ da regra se aplica (linha 6). Em caso positivo, aplica-se a Expressão regular$_2$ selecionado o fragmento a ser substituído pela *String* de Substituição (linha 7).

Nome: applyRule

Entrada:

- `rule` (Regra a ser aplicada)
- `inFile` (Texto original)

Saída:

- `outFile` (Texto alterado pela regra)

Descrição:

1. `searchRule` ← Expressão Regular$_1$ de `rule`
2. `substRule` ← Expressão Regular$_2$ de `rule`
3. `substExpr` ← *String* de Substituição de `rule`
4. faça enquanto houver palavras em `inFile`
5. `word` ← próxima palavra em `inFile`
6. se `word` =~ `/searchRule/i`
7. `word` =~ `s/substRule/substExpr/i`
8. fim-se
9. grava-se `word` em `outFile`
10. fim-faça

Novamente, pode-se constatar que a complexidade deste algoritmo é $O(n)$, sendo que n corresponde ao número de palavras do texto-referência.

3.2.2 Seleção Incremental

Nesta etapa, as regras são selecionadas de acordo com a variação da precisão de etiquetagem considerando as versões decorrentes de transformações de regras-candidatas anteriormente escolhidas.

Ou seja, cada regra tem sua precisão de etiquetagem comparada à precisão de etiquetagem da versão do texto-referência alterada por todas as regras anteriormente selecionadas. Assim, se a regra aumentar a precisão de etiquetagem, então ela será incluída no conjunto de regras selecionadas no processo de seleção das próximas regras.

Para isso, as regras filtradas pelo processo anterior são inicialmente **ordenadas pela precisão de etiquetagem**, ou seja, em ordem decrescente de erros de etiquetagem. Assim, as regras que mais aumentam a precisão de etiquetagem são selecionadas em primeiro lugar, minimizando o total de erros de etiquetagem considerando todas as regras selecionadas por este processo.

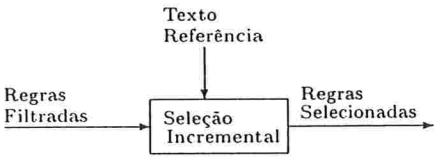

Figura 3.5: Seleção Incremental

Entrada:

- `reference` (Texto referência etiquetado manualmente)
- `untagged` (Texto referência sem etiquetas)
- `tmpRules` (Conjunto de regras filtradas)

Saída:

- `currentRules` (Conjunto de regras selecionadas)

Descrição:

1. ordene as regras-candidatas
2. `currentScore` ← *score*(`reference`, `untagged`)
3. copie `untagged` para `current`
4. faça enquanto existem regras-candidatas
5. tmp ← *applyRule*(regra-candidata, `current`)
6. `changes` ← # palavras alteradas pela regra-candidata
7. se `changes` > 0
8. `tmpScore` ← *score*(`reference`, `tmp`)
9. se `currentScore` > `tmpScore`
10. seleciona regra-candidata
11. `currentScore` ← `tmpScore`

12. mova `tmp` para `current`
13. senão
14. ignora regra-candidata
15. fim-se
16. fim-se
17. fim-faça

Este algoritmo aplica cada uma das regras (linha 4), e seleciona somente as regras que aumentam a precisao de etiquetagem automática do texto (linhas 8 a 14), e neste caso, considera a nova versão do texto-referência para a próxima regra.

Já vimos que a contagem de erros de etiquetagem apresenta uma complexidade $O(\mathtt{n})$ (Ver seção 3.2.1), sendo que a aplicação de uma regra-candidata também tem uma complexidade $O(\mathtt{n})$ (Ver seção 3.2.1), em que `n` corresponde ao número de palavras do texto-referência.

Assim, este algoritmo tem uma complexidade $O(\mathtt{n} * \mathtt{m})$, sendo que `n` corresponde ao número de palavras do texto-referência e `m` corresponde ao número total de regras-candidatas.

3.3 Processo de Normatização

Uma vez que o treinamento foi realizado, esta ferramenta está apta a aumentar o grau de normatização de textos.

Este processo consiste em aplicar o conjunto de regras pré-selecionadas pelo processo de Treinamento em textos dos quais se deseje aumentar o grau de normatização.

Figura 3.6: Normatização de Textos

De maneira análoga ao processo utilizado no processo de Treinamento, as regras são aplicadas de maneira ordenada (linhas 4 a 12) em cada uma das palavras do texto a ser normatizado (linha 2).

Se, no entanto, ocorrer uma *desnormatização*, isto é, se a palavra original já for normatizada e a palavra transformada não for normatizada, então a transformação é ignorada.

Entrada:

- `rules` (Conjunto de Regras selecionadas)
- `inFile` (Texto pouco normatizado)

Saída:

- `outFile` (Texto mais normatizado)

Descrição:

1. faça enquanto houver palavras em `inFile`
2. `word` ← próxima palavra em `inFile`
3. `originalWord` ← `word`
4. faça enquanto houver regras em `rules`
5. `rule` ← próxima regra em `rules`
6. `searchRule` ← Expressão Regular$_1$ de `rule`
7. `substRule` ← Expressão Regular$_2$ de `rule`
8. `substExpr` ← *String* de Substituição de `rule`
9. se `word` =~ /`searchRule`/i
10. `word` =~ s/`substRule`/`substExpr`/i
11. fim-se
12. fim-faça

13. se `originalWord` $\in$ léxico
14. se `word` $\notin$ léxico
15. `word` $\leftarrow$ `originalWord`
16. fim-se
17. fim-se
18. grava-se `word` em `outFile`
19. fim-faça

A complexidade deste algoritmo é de $O(\mathtt{n} * \mathtt{m})$, sendo `n` correspondente ao número de palavras no texto, e `m` correspondente ao número de regras pertencentes ao "conhecimento".

Capítulo 4

Resultados

Após a implementação da ferramenta, realizamos um experimento para tentar comprovar sua eficácia.

Veja-se a descrição do experimento e dos resultados obtidos. No capítulo seguinte, fazemos uma análise desses resultados.

4.1 Descrição do Experimento

Amostra

O primeiro passo do experimento foi a escolha da amostra do experimento, ou seja, a determinação de qual seria o texto-referência utilizado no treinamento e de qual seria o texto utilizado para constatar se ocorre aumento do grau de normatização. Em ambos os casos, o texto deveria ter uma versão etiquetada manualmente, para permitir verificar a precisão de etiquetagem.

Para isto, escolhemos um texto do Corpus Tycho Brahe [23] intitulado *Rellaçaõ da Vida e Morte da Serva de Deos a Venerável Madre Elenna da Crus*, do ano de 1721, escrita por Maria do Céu [1] e o dividimos em 2 partes:

- A primeira, com 75% das sentenças (frases) escolhidas aleatoriamente, foi utilizada como texto-referência na fase de treinamento;
- A segunda, com os 25% restantes das sentenças, foi utilizada como amostra de controle.

A princípio, consideramos como normatizada uma palavra que pertence ao léxico de treinamento do etiquetador, uma vez que estamos partindo da premissa

[1] Esse texto, dentre os que têm uma versão etiquetada manualmente, pertence a uma época em que a norma ortográfica mais difere da atual

de que o etiquetador está atualmente treinado com textos, em sua maioria, ortograficamente modernos.

Assim, inicialmente chegamos aos seguintes números:

Amostra	Total de Ocorrências	Ocorrências já Normatizadas	Ocorrências não Normatizadas	Erros de Etiquetagem
Referência	22.025	21.525	500	3.846
Controle	7.750	7.578	172	1.349
Total	29.775	29.103	672	5.195

Tabela 4.1: Dados da Amostra

Dicionário Normatizador

A seguir, montamos o Dicionário Normatizador, ou seja, um conjunto de pares de palavras, em que a primeira corresponde à versão normatizada da segunda (Ver seção 3.1).

Para isso, foi realizada uma breve leitura do texto-referência, tendo sido selecionadas aleatoriamente, 70 palavras não normatizadas, segundo um critério simples : a normatização da palavra deveria ser imediata, ou seja, não deveriam ser necessários maiores conhecimentos de lingüística para deduzir a versão normatizada da palavra.

Assim, obtivemos uma amostra constituída por uma lista de pares de palavras que se encontra no Apêndice A.

4.2 Resultados Obtidos

Para comparar a heurística proposta (Qualidade de Etiquetagem) com uma heurística mais trivial (Quantidade de Palavras Normatizadas), foram realizados dois experimentos:

- Experimento 1: o grau de normatização de um texto foi medido pela variação da qualidade de etiquetagem;
- Experimento 2: o grau de normatização de um texto foi medido pela variação da quantidade de palavras que pertencem ao léxico de treinamento, ou seja, pela variação de palavras normatizadas.

Estes experimentos foram realizados variando-se somente o algoritmo de *scoring* utilizado.

Inicialmente, o Gerador de Regras montou 4.874 regras-candidatas, a partir do Dicionário de Treinamento, ou seja, uma média de 37,88 regras-candidatas por par de palavras.

Após o treinamento, obtivemos os seguintes resultados:

Métrica de Eficácia Utilizada no Treinamento	Regras Selecionadas Não-Incremental	Regras Selecionadas Incremental
Experimento 1	1.644	17
Experimento 2	2.013	44

Tabela 4.2: Resultados de Treinamento

Realizamos, então, a normatização da amostra de controle.

Inicialmente, analisamos a qualidade de etiquetagem e a comparamos com a versão não normatizada que possui 1.349 erros de etiquetagem (Ver tabela 4.1), obtendo os seguintes resultados:

Métrica de Eficácia Utilizada no Treinamento	Erros de Etiquetagem	Variação (%)	Tempo Gasto (hh:mm)
Experimento 1	936	30.61 %	01:11
Experimento 2	1.294	4.07 %	01:52

Tabela 4.3: Variação da Precisão de Etiquetagem

Novamente, consideramos como normatizada uma palavra que pertence ao léxico de treinamento do etiquetador.
Deste modo, analisamos o grau de normatização comparando com a versão original que possui 172 palavras não normatizadas (Ver tabela 4.1), obtendo os seguintes resultados:

Métrica de Eficácia Utilizada no Treinamento	Palavras Alteradas Normatizadas	Variação (%)	Total de Palavras não Normatizadas
Experimento 1	36	20.93 %	136
Experimento 2	10	5.81 %	162

Tabela 4.4: Resultados de Normatização

Adicionalmente, realizamos também uma **contagem manual** de palavras alteradas que estavam na forma moderna, ou seja, de acordo com a ortografia atual. Para isso verificamos se cada uma das palavras alteradas pode ser encontrada no dicionário Houaiss[12].

Analisando a normatização da amostra de controle feita pelas 17 regras, obtivemos os seguintes resultados:

Métrica de Eficácia Utilizada no Treinamento	Palavras Alteradas	Palavras Encontradas no Houassis	Variação (%)
Experimento 1	899	713	81.31 %
Experimento 2	96	36	40.62 %

Tabela 4.5: Contagem de Palavras encontradas no Houassis

Veja no Apêndice C, o conjunto de regras selecionadas pelo Experimento 1.

Os fontes (incluindo uma versão do Etiquetador Tycho Brahe) e as amostras do Experimento 1 podem ser encontrados em

`http://www.ime.usp.br/~hiro/normatizador.tar.gz`.

Capítulo 5

Conclusões

Neste estudo, implementamos um método para aumentar o grau de normatização de textos. O método consiste na aplicação de regras de substituição de cadeia de caracteres, selecionadas em uma fase de treinamento.

Estávamos particularmente interessados em verificar duas heurísticas:

- A utilização de regras de substituição de cadeia de caracteres para aumentar o grau de normatização de palavras (Ver seção 2.1.1);
- A utilização da variação da precisão de etiquetagem para medir o grau de normatização de textos (Ver seção 2.2.3).

5.1 Utilização de Regras de Substituição de Cadeia de Caracteres

As regras de substituição se mostraram eficazes no aumento do grau de normatização de palavras. Como observamos (Ver tabela 4.4), houve realmente um aumento no grau de normatização na amostra de controle do experimento.

Vale a pena lembrar que estas regras possuem diferentes níveis de generalização, possibilitando inclusive a normatização de palavras que não estão no Dicionário Normatizador utilizado na fase de treinamento.

Independentemente do modo pelo qual são geradas/filtradas, essas regras dividem o problema de normatização em dois níveis:

- Seleção das palavras elegíveis a uma correção (escopo da regra no texto);
- Correção propriamente dita (escopo da regra na palavra + substituição).

Desse modo, um normatizador que as utilize minimiza as tentativas desnecessárias de correção em palavras indevidas, aumentando sua eficiência e performance.

Finalmente, o formato proposto dessas regras apresenta características ideais para a representação explícita do "conhecimento" do normatizador.

5.2 Utilização da Variação da Precisão de Etiquetagem

Verificou-se que medir o grau de normatização utilizando a variação da precisão de etiquetagem se mostrou mais eficaz para seleção de regras durante a fase de treinamento, do que a utilização da variação do número de palavras normatizadas.

Isso baseia-se no fato de que, utilizando a variação da precisão de etiquetagem, um maior número de palavras da amostra de controle foi normatizado (Ver tabela 4.4), com um menor número de regras selecionadas (Ver tabela 4.2), ou seja, regras mais eficazes foram selecionadas no treinamento.

Vale a pena ressaltar que utilizamos um corpus muito específico, pois estamos trabalhando com um conjunto de textos não normatizados que foram etiquetados morfossintaticamente por um lingüista.

5.3 Eficácia do Normatizador

Vimos, nos experimentos, que as regras selecionadas aumentaram o grau de normatização da amostra de controle (Ver tabela 4.4).

Como essa amostra de controle é independente do texto-referência utilizado no treinamento (isto é, essa amostra não foi utilizada na fase de treinamento), concluimos, então, que essas regras aumentam o grau de normatização de textos de uma maneira geral.

Já que a eficácia do normatizador depende desse conjunto de regras, consideramos para análise as seguintes características:

- Qualidade das regras;
- Quantidade das regras.

5.3.1 Qualidade das Regras

Quanto mais genéricas as regras, mais palavras essas regras afetam.

O processo de treinamento implementado dá preferência às regras genéricas, assimilando aquelas que normatizam o texto como um todo. Ou seja, se determinada regra, isoladamente, normatiza mais palavras do que ela "desnormatiza",

então ela é assimilada ao "conhecimento" do normatizador (Normatização Cega).

Posteriormente, no processo de normatização, as regras são aplicadas em lote, palavra por palavra, isto é, todas as regras são aplicadas em cada uma das palavras, isoladamente. Desse modo, uma tentativa de normatização de uma palavra só é efetivada se o resultado final for uma normatização da versão inicial.

Assim, possibilitamos a inclusão de regras que realizam *normatizações parciais*, ou seja, de regras que **aproximam** palavras à sua forma mais moderna.

Novamente, ressaltamos que as correções efetuadas pelas regras selecionadas atualmente **não** se limitaram somente às palavras que estavam no Dicionário Normatizador criado para o experimento (Ver seção 4.1), ou seja, o método selecionou regras mais genéricas, aumentando sua eficácia no aumento do grau de normatização de textos.

5.3.2 Quantidade das Regras

Quanto mais regras existirem, mais palavras todas essas regras afetam.

Mas, se houver regras demais, o normatizador não será eficiente, isto é, o processo de normatização levará muito tempo para ser concluído.

Isto se deve ao fato de que a complexidade do processo é $O(\mathtt{n} * \mathtt{m})$, em que `n` corresponde ao número de palavras do texto e `m` corresponde ao número total de regras-candidatas (Ver seção 3.3).

Esses dois pontos de atenção podem ser (mais) considerados na etapa de Geração de Regras Candidatas (Ver seção 3.1), de duas maneiras:

- Aumentando o Dicionário Normatizador;
- Aumentando o número de regras geradas por diferenças encontradas.

Assim, teríamos mais regras-candidatas, aumentando a qualidade das regras selecionadas.

5.4 Conclusões Adicionais

Estudando os resultados do experimento que utilizou a variação da precisão de etiquetagem para medir o grau de normatização de textos durante a fase de treinamento, chegamos a alguns fatos que devem ser ressaltados.

Inicialmente, verificou-se que nem todas as palavras do Dicionário Normatizador foram normatizadas. Isso se deve a dois fatos :

- Atualmente, o etiquetador está treinado **também** com textos históricos. Isto faz com que algumas palavras pareçam normatizadas ao etiquetador (em conseqüência, ao normatizador), já que pertencem ao léxico de treinamento do etiquetador;

- Muitas dessas palavras não interferem atualmente na qualidade de etiquetagem, ou seja, com o nível de treinamento atual, o fato de essas palavras não serem conhecidas (Ver seção 2.2.3) não influi na qualidade de etiquetagem. Novamente, essas palavras tornam-se "invisíveis" ao etiquetador e, em conseqüência, ao normatizador.

Ou seja, a qualidade do corpus de treinamento do etiquetador tem alguma influência na qualidade das regras selecionadas pelo normatizador.

Constatou-se, também, que grande parte das palavras alteradas na fase de normatização foram realmente normatizadas de acordo com a norma corrente (Ver tabela 4.5), ignorando o fato de que a palavra original se encontra no léxico de treinamento do etiquetador. Ou seja, as regras selecionadas normatizaram palavras consideradas ortograficamente corretas pelo etiquetador, sem prejudicar a qualidade de etiquetagem.

Finalmente, a qualidade de etiquetagem aumentou (Ver tabela 4.3), já que a quantidade de erros de etiquetagem diminuiu significativamente. Assim, provavelmente essa ferramenta pode servir de apoio para aumentar a qualidade de etiquetagem em treinamentos futuros do etiquetador.

Apêndice A

Dicionário Normatizador

A seguir, a lista de palavras utilizadas no treinamento do experimento realizado.

Esta lista foi montada a partir de uma breve leitura do texto *Rellaçaõ da Vida e Morte da Serva de Deos a Venerável Madre Elenna da Crus*, do ano de 1721, escrita por Maria do Céu. Usamos um critério simples : a normatização da palavra deveria ser imediata, ou seja, não deveriam ser necessários maiores conhecimentos de lingüística para deduzir a versão normatizada da palavra.

1. `elegância elegancia`
2. `Irmãs Irmaas`
3. `proporcionada proporsionada`
4. `insuportáveis insoportaveis`
5. `clausura clauzura`
6. `repetir repitir`
7. `presentes prezentes`
8. `mortificava mortificaua`
9. `açucar assucar`
10. `atrevimento attreuimento`
11. `teima teyma`
12. `Prelados Prellados`
13. `vestiu vistio`
14. `belo bello`

15. disposição disposiçaõ
16. acaso acazo
17. promessa promeça
18. talentos tallentos
19. precisa preciza
20. estrangeiros estrangeyros
21. levava leuaua
22. cuidadosa cuydadosa
23. comunicação commonicaçaõ
24. dano danno
25. celeste celleste
26. noites noytes
27. sufrágio suffragio
28. estavam estauaõ
29. visão vizão
30. deleite deleyte
31. falecida fallecida
32. suspeitoso sospeytoso
33. enfeites enfeytes
34. contemporizações contemporisaçoens
35. relatarei relatarey
36. chocolate choculate
37. ausente auzente
38. vozes voses
39. lágrima lagrima
40. potências potencias
41. gravíssimo grauissimo
42. serviços servissos

43. conservavam conservavaõ
44. destruir distruir
45. transformações transformaçoens
46. imundo immumdo
47. cantava cantaua
48. músico musico
49. viola violla
50. opõem oppoem
51. perseguições perseguiçoens
52. Divinos Diuinos
53. diferenças differencas
54. extensão extençaõ
55. Música Muzica
56. natureza naturesa
57. capitães cappitaes
58. fundação fundaçaõ
59. discrição discripçaõ
60. operações opperaçoens
61. explicações explicaçoens
62. raízes rayzes
63. natureza natuersa
64. suspeitoso suspeitozo
65. conheciam conheciaõ
66. multidão multidaõ
67. debilidade debelidade
68. confusão confuzaõ
69. devemos deuemos
70. piedosa piedoza

Apêndice B

Regras Geradas

Durante o experimento descrito, o par de palavras:

`elegância elegancia`

originou as seguintes regras-candidatas:

1. `elegancia a â`
2. `elegancia elegancia elegância`
3. `elegancia gan gân`
4. `elegancia elega elegâ`
5. `elegancia ga gâ`
6. `elegancia ancia ância`
7. `elegancia an ân`
8. `gan a â`
9. `gan elegancia elegância`
10. `gan gan gân`
11. `gan elega elegâ`
12. `gan ga gâ`
13. `gan ancia ância`
14. `gan an ân`
15. `[^aeiou]a[^aeiou] a â`
16. `[^aeiou]a[^aeiou] elegancia elegância`

17. [^aeiou]a[^aeiou] gan gân
18. [^aeiou]a[^aeiou] elega elegâ
19. [^aeiou]a[^aeiou] ga gâ
20. [^aeiou]a[^aeiou] ancia ância
21. [^aeiou]a[^aeiou] an ân
22. elega a â
23. elega elegancia elegância
24. elega gan gân
25. elega elega elegâ
26. elega ga gâ
27. elega ancia ância
28. elega an ân
29. eleg a â
30. eleg elegancia elegância
31. eleg gan gân
32. eleg elega elegâ
33. eleg ga gâ
34. eleg ancia ância
35. eleg an ân
36. ga a â
37. ga elegancia elegância
38. ga gan gân
39. ga elega elegâ
40. ga ga gâ
41. ga ancia ância
42. ga an ân
43. [^aeiou]a a â
44. [^aeiou]a elegancia elegância

45. [^aeiou]a gan gân
46. [^aeiou]a elega elegâ
47. [^aeiou]a ga gâ
48. [^aeiou]a ancia ância
49. [^aeiou]a an ân
50. ancia a â
51. ancia elegancia elegância
52. ancia gan gân
53. ancia elega elegâ
54. ancia ga gâ
55. ancia ancia ância
56. ancia an ân
57. ncia a â
58. ncia elegancia elegância
59. ncia gan gân
60. ncia elega elegâ
61. ncia ga gâ
62. ncia ancia ância
63. ncia an ân
64. an a â
65. an elegancia elegância
66. an gan gân
67. an elega elegâ
68. an ga gâ
69. an ancia ância
70. an an ân
71. a[^aeiou] a â
72. a[^aeiou] elegancia elegância

73. a[^aeiou] gan gân

74. a[^aeiou] elega elegâ

75. a[^aeiou] ga gâ

76. a[^aeiou] ancia ância

77. a[^aeiou] an ân

78. a a â

79. a elegancia elegância

80. a gan gân

81. a elega elegâ

82. a ga gâ

83. a ancia ância

84. a an ân

Apêndice C

Regras Selecionadas

Durante o experimento descrito, as seguintes regrals s-candidatas foram selecionadas :

1. `[^aeiou]aõ aõ ão`
2. `y y i`
3. `[^aeiou]l[aeiou] ll l`
4. `[aeiou]u[aeiou] u v`
5. `gio a á`
6. `o z s`
7. `[aeiou]u iu iv`
8. `a[^aeiou] lag lág`
9. `[^aeiou]f ff f`
10. `[^aeiou]n[aeiou] nn n`
11. `[aeiou]z[aeiou] preciz precis`
12. `natu e`
13. `[aeiou]s[aeiou] ses zes`
14. `mu us ús`
15. `[aeiou]z[aeiou] zente sente`
16. `[aeiou]ss[aeiou] serviss serviç`
17. `[aeiou]ç[aeiou] promeça promessa`

Bibliografia

[1] E. Brill. *Transformation-Based Error Driven Parsing.* 3rd Workshop on Parsing Technologies, Netherlands, 1993.

[2] E. Brill. *A Corpus-Based Approach to Language Learning.* PhD thesis. Department of Computer and Information Science, University of Pennsylvania, 1993.

[3] E. Brill. *Automatic Grammar Induction and Parsing Free Text: A Transformation-Based Approach.* ACL 1993. Department of Computer and Information Science, University of Pennsylvania, 1993.

[4] P. Resnik, E. Brill. *A Rule-Based Approach to Prepositional Phrase Attachment Disambiguation.* Article. `http://research.microsoft.com~brill`, 1994.

[5] E. Brill. *Some Advances in Transformation-Based Part-Of-Speech Tagging.* In Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), 1994.

[6] E. Brill. *Transformation-Based Error Driven Learning and Natural Language. A Case Study in Part-of-Speech Tagging.* Computational Linguistics, 21(4):543-563, 1995.

[7] E. Brill. *Unsupervised Learning of Disambiguation Rules of Part-of-Speech Tagging.* Natural Language Processing Using Very Large Corpora, Kluwer Academinc Press, 1997.

[8] Carlos D. Chacur Alves. *Etiquetagem do Português Clássico Baseada em Corpus.* Master's thesis.

[9] F. J. Damerau. *A Technique for Computer Detection and Correction of Spelling Errors.* Communications of ACM, 7(3), 171-176. 1964.

[10] F. J. Damerau, E. Mays. *An Examination of Undetected Typing Errors.* Information Processing and Management, 25(6), 659-664. 1989.

[11] M. Finger. *Técnicas de Otimização da Precisão Empregadas no Etiquetador Tycho Brahe.* V Encontro para o Processamento Computacional da

Língua Portuguesa Escrita e Falada (PROPOR2000). Atibaia, Brasil, 2000. Instituto de Matemática e Estatística, USP, 1999.

[12] A. Houaiss, M. S. Villar. *Dicionário Houaiss da Língua Portuguesa.* Ed. Objetiva. Rio de Janeiro. 2004

[13] *International Ispell.* home page available at `http://ficus-www.cs.ucla.edu/geoff/ispell.html`.

[14] K. Kukich. *Techniques for Automatically Correcting Words in Text.* ACM Computing Surveys, 24(4), 377-439. 1992.

[15] W. Labov, Weinreich, Uriel, M. Herzog. *Empirical Foundations for a Theory of Language Change.* In W. Lehmann and Y. Malkiel (eds.), *Directions for Historical Linguistics.* University of Texas Press, Austin, US, 1968.

[16] L. Mangu, E. Brill. *Automatic Rule Acquisition for Spelling Correction* Article. `http://research.microsoft.com/~brill`, 1997.

[17] T. McEnery, A. Wilson. *Corpus Linguistics.* Edinburg University Press, 1st Edition, 1996.

[18] R. J. Mooney, E. Brill (eds.), *AI Magazine, vl. 18, n. 4.* American Association for Artificial Intelligence, 1997.

[19] M. G. V. Nunes *Desenvolvimento de um Revisor Gramatical para o Português Contemporâneo.* NILC/ICMC. Relatório Técnico no. 46, 1996.

[20] G. Satta, E. Brill. *Efficient Transformation-Based Parsing.* 34th Annual Meeting of the Association for Computational Linguistics, 1996.

[21] J. L. Peterson. *A Note on Undetected Typing Errors.* Communications of the ACM, 29(7), 633-637. 1986.

[22] *Pratical Extraction and Reporting Language (PERL).* home page available at `http://www.perl.com`.

[23] Project *Rythimic Patterns, Parameter Setting & Language Change.* home page available at `http://www.ime.usp.br/~tycho`. Instituto de Matemática e Estatística, USP, 1999.

[24] A. Villavicencio, N. M. C. Marques, J. G. P. Lopes, F. Villavicencio. *Part-of-Speech Tagging for Portuguese Texts.* Proceedings of the 12th Brazillian Conference on Artificial Inteligence (SBIA'95), 1995.