Instituo de Matemática e Estatística
Universidade de São paulo

Corretor Gramatical Para o Emacs

Aluno: Thiago Maciel Batista (5749922)
(*Orientador*: Prof. Dr. Marcelo Finger)

São Paulo, 01 de Dezembro de 2010

Sumário

Parte I

Parte Objetiva

1 Introdução

As linguagens que, normalmente, são compreendidas e manipuladas por computadores são formais e precisas, diferentemente de linguagens humanas, que possuem características que dificultam ao computador compreendê-las, como por exemplo, presença de ambiguidade, dependência de contexto externo, características culturais, regras gramaticais etc.

Processamento de linguagem natural (PLN) é uma área da computação que pesquisa e desenvolve algoritmos para processar uma linguagem natural, de modo a torná-la compreensível para um computador. Dessa forma, um computador será capaz de interpretar e de gerar textos em linguagens naturais, tornando a comunicação homem-máquina o mais natural possível.

PLN é uma área em plena evolução e que ainda tem muito a evoluir, mas já existem muitas aplicações que utilizam os conceitos de PLN. Alguns exemplos da utilidade de PLN são, extração de informação, recuperação de informação, tradução automática, geração e interpretação de linguagem natural, resumos e correção de textos. Nenhum destes aplicativos está livre de erros, que são advindos, principalmente, da dificuldade dos computadores em trabalhar com uma linguagem natural.

Um exemplo de programa que utiliza PLN é o cogroo[Col10a]. Um corretor gramatical para a língua portuguesa do Brasil totalmente acoplável ao OpenOffice.org[ope10b], desenvolvido por alunos da Universidade de São Paulo. O cogroo utiliza análise sintática, morfológica e regras gramaticas para detectar erros em textos. Para fazer a análise gramatical de um texto, o cogroo realiza um processo que é dividido em fases, que são executadas uma após a outra. Atualmente, o cogroo é composto por módulos e cada módulo realiza uma fase do processo.

Apesar de ser feito, a princípio, para o OpenOffice, o cogroo pode ser acoplado a outros aplicativos. Aproveitando essa mobilidade do cogroo, este trabalho tem como objetivo estudar formas de acoplar o corretor gramatical ao editor de textos Emacs[GNU10], que apesar de ser um editor de texto popular no contexto de software livre, ainda não possui um corretor para a língua portuguesa do Brasil. Ao final do acoplamento, o emacs usará o cogroo para analisar e encontrar erros em textos digitados pelos usuários.

O cogroo processará um trecho de um texto enviado pelo emacs a procura de erros e

devolverá ao emacs os erros encontrados e sugestões para correção, quando existirem. O emacs poderá usar os erros apontados pelo cogroo para mostrar ao usuário onde os erros ocorreram no texto, de forma que o usuário possa identificá-los e arrumá-los. Para isso, o emacs destacará os trechos do texto em que foram detectados erros.

Ao decorrer do trabalho teremos a definição formal de PLN e a descrição de conceitos utilizados em PLN, assim como uma descrição do corretor gramatical cogroo, mostrando seus módulos, técnicas e algoritmos utilizados. Ao fim, haverá a descrição de como foi feito o acoplamento do cogroo ao emacs e como o emacs utiliza o corretor, apresentando os resultados finais.

2 PLN

Linguagens naturais são as linguagens utilizadas pelas pessoas para se comunicarem por intermédio da fala ou da escrita, como por exemplo, português e inglês. Estas linguagens são usadas de forma intuitiva pelas pessoas, mesmo quando suas regras não são conhecidas profundamente. Processar uma linguagem natural é a capacidade de compreendê-la e de se expressar coerentemente, por intermédio da fala ou escrita, usando a linguagem.

Para que os computadores sejam capazes de manipular uma linguagem, esta precisar ser extremamente formal e precisa. Porém, tais linguagens não são intuitivas para a maioria das pessoas, tornando a interação com a máquina mais difícil. Para superar essa dificuldade e oferecer novas funcionalidades e aplicações, pesquisadores tentam encontrar um modo de tornar as interfaces dos computadores mais natural usando linguagens naturais. Esta, é a área de pesquisa na computação chamada de processamento de linguagem natural.

Processamento de linguagem natural (PLN ou NLP do inglês, *Natural Language Processing*) é uma área da inteligência artificial que se concentra no desenvolvimento de algoritmos para manipular linguagens naturais, de forma a torná-las compreensíveis para o computador. Sendo capaz de reconhecer e manipular uma linguagem natural, o computador poderá interpretar e gerar textos em linguagem humana. Para realizar estas tarefas, um computador terá que ser capaz de identificar contextos, fazer análise sintática, morfológica, semântica e léxica.

Desenvolver métodos que realizam o processamento de textos não é fácil. Uma linguagem natural possui características que dificultam ao computador seu completo entendimento. Ambiguidade, contexto externo, culturas locais e regras gramaticais são algumas dessas características que tornam PLN uma área extremamente difícil e desafiante.

Para resolver alguns dos problemas citados, os sistemas de PLN necessitam de grande conhecimento do mundo externo para que possa reconhecer todas características e variações da linguagem. Além disso, pode ser aplicado conceitos de estatística à linguagem. A utilização de estatística para processar uma linguagem é chamada de processamento estatístico de linguagem natural. Abaixo temos algumas aplicações de PLN, uma descrição mais detalhada de PLN estatístico e seus conceitos.

2.1 Utilidades de PLN

Apesar de ser uma área em desenvolvimento, com várias dificuldades e não possuir soluções tão eficazes para alguns problemas encontrados, processamento de linguagem natural é aplicado em diversos dispositivos que manipulam textos, ou até mesmo, arquivos de áudios. Abaixo exemplificamos algumas dessas aplicações.

- Extração de Informação:

  Extração de informação(EI) é uma área que pesquisa maneiras de se fazer buscas por informações específicas em um texto. A busca não precisa compreender o texto por completo e nem devolver um arquivo, mas sim identificar trechos que supram a necessidade do usuário. Usa-se extração de informação em grandes conjuntos de dados não estruturados para responder à consultas de usuários. Os resultados obtidos de uma consulta devem possuir informações que sejam relevantes ao usuário. Uma das técnicas utilizadas em EI é a busca baseada em palavras chaves. Para tornar a busca mais eficaz, pode ser aplicado os conceitos de PLN. Atualmente, com a grande quantidade de conteúdos de textos na internet, a extração de informação tem sido empregada nas buscas de informações na *web*.

- Recuperação de Informação:

  Recuperação de informação (RI) tem como objetivo trabalhar com armazenamento de grande quantidade documentos, cujo conteúdo, geralmente, são textos. Seu principal objetivo é desenvolver maneiras eficazes e eficientes de recuperar arquivos , de forma que os documentos encontrados contenham informações relevantes para uma pesquisa do usuário. Atualmente, os dispositivos de busca na *web* são as aplicações mais usadas, que empregam conceitos de recuperação de informação. Como maior

exemplo, podemos citar o *google*, o motor de pesquisa mais utilizado no mundo. Outro exemplo importante da utilização de RI são os dispositivos de busca em banco de dados relacionais.

Geralmente, confundi-se RI com EI. Sistemas de RI respondem à uma consulta com um conjunto de documentos relevantes ao usuário. Então, o usuário pode selecionar um documento deste conjunto que ele julgue como a melhor resposta para sua consulta. Em sistemas de EI, as buscas são, apenas, por trechos de textos.

Um sistema de RI vai além da procura por documentos. Ele deve ser capaz de fazer indexação dos documentos, realizar buscas e fazer um ranqueamento dos resultados obtidos. Além disso, pode fazer uma classificação dos documentos conforme seu conteúdo.

Algumas técnicas aplicadas à RI são, buscas por palavras chaves, indexação de arquivos, busca por padrões de comparação, utilização de n-gramas, linguagem natural, dentre outras técnicas. O uso de PLN pode melhorar os resultados por meio de interpretação da consulta e dos textos dos documentos. Porém, PLN é pouco usado em sistemas de buscas, principalmente por que as consultas dos usuários, normalmente, são feitas com palavras chaves e não com uma frase específica.

- Tradução Automática:

  Tradução automática, também chamada apenas de TA, é uma área da linguística computacional voltada para o desenvolvimento de softwares que sejam capazes de traduzir, automaticamente, textos de uma linguagem natural para outra. A tarefa de tradução entre duas linguagens não é simples. Não há tradutores automáticos no mercado que estejam livres de erros. As diferenças entre as características das linguagens, como regras gramaticais, sintaxe, morfologia, semântica e palavras com mais de um sentido são alguns dos principais empecilhos na construção de um tradutor automático. Mesmo com esses problemas, os sistemas de tradução automática tornaram-se populares. A maioria deles usam técnicas de estatística e corpus bilíngues. Porém, dessa forma os resultados da tradução são dependentes das características do corpus.

- Interpretação de Linguagem Natural:

  Ao receber dados de textos ou comandos em linguagem natural, o computador é capaz de interpretá-los. Dessa forma, a comunicação com computadores pode ser

feita de maneira mais fácil. Pode ser aplicado em dispositivos que fazem reconhecimento de voz para entender comandos.

- Resumo de Textos

  Usando técnicas e conceitos de PLN, alguns sistemas conseguem resumir um texto automaticamente, mantendo a coerência do conteúdo. Identifica-se pontos importantes nos textos originais, que são utilizados no resumo final. Geralmente, esses sistemas obtêm melhores resultados com textos que possuem uma estrutura bem definida, como artigos, documentos científicos etc.

- Correção de Textos

  PLN pode ser utilizado para fazer correção gramatical e ortográfica de textos. São utilizados conceitos de linguística, estatística e a utilização de corpus. Como exemplos, temos o corretor gramatical cogroo e os dispositivos de correção ortográfica e gramatical disponíveis em editores de textos.

2.2 PLN estatístico

O processamento estatístico de linguagem natural emprega conceitos de estatística para realizar o processamento de textos em linguagens naturais. Em PLN estatístico utiliza-se um corpus, modelos probabilísticos e técnicas de aprendizado de máquina. Corpus é uma base de dados com arquivos de textos em uma linguagem natural. A partir dele coleta-se características da linguagem, como sintaxe, morfologia etc. Os modelos probabilísticos são usados para gerar uma distribuição de probabilidade para linguagem. Analisando-se o corpus observa-se que a linguagem segue um padrão com relação às características citadas anteriormente. Esses padrões são modelados para um modelo estatístico. O modelo formado possui as características de um modelo de markov. O aprendizado de máquina é aplicado ao sistema para que este possa adquirir os conhecimentos necessários para processar um texto. O sistema de PLN estatístico tem que ser capaz de generalizar o conhecimento obtido, a partir do corpus, para todos as ocorrências da linguagem.

Com a utilização de estatística para processamento de textos, tenta-se resolver alguns problemas que são comuns em sistemas de PLN. Problemas como ambiguidade e análise da estrutura de sentenças, por exemplo, podem ser resolvidos com mais eficacia com o uso de modelos probabilísticos.

Ao usar este tipo de abordagem, os sistemas de PLN ficam restritos às linguagens que são descritas no corpus usado na construção do sistema. Além disso, o tipo de texto que compõe o corpus pode influenciar no resultado do processamento de um texto. Um corpus com textos que possuem características da linguagem escrita pode ser inadequado para processar textos com características da linguagem falada.

Apesar dessas restrições, PLN estatístico é bastante utilizado por sistemas que processam textos. Alguns conceitos para se fazer processamento estatístico de linguagem natural são explicados a seguir. São dados suas definições e seu papel em PLN estatístico.

2.3 Token

Um token é a menor unidade em que um texto pode ser dividido. Ele pode ser composto por caracteres, números e símbolos, e possui um significado no contexto da sentença em que aparece. Dessa forma, os tokens podem ser vistos como subsequências de um texto. Normalmente, toma-se como padrão que os tokens são as palavras que compõem os textos. E uma palavra é definida como um conjunto de caracteres consecutivos e ordenados entre dois espaços. Porém, um token pode ser definido pensando em como ele será útil no momento de processar o texto ao qual ele pertence. Logo, um token pode não ser uma palavra, mas um conjunto de palavras ou um caractere, por exemplo.

O processo de gerar tokens ou dividir um texto em tokens é chamado de *tokenization*. Em PLN os tokens são utilizados para dividir sentenças. Cada sentença é dividida em tokens para que cada token possa ser analisado separadamente e posteriormente determinar sua função na estrutura da sentença. *Tokenization* é aplicado ao corpus, antes que este seja empregado no processo de treinamento, e a textos que serão processados pelo aplicativo de PLN.

Apesar de parecer simples, este processo apresenta algumas dificuldades. Dentre essas dificuldades, uma das maiores é definir se uma subsequência do texto é, realmente, um token. Para exemplificar essa dificuldade, podemos analisar o trecho "comunicação homem-máquina". Neste caso fica difícil definir qual é o token para a subsequência "homem-máquina". Qual seria a maneira correta de classificar o token? "homem-máquina", "homemmáquina", "homem máquina"ou "homem"e "máquina"como dois tokens diferentes?

As pontuações no meio de sentenças são os elementos que mais dificultam o processo de *tokenization*. Essas pontuações são, normalmente, ignoradas. Porém, com a remoção dessas pontuações, informações sobre a estrutura da sentença podem ser perdidas. Por exemplo, um ponto pode significar uma abreviação e ignorá-lo irá prejudicar o resultado final da análise do texto. Os aplicativos de PLN devem aplicar uma solução eficaz para

esse problema, de forma a causar menos dano na análise do texto. A escolha de como tratar este problema deve ser feita por quem desenvolve o sistema.

2.4 n-grama

Um n-grama é definido como uma sequência de n elementos consecutivos, que estão contidos em outra sequência de tamanho maior ou igual a n. Ou seja, n-grama é uma subsequência de uma sequência de elementos. Geralmente, um n-grama pode ser uma sequência de n caracteres, podendo incluir espaços e pontuações, ou um conjunto de n palavras. Os n-gramas mais utilizados são os monograma, bigrama e trigrama, que são aqueles em que o valor de n é 1, 2 e 3, respectivamente. Os n-gramas em que n é maior ou igual a 4 são, comumente, chamados de n-grama. Na figura 1 podemos ver exemplos de palavras e sentenças divididas em n-gramas.

A utilização de n-gramas não fica restrita ao processamento de linguagem natural. Seu conceito pode ser utilizado para outros tipos de aplicações, pois, praticamente, qualquer tipo de dado pode ser processado para ser dividido em n-gramas, como arquivos de áudio ou imagem. Também pode ser usado para comparar palavras com o intuito de determinar se elas são iguais ou semelhantes. Esta técnica é utilizada em recuperação de informação. Além disso, n-gramas são aplicados também à correção ortográfica, reconhecimento de padrão, reconhecimento de voz, tradução automática, dentre outras aplicações da computação.

Em processamento estatístico de linguagem natural, o n-grama é usado para auxiliar na construção do modelo de probabilidade da linguagem descrita no corpus. A partir dos dados de aprendizagem, gera-se uma distribuição de probabilidade usando o conceito de n-gramas. O modelo de probabilidade gerado possui as características de um modelo de markov.

Exemplo de uma sequência de caracteres divididos em n-gramas. Neste caso, usa-se bigramas.
Sequência de caracteres: Choveu ontem
Os bigramas gerados para a sequência acima são: "Ch", "ho", "ov", "ve", "eu", "u ", " o", "on", "nt", "te" e "em"

Exemplo de uma sequência de palavras divididas em n-gramas. neste caso, usa-se trigramas.
Sequência de palavras: Fiquei cansado depois do jogo
Os trigramas gerados para a sequência acima são: "Fiquei cansado depois", "cansado depois do" e "depois do jogo"

Figura 1: Exemplo de divisão em n-gramas

2.5 Modelos de n-gramas

Em uma linguagem natural observa-se que algumas palavras tendem a seguir outras palavras. Dessa forma, pode-se assumir que "o contexto prévio pode influenciar na decisão de qual é a palavra seguinte"[MS99]. Baseado nesta ideia, o processamento estatístico de linguagem natural usa conceitos de probabilidade, mais precisamente, um modelo estocástico utilizando n-gramas, para prever a próxima palavra de uma sentença. Este modelo é conhecido como modelo de n-gramas, um modelo de probabilidade baseado em n-gramas que ajuda a determinar qual pode ser a próxima palavra de uma sentença, dada as palavras anteriores.

"Esta ideia remonta a um experimento de Claude Shannon. Sua ideia era a seguinte: dada uma sequência de letras (por exemplo, a sequência "por ex"), qual poderá ser a próxima letra? Por meio de treinamento, pode-se obter uma distribuição de probabilidade para a próxima letra: a=0,4, b=0,00001, c=0....; em que as probabilidades de todas as "próximas letras" possíveis somem 1.0"[GSC06].

Como este tipo de abordagem se baseia em frases vistas anteriormente para predizer qual será a próxima palavra de uma determinada frase, "não se pode eventualmente considerar cada sequência de texto separadamente, pois a maioria das vezes será encontrada uma frase que nunca foi vista antes, e assim não há nenhuma frase anterior que seja idêntica para basear as previsões, e mesmo se já foi visto o início da frase anteriormente, poderá ter um final diferente desta vez. Por essa razão, precisa-se de um método de agrupamento de frases já analisadas e que sejam similares, de modo a dar previsões razoáveis quanto as palavras que podem aparecer em seguida em um texto texto"[MS99]. Uma maneira possível de agrupá-las é supor que só o contexto local, mais precisamente, as últimas palavras anteriores, afetam a ocorrência da palavra seguinte. "Constrói-se um modelo em que todas as frases já vistas que têm as mesmas n - 1 últimas palavras são colocadas na mesma classe de equivalência, e obtém-se um modelo n-gramas (a última palavra do n-grama é a palavra que se quer prever), este modelo é semelhante a um modelo de markov de ordem (n - 1)"[MS99].

Para a construção de um aplicativo de PLN que usa modelos de n-gramas, usa-se uma grande quantidade de arquivos de textos, conhecidos como corpus, para que seja feito o modelo de n-gramas. O modelo é gerado durante um processo conhecido como aprendizagem de máquina. Após o aprendizado de máquina, o sistema será capaz de analisar textos baseado no modelo montado a partir do corpus.

2.6 O princípio da máxima entropia

"A noção de entropia é originária de estudos de termodinâmica, em que é utilizada para mensurar o grau de desordem de um sistema. O conceito de entropia da informação, introduzido por Shannon em seu artigo A Mathematical Theory of Communication (Uma teoria matemática da comunicação), em 1948, refere-se à incerteza probabilística"[GSC06], ou seja, está relacionada a falta de informações sobre um sistema que se quer modelar estatisticamente. Quanto menos informação é conhecida sobre o problema que se pretende gerar uma distribuição probabilística, maior a entropia.

"Pode-se dizer que há três tipos de incerteza: a determinística, em que não são conhecidos os estados que um sistema pode assumir; a entrópica, em que são conhecidos os estados possíveis, mas não as probabilidades de ocorrência de cada um deles; e a probabilística, em que são conhecidos não só os estados possíveis, mas também a distribuição de probabilidade para eles, associada a uma distribuição de probabilidade"[GSC06]. Cada uma delas está associada a um modelo de distribuição de probabilidade diferente, pois possuem diferenças em relação a quantidade de informações que são conhecidas.

Ao fazer uma modelagem estatística sem supor nada sobre as informações que não se tem, diz-se que trabalha-se com informações parciais. Para fazer inferências sobre informações parciais deve-se usar uma distribuição de probabilidade que tenha a maior entropia, considerando-se tudo o que é conhecido e não assumindo nada sobre o que é desconhecido. Dessa forma, procura-se formar uma distribuição de probabilidade que seja o mais uniforme possível. "Um exemplo simples pode ser observado quando se lança uma moeda, sem saber se a moeda é viciada ou não. A probabilidade mais imparcial, ou seja, sem considerar algo que não se sabe a priori, como, por exemplo, a moeda ser viciada, a ser atribuída a cada evento possível é de 1/2, retratando a incerteza por meio de uma distribuição uniforme"[GSC06].

2.7 Processamento de Linguagens Naturais e o Princípio da Máxima Entropia

Para fazer a modelagem estatística de uma linguagem é necessário a utilização de uma distribuição probabilística que seja consistente com os dados do corpus. Para isso, "muitos problemas em processamento de linguagens naturais podem ser reformulados como problemas de classificação, nas quais a tarefa é observar algum contexto linguístico $b \in B$ e predizer classes linguísticas corretas $a \in A$. Isto envolve construir o classificador $cl : B \to A$ que por sua vez pode ser implementado com uma distribuição p com uma

probabilidade condicional, de forma que $p(a|b)$ é a probabilidade de ocorrência da classe a dado algum contexto b .

Representam-se evidências com funções chamadas de predicados contextuais ou *features*. Se A = $\{a_1...a_n\}$ representa o conjunto de possíveis classes que se está interessado em predizer, e B representa o conjunto de possíveis contextos ou material textual que se pode observar, então um predicado contextual é uma função: $cp : B \rightarrow \{true, false\}$ que retorna true ou false, correspondendo à presença ou ausência de informações úteis em algum contexto, ou *histórico* $b \in B$. O conjunto exato de predicados contextuais $cp_1...cp_m$ que está disponível para uso varia de acordo com o problema, mas em cada problema, esses predicados devem ser fornecidos pelo usuário. Predicados contextuais são usados em *features*, que são funções da forma $f : A \times B \rightarrow \{0, 1\}$. Todas as *features* apresentadas aqui têm a forma

$$f_{cp,a'}(a, b) = \begin{cases} 1, & \text{se } a = a' \text{ e } cp(b) = true \\ 0, & \text{Caso contrário} \end{cases}$$

O conjunto de *features* utilizado em um problema é determinado por uma estratégia de seleção de *features*, que, em geral, é específico para cada problema"[GSC06]. [1]

Percebe-se, agora, qual a relação desta distribuição com o modelo de n-gramas. As n - 1 palavras de um n-grama representam os contextos $b \in B$ e a palavra que completa o n-grama representa as classes linguísticas $a \in A$, que é a palavra que queremos predizer. Dessa forma, a linguagem pode ser modelada completamente.

2.8 Corpus

Corpus é um conjunto de dados cujo conteúdo são grandes quantidades de textos com informações linguísticas. Os textos que compõem o corpus podem ser em uma única linguagem ou em várias. O corpus com mais de uma linguagem é usado, principalmente, para fazer traduções automáticas e costumam ter duas linguagens diferentes. Além disso, os textos do corpus podem possuir características da linguagem escrita ou da linguagem falada. A principal utilização de um corpus é para pesquisa linguística, de forma a observar características da linguagem, como padrões da linguagem, por exemplo.

[1] O texto original, em inglês, pode ser encontrado em[Rat98]

Os dados de um corpus devem ser legítimos, ou seja, não inventados e cobrir variações das características da língua que se deseja estudar. Por essa razão, a construção de um corpus envolve várias pessoas e pode levar um longo tempo para ser construído. Normalmente, os textos do corpus são extraídos de outros textos, que seguem as regras formais da língua.

Um corpus pode ser divido em tokens e além de possuir textos, pode conter anotações. Estas anotações são constituídas de informações sobre cada token que compõe o texto. As informações podem ser sobre a morfologia de cada token, como dizer se o token é substantivo ou adjetivo, por exemplo, informações sobre sintaxe e também pode-se dizer qual é a primitiva de cada token. Por exemplo, a primitiva de meninas é menino. São estas informações que ajudam a entender e observar os padrões na estrutura da linguagem.

O estudo do corpus possibilita coletar as características da linguagem estudada. Analisando essas características, nota-se que cada linguagem segue um certo padrão. Percebe-se que as palavras costumam aparecer nos mesmos lugares e ter a mesma função nas estruturas das sentenças. A partir destes padrões observados, pode-se fazer a modelagem probabilística da linguagem. Para isso, são utilizados o modelo de n-gramas e o princípio da máxima entropia.

Uma constatação importante ao analisar as linguagens é que algumas poucas palavras ocorrem com grande frequência nos textos. E por outro lado, uma grande quantidade de palavras são raras nos textos. As palavras que aparecem com menos frequência podem causar maior influência na análise de textos e tomadas de decisão sobre uma correção gramatical.

A função do corpus no processamento de linguagem natural é servir como dados de aprendizagem para o aplicativo que fará o processamento da linguagem. Porém, o uso de corpus cria uma restrição no sistema. As características presentes no corpus influenciam na análise dos textos. Um corpus com características da linguagem escrita pode ser inadequado para processar textos com características da linguagem falada. Além disso, uma mesma língua pode ter variações de região para região. Estas diferenças regionais é uma das dificuldades no processamento de linguagem natural.

2.9 Aprendizagem de Máquina

"A aprendizagem de máquina é um sub-campo da inteligência artificial dedicado ao desenvolvimento de algoritmos e técnicas que permitam ao computador aprender, isto é, que permitam ao computador aperfeiçoar seu desempenho em alguma tarefa"[Wik10b]. Considera-se que um computador é capaz de aprender se, baseado em uma experiência

obtida pela realização de uma certa tarefa, ele consegue aumentar sua performance ao realizar tarefas semelhantes futuramente.

"Algumas partes da aprendizagem de máquina estão intimamente ligadas à mineração de dados e estatística. Sua pesquisa foca nas propriedades dos métodos estatísticos, assim como sua complexidade computacional. Sua aplicação prática inclui o processamento de linguagem natural, motores de busca, diagnósticos médicos, bioinformática, reconhecimento de fala, reconhecimento de escrita, visão computacional, locomoção de robôs"[Wik10b] e jogos eletrônicos, por exemplo.

Para aprender, a máquina precisa adquirir conhecimentos de alguma fonte de dados com vários exemplos cobrindo todos os aspectos da linguagem. No processamento estatístico de linguagem natural, é utilizado um corpus como a fonte de informações. Dos corpus são extraídas as características da linguagem descrita e montado o modelo de n-gramas. Essas informações são armazenadas pelo aplicativo que fará processamento de linguagem natural.

O processo de aprendizagem, também chamado de treinamento, não deve servir apenas para memorizar as características da linguagem e o modelo probabilístico obtido, mas também, fazer com que a máquina seja capaz de generalizar essa base de conhecimento para que, em situações nunca vistas antes, ela possa tomar decisões corretas , baseado no que foi analisado no corpus.

3 Cogroo

Um exemplo de um programa que utiliza PLN estatístico é o cogroo. Um corretor gramatical para a língua portuguesa do Brasil, totalmente acoplável ao OpenOffice.org, desenvolvido por alunos da Universidade de São Paulo. O cogroo utiliza análise sintática, morfológica e regras gramaticas para detectar erros gramaticais em textos. O processo realizado pelo cogroo para a análise gramatical de um texto é dividido em fases, que são executadas uma após a outra.

A última versão disponível do cogroo foi desenvolvida utilizando o anotador apache UIMA[Com10], uma ferramenta para auxiliar no desenvolvimento de aplicativos de processamento de linguagem natural. Neste trabalho será usada e descrita a versão 3.0.5 do cogroo, a última versão desenvolvida sem usar o UIMA. Então, a partir deste momento, o termo cogroo referirá a versão 3.0.5 do cogroo.

A estrutura básica do cogroo pode ser dividida em três partes principais. Estas partes são a interface, o núcleo e os dados, como podemos ver na figura 2. A camada de interface tem como função promover a interação entre o sistema que irá utilizar o cogroo e a camada do núcleo. Essa camada foi projetada para interagir com o OppenOffice.org, porém, podemos modificá-la ou inserir novas funcionalidades para que o cogroo possa ser acoplado a outros dispositivos. O núcleo possui os módulos que fazem o processamento dos textos. A camada de dados contém os dados usados no aprendizado de máquina e durante a análise de textos. Seu conteúdo é o corpus e um um dicionário. O corpus usado foi o CETENFolha[cet10]. O dicionário é composto por palavras, com suas anotações morfológicas, pelo relacionamento entre palavras derivadas de outra palavra e por abreviaturas. Estas informações ajudam na fase de anotação morfológica, ao processar um texto.

O núcleo do cogroo possui um conjunto de módulos e cada módulo realiza uma fase no processo de correção de um texto. Para auxiliar na construção desses módulos, foram utilizadas duas ferramentas. O framework openNLP[ope10a], para auxilio no processamento da linguagem e o maxent[max10], para gerar o modelo estatístico de aprendizagem.

É importante citar que o cogroo também possui um módulo de treinamento e "algumas outras classes para realizar tarefas secundárias e interagir com o OpenNLP e o maxent. O módulo de treinamento tem como objetivo criar e usar modelos de aprendizagem de máquina para aprimorar a análise de texto"[cog10].

Como qualquer sistema de PLN, o corretor cogroo não está livre de erros. Alguns dos principais problemas encontrados pelo cogroo ao processar um texto são, detecção de limites de palavras e sentenças e ambiguidades nos sentidos das palavras. Além disso,

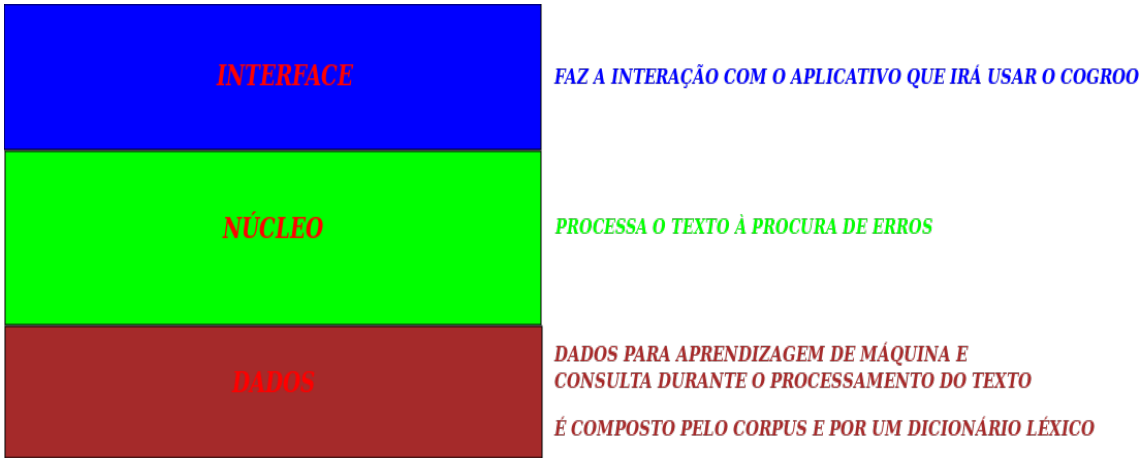

Figura 2: Estrutura geral do cogroo

ao analisar um texto, o cogroo pode não detectar erros presentes ou indicar que há erro em um trecho que está correto. Estes tipos de erros em um corretor gramatical são classificados como falsos negativos e falsos positivos, respectivamente.

- "Falsos negativos: o analisador de desvios não encontra um erro onde o erro existe"[GSC06].
- "Falsos positivos: o analisador de desvios afirma haver um erro onde não existe. É o tipo de erro cometido por corretores gramaticais que mais perturba os usuários, pois o usuário pode trocar uma estrutura correta por uma incorreta ou mesmo ficar receoso quanto à veracidade dos outros erros (outros falsos positivos ou desvios verdadeiros) assinalados pelo corretor, abalando sua confiança no mesmo"[GSC06].

3.1 OpenNLP

O OpenNLP é um framework, feito na linguagem java, para auxiliar no desenvolvimento de projetos de processamento de linguagem natural. Ele facilita fazer a divisão do projeto em módulos, tornando-o mais flexível. O framework possui módulos para fazer a detecção de sentenças, separação de tokens, anotações morfológicas, detecção de entidades nomeadas, agrupamento e análise sintática simples. Para o processo de aprendizagem desses módulos, o framework faz uso do pacote de aprendizado de máquina maxent. Os principais usuários da ferramenta OpenNLP são, pesquisadores e desenvolvedores que colaboram para o desenvolvimento de projetos de código livre sobre linguagens naturais.

O framework OpenNLP foi desenvolvido para a língua inglesa. Porém, seus módulos podem ser modificados e estendidos para se adaptarem à outras linguagens. Teoricamente, basta elaborar os dados de treinamento para o maxent, em qualquer língua. Além disso,

para o uso do framework no cogroo, foram realizadas especializações nos algoritmos de aprendizagem de máquina para que o OpenNLP pudesse tratar algumas particularidades dos dados de treinamento e da linguagem.

O OpenNLP sozinho não possui utilidade. Ele deve ser integrado ao software que fará o processamento de linguagem natural. No cogroo, o framework foi integrado aos módulos que compõem a camada do núcleo. Dessa forma, procurou-se deixar os módulos que constituem o cogroo o mais independentes possível, para facilitar trabalhos futuros.

3.2 Maxent

Maxent é um pacote que utiliza o conceito do modelo da máxima entropia para fazer aprendizado de máquina. Antes de ser usado no cogroo, ele recebeu algumas modificações para ser capaz de trabalhar com o português do Brasil.

3.3 Tipos de Erros Detectados Pelo Cogroo

Antes de mostrar os tipos de erros gramaticais que o cogroo é capaz de encontrar, devemos fazer uma distinção entre corretor gramatical e corretor ortográfico. Muitas pessoas não sabem a diferença entre as duas ferramentas e costumam achar que elas fazem o mesmo trabalho.

- "Corretor ortográfico: detecta erros de grafia nas palavras, por exemplo, a palavra "sugeito"com a letra "g", em vez de "sujeito"com a letra "j". Além disso, sugere uma medida corretiva para a solução do erro, que é a substituição da palavra grafada incorretamente pela palavra grafada corretamente"[Col10b].

- "Corretor gramatical: detecta erros nas relações entre as palavras, por exemplo, na sentença "Os menino estudam demais.", o corretor gramatical detecta o erro na concordância entre o artigo "os"e o substantivo "menino". E também sugere uma medida corretiva, como a substituição da palavra "menino"pela mesma palavra flexionada no plural, que é "meninos""[Col10b].

Dentre os tipos de erros gramaticais que o cogroo é capaz de detectar, os listados a seguir são os mais comuns:

- colocação pronominal:
  - uso de próclise

- concordância nominal:
  - concordância de gênero entre substantivos e adjetivos
  - concordância de número entre substantivos e adjetivos
  - concordância de gênero entre artigo e substantivo
  - concordância de número entre artigo e substantivo
- concordância entre sujeito e verbo
- concordância verbal:
  - Erros ao usar “fazer” indicando tempo
  - Erros ao usar “haver” no sentido de existir
- uso de crase:
  - ocorrência de crase antes de substantivo masculino singular ou plural
  - ocorrência de crase antes de verbo
  - ocorrência de crase em expressões indicativas de horas
  - ocorrência de crase em expressões indicativas de modo, tempo, lugar etc.
  - ocorrência de crase antes de pronomes de tratamento
  - ausência de crase em expressões como “em relação à”, “com relação à” e “devido à”
  - expressão “de segunda a sexta” e variantes
- regência nominal:
  - “valorização de”
- regência verbal:
  - “assistir” com o sentido de presenciar
  - “evitar” ou “usufruir”
  - “habituar”
  - “obedecer” e “desobedecer”
  - “preferir isso a aquilo”

- erros comuns da língua portuguesa escrita:
  - uso de “meio” como adjetivo
  - uso de “meio” como advérbio
  - emprego de “mim” como pronome pessoal do caso reto
  - emprego de “mim” e “ti”
  - emprego de vírgulas para separar expressões como “ou seja” e “no entanto” do fluxo da sentença
  - redundância no uso do verbo “haver”

3.4 Módulos do Núcleo do Cogroo

Esta versão do cogroo tem seu núcleo composto por módulos e cada módulo realiza uma fase no processo de correção de um texto. Portanto, todos conceitos e técnicas de PLN estatístico estão no núcleo. Por esta razão, vamos descrever alguns de seus principais módulos.

A entrada do processo de correção é o texto que o usuário quer corrigir. Durante o fluxo do processo, cada módulo executa uma tarefa diferente com o texto. A saída de cada módulo é encaminhada para a entrada do próximo módulo no fluxo. Ao final, obtêm-se os erros encontrados e sugestões de correção. Na figura 3 podemos ver a ilustração desse fluxo, mostrando os principais módulos que realizam o processo.

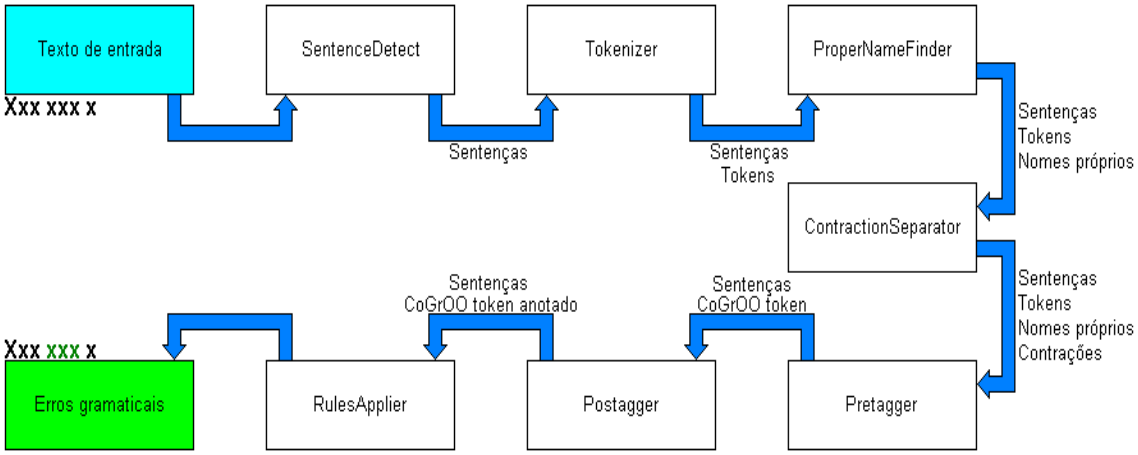

Figura 3: Fluxo de processamento de texto no cogroo[2]

[2]Ilustração gentilmente cedida por Edwin Miguel.

Abaixo há a descrição dos principais módulos do núcleo do cogroo. Será mostrado qual sua função, técnica aplicada e quais são os dados de entrada e saída para cada um. Antes de serem usados, os módulos *SentenceDetect*, *Tokenizer*, *ProperNameFinder* e *Postagger* passaram pela fase de aprendizado de máquina. Os dados de aprendizado utilizado foram o corpus CETENFolha e um dicionário léxico.

- *SentenceDetect*:
  É o módulo responsável por fazer a separação do texto em sentenças. Ele recebe como entrada o texto a ser corrigido. A saída deve ser uma lista de sentenças na mesma ordem do texto original. Para cada sentença é marcado seu início e seu final. Para delimitar uma sentença é preciso determinar quais são os padrões no texto que indicam que foi encontrado o final da sentença. A técnica usada no cogroo é de considerar sinais de pontuação como padrão para identificar o final de uma sentença. Os sinais de pontuação usados para esta tarefa são, ponto final("."), exclamação("!") e interrogação("?"). Porém, esta técnica pode ocasionar erros no resultado final do processamento por causa de ambiguidades que o uso de uma pontuação pode causar. Por exemplo, na sentença, "O Sr. João chegou.", o ponto do trecho "Sr."pode ser confundido com ponto de final de sentença.

- *Tokenizer*:
  O *Tokenizer* tem a função de separar uma sentença em tokens. A entrada deve ser uma lista de sentenças gerada pelo detector de sentenças. A saída é uma lista com todos tokens encontrados nas sentenças, ordenados na mesma sequência em que aparecem na entrada. Ele marca onde é o início e fim de cada token. No caso do cogroo, considera-se que um token pode ser uma palavra, números ou pontuações. As principais dificuldades encontradas pelo módulo são, detectar tokens que são nomes próprios e trabalhar com pontuações. Além disso, o resultado final pode ser influenciado por erros no conteúdo do corpus.

- *ProperNameFinder*:
  Módulo para identificar nomes próprios. Ele recebe como entrada uma lista de tokens que foi gerada pelo detector de tokens. Ele gera uma lista com “os possíveis nomes próprios, incluindo nomes de pessoas, lugares e organizações, mas sem os classificar”[GSC06]. Esta tarefa é importante para que se possa fazer uma melhor classificação morfológica dos tokens e, dessa forma, obter melhor resultado na correção do texto.

- *ContractionSeparator*:
  O módulo recebe como entrada uma lista de tokens. Ele deve identificar contrações e gerar como saída uma lista de tokens com as contrações expandidas.

- *Pretagger*:
  Recebe todas as listas anteriores e gera uma lista de *CogrooTokens*, que contém todos os tokens, incluindo se são nomes próprios ou contrações. Ele prepara os tokens para receber anotações morfológicas.

- *Postagger*:
  O módulo recebe como entrada as sentenças e a lista gerada pelo módulo *Pretagger*. Esta lista possui os tokens ordenados conforme o texto original da entrada. Deve ser devolvida uma lista de etiquetas morfológicas associadas a cada token da entrada. A marcação é feita de acordo classificação morfológica do token no contexto na sentença. "Para esta tarefa, são feitos cálculos estatísticos, cujos parâmetros foram obtidos de um corpus já etiquetado"[GSC06]. Um bom desempenho na anotação morfológica de cada token possui grande importância para o momento da aplicação das regras gramaticais. "Estima-se que etiquetadores baseados em estatística possuem taxa de acerto entre 95 e 97%"[GSC06].

- *Rulesapplier*:
  Módulo responsável por aplicar as regras gramaticais à lista de tokens e gerar uma lista de erros, quando existir. As regras são armazenadas em um arquivo XML. Dessa forma, elas seguem uma certa estrutura definida em um arquivo esquema de XML (XML Schema). A entrada do módulo deve ser os tokens com suas anotações morfológicas, sintagmas e agrupamentos sujeito verbo(Estes dois últimos serão explicados a seguir). Como saída, o módulo devolve uma estrutura de dados contendo a mensagem de erro, o trecho da sentença em que o erro foi encontrado e , quando possível, sugestões para correção do erro.

Além desses módulos citados acima, o cogroo possui mais dois módulos que devem ser citados por suas importâncias no processo de análise. O primeiro é um módulo chamado de agrupador, que tem como função agrupar tokens já etiquetados, que juntos podem ser considerados como um único componente da sentença, possuindo um significado e uma função na estrutura da sentença. Estes agrupamentos são conhecidos como sintagmas. Deve ser identificado dois tipos de sintagmas, os nominais, que tem como núcleo um substantivo, e os verbais, que tem como núcleo um verbo. O agrupador recebe como

entrada uma lista de tokens com suas etiquetas morfológica. A saída será informações que indicam onde ocorrem os sintagmas.

O segundo módulo é o analisador sintático simples, que serve para detectar agrupamento entre sujeito e verbo. Como entrada ele recebe os tokens e como saída, ele devolve etiquetas que informam as formações sujeito verbo.

4 Emacs

Nesta seção será descrito o editor emacs. O principal foco são nas características e funcionalidades oferecidas pelo emacs e emacs lisp que foram essenciais para a implementação do acoplamento. Será explicada as características da linguagem emacs lisp, como criar funções e novos comandos relacionados a essas funções, como trabalhar com subprocessos no emacs e como adicionar extensões ao emacs. Estes, foram alguns dos principais conceitos utilizados no acoplamento.

O emacs é um editor de texto disponível para diversos sistemas operacionais, como Linux, windows e mac, por exemplo. Sua versão mais popular é o GNU Emacs, que, normalmente, é chamado apenas de emacs.

Apesar de ser denominado como um editor de texto, o emacs é capaz de oferecer várias funcionalidades que vão além da edição de textos. Ele pode ser um poderoso ambiente de trabalho, sendo usado para uma grande diversidade de tipos de projetos. Além disso, no emacs podemos usar comandos shell, acessar internet, ler e-mail, manipular arquivos, compilar códigos e efetuar outras várias atividades. Os principais usuários do emacs são programadores e pessoas que escrevem textos técnicos.

Uma das principais características do emacs é ser facilmente extensível e customizável, permitindo que os usuários o configure, podendo criar comandos próprios, alterar teclas de atalho etc. Por esta razão, há uma grande quantidade de extensões para o emacs disponíveis para os usuários. Estas extensões são construídas utilizando a linguagem de programação emacs lisp, um dialeto da linguagem lisp. Neste trabalho, o emacs lisp foi usado para a construção de uma interface para prover a interação do emacs com o cogroo.

Normalmente, o emacs aceita comandos escritos ou combinações de teclas para efetuar alguma operação. Uma operação, como mover um caractere ou copiar uma sequência, por exemplo, corresponde a uma função em emacs lisp que realiza a tarefa.

4.1 Emacs Lisp

Emacs Lisp é uma linguagem de programação, sendo um dialeto da linguagem lisp. É uma linguagem baseada nos conceitos de listas e notação prefixa. Uma função em emacs lisp é escrita como uma lista em que, o primeiro elemento da lista é o nome da função e os demais elementos são os parâmetros da função. Para manipular dados, podemos criar listas para armazená-los. Para exemplificar esta ideia de trabalhar com lista, podemos analisar o comando (+ 1 2), escrito em emacs lisp. Este comando é uma lista em que o primeiro elemento é a função de adição e os inteiros 1 e 2, que são o segundo e terceiro

elementos da lista, são os parâmetros a ser passado para a função de adição.

A linguagem foi construída para ser usada na construção das versões do emacs. Eles são construídos, principalmente, com emacs lisp e a linguagem de programação C, em alguns trechos. Além disso, a linguagem é usada para estender e personalizar o emacs. Para trabalhar com emacs lisp, o emacs possui um interpretador de emacs lisp.

A linguagem emacs lisp oferece maneiras de criar novos comandos, iniciar e gerenciar processos, associar comandos a funções em emacs lisp, dentre outras funcionalidades. A principal característica da linguagem é a facilidade para manipular textos por meio de comandos e expressão regular. Estas funcionalidades foram usadas no trabalho para manipular o texto a ser corrigido.

4.2 Comandos no Emacs

No emacs o usuário pode usar comandos por meio de combinações de teclas do teclado ou digitando o nome do comando. Para digitar um comando, o usuário deve pressionar *alt* e *x* simultaneamente, escrever o comando desejado e então pressionar *enter*. Todos os comandos do emacs estão associados a alguma função em emacs lisp que executa alguma operação. Para relacionar um comando a uma função o emacs utiliza uma estrutura chamada *keymap*. Esta estrutura mapeia um comando para uma função.

Utilizando emacs lisp é possível manipular o *keymap*. Dessa forma, podemos remover comandos, criar novos comandos e associar um comando a outra função. Por intermédio do *keymap*, também é possível tratar eventos, como clique de algum botão do mouse ou o pressionamento de uma tecla, por exemplo. Da mesma maneira que os comandos são relacionados a alguma função, os eventos também podem ser mapeados para uma função.

O *keymap* foi usado no trabalho para criar comandos que possibilitam iniciar e encerrar o corretor gramatical, corrigir trechos do texto e para capturar eventos do mouse e do teclado.

4.3 Arquivo .emacs

O .emacs é um arquivo de inicialização que fica no diretório *home* do usuário e que o emacs carrega quando é iniciado. Ele também pode estar com o nome .emacs.el ou init.el. Neste último caso, o init.el fica em um subdiretório da *home* chamado .emacs.d. O usuário também tem a opção de compilá-lo e gerar um arquivo .emacs.elc ou init.elc.

Por meio do arquivo de inicialização, o usuário pode personalizar seu emacs, mudar comandos, criar comandos, carregar extensões, especificar comandos para serem executa-

dos ao iniciar o emacs etc. Comandos que são usados dentro do emacs também podem ser colocados no .emacs. No contexto deste trabalho, o .emacs será usado para que o emacs carregue a extensão que fará a correção gramatical.

4.4 Adicionando uma extensão ao emacs

Para carregar uma extensão que não está acoplada ao emacs, ou seja, que não é instalada no mesmo momento da instalação do emacs, usa-se o arquivo de inicialização .emacs. Neste arquivo é preciso especificar o diretório onde se encontra a extensão que irá ser usada. O emacs possui uma lista em que armazena estes diretórios. Por meio do .emacs, pode-se adicionar ou remover diretórios dessa lista. Além disto, deve ser especificado, no .emacs, o nome da extensão que deve ser carregada. Estas duas tarefas são realizadas com comandos da linguagem emacs lisp. Dessa forma, ao ser iniciado o emacs irá procurar pelas extensões especificadas nos diretórios contidos na lista. Esta abordagem foi usada para fazer a instalação da extensão que faz a correção gramatical.

4.5 Subprocessos no emacs

Em um sistema operacional, um processo pode ser definido, de uma forma mais geral, como sendo um programa em execução. Ou ainda, como tarefas em execução que trabalham juntas para chegar a um determinado objetivo. Todos aplicativos, ao serem executados, geram processos e um aplicativo pode conter vários processos.

Para que os processos possam realizar seus objetivos, eles utilizam recursos do computador como, processador, memória etc. Nos computadores há vários processos sendo executados simultaneamente, ou seja, utilizando os mesmos recursos.

O emacs é executado em um único processo. Porém, programas desenvolvidos com emacs lisp podem criar novos processos por meio de comandos. Estes comandos são capazes de invocar programas externos ao emacs, que irão executar no processo que foi criado. No emacs, estes processos são chamados de subprocessos ou processos filhos e o processo pai é o processo em que o emacs é executado.

Emacs lisp possibilita criar dois tipos de processos, síncrono e assíncrono. Eles podem ser criados, respectivamente, pelas funções *call-process* e *start-process*, que recebem como principal parâmetro o nome do programa externo a ser executado. Ao criar um subprocesso síncrono, o programa que criou o subprocesso precisa esperar que o subprocesso termine sua execução. Um subprocesso assíncrono, ao ser criado, será executado paralelamente ao programa que o criou.

Os programas feitos em emacs lisp que criam um subprocesso, possuem uma referência para este. Com esta referência, o programa que criou o subprocesso é capaz de se comunicar e controlar o subprocesso por meio de envio de sinais, recebendo saídas do processo, enviando entradas para o subprocesso, ou obtendo o status do processo. Para a implementação do projeto, usou-se um subprocesso assíncrono gerado com emacs lisp para executar o cogroo.

5 Processo de Acoplamento

Para implementar o acoplamento é necessário fazer o emacs interagir com o cogroo. Por utilizarem linguagens diferentes, foi preciso pesquisar alguma maneira de fazer os dois sistemas se comunicarem. Foram feitas tentativas com *sockets* e com um aplicativo chamado jlinker[Des10]. Porém, o jlinker funciona apenas com a linguagem common lisp. Apesar da utilização de *sockets* funcionar, no final foi usado o conceito de subprocesso do emacs, que possibilita o emacs executar um programa externo. Esta ideia foi usada para executar o cogroo em um subprocesso do emacs. Dessa forma, será possível manter uma comunicação com o cogroo e solicitar a correção de um texto, quando necessário.

Para providenciar a interação entre o emacs e o cogroo foram criadas duas interfaces. Uma interface do lado do emacs e outra do lado do cogroo. As interfaces interagem entre si para prover a correção do texto. A interface do lado do emacs é um arquivo escrito com a linguagem emacs lisp chamado *cogroo.el*, que após ser compilado pelo emacs gera o arquivo *cogroo.elc*. No lado no cogroo foi criada a interface *CorretorGramatical.java*, que intermedeia a comunicação da interface do emacs com o núcleo cogroo. Abaixo há uma descrição mais detalhada das interfaces.

5.1 Interfaces Para Prover Comunicação

O arquivo cogroo.el é uma interface para que o emacs possa interagir com o cogroo. A interface foi construída usando a linguagem emacs lisp. Como todas as extensões do emacs, ela é carregada quando o emacs é iniciado. Para isso, devemos especificar no arquivo de inicialização .emacs, que a extensão cogroo.el deve ser carregada.

Sua principal função é iniciar e manter uma conexão com o cogroo. Para executar o cogroo há uma função que usa o comando do emacs lisp que cria subprocessos. Então, cria-se um subprocesso assíncrono que invoca o cogroo. A partir desse momento, o emacs consegue se comunicar com o cogroo por meio de mensagens enviadas pela interface cogroo.el à interface do cogroo. Porém, o emacs não consegue invocar o cogroo diretamente. Para resolver este problema foi usado um script chamado cogrooemacs, que será explicado a seguir.

Além de iniciar o cogroo e manter uma comunicação, o arquivo cogroo.el possui outras funções. Ele provê comandos para que o usuário possa solicitar a correção de um trecho do texto. Para criar estes comandos foi utilizada estrutura *keymap*. Cada comando está associado a alguma função que executa uma tarefa. A interface também tem a função de capturar o trecho de texto que usuário quer corrigir e enviar ao cogroo. Após a correção

de um texto pelo cogroo, a interface cogroo.el deve ler os erros encontrados e destacá-los no texto que está no emacs.

O arquivo CorretorGramatical.java é a interface feita para o cogroo interagir, especificamente, com a interface cogroo.el do emacs. A principal função dessa interface é receber os textos enviados pelo emacs, encaminhá-los ao núcleo do cogroo, para ser realizada a correção gramatical, e devolver ao emacs os erros que foram encontrados. A interface CorretorGramatical foi feita usando a linguagem de programação java.

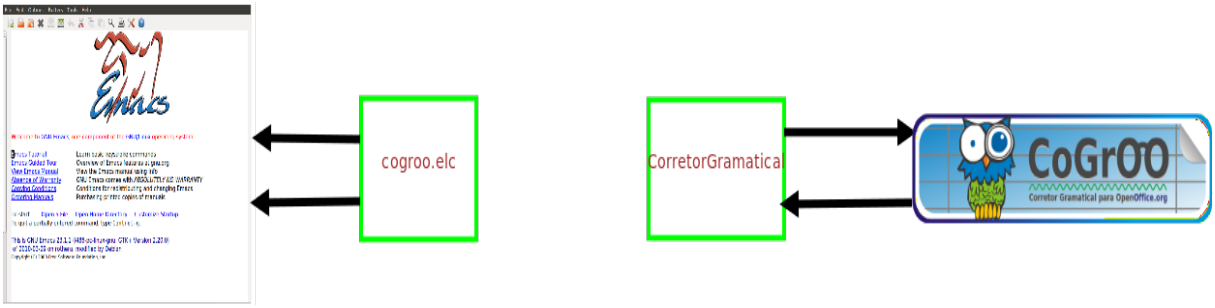

Figura 4: Interfaces para a interação entre o emacs e o cogroo

5.2 Script cogrooemacs

As interfaces cogroo.el e CorretorGramatical.java não se comunicam diretamente. Para executar o cogroo, o emacs cria um processo para executar o script cogrooemacs. Então, este script inicia o corretor cogroo e transmite as mensagens de uma interface à outra.

O emacs consegue enviar e receber mensagens por meio do subprocesso que foi criado para executar o script cogrooemacs, e este, encaminha as mensagens à interface do cogroo. A interface do cogroo recebe e envia mensagens por meio dos comandos de entrada e saída padrões da linguagem java. Na figura 5 podemos ver o modelo do acoplamento com o script e as interfaces.

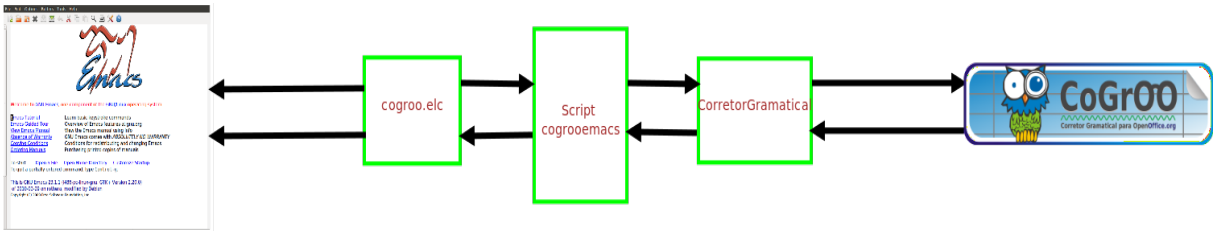

Figura 5: script cogrooemacs para intermediar a comunicação entre as interfaces

5.3 Arquivos auxiliares

As interfaces já conseguem se comunicar usando um script e enviando mensagens, porém, os textos a serem corrigidos e os erros encontrados pelo cogroo ainda não são enviados de um sistema para o outro. Apesar do emacs conseguir enviar e receber mensagens, o envio dos textos e dos erros por meio de mensagens fica inconveniente, pois receber o texto como entrada no cogroo fica mais difícil. Para resolver este problema foram usados arquivos auxiliares. Para enviar um texto do emacs para o cogroo, usa-se o arquivo cogrootexto, que fica armazenado no diretório *tmp*. E para enviar os erros encontrados pelo cogroo para o emacs, usa-se o arquivo cogrooerros, que também fica armazenado no diretório *tmp*.

Quando o usuário solicita ao emacs a correção de um trecho do texto, este trecho é gravado no arquivo cogrootexto pela interface cogroo.el. Após gravar o texto, o emacs envia uma mensagem para o corretor corrigir o texto. A interface do cogroo lê o texto do arquivo cogrootexto e o envia ao núcleo do cogroo, para realizar a correção. Após a correção, a interface do cogroo grava os erros encontrados no arquivo cogrooerros. Então, a interface do cogroo envia uma mensagem à interface do emacs, avisando que o texto já foi corrigido. O emacs lê os erros do arquivo cogrooerros e os destacam no *buffer* que contém o texto que o usuário solicitou a correção. Na figura 6 podemos ver a ilustração do modelo do acoplamento com os arquivos auxiliares.

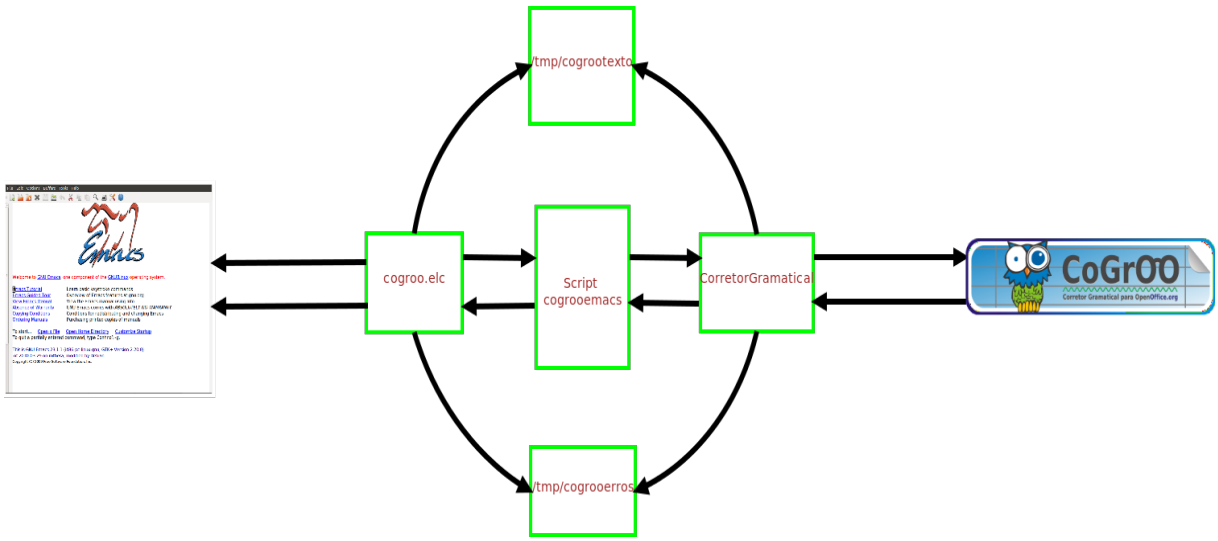

Figura 6: Inclusão dos arquivos auxiliares para o envio de textos e erros

5.4 Modelo do Acoplamento

Com essas definições, o modelo do acoplamento está completo. Na figura 7 podemos ver o modelo completo com os passos necessários para a correção de um texto.

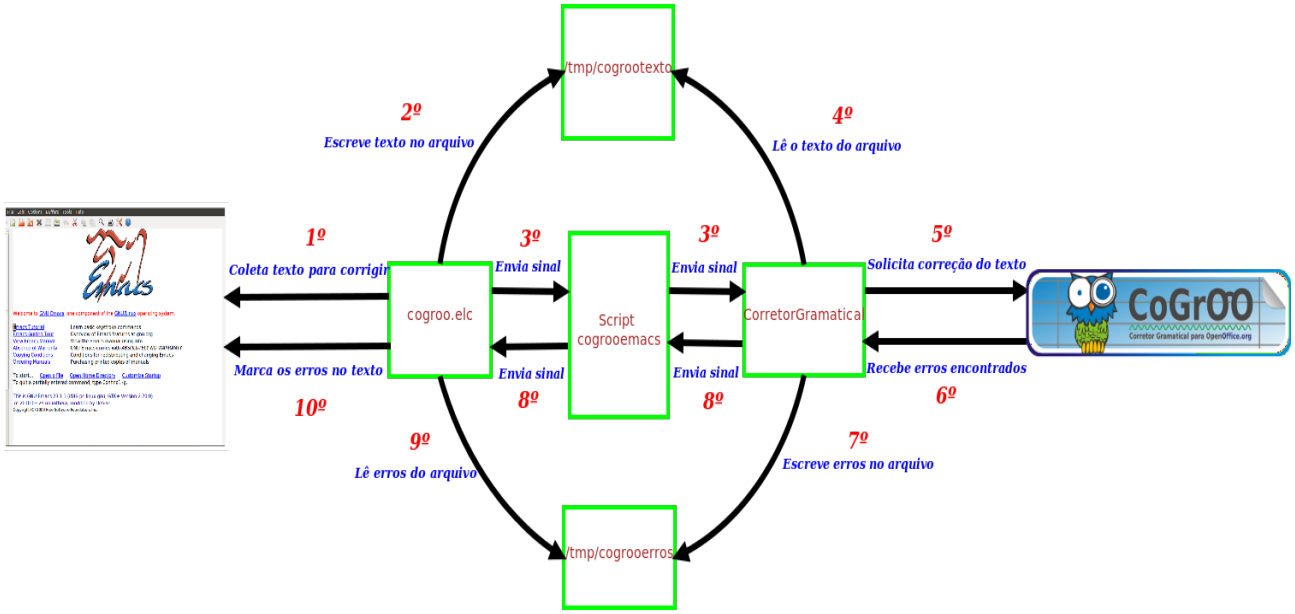

Figura 7: Modelo do acoplamento. Os números indicam a sequência de passos para corrigir um texto

5.5 Instalando o Corretor

O corretor gramatical para o emacs já possui uma versão que pode ser disponibilizada aos usuários. Esta versão do corretor foi testada apenas nas versões 23.1.1 e 22.1.1 do GNU emacs.

Para instalar esta primeira versão, o usuário deve seguir os seguintes passos. Descompactar o pacote *cogrooemacs.tar.gz*, que contém o corretor, em algum diretório da escolha do usuário. Após descompactar, será criado o diretório *cogrooemacs*. Por meio do console, o usuário deve seguir até o diretório gerado e executar o arquivo *install*, que irá criar o script *cogrooemacs* no diretório */usr/bin*.

Após o procedimento citado acima, o usuário deve acrescentar ao arquivo .emacs os comandos *(add-to-list 'load-path "/caminho/para o/diretório/cogrooemacs")* e *(load "cogroo")*. O primeiro comando indica ao emacs para adicionar na lista chamada *load-path* o diretório onde o corretor foi descompactado. Esta é a lista que armazena os diretórios onde o emacs deve procurar por extensões. O segundo comando pede ao emacs

para carregar a extensão *cogroo.el*, que é a interface do lado do emacs feita com emacs lisp. Ao escrever o comando *(load "cogroo")*, não é necessário especificar a extensão do arquivo que será carregado. Por esta razão colocamos apenas o nome *cogroo*. Após estes passos, o corretor já poderá ser usado no emacs.

Para remover o corretor, o usuário deve apagar os comandos colocados no seu *.emacs* e executar o arquivo *remove*, que está no diretório *cogrooemacs*, que foi gerado ao descompactar o corretor.

5.6 Testes

- Envio de textos e erros:
  Para que a correção gramatical do texto seja correta, é essencial que o texto a ser corrigido seja enviado corretamente do emacs para o cogroo. Para testar se o emacs envia o texto corretamente para o cogroo, foi necessário analisar o arquivo *cogroo-texto* para ver se o texto gravado nele, realmente, era o texto o qual foi solicitada a correção. Também foi verificado se o cogroo lia corretamente o texto do arquivo *cogrootexto*. Nesta etapa não foram encontrados erros. Ou seja, os textos são enviados corretamente, sem perda de informações.

  Para testar o envio de erros encontrados no texto, do cogroo para o emacs, foi usada a mesma ideia, com o arquivo *cogrooerros*. Também não há perda de informações durante o envio dos erros.

  Além do envio dos textos e erros por meio de arquivos, foi testado também o envio diretamente entre o emacs e cogroo por meio do envio de mensagens para processos e o uso de entrada e saída padrão da linguagem java. Porém, esta maneira mostrou-se inconveniente no momento de receber os dados de entrada, tanto do lado do cogroo como do emacs.

  Desta forma, o uso de arquivos auxiliares é a maneira mais eficaz e eficiente encontrada até o momento. Pretendo pesquisar novas maneiras para manipular os textos e os erros que possam ser mais eficiente.

- Destaque dos erros:
  Foram feitos testes para verificar se o emacs estava destacando os erros corretamente. Ele destaca um erro da posição inicial até a posição final do erro, usando uma linha para sublinhar. Estas posições são indicadas por dois números inteiros que são passados ao emacs pelo cogroo. Para analisar esta etapa, foi preciso verificar, no

cogroo, as posições dos erros e ver se o emacs destacava corretamente conforme estas posições. Além do destaque que sublinha o erro, também foi verificado se o emacs mostra corretamente as informações sobre os erros encontrados. Em todos os testes feitos não foram encontrados erros.

- Falsos positivos e falsos negativos:
  Apesar destes erros serem do cogroo, não fazendo parte da minha implementação, fiz alguns testes para encontrar casos de falsos positivos e falsos negativos. Identificando esses erros por parte do cogroo, pode-se aperfeiçoá-lo fazendo aprendizado de máquina com os falsos positivos e negativos encontrados.

  Foram Encontrados alguns falsos positivos relacionados, principalmente, a concordância entre sujeito e verbo. A principal dificuldade é trabalhar com sujeito composto. Neste caso ocorreram casos de falsos negativos. Para a frase "João e Maria foi para a casa", por exemplo, não é detectado erro de concordância entre sujeito e verbo. Alguns outros casos de falso negativo ocorreram no uso de crase.

6 Resultados e Produtos Obtidos

O emacs já consegue usar o cogroo para corrigir textos, porém algumas melhorias ainda podem ser feitas. A correção de um texto não pode ser inconveniente para o usuário, seja demorando muito para terminar ou usando grande quantidade de recursos do computador. O ideal é que a correção seja imperceptível ao usuário, sendo feita de maneira automática. Dessa forma, o principal ponto do trabalho era usar o corretor gramatical cogroo de maneira eficiente e destacar corretamente os erros encontrados. Para usar o cogroo, foi implementado o modelo de acoplamento citado acima. O destaque dos erros é feito por meio de uma linha que sublinha o trecho com erro. Porém, a correção ainda não ocorre de forma automática.

Para iniciar o cogroo no emacs, o usuário tem duas opções. O cogroo pode ser iniciado automaticamente no mesmo momento em que o emacs é iniciado ou ser iniciado, a qualquer momento, de dentro do emacs. Na última opção, o usuário deve usar um comando para iniciar o corretor. Para habilitar a iniciação automática do cogroo, o usuário deve colocar um comando no arquivo .emacs, que indicará ao emacs para iniciar o cogroo. Abaixo podemos ver os comandos disponíveis aos usuários.

- Comandos para serem usados dentro do emacs

  - *cogroo-inicia* - Para iniciar o corretor de dentro do emacs. O corretor pode ser iniciado a qualquer momento. E não é permitido iniciar dois corretores ao mesmo tempo.
  - *cogroo-corrige* - Este comando pode ser usado para corrigir o parágrafo em que o marcador estiver ou um texto selecionado pelo usuário.
  - *cogroo-finaliza* - Comando usado para encerrar o cogroo, ou seja, parar sua execução. Dessa forma, o usuário pode usar o corretor apenas quando quiser.

- Comandos para serem usados no arquivo .emacs
  - *(setq cogroo-inicia-automaticamente t)* - Indica ao emacs para iniciar o cogroo automaticamente.
  - *(setq cogroo-corrige-automaticamente t)* - Indica que um parágrafo deve ser corrigido automaticamente quando o usuário pressionar seta para baixo. Quando o marcador passar pelo fim de um parágrafo, este será corrigido.

Além do erro ser sublinhado, ao passar o *mouse* sobre o texto destacado, o emacs mostra uma janela contendo a regra gramatical que foi desrespeitada, sugestões de correção, quando possível, e uma explicação de como aceitar a sugestão. O cogroo nem sempre é capaz de sugerir uma correção e em alguns casos, ele sugere duas correções.

Fiz alguns testes do corretor na minha própria monografia e para ilustrar como o corretor funciona podemos ver nas figuras seguintes, exemplos que nos mostram como o emacs destaca o texto e como ele mostra as informações sobre o erro.

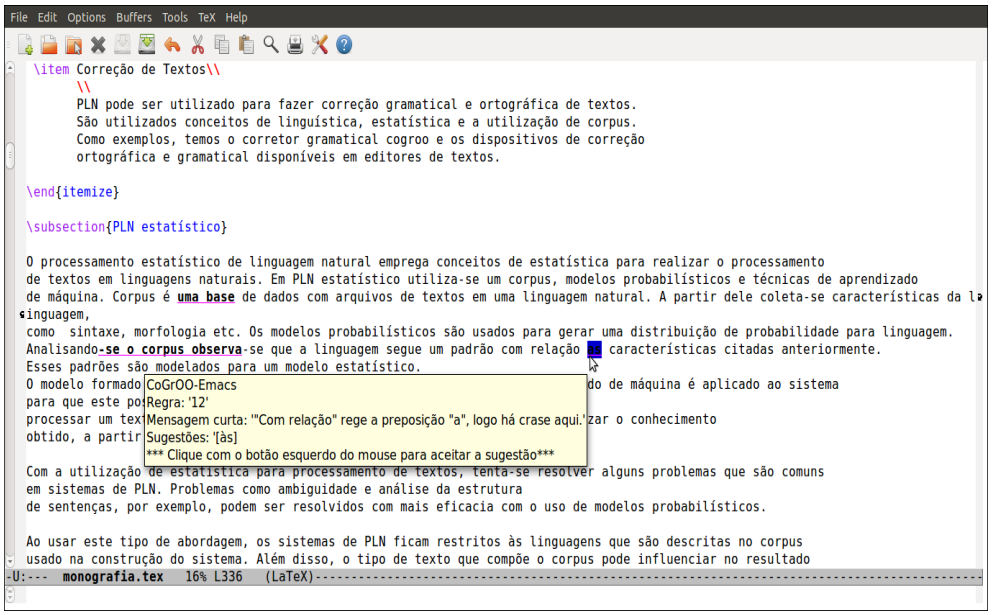

Figura 8: Janela mostrando a explicação do erro e uma sugestão de correção. Neste caso foi encontrado um erro com relação ao uso de crase.

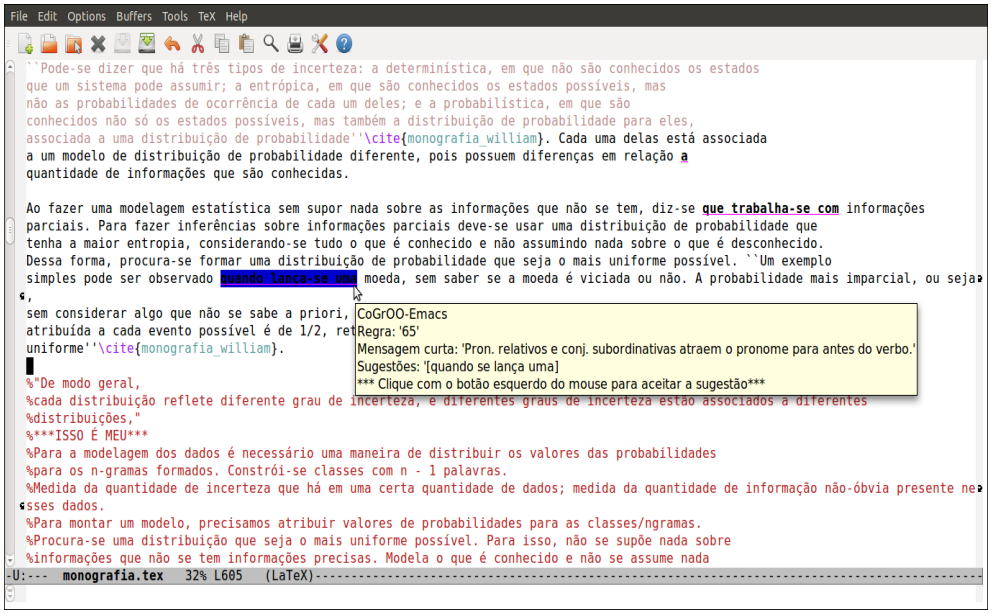

Figura 9: Erro encontrado no trecho "...quando lança-se uma moeda..."

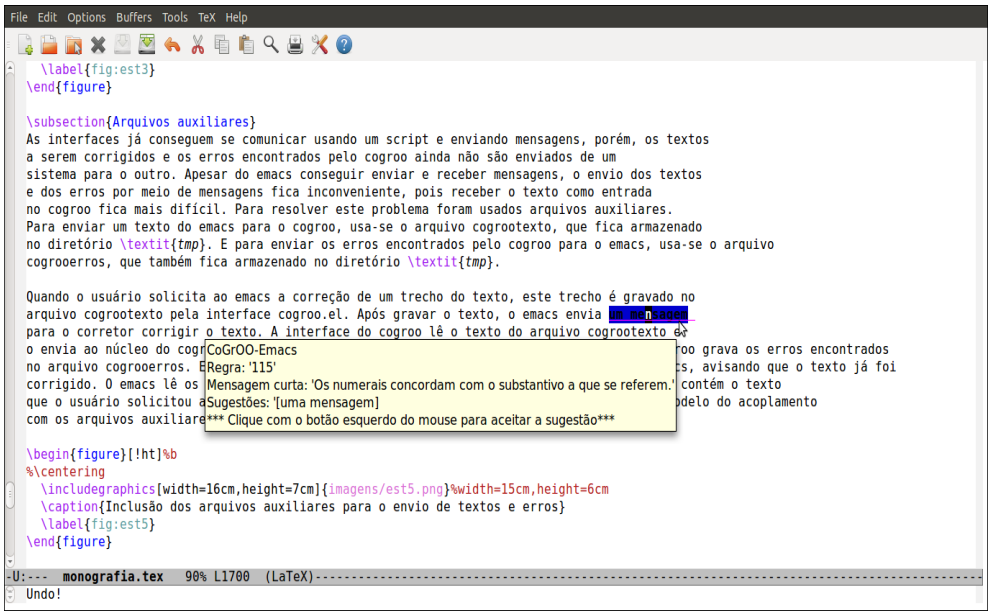

Figura 10: Erro de concordância no trecho "...um mensagem". Neste caso, ele sugere a correção para "uma mensagem".

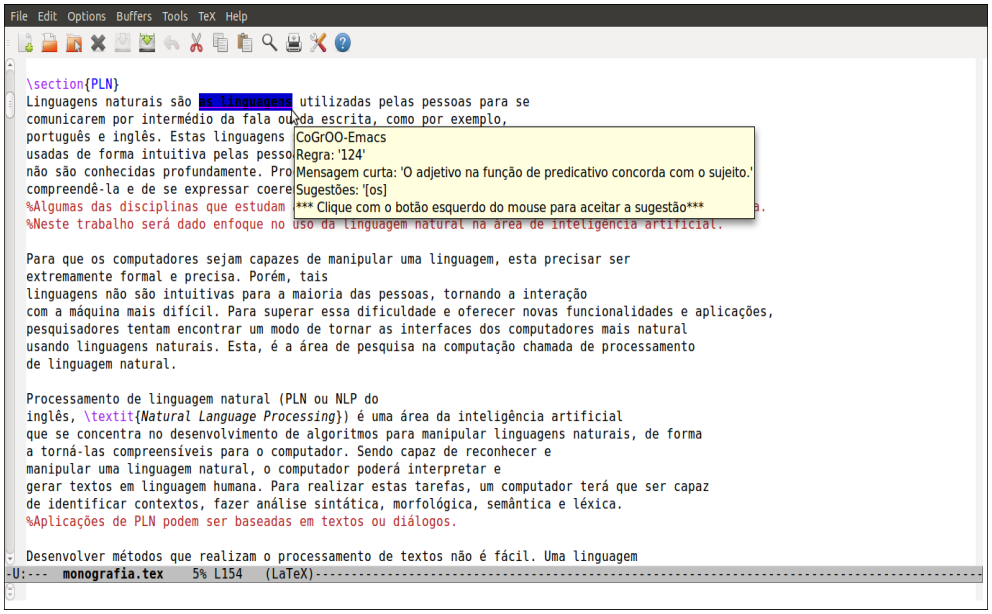

Figura 11: Neste caso ocorre um falso positivo. Ele indica um erro onde não há erro.

Referências

[cet10] Cetenfolha. Disponível em: http://www.linguateca.pt/cetenfolha/index_info.html., Acesso em: 2010.

[Cha10] Robert J. Chassell. An introduction to programming in emacs lisp. Disponível em: http://www.gnu.org/software/emacs/emacs-lisp-intro/html_node/index.html., Acesso em: 2010.

[cog10] Cogroo. Disponível em: http://ccsl.ime.usp.br/wiki/index.php/SugestoesCogroo., Acesso em: 2010.

[Col10a] William Colen. Cogroo - corretor gramatical acoplável ao openoffice.org. Disponível em: http://cogroo.sourceforge.net/., Acesso em: 2010.

[Col10b] William Colen. Cogroo - corretor gramatical acoplável ao openoffice.org - o que faz o cogroo. Disponível em: http://ccsl.ime.usp.br/cogroo/pt-br/node/2., Acesso em: 2010.

[Col10c] William Colen. Cogroo corretor gramatical para openoffice.org-guia do desenvolvedor. Disponível em: http://ccsl.ime.usp.br/cogroo/en/node/5., Acesso em: 2010.

[Com10] Apache UIMA Development Community. Apache uima documentation. Disponível em: http://uima.apache.org/documentation.html., Acesso em: 2010.

[Des10] Desconhecido. jlinker - a dynamic link between lisp and java. Disponível em: http://franz.com/support/documentation/current/doc/jlinker.htmintro-1., Acesso em: 2010.

[EL07] Martin Emms and Saturnino Luz. Machine learning for natural language processing. 2007.

[GNU10] GNU. Gnu emacs. Disponível em: http://www.gnu.org/software/emacs/., Acesso em: 2010.

[GSC06] Diogo M. Pires , Fábio W. Gusukuma, Marcelo Suzumura, and William Colen. Corretor gramatical acoplável ao openoffice.org cogroo 2.0. 2006.

[Inb10] Inbot. Inbot inteligÊncia artificial. Disponível em: http://www.inbot.com.br/., Acesso em: 2010.

[Lee10] Xah Lee. Xah's emacs lisp tutorial. Disponível em: http://xahlee.org/emacs/elisp.html., Acesso em: 2010.

[max10] opennlp maxent. Disponível em: http://maxent.sourceforge.net/., Acesso em: 2010.

[MS99] Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing.* MIT press, second edition, 1999.

[ope10a] opennlp. Disponível em: http://opennlp.sourceforge.net/., Acesso em: 2010.

[ope10b] openoffice.org. Openoffice.org the free and open productivity suite. Disponível em: http://www.openoffice.org/., Acesso em: 2010.

[Rat98] Ratnaparkhi. *A. Maximum Entropy Models for Natural Language Ambiguity Resolution. Ph.D. Dissertation.* Julho de 1998.

[Wik10a] Wikilingue. Processamento de linguagens naturais. Disponível em: http://pt.wikilingue.com/es/Processamento_de_linguagem_natural., Acesso em: 2010.

[Wik10b] Wikipedia. Aprendizagem de máquina. Disponível em: http://pt.wikipedia.org/wiki/Aprendizagem_de_máquina., Acesso em: 2010.

[Wik10c] Wikipedia. Machine learning. Disponível em: http://en.wikipedia.org/wiki/Machine_learning., Acesso em: 2010.

[Wik10d] Wikipedia. Natural language processing. Disponível em: http://en.wikipedia.org/wiki/Natural_language_processing., Acesso em: 2010.

[Wik10e] Wikipedia. Tradução automática. Disponível em: http://pt.wikipedia.org/wiki/Tradução_automática., Acesso em: 2010.

Parte II

Parte Subjetiva

1 Escolha do Tema

Ao final de 2009 eu estava preocupado em escolher um tema para o trabalho de conclusão de curso, porém, ainda não tinha ideia de qual área escolher. Comecei a pensar em algumas possibilidades, mas deixei passar o fim de ano para então escolher um tema.

No início de 2010 comecei a procurar uma área que me interessasse. A principio, pensei em algum tema relacionado a grafos. Não encontrei algo que me chamasse a atenção, então comecei a pensar em outras áreas, sem abandonar a possibilidade de um trabalho com grafos. Pesquisei temas relacionados a banco de dados, mas também não me atraiu. A partir deste momento, procurei por outros temas e encontrei uma área chamada de PLN. No começo eu não tinha ideia do que se tratava, mas fiquei curioso, devido a grande quantidade de assuntos relacionados a PLN.

Pesquisando sobre PLN e suas aplicações, encontrei na internet uma explicação sobre o sistema Inbot[Inb10]. Um sistema que gera personagens virtuais que são capazes de interagir com as pessoas como se fosse outra pessoa. Estes personagens usam PLN para conversar com as pessoas. A partir deste momento, decidi fazer o trabalho de conclusão com algum tema voltado a PLN. Procurei por professores que pesquisam nesta área e encontrei o professor Marcelo Finger. Na nossa primeira conversa ele me passou a ideia básica de um projeto e indicou alguns capítulos de um livro de PLN para eu ler. Ao terminar a leitura desses capítulos, eu deveria confirmar para ele se aceitaria o projeto sugerido. Após ler os trechos do livro, confirmei com o professor que gostaria de fazer o trabalho. O projeto era o corretor gramatical para o emacs. O corretor já estava pronto e eu teria o trabalho de acoplá-lo ao emacs. Porém, além do acoplamento, eu deveria estudar sobre PLN e sobre o corretor gramatical que eu iria usar, o cogroo.

2 Desenvolvimento do trabalho, Desafios e Frustrações

Comecei meu projeto estudando conceitos básicos de processamento de linguagem natural e vendo algumas aplicações e sistemas que usam PLN. Estes conceitos foram importantes para entender como o cogroo consegue processar um texto a procura de erros.

O segundo passo foi estudar o cogroo. Nesta etapa aproveitei o projeto que eu participei na matéria de laboratório de programação extrema. O projeto era desenvolver uma página *web* para o cogroo, em que os usuários pudessem fazer análises de textos, buscar palavras, reportar erros na correção do cogroo etc.

No final de Maio e começo de Junho comecei a estudar o emacs, dando ênfase na sua estrutura e na linguagem emacs lisp. Porém, o estudo de emacs lisp estava lento e desgastante, devido a grande quantidade de conteúdo do seu manual. Então estudei alguns conceitos básicos da linguagem e comecei a implementar o projeto, e o estudo do emacs lisp era dirigido conforme as necessidades durante o acoplamento.

O primeiro desafio era fazer o emacs se comunicar com o cogroo. Após algumas tentativas usando algumas ferramentas e *sockets*, consegui fazer a comunicação usando processos, que podem ser criados pelo emacs.

Estudar emacs lisp tomou bastante tempo, mas consegui implementar o que era necessário, como comandos para o usuário interagir com o corretor, captura do texto para enviar ao cogroo, marcação dos erros encontrados e aceitar sugestões de correção dos erros. No decorrer do trabalho, o professor foi sugerindo melhorias na implementação. Porém, ainda há melhorias a serem feitas, como tornar a correção de texto o mais imperceptível possível para o usuário, por meio de correção automática, e aperfeiçoamento do cogroo, por exemplo.

Ao final de setembro, parei a implementação e me dediquei ao pôster, apresentação e monografia. O principal foco nesta fase do trabalho de conclusão era descrever PLN estatístico, o cogroo, como foi feito o acoplamento e os resultados obtidos.

3 Futuro

No começo do trabalho de conclusão tínhamos a intenção de disponibilizar o projeto para os usuários do emacs. Atualmente, estamos conversando sobre a possibilidade de disponibilizar esta primeira versão. A ideia é continuar a implementação para melhorar algumas características e criar novas funcionalidades para as novas versões. Estas características e funcionalidades são citadas abaixo.

- Funcionalidades a serem criadas
  - Tornar a correção mais imperceptível ao usuário. Este, ainda precisa interagir com o emacs por meio de comandos ou de teclas para solicitar a correção de

um texto. O ideal é o próprio emacs identificar uma mudança no texto e tentar corrigi-lo. No futuro tentarei implementar esta ideia.

  - Quando não há sugestão de correção para um erro, o usuário precisa corrigi-lo e solicitar a correção manualmente para que o destaque possa ser removido. Com a implementação do item anterior, tenho a intenção de que o texto modificado seja corrigido novamente sem a necessidade do usuário usar comandos.
  - Criar uma interface gráfica simples, que por meio do menu o usuário possa escolher para iniciar o corretor automaticamente e que possa habilitar correção automática ou não.

- Melhorias de algumas características
  - Aperfeiçoar o resultado da correção. Desta forma, pode-se evitar casos de falsos positivos e falsos negativos. Para isto, precisa-se trabalhar na melhoria do cogroo.
  - Cooperar para a melhoria do cogroo, reportando erros e usando aprendizado de máquina com novas informações no corpus e com erros reportados. Para verificar um aperfeiçoamento, fazer medidas da eficiência dos módulos do cogroo.
  - Fazer medidas da eficiência do sistema de acoplamento. Verificar o quanto se demora para corrigir textos de tamanhos variáveis e tentar aperfeiçoar, ao máximo, o sistema, para que o tempo de correção seja o menor possível. Esta mudança pode ser feita por meio da maneira de manipular os textos e das estruturas usadas no acoplamento.

Eu não vou seguir para o mestrado, mas isto não impede que eu continue a implementação do projeto e coopere na evolução do cogroo.

4 Disciplinas Relevantes

- **Introdução à Computação, Princípios de Desenvolvimento de Algoritmos -** Estas matérias introdutórias me deram uma noção básica de algoritmos e lógica.
- **Estrutura de Dados -** Matéria que me mostrou as melhores maneiras de trabalhar com estruturas de dados. Precisei usar estruturas de dados para manipular os textos.

- **Desafios de Programação -** Foi uma matéria que me ajudou a desenvolver e conhecer algoritmos, melhorando minha maneira de programar, além de não me deixar restrito a uma linguagem, usando a linguagem que era mais conveniente para resolver um problema. Dessa forma, estudar emacs lisp não foi tão complicado.
- **Laboratório de Programação Extrema -** Me ajudou a ser mais organizado e disciplinado com o desenvolvimento de projetos maiores.
- **Programação Concorrente -** Utilizei ideias básicas para sincronizar o processo do cogroo com o emacs.
- **MAE0121 Introdução à Probabilidade e Estatítica I, MAE0212 Introdução à Probabilidade e Estatítica II -** Matérias importantes para eu entender os conceitos de estatística utilizados em PLN estatístico.