Universidade de São Paulo
Escola de Artes, Ciências e Humanidades

Uso da Ferramenta NLTK e Python para Desenvolvimento de Aplicações de Processamento de Linguagem Natural

Victor Hugo Jabur Passavaz

Monografia apresentada à Escola de Artes, Ciências e Humanidades, da Universidade de São Paulo, como parte dos requisitos exigidos na disciplina ACH 2017 – Projeto Supervisionado ou de Graduação I, do curso de Bacharelado em Sistemas de Informação.

São Paulo, Junho de 2008

Universidade de São Paulo
Escola de Artes, Ciências e Humanidades

Uso da Ferramenta NLTK e Python para Desenvolvimento de Aplicações de Processamento de Linguagem Natural

Victor Hugo Jabur Passavaz

Orientação

_____

Prof. Dr. Ivandré Paraboni

Banca Examinadora:

_____

Prof. Dr. Patrícia Rufino Oliveira

_____

Prof. Dr. Raphael Camargo

São Paulo, Junho de 2008

Agradecimentos

A Deus

“Por me fazer mais forte nos momentos difíceis e por me ensinar que na vida, devemos ser verdadeiros e humildes para com todos os nossos irmãos. O mundo é a sua escola, para onde viemos compartilhar conhecimento e amor.”

Aos Meus Pais

“Por todo o carinho e amor que tiveram comigo, fizeram-me um verdadeiro guerreiro para combater as injustiças da vida, sou eternamente grato por tudo o que me ensinaram, é maravilhosa a vida ao lado de vocês”

Ao Meu Avô

“Meu eterno ídolo, por todos os momentos de alegria que me proporcionou ao longo desta linda estrada da vida, me ensinou que para amar ao próximo, é preciso abrir mão de nós mesmos, sem egoísmo, sem nenhuma pretensão. Sinto muitas saudades...”

À Minha Noiva

“Muito obrigado pelo seu amor, sempre me amando mais do que a si mesma, tenho certeza de que Deus a colocou em minha vida, para me ensinar o seu verdadeiro significado. Juntos, viveremos os momentos mais lindos de amor...”

Dedicatória

À comunidade EACH

É um orgulho para todos nós, alunos da EACH, participar de uma revolução que ocorreu na USP, com o surgimento de uma escola flexível e inovadora como a EACH. Temos ciência de que a aprovação e construção do projeto desta escola não foi nada fácil. Há vários anos a comunidade pobre da zona leste reivindicou por uma universidade renomada como a USP na zona leste.

Somos a primeira turma formada após todas estas reivindicações e lutas de nosso povo, devemos por tanto, agradecer a esta comunidade, pois sem ela, não estaria escrevendo hoje este documento. Muito Obrigado ! Cabe a nós alunos, valorizar o conhecimento que nos foi dado, gratuitamente, e devolver em forma de benefícios para a comunidade, disseminando conhecimento e fazendo o nome de nossa escola pelo mundo afora.

Agradeço a todos os professores, pela dedicação e carinho com que nos receberam em classe e compartilharam conosco a fundação desta maravilhosa escola. Uma nova semente foi plantada e já está dando ótimos frutos, tanto no mercado de trabalho, quanto na área de pesquisa científica.

Fica aqui os meus parabéns a toda a comunidade EACH, que a cada dia luta por fazer a escola crescer e que está proporcionando a formação superior de muitos jovens que antes não tinham condições de realizar tal sonho.

Resumo

Neste trabalho, foi desenvolvido um protótipo de aplicação prática com o objetivo de se avaliar um recente conjunto de ferramentas para desenvolvimento de aplicação de PLN chamado Natural Language Toolkit (NLTK). Os autores do kit optaram por desenvolvê-lo usando a linguagem de programação Python, linguagem que também vêm fazendo bastante sucesso por ser bastante dinâmica, além de ser de alto nível, o que a torna uma linguagem fácil de se aprender e utilizar.

A aplicação escolhida foi um algoritmo que realiza a resolução anafórica, mais especificamente, implementa a resolução de co-referência de pronomes no idioma português brasileiro.

Palavras Chave

- Processamento de Linguagem Natural (PLN)
- Linguagem Python
- Natural Language ToolKit (NLTK)
- Resolução de Co-referência de Pronomes (Em Português)
- Resolução de Anáforas

Índice

Agradecimentos iii

Dedicatória iv

Resumo v

Lista de Figuras vii

Lista de Tabelas viii

Glossário ix

1. Introdução 1

1.1 Objetivos 1

1.2 Motivação 2

1.3 Metodologia 2

1.4 Estrutura do Trabalho 3

2. Revisão Bibliográfica 5

2.1 Introdução à Linguagem Python 5

2.2 Introdução ao Kit NLTK 8

2.3 Resolução de Co-Referência de Pronomes 13

2.3.1 – Co-referência de pronomes por Cuevas et. al. (2008) 15

3. Trabalho Desenvolvido 19

3.1 – O algoritmo desenvolvido 19

4. Resultados 26

5. Conclusões 30

Referências Bibliográficas 33

Anexo A 34

Lista de Figuras

Figura 1 – Árvore de Decisão 15

Figura 2 – Texto Anotado 17

Figura 3 – Estrutura de Dados 20

Figura 4 – Estrutura de Regras 21

Figura 5 – Arquivo XML de Dados 22

Figura 6 - Arquivo XML de Regras 23

Figura 7 – Checkbuttons Renderizados Dinamicamente 25

Figura 8 – Arquivo HTML de resposta gerado pelo algoritmo desenvolvido 27

Figura 9 – Interface Gráfica Desenvolvida 34

Lista de Tabelas

Tabela 1 – Tabela de Regras de Co-Referência de Pronomes 16

Tabela 2 – Tabela de Código-Fontes desenvolvidos..... 20

Tabela 3 – Resultados Obtidos – Análise Manual X Resposta do Algoritmo 28

Glossário

Número	Termo	Significado
1	Corpus / Corpora	Repositório de arquivos textos utilizados por aplicações PLN, geralmente acompanham kits de desenvolvimento ou são distribuídas em um pacote de arquivos, geralmente em vários idiomas e constituem arquivos de entrada para as aplicações PLN.
2	NLTK	Natural Language Toolkit – É um kit de desenvolvimento feito na linguagem Python e voltado especialmente para se trabalhar com Processamento de Linguagem Natural
3	Parser	Parser é um componente que serve para analisar a estrutura gramatical de uma entrada, que são segmentos de texto ou símbolos que podem ser manipulados. O parser pode ser um leitor que ajuda na conversão do arquivo para manipulação dos dados contidos no mesmo, é uma espécie de tradutor que converte uma dada entrada não estruturada em uma saída fácil de ser manipulada, como transformar um texto em XML por exemplo.
4	PLN	Processamento de Linguagem Natural
5	Python	É uma linguagem de programação.
6	Tagger	Um Tagger é um componente responsável por anotar ou fazer marcas em um determinado texto. Por exemplo, dado um texto “cru” de entrada, o tagger analisa gramaticalmente cada palavra do texto e faz uma marca. Frase de exemplo: O Armando caiu. O<artigoDefinido> Armando<nomePróprio> caiu <verbo>.
7	W3C	**World Wide Web Consortium** é um consórcio de empresas de tecnologia. O W3C desenvolve padrões para a criação e a interpretação dos conteúdos para a Web.
8	XML	XML = e**X**tensible **M**arkup **L**anguage é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais.

1. Introdução

Este trabalho realiza um estudo e avaliação crítica de um conjunto de ferramentas para Processamento de Linguagem Natural (PLN), que tem se popularizado bastante em tempos modernos.

O kit se chama Natural Language Toolkit (NLTK) e foi desenvolvido inteiramente em Python, que é uma linguagem de programação dinâmica, de alto nível, apresenta um fácil aprendizado e possui uma manipulação otimizada de strings, características que chamam a atenção da comunidade científica de PLN, pois a área requer um grande volume de processamento textual e os recursos citados auxiliam os desenvolvedores de software em PLN.

Para a avaliação do NLTK, foi escolhido um problema clássico na área de PLN para ser codificado. O problema é conhecido como co-referência de pronomes em terceira pessoa no plural (Eles, Elas) que é uma sub-área de Resolução de Anáforas.

Neste problema o algoritmo deve procurar pelos pronomes citados e localizar no texto a sua referência, por exemplo, “As mulheres” e “elas” são termos co-referentes na frase seguinte:

“As mulheres gostam de fazer compras, pois elas se sentem bem”.

O Corpus (arquivos texto base) utilizado neste trabalho está no idioma português brasileiro.

1.1 Objetivos

O objetivo deste trabalho é a correta configuração, aprendizado e utilização do pacote NLTK, bem como da linguagem de programação Python e técnicas básicas de PLN, realizando uma análise crítica do pacote NLTK através de uma implementação prática na área.

Pretende-se avaliar se o kit possui todas as qualidades divulgadas por seus autores, tais como a facilidade de uso, a legibilidade de código, o tamanho reduzido de linhas de código-fonte e a experiência de desenvolvimento de um protótipo de aplicação de PLN neste ambiente.

1.2 Motivação

O Natural Language Toolkit tem feito bastante sucesso entre a comunidade de desenvolvedores, cientistas, alunos e demais interessados na manipulação de textos e strings, em especial na área de Processamento de Linguagem Natural (PLN), que requer manipulação de grandes volumes de textos com alta performance e capacidade de processamento.

Como estes requisitos são muitas vezes difíceis de se atender em linguagens de programação mais antigas, esta nos parece uma boa oportunidade de testar o kit NLTK e a linguagem Python no desenvolvimento de uma aplicação deste tipo.

O desenvolvimento de novas versões e funcionalidades tanto para a linguagem Python, quanto para o pacote NLTK estão em andamento, somando-se o fato de serem tecnologias totalmente gratuitas e constantemente aperfeiçoadas e evoluídas pela comunidade de desenvolvedores ao redor do mundo.

1.3 Metodologia

Para o desenvolvimento deste trabalho, foi seguida a seguinte metodologia, numerada em ordem de ocorrência:

1. Instalação, Configuração e aprendizado da linguagem Python
2. Instalação, Configuração e aprendizado do pacote NLTK

3. Estudo de técnicas básicas de PLN

4. Implementação de uma aplicação prática que resolvesse algum problema da área de PLN, servindo para a avaliação crítica tanto do kit NLTK, quanto da linguagem Python. A escolha recaiu sobre o modelo de resolução de pronomes proposto em Cuevas et. al. (2008).

5. Dado que foi escolhido o problema de resolução anafórica, foi selecionado um Corpus (conjunto de textos anotados) no idioma português brasileiro para servir como entrada desta aplicação.

6. Desenvolvimento de um algoritmo de resolução de co-referência de pronomes utilizando-se a linguagem Python e também rotinas prontas do kit NLTK.

7. Foram realizados testes e obtidos alguns resultados do algoritmo desenvolvido neste trabalho e realizada uma comparação com o índice de acertos em Cuevas et. al. (2008) para verificar se o algoritmo desenvolvido obteve resultados próximos do trabalho de Cuevas et. al. (2008).

1.4 Estrutura do Trabalho

O restante do trabalho está organizado conforme descrito abaixo:

Seção 2. Revisão Bibliográfica – Dividido em três partes:

2.1. Teoria sobre Resolução Anafórica – Breve embasamento teórico a respeito de resolução de co-referência de pronomes.

2.2. Introdução ao Python – Breve apresentação da linguagem de programação Python, de seus principais conceitos e trechos de código para melhor compreensão do trabalho desenvolvido e disponível na seção de anexos.

2.3. Introdução ao kit NLTK – Descrição dos recursos que o kit oferece e sua utilização, além de trechos de código ilustrativos.

Seção 3. Trabalho Desenvolvido – Detalhamento de todo o trabalho prático desenvolvido, explicando desde alguns trechos do código-fonte até o funcionamento da aplicação prática e como ela resolve o problema de PLN proposto.

Seção 4. Resultados – Contém os resultados obtidos com a execução da aplicação desenvolvida. É feita uma comparação entre a análise manual e a análise automática para se verificar o índice de acerto da aplicação na resolução de pronomes de Cuevas et. al.(2008).

Seção 5. Conclusões – Avaliação crítica da linguagem Python e do kit NLTK, relatando os conhecimentos aprendidos durante a execução deste trabalho, uma síntese das contribuições do mesmo e sugestões para trabalhos futuros.

Anexos – Seção que contém todo o trabalho prático desenvolvido, desde o código-fonte até figuras das interfaces gráficas da aplicação desenvolvida.

2. Revisão Bibliográfica

Realizamos nesta seção uma contextualização geral, descrevendo todos os recursos pesquisados e utilizados durante a execução deste trabalho, para um maior esclarecimento acerca do trabalho prático que foi desenvolvido. Iniciamos com uma breve apresentação da linguagem Python e de alguns recursos do Natural Language Tool Kit (NLTK) (TOOLKIT, Natural Language), um kit para desenvolvimento de aplicações de PLN em Python. A seguir, descrevemos a base teórica do problema computacional de Resolução de Anáforas, enfocando especificamente o modelo de resolução de pronomes baseado em regras proposto em Cuevas et. al. (2008) que será utilizado como exemplo de aplicação de PLN a ser implementada em Python/NLTK.

2.1 Introdução à Linguagem Python

O objetivo desta seção é contemplar os recursos da linguagem Python que foram utilizados no trabalho. Para maiores informações sobre diversos recursos da linguagem, deve-se consultar a documentação oficial da comunidade Python disponível em http://www.Python.org/doc/, nos idiomas Inglês e Português e em diversos formatos de arquivo.

Um pouco da história e conceito de Python

Python é uma linguagem de programação que foi criada em 1991, pelo seu criador Guido van Rossum. É uma linguagem de programação que objetiva uma maior facilidade de aprendizado e legibilidade de código. Segue abaixo uma lista contendo suas principais características:

- É uma linguagem interpretada.

- É bastante dinâmica, suportando declaração de variáveis e seus tipos, criação de funções ou métodos em tempo de execução.

- É uma linguagem de alto nível, contendo muitas estruturas prontas para o uso e possui muitos recursos em sua biblioteca padrão.

- Suporta o paradigma de orientação a objetos, com todos os seus conceitos teóricos, tais como herança, polimorfismo, conceito de classes, além de suportar outros tipos de paradigmas tais como o funcional e o modular.

- Possui elevada portabilidade, trabalhando com muitos sistemas operacionais

Para exemplificar e discutir alguns recursos básicos da linguagem Python com o intuito de uma maior familiarização coma linguagem e com o código-fonte desenvolvido neste trabalho segue abaixo, alguns trechos de código escritos na linguagem Python:

Construindo um Hello World em Python

```
# -*- coding: iso-8859-15 -*-
texto = 'Hello World'
print texto
print type(texto)
```

Retorno:

```
Hello World
<type 'str'>
```

Neste simples exemplo acima, note algumas coisas interessantes:

Em Python é desejável declarar o tipo de encoding dos código-fontes, como visto na primeira linha, para utilização de caracteres especiais, tais como acentuação etc. Na segunda linha existe uma criação de uma variável de nome “texto” do tipo string e a atribuição de um valor para ela. Python admite tanto aspas simples como aspas duplas para o tipo string, mas não é preciso declarar o seu tipo. Na terceira linha é exibido o valor da variável texto, e na última linha o seu tipo. Observamos que a tipagem da variável ocorreu de maneira dinâmica, pelo próprio interpretador da linguagem.

Python possui uma série de facilidades para manipulação de strings, algumas das quais serão exemplificadas em conjunto com outros recursos do NLTK na próxima seção. Para fins ilustrativos, citamos aqui o recurso conhecido por "slice". Slice do inglês quer dizer fatiar e é utilizado na linguagem Python para fatiar diversos tipos de elementos tais como variáveis do tipo string, estrutura de dados tais como listas e dicionários. Exemplo de sua utilização:

```
# -*- coding: iso-8859-15 -*-
texto = 'Hello World'
print texto[0]
print texto[0:3]
print texto[:-1]
```

Retorno:

```
H
Hel
Hello Worl
```

Como pode ser visto acima, o recurso slice consegue fatiar uma variável do tipo string através da indexação, que se inicia em zero.

2.2 Introdução ao Kit NLTK

O pacote Natural Language Toolkit (TOOLKIT, Natural Language), é uma biblioteca para a linguagem Python distribuída gratuitamente sob a licença open source com a finalidade de pesquisa e desenvolvimento na área de Processamento de Linguagem Natural (PLN).

Este kit já possui implementadas muitas tarefas básicas de PLN, acompanha mais de quarenta coleções de arquivos texto populares, conhecidos como Corpus e possui uma extensa documentação, tanto on-line quanto em formato pdf, disponibilizada gratuitamente no site dos desenvolvedores do NLTK [1].

Nesta seção são abordados e explicados alguns recursos do kit NLTK, para que o código-fonte possa ser entendido de maneira mais fácil. Maiores detalhes podem ser visualizados na documentação on-line do kit, no site dos seus desenvolvedores.

Desenvolvimento prático com o kit NLTK

Neste tópico são discutidos alguns recursos mais utilizados do kit NLTK para uma breve introdução e orientação sobre sua utilização. Antes de qualquer coisa, para utilizar o kit NLTK deve-se importar o pacote no início de cada arquivo Python onde se deseja utilizá-lo.

Em Python, existem duas maneiras de se realizar importação, a importação normal, onde importa-se o pacote nltk inteiro, ou então utiliza-se a importação estática, que consegue importar apenas o recurso desejado do kit. Abaixo segue um exemplo contendo as duas maneiras citadas:

---

[1] NLTK – Natural Language Toolkit - http://nltk.org/index.php/Main_Page - Kit de desenvolvimento para aplicações PLN, desenvolvido em Python.

1) Importação Normal (pacote inteiro)

```
# -*- coding: iso-8859-15 -*-
import nltk
count = nltk.defaultdict(int)
```

2) Importação Estática (apenas o recurso desejado)

```
# -*- coding: iso-8859-15 -*-
from nltk import defaultdict
count = defaultdict(int)
```

Nos exemplos acima, estamos utilizando um recurso do kit NLTK chamado "defaultdict". Note que dependendo da utilização da importação normal ou estática, o código-fonte se modifica, na importação normal é necessário referenciar nltk.defaultdict, enquanto que na importação estática referencio apenas como defaultdict. Este é um recurso de importação de pacotes similar ao utilizado na linguagem Java. Deve-se tomar cuidado na importação estática para não ocorrer conflitos de nome, caso seja importados dois pacotes que contenham um recurso com o mesmo nome.

O recurso defaultdict

O kit NLTK fornece uma poderosa estrutura de dados, ela é uma espécie de dicionário e se chama defaultdict. Vamos utilizar um exemplo para facilitar a explicação:

```
# -*- coding: iso-8859-15 -*-
import nltk
count = nltk.defaultdict(int)

frase = """Em 30 de Novembro de 2007 foi decidido que a
           Copa do Mundo de 2014 será no Brasil. As cidades
           sedes dos jogos serão definidas em dezembro de 2008.
           Atualmente, 18 cidades estão na disputa e entre 10 e
           12 serão escolhidas."""

for palavra in frase.split():
    palavra = palavra.lower()
    count[palavra] += 1
print count['cidades']
print count
```

Retorno:

```
2
defaultdict(<type 'int'>, {'em': 2, 'mundo': 1, 'jogos': 1, 'as':
1, 'escolhidas.': 1, 'est\xe3o': 1, 'no': 1, 'na': 1, 'dezembro':
1, 'cidades': 2, 'decidido': 1, '2014': 1, 'disputa': 1, 'foi': 1,
'brasil.': 1, 'do': 1, 'novembro': 1, 'copa': 1, '10': 1, 'sedes':
1, 'definidas': 1, 'de': 4, '2008.': 1, 'que': 1, 'ser\xe3o': 2,
'a': 1, 'atualmente,': 1, '12': 1, 'e': 2, 'entre': 1, '18': 1,
'30': 1, '2007': 1, 'ser\xe1': 1, 'dos': 1})
```

No trecho de código acima, o dicionário armazena o número de vezes que cada palavra aparece no texto e podemos buscar por uma determinada palavra assim como foi demonstrado acima. Qual o número de vezes que a palavra “cidades” aparece no texto ? Como retorno foi impresso o número 2.

No retorno foi impresso também a estrutura “count” inteira, é uma espécie de dicionário, contendo a palavra e o número de vezes que ela aparece. O defaultdict é uma estrutura potencialmente muito interessante para o desenvolvimento de aplicações PLN.

Na linguagem Python podemos utilizar aspas simples, duplas ou triplas para se trabalhar com variáveis do tipo string. Não existe nenhuma diferença entre aspas simples e duplas, mas com as aspas triplas conseguimos utilizar um texto com quebras de linha, assim como demonstrado no exemplo acima, quando se colocou um texto quebrado em diversas linhas dentro de uma variável do tipo string.

Além disso, em nenhum momento precisamos declarar o tipo das variáveis, nem no caso do defaultdict e nem no caso da variável frase, pois Python é uma linguagem dinâmica e que identifica o tipo das variáveis em tempo de execução.

Corpus que acompanham o NLTK

O pacote NLTK acompanha muitos Corpus populares em diversos idiomas, vamos demonstrar através do trecho de código abaixo uma aplicação que consegue ler um texto de um dos Corpus do NLTK e apresentá-lo na tela.

```
# -*- coding: iso-8859-15 -*-
import nltk
for palavra in nltk.corpus.gutenberg.words('shakespeare-macbeth.txt'):
        print palavra,
```

Retorno

```
"Project Gutenberg ' s Etext of Shakespeare ' s The Tragedie of
Macbeth Executive Director ' s Notes :" …
```

O exemplo acima mostra um arquivo texto contido no Corpus Gutenberg [1]. Repare que ao final do comando print existe uma vírgula, o que faz com que todas as palavras iteradas no laço sejam impressas na mesma linha, ao invés de colocar uma palavra por linha, como aconteceria se não existisse esta vírgula, este é mais um recurso da linguagem Python que pode ser útil ao desenvolvimento de aplicações PLN.

Expressões Regulares

Iremos demonstrar agora algumas das funcionalidades disponíveis no kit NLTK e na linguagem Python para se realizar buscas através de expressões regulares.

Citaremos aqui três comandos para buscas:

1) O comando re_show (NLTK)

2) O comando findall (Python)

3) O comando sub (Python)

Um exemplo onde são utilizados os comandos citados é apresentado a seguir:

```
# -*- coding: iso-8859-15 -*-
import nltk
import re
padrao = r'\bo[a-zA-Z]+\b'
texto = """As mais belas descobertas ocorrem quando
           as mesmas coisas são vistas com um novo olhar."""
nltk.re_show(padrao, texto)
print re.sub(padrao, 'xxx', texto)
```

[1] Projeto Gutenberg - http://promo.net/pg/ - Fundado por Michael Hart, em 1971. Decidiu elaborar um projeto que disponibilizasse uma coleção de textos famosos e importantes, gratuitamente.

```
print re.findall(padrao, texto, re.IGNORECASE)
```

Retorno:

```
As mais belas descobertas {ocorrem} quando
           as mesmas coisas são vistas com um novo {olhar}.
As mais belas descobertas xxx quando
           as mesmas coisas são vistas com um novo xxx.
['ocorrem', 'olhar']
```

Neste exemplo, foram buscadas através de expressão regular todas as palavras que começam com a letra “o. O comando re_show pertence ao NLTK e os comandos sub e findall pertencem ao Python, utilizados através do pacote padrão “re”. O comando re_show imprime na tela as palavras buscadas apresentadas entre chaves, enquanto que a função sub, neste exemplo, realiza uma substituição nas palavras encontradas por ‘xxx’ e a função findall devolve uma lista com todas as palavras encontradas.

Vale ressaltar que existem outras maneiras de se buscar expressões regulares em Python, tais como as funções match e compile, mas estas não serão abordadas neste documento.

Processando arquivos XML com o NLTK

O NLTK nos fornece uma variedade bastante grande de recursos para trabalhar com arquivos XML, nesta seção demonstraremos a maneira que foi utilizada no trabalho prático para este fim. Veja o trecho de código abaixo ilustra este uso:

```
# -*- coding: iso-8859-15 -*-
from nltk.etree.ElementTree import Element, dump
from nltk.etree import ElementTree as ET
noHtml = Element('html')
noBody = Element('body')
noHtml.append(noBody)
noHtml.text = 'texto html'
noBody.text = 'texto body'
dump(noHtml)
tree = ET.ElementTree(noHtml)
tree.write("C:/arquivo.xml")
```

Retorno:

```
<html>texto html<body>texto body</body></html>
```

Neste exemplo foram criados dois nós, um chamado noHtml e outro chamado noBody. Com a função append definimos noBody como filho do NoHtml. Incluímos também textos nos dois nós. A função dump imprime na tela a árvore xml a partir de um nó raiz dado. A função write grava a árvore xml em um arquivo xml no diretório especificado.

2.3 Resolução de Co-Referência de Pronomes

Para exemplificar o uso de Python e NLTK foi escolhido um exemplo simples de aplicação de PLN, o tratamento computacional de expressões anafóricas.

Uma expressão anafórica ou simplesmente anáfora é uma expressão lingüística que faz referência – geralmente de forma abreviada – a um termo que a antecede no discurso falado ou escrito. Exemplos de expressões anafóricas incluem pronomes (e.g., “ele”), descrições indefinidas (e.g., “um homem”) ou definidas (e.g., “o segundo homem da esquerda, de terno marrom”), nomes próprios (e.g., “João”) e muitas outras. A tarefa computacional de “resolver” expressões anafóricas (ou seja, encontrar o termo antecedente a que correspondem) é um dos mais tradicionais desafios do PLN, e um requisito essencial à maioria das aplicações de interpretação de linguagem, de tradução automática à extração de informações (FELDMAN, 2006). Ou nas palavras de (CHAVES, 2007):

“Um dos problemas encontrados em sistemas de processamento de línguas naturais é conseguir manter a coesão referencial de um texto, propriedade que permite estabelecer as ligações entre os seus constituintes, tornando-o inteligível. Dentre os fatores de coesão referencial destacamos a anáfora, que ocorre quando duas ou mais expressões de um texto estabelecem uma relação de referência entre si, isto é, a interpretação da anáfora depende de um antecedente ao qual ela se refere no texto.” (CHAVES, 2007).

Existe um tipo de anáfora, denominado anáfora pronominal e é o foco de estudo deste trabalho, segundo Chaves (2007), a resolução automática de anáforas pronominais consiste em 3 passos:

1. Identificar a Anáfora
2. Determinar um conjunto de Possíveis Antecedentes
3. Identificar e selecionar o antecedente da anáfora

"Quando uma anáfora e seu antecedente referenciam uma mesma entidade, dizemos que eles são co-referentes". (CHAVES, 2007).

A resolução de co-referência de pronomes utiliza o mesmo princípio da resolução de anáforas, que é o processo pelo qual se procura identificar em um texto, a qual expressão a anáfora se refere. É um dos problemas clássicos da lingüística computacional. A resolução de anáforas é um componente essencial para a maioria das aplicações PLN desde um simples entendimento do texto até um processo de tradução automática.

A Resolução de co-referência de pronomes é uma tarefa mais específica que a resolução de anáforas e funciona da seguinte maneira: dado um texto anotado com as corretas análises gramaticais e morfológicas, é feita uma busca por pronomes tais como o pronome "ele" e procura-se então um antecedente que, por exemplo, poderia ser a expressão "O presidente".

É cada vez maior o número de pessoas que estão estudando resolução de anáforas no idioma português brasileiro. Isso se deve possivelmente ao aumento de ferramentas básicas, aplicações PLN e grandes Corpus em diversos idiomas. Existem técnicas de resolução de anáforas puramente algorítmicas e recentemente surgiram algumas técnicas baseadas em aprendizagem de máquina, supervisionadas e não-supervisionadas.

2.3.1 – Co-referência de pronomes por Cuevas et. al. (2008)

A aplicação prática deste trabalho utiliza a técnica apresentada em Cuevas et. al. (2008) para ilustrar os recursos da linguagem Python e do kit NLTK. Trata-se de uma abordagem de indução de árvores de decisão utilizando o algoritmo J48 do pacote WEKA (WITTEN et. al. 2005) para resolução de pronomes pessoais de terceira pessoa Eles/Elas.

Para ilustrar melhor a técnica de Cuevas et. al. (2008), segue abaixo, a árvore de decisão citada com sua respectiva tabela de regras, contendo as explicações sobre cada uma delas.

```
number_agreement = true
|   gender_agreement = true
|   |   i_subject = true
|   |   |   distance <= 2: true (135.0/15.0)
|   |   |   distance > 2
|   |   |   |   j_subject = true: true (4.0/1.0)
|   |   |   |   j_subject = false: false (3.0)
|   |   i_subject = false
|   |   |   distance <= 2
|   |   |   |   is_civ = true: false (12.0/2.0)
|   |   |   |   is_civ = false
|   |   |   |   |   j_subject = true
|   |   |   |   |   |   i_defined = true
|   |   |   |   |   |   |   i_direct = true
|   |   |   |   |   |   |   |   words_between <= 17
|   |   |   |   |   |   |   |   |   words_between <= 3: false (4.0/1.0)
|   |   |   |   |   |   |   |   |   words_between > 3: true (12.0/3.0)
|   |   |   |   |   |   |   |   words_between > 17: false (5.0)
|   |   |   |   |   |   |   i_direct = false
|   |   |   |   |   |   |   |   distance <= 0
|   |   |   |   |   |   |   |   |   words_between <= 0: false (3.0)
|   |   |   |   |   |   |   |   |   words_between > 0: true (31.0/10.0)
|   |   |   |   |   |   |   |   distance > 0
|   |   |   |   |   |   |   |   |   words_between <= 46: true (41.0/13.0)
|   |   |   |   |   |   |   |   |   words_between > 46: false (3.0)
|   |   |   |   |   |   i_defined = false: false (208.0/70.0)
|   |   |   |   |   j_subject = false
|   |   |   |   |   |   words_between <= 1
|   |   |   |   |   |   |   pronoun_type = 1: true (6.0/1.0)
|   |   |   |   |   |   |   pronoun_type = 2: false (13.0)
|   |   |   |   |   |   |   pronoun_type = 3: false (0.0)
|   |   |   |   |   |   words_between > 1
|   |   |   |   |   |   |   words_between <= 23: true (195.0/64.0)
|   |   |   |   |   |   |   words_between > 23
|   |   |   |   |   |   |   |   i_defined = true
|   |   |   |   |   |   |   |   |   i_direct = true: false (2.0)
|   |   |   |   |   |   |   |   |   i_direct = false
|   |   |   |   |   |   |   |   |   |   same_sentence = true: true (3.0)
|   |   |   |   |   |   |   |   |   |   same_sentence = false: false (5.0/2.0)
|   |   |   |   |   |   |   |   i_defined = false
|   |   |   |   |   |   |   |   |   i_direct = true
|   |   |   |   |   |   |   |   |   |   pronoun_type = 1: false (2.0)
|   |   |   |   |   |   |   |   |   |   pronoun_type = 2
|   |   |   |   |   |   |   |   |   |   |   words_between <= 26: false (2.0)
|   |   |   |   |   |   |   |   |   |   |   words_between > 26: true (6.0/1.0)
|   |   |   |   |   |   |   |   |   |   pronoun_type = 3: true (0.0)
|   |   |   |   |   |   |   |   |   i_direct = false: false (19.0/5.0)
|   |   |   distance > 2
|   |   |   |   distance <= 12: false (49.0/4.0)
|   |   |   |   distance > 12: true (3.0)
|   gender_agreement = false: false (244.0)
number_agreement = false: false (1383.0/24.0)
```

Figura 1 – Árvore de Decisão

A árvore de decisão é um modelo de resolução treinado a partir de um gênero específico de texto e a árvore utilizada neste trabalho (figura 1) foi obtida a partir de textos científicos, corpus utilizado em Cuevas et. al. (2008). As características de cada gênero de texto são diferentes, por exemplo, a distância média entre o pronome e o antecedente e por isso o modelo (árvore de decisão) não se aplica ao processamento de texto irrestrito e nem possui tal intenção.

Table 1. Set of learning features extracted from the tagged text.

Feature name	Description
distance	sentences between i and j.
words_between	number of words between i and j.
pronoun_type	1=personal (eles/as); 2=possessive (deles/as); 3=location (neles/as.)
number_agreement	true if i and j agree in number.
gender_agreement	true if i and j agree in gender.
same_sentence	true if i and j occur in the same sentence.
i_defined	true if i is a definite description.
i_subject	true if i is the sentence subject.
i_direct	true if i is a direct object.
j_subject	true if j is the sentence subject.
is_hh	true if i is a group of humans.
is_inst	true if i is an institution.
is_civ	true if i is a city, country, province etc.

Tabela 1 – Tabela de Regras de Co-Referência de Pronomes

A árvore de decisão acima é uma estrutura composta de diversas regras e têm a função de dizer se uma palavra antecedente consegue resolver um determinado pronome.

Para exemplificar a técnica de Cuevas et. al. (2008), segue um exemplo prático:

Considere a seguinte frase:

A mulher caiu da escada ontem e se machucou. Ela nos disse que já está melhor.

Um algoritmo de co-referência de pronomes deve procurar no texto os pronomes de terceira pessoa (e.g. “Ela”) e recuar no texto procurando uma palavra antecedente que consiga “resolver” o pronome encontrado.

O algoritmo de co-referência de pronomes não pode executar sua análise sobre um texto cru. É necessário um texto especial, que é chamado de texto anotado ou texto marcado, ilustrado pela figura abaixo:

```
1  Brasil    [Brasil] <civ> PROP M S @SUBJ>
2  $,        [$,] <co-subj> PU @CO
3  01/01/94          [01/01/94] <date> <card> <NER:date> NUM M S @SUBJ>
4  $,
5  p.        [p.] <NER2> <np-close> N F S @N<PRED
6  1-5       [1-5] <NER:prednum> <card> <np-close> NUM M/F P @N<
7  Imposto=PEGA=50  [Imposto=PEGA=50] <plan> PROP M S @SUBJ>
8  $%        [%] N M P @PRED>
9  De        [de] <sam-> <prop1> <np-close> PRP @N<
10 o         [o] <prop2> <artd> <-sam> DET M S @>N
11 Juro      [juro] <prop> <prop2> <amount> N M S @P<
12 $(
13 De        [de] <sam-> <prop1> PRP @ADVL>
14 a         [o] <prop2> <artd> <-sam> DET F S @>N
15 Reportagem        [reportagem] <prop> <prop2> <sem-r> N F S @P<
16 Local     [local] <prop> <prop2> <np-close> ADJ F S @N<
17 $)
18 O         [o] <artd> DET M S @>N
19 aumento   [aumento] <act> N M S @SUBJ>
20 de        [de] <np-close> PRP @N<
21 impostos          [imposto] <mon> N M P @P<
22 em        [em] <sam-> <np-close> PRP @N<
23 o         [o] <artd> <-sam> DET M S @>N
24 setor     [setor] <Labs> <L> N M S @P<
25 bancário          [bancário] <np-close> ADJ M S @N<
26 levará    [levar] <fmc> <mv> V FUT 3S IND VFIN @FS-STA
27 a         [a] PRP @<ADVL
28 uma       [um] <arti> DET F S @>N
29 alta      [alta] <sit> N F S @P<
30 de        [de] <sam-> <np-close> PRP @N<
31 as        [o] <artd> <-sam> DET F P @>N
32 taxas     [taxa] <mon> N F P @P<
33 de        [de] <np-close> PRP @N<
34 juros     [juro] <amount> N M P @P<
35 de        [de] <sam-> <np-close> PRP @N<
36 os        [o] <artd> <-sam> DET M P @>N
37 empréstimos       [empréstimo] <mon> N M P @P<
38 $.
```

Figura 2 – Texto Anotado

O texto acima possui uma única palavra por linha e ao lado possui seus atributos que contém a classificação gramatical e análise morfológica de cada palavra no texto. Não é o foco deste trabalho saber como o texto anotado foi gerado, mas basicamente textos anotados são gerados por algoritmos clássicos em PLN, conhecidos como Tagger's, que realizam uma análise sintática e morfológica para classificar cada palavra no texto e gerar os atributos visualizados acima.

A relação entre a árvore de decisão, a tabela de regras e o texto anotado é a seguinte:

1 – A árvore de decisões contém regras a serem verificadas pelo algoritmo de co-referência.

2 – Na tabela de regras, existem todas as regras utilizadas pela árvore de decisão com uma breve explicação de quais informações devem ser verificadas para cada regra.

3 – Cada palavra do texto anotado contém atributos que são utilizados pelas regras para verificar se a palavra candidato e o pronome satisfazem seu requisito.

Um exemplo prático para ilustrar essas relações:

Palavra candidata: **membros** [membro] <H> N M P @P<

Pronome encontrado: **eles** [eles] PERS M 3P NOM @SUBJ>

Regra em questão: "gender_agreement" [1]

1) A árvore de decisão contém a regra "gender_agreement".

2) A tabela de regras informa à árvore de decisão o que fazer com esta regra, por exemplo, se palavra candidato e pronome forem ambos masculino ou ambos feminino, i.e, contiverem ambos os atributos 'M' ou 'F', então é regra é satisfeita (true), caso contrário, a regra não é satisfeita (false).

3) Por último o texto anotado contém os atributos necessários para que a regra faça a sua validação.

---

[1] gender_agreement - A regra "gender_agreement" verifica se uma palavra candidata combina em gênero (masculino/feminino) com o pronome.

3. Trabalho Desenvolvido

O trabalho prático desenvolvido é uma implementação em Python/NLTK da técnica desenvolvida por Cuevas et. al. (2008).

O algoritmo desenvolvido aplica os seguintes passos para executar a co-referência de pronomes de terceira pessoa:

1 - Percorre a frase palavra a palavra até encontrar um pronome em terceira pessoa (Ele, Ela, Eles, Elas).

2 - Após identificar o pronome (e.g. “Ela”), recua uma palavra e obtém a palavra candidata (e.g. “machucou”). Neste trabalho, as palavras candidatas e os pronomes receberão um apelido, respectivamente “i” e “j”.

3 - Pergunta a uma árvore de regras se a palavra candidata “i” resolve o pronome “j” encontrado.

4 – Caso não resolva, obtém a palavra anterior e assim sucessivamente, até acabar o texto ou até encontrar uma palavra que resolva o pronome em questão.

5 – Caso resolva, realizo uma ligação entre o pronome j e a palavra considerada pelo algoritmo, segundo as regras dadas, como referência correta.

3.1 – O algoritmo desenvolvido

Esta seção objetiva apresentar o algoritmo desenvolvido e qual é a função de cada trecho de código e como ele está estruturado, para que possa servir como consulta e aprendizado a outras pessoas interessadas e facilitar o entendimento para alguém que queira realizar alterações no código desenvolvido.

O código-fonte do trabalho prático desenvolvido está inteiramente disponível no Anexo A, aqui serão apresentados alguns trechos apenas para explicação. A aplicação PLN desenvolvida possui os seguintes arquivos fonte:

1 – DadosEstrutura.py	2 – RegrasEstrutura.py
3 – NoFrase.py	4 – NoPalavra.py
5 - NoCondicao.py	6 – CorreferenciaPronome.py
7 – Executor.py	8 – InterfaceGrafica.py
9 – Útil.py	10 – Main.py
11 – Regras.py	

Tabela 2 – Tabela de Código-Fontes desenvolvidos

Cada arquivo acima é um script na linguagem Python e é também uma classe, este trabalho foi desenvolvido utilizando o paradigma de orientação a objetos. Vamos detalhar agora um pouco sobre cada uma das classes desenvolvidas, explicando sua função e alguns trechos de códigos mais importantes.

1. **DadosEstrutura** – Esta classe tem a função de realizar uma transformação do arquivo texto anotado recebido como entrada em uma estrutura de dados desenvolvida especificamente para este trabalho. Abaixo segue uma figura ilustrando esta estrutura de dados desenvolvida:

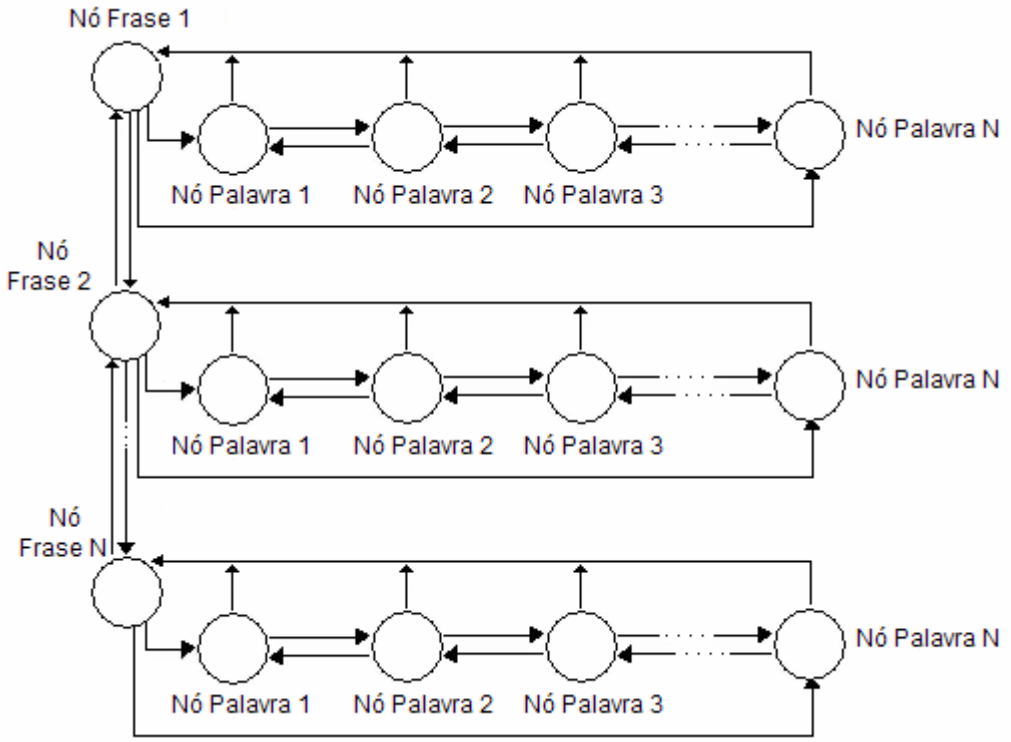

Figura 3 – Estrutura de Dados

Observações: Seria possível não criar uma estrutura de dados como esta acima, seria possível o preenchimento direto para uma estrutura Xml, mas por decisão de projeto optou-se por criar a estrutura de dados personalizada por alguns motivos. Descritos abaixo:

1 – Legibilidade de Código – Já que a estrutura personalizada são objetos que possuem métodos gets() e sets() e a estrutura Xml teria métodos do tipo getChildren(), getIterator(), etc ...

2 – Facilidade na hora de navegar por um texto - Já que a estrutura personalizada possui "ponteiros" personalizados, representados pelas setas do desenho acima e melhoram bastante a performance do algoritmo na hora de realizar o algoritmo de co-referência de pronomes, pois o algoritmo exige um recuo no texto palavra a palavra, frase a frase.

2. RegrasEstrutura – Semelhante à classe anterior (DadosEstrutura), a RegrasEstrutura realiza a transformação da árvore de regras recebida como entrada (Figura 1) em uma estrutura personalizada do tipo árvore. Esta estrutura pode ser visualizada abaixo e também foi criada pelos mesmos motivos da estrutura anterior:

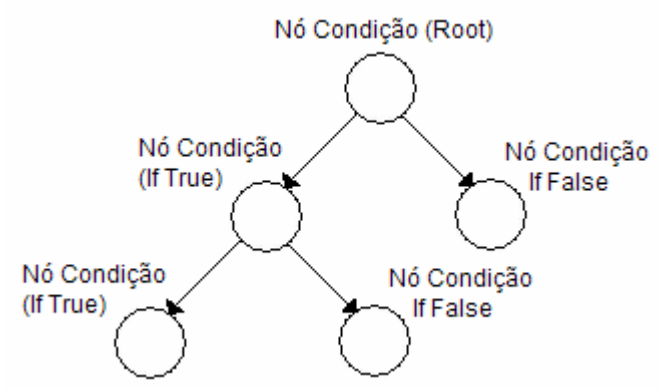

Figura 4 – Estrutura de Regras

3. NoFrase – Esta é uma classe que representa uma frase real em um texto, ela pode ser visualizada na Figura 3. Basicamente é uma classe que contém os ponteiros Frase Anterior, Próxima Frase, Palavra Root e Ultima Palavra. Toda esta estrutura de ponteiros está representada na Figura 3 (Estrutura de Dados). Esta classe contém um método especial que se chama parseXml e transforma a estrutura de dados em um arquivo XML, segue abaixo a figura de um arquivo de dados XML

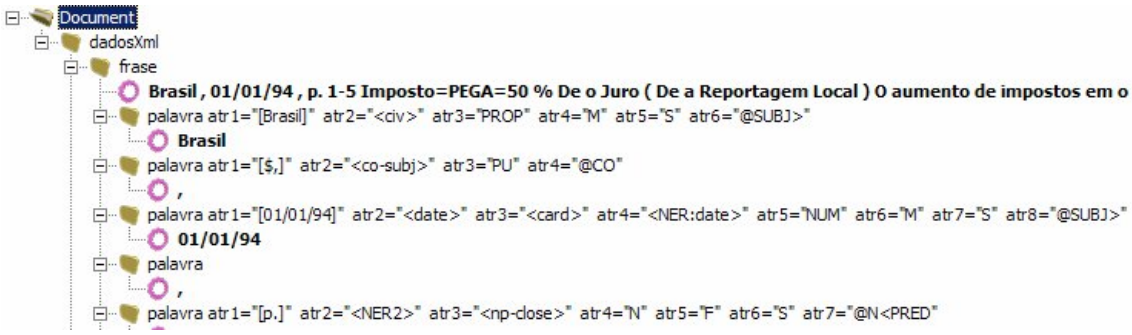

Figura 5 – Arquivo XML de Dados

4. NoPalavra – Esta é uma classe que representa uma palavra real em um texto, ela pode ser visualizada na Figura 3. Basicamente é uma classe que contém uma string com um conteúdo de palavra, uma lista de atributos e os ponteiros Palavra Anterior, Próxima Palavra e Frase Root Pai. Toda esta estrutura de ponteiros está representada na Figura 3 (Estrutura de Dados).

5. NoCondicao – Esta classe representa uma única regra, e quando associado à outros nós de condição, constitui uma árvore de regras que pode ser visualizada na figura 4 (Estrutura de Regras). Basicamente ela contém um atributo chamado código, que é utilizado na hora da montagem da árvore durante a execução do script RegrasEstrutura de acordo com a indentação do arquivo de regras (Figura 1), possui também um atributo chamado texto que armazena a condição, por exemplo, “number_agreement = true” e por último possui dois ponteiros para outros nós condição, um que aponta para o caso da condição ser verdadeira e outro para o caso de condição falsa.

Esta classe contém um método especial que se chama parseXml e transforma a estrutura de regras em um arquivo XML, segue abaixo a figura de um arquivo de regras XML

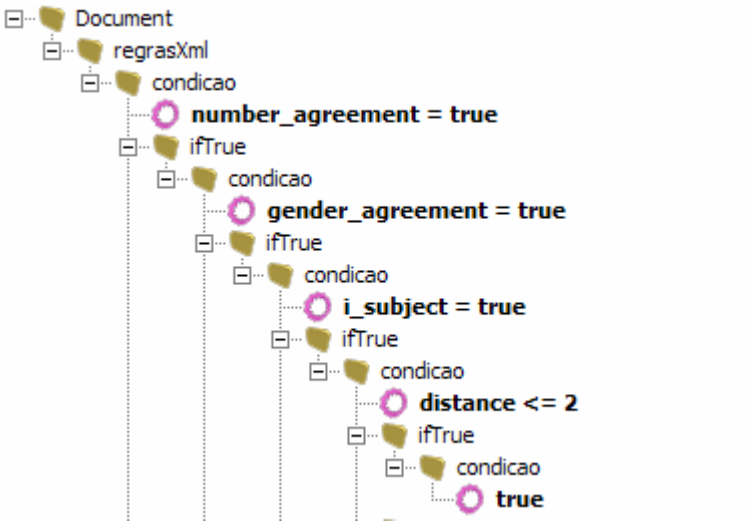

Figura 6 - Arquivo XML de Regras

6. CorreferenciaPronome – Esta é uma classe bem simples, responsável por armazenar o arquivo HTML, produto resultante do algoritmo de co-referência deste trabalho.

7. Executor – Esta é a principal e mais importante classe do projeto, como o próprio nome diz, é ela quem executa o algoritmo de resolução de co-referência de pronomes. É interessante comentar apenas dois métodos principais desta classe.

1 – executaCorreferenciaPronomes – Método responsável por executar o algoritmo de resolução de pronomes. Está bem legível e fácil de ser entendido e se comporta de acordo com o que foi explicado na seção 3.1. Resumindo rapidamente, avança no texto buscando por pronomes (Ele, Ela, Eles, Elas), assim que encontra, recua no texto buscando por seus antecedentes relacionados.

2 – aplicarRegras – O método 1 anterior, realiza chamadas a este método, para aplicar as regras utilizando-se da árvore de decisão (Figura 1), com o objetivo de verificar se a palavra candidato i resolve o pronome j em questão. Um ponto

muito interessante deste método é que ele realiza chamadas dinâmicas ao arquivo Regras.py utilizando-se da função eval() do Python. Desta maneira, esta aplicação prática pode ser facilmente alterada para suportar novas regras, alterando-se apenas o arquivo Regras.py incluindo-se a nova regra desejada e implementando a sua funcionalidade, não é preciso alterar mais nenhum ponto no sistema, a regra é enxergada dinamicamente através deste método. Exemplo:

No arquivo Regras.py existem muitas regras, uma delas é a regra "gender_agreement" que informa se a palavra candidato i e o pronome j combinam em gênero. Segue abaixo o código implementado desta regra no arquivo Regras.py:

```
def gender_agreement(self, iPalavraCandidato, jPronomeCandidato,
distFrases, distPalavras, operador, esperado):
  if 'M' in jPronomeCandidato.getAtributos() and 'M' in
iPalavraCandidato.getAtributos():
      resultado = 'true'
        elif 'F' in jPronomeCandidato.getAtributos() and 'F' in
iPalavraCandidato.getAtributos():
      resultado = 'true'
  else:
      resultado = 'false'
return self.comparaResultados(resultado, esperado)
```

No código acima, para serem considerados do mesmo gênero, palavra e pronome devem apresentar ambos o atributo 'M' ou o atributo 'F'.

Na árvore de decisão existe a seguinte chamada a esta regra:

gender_agreement = true

Como foi citado acima, caso algum usuário do software codificado neste trabalho queira incluir mais uma regra, basta criar uma nova função no arquivo de regras e poderá utilizar na árvore de decisão.

Outro ponto bastante importante é o seguinte trecho de código desta classe:

```
self.listaPronomesValidos = ['Ele','Eles','Ela','Elas']
self.listaCoresPronomes =['blue','green','red','yellow']
```

O trecho acima representa duas listas em Python, uma lista de pronomes que o algoritmo é capaz de resolver e outra lista indicando quais são as cores destes pronomes, essas cores serão colocadas no arquivo html de saída, após o

algoritmo de co-referência. Nestas cores podem ser especificadas os nomes das cores por escrito ou também uma cor no formato RGB, por exemplo, #7FFFD4, que representa a cor Aquamarine 1.

É muito importante dizer também que o arquivo InterfaceGrafica acessa esta lista de pronomes para instanciar dinamicamente os Checkbuttons de pronomes que são renderizados junto com a interface. A figura abaixo mostra os Checkbuttons:

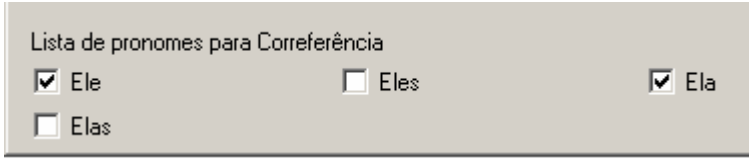

Figura 7 – Checkbuttons Renderizados Dinamicamente

8. InterfaceGrafica – Esta classe é responsável por renderizar uma interface gráfica utilizando a biblioteca padrão do Python chamada Tkinter.

9. Util – Esta classe contém métodos utilitários que servem de ferramenta à todas as demais classes do projeto. Nenhum método em especial deve ser comentado nesta classe.

10.Main – Esta é a classe responsável por instanciar um objeto do tipo InterfaceGrafica e fazer com que o sistema chame o método renderizaInterface do objeto InterfaceGrafica, esta classe não tem nada de especial. Ela apenas inicializa o sistema.

11.Regras – Esta é a classe que contém todas as regras para o algoritmo de co-referência de pronomes, é importante que esta classe contenha todos os métodos existentes na árvore de regras (Figura 1), pois se estiver faltando alguma regra, quando o método aplicarRegra da classe Executar vier aqui procurar uma regra inexistente, o sistema vai apresentar um erro de método não encontrado.

4. Resultados

Nesta seção de resultados é importante ressaltar que os resultados do algoritmo de co-referência dependem única e exclusivamente da árvore de decisão de entrada e da qualidade do texto anotado de entrada. É possível passar para esta aplicação uma árvore de decisão diferente e obter resultados totalmente diferentes.

Dito isto, os resultados aqui apresentados foram gerados utilizando-se a árvore de decisão ilustrada neste documento (figura 1) e um corpus contendo 300 arquivos científicos que foram utilizados no trabalho de Cuevas et. al. (2008), o corpus é adequadamente anotado e pronto para o processo de co-referência de pronomes.

O processo para obter os resultados aqui descritos basicamente segue os seguintes passos:

1. Escolher um arquivo texto anotado para a entrada.

2. Executar o processo de co-referência de pronomes e realizar uma análise manual e comparar com os resultados obtidos pelo algoritmo.

3. Repetir este processo para todos os arquivos anotados existentes no Corpus e gerar tabelas de resultados.

Abaixo seguem algumas tabelas que resumem os resultados obtidos através da execução do algoritmo de co-referência de pronomes com a árvore de decisão ilustrada neste documento (figura 1), utilizando-se o corpus científico utilizado por Cuevas et. al. (2008).

Todos os resultados disponíveis abaixo foram obtidos através da execução do algoritmo desenvolvido neste trabalho. Logo abaixo está uma figura do arquivo HTML de resposta gerado pelo algoritmo:

interesse por os átomos o tornou um especialista em a questão nuclear e férreo defensor de a energia elétrica gerada a=partir=de urânio enriquecido
conhecimento e leve ironia , ele explica por=que o Brasil deve redirecionar os investimentos em esse setor . Seu leque de interesses(0)
música é um de eles(0) -- e não o menos importante . Com a mulher , Lúcia , e alguns amigos , Damy chegou a formar um grupo que tocava
com=preferência para o barroco , em sua casa . Lúcia em o piano , o marido em a flauta doce . Em a casa onde moram é fácil perceber- lhes o gost
, um piano e um pianoforte autêntico , além=de instrumentos=de=corda e sopro . As paredes são forradas de quadros , quase todos pintados por L
premiados em o exterior . Com raro talento , Damy conseguiu unir algumas paixões : como ama a música e é um pioneiro de a física experimental em
lhe pareceu difícil inventar um aparelho para afinar instrumentos(2) . A seguir , os principais trechos de a entrevista . Causou polêmi
a Ciência e Tecnologia , Roberto=Amaral , ter dito que o Brasil deveria dominar todo=o conhecimento nuclear , incluindo o de a bomba=atômica .
vê a possibilidade de o uso de esse tipo de energia em o país ? -- Se olharmos o panorama internacional , veremos que a energia hidrelétrica , que é
motriz de o Brasil , representa menos=de 5 % de a geração de energia elétrica em o mundo inteiro . Agora , a energia hidrelétrica vai muito bem qua
condições(1) favoráveis e há chuvas regulares . Mas quando São=Pedro briga um=pouco com nós , entramos em o apagão . A energi
interessante quando há uma queda-d' água natural que possa ser aproveitada e tenha um regime de funcionamento razoável por todo=o ano . A font
é a mais comum em o mundo é a térmica , de petróleo , carvão , gás . Foi a que Fernando=Henrique=Cardoso escolheu para o Brasil . O país com
número de essas usinas e elas(1) até hoje não entraram em funcionamento . O senhor se refere a o gás ? -- Sim . Então tem que importar gás
gasoduto vai de o Brasil a a Bolívia , para retirar o gás de eles(2) -- o=que é uma coisa absurda , mesmo de o ponto=de=vista estratégico ,

Figura 8 – Arquivo HTML de resposta gerado pelo algoritmo desenvolvido

Os números de acertos e erros foram contabilizados através de uma análise manual sobre os arquivos HTML de saída como o da figura acima. Na realização destas análises, foram considerados apenas os pronomes de terceira pessoa no plural eles / elas, para se realizar uma comparação com o trabalho original Cuevas et. al. (2008).

Resultados Obtidos

Pronome	N° Total no Corpora
Eles	128
Pronomes co-referentes (algoritmo)	
N° Pronomes	114
N° Acerto	59
% Acerto	51.75
Pronomes não co-referentes (algoritmo)	
N° Pronomes	14
% Acerto Total	46.09

Pronome	N° Total no Corpora
Eles	76
Pronomes co-referentes (algoritmo)	
N° Pronomes	71
N° Acerto	57
% Acerto	80.28
Pronomes não co-referentes (algoritmo)	
N° Pronomes	5
% Acerto Total	75.00

% Acerto Total do Algoritmo:	56.86

Tabela 3 – Resultados Obtidos – Análise Manual X Resposta do Algoritmo

Discussão dos Resultados

A tabela 3 é dividida em duas colunas principais, uma para o pronome Eles e outra para o pronome Elas. No topo da tabela 3 existe o número total de pronomes que o corpus científico contém (Eles,Elas) . Existem duas seções (co-referentes e não co-referentes). A seção co-referentes significa que o algoritmo conseguiu encontrar um antecedente julgado como válido para o pronome, e a seção não co-referentes significa que o algoritmo percorreu o texto inteiro, mas não encontrou um antecedente que julgasse válido para a resolução do pronome. Na tabela 3 os casos não co-referentes foram contabilizados como erros do algoritmo, pois os textos foram analisados manualmente e todos tinham termos antecedentes válidos. Observa-se que o algoritmo conseguiu um maior número de acerto para o pronome “Elas” com uma taxa de acerto total de 75 % contra apenas 46.09 % para o pronome “Eles”.

Levando-se em conta os dois pronomes, o algoritmo obteve um índice de acerto geral de 56.86 %. Este índice relativamente baixo revela na verdade uma limitação da abordagem em Cuevas et. al. (2008), cujo índice de acertos original (86,6%) considera cada par pronome-candidato de forma isolada. Em outras palavras, na implementação deste modelo o algoritmo é forçado a escolher apenas um candidato como co-referente (considerando-se neste corpus uma média de 4,5 candidatos por pronome). Isso explica o índice inferior de acertos, e nos indica que o modelo não é suficientemente expressivo para a resolução prática do problema. Entretanto, observamos que este índice de acerto ainda é bastante superior ao que seria obtido, por exemplo, por um baseline que simplesmente escolhesse um candidato de forma aleatória (acertando em média uma vez a cada 4,5 tentativas).

5. Conclusões

A aplicação prática desenvolvida funcionou bem, de acordo com os resultados que eram esperados. Este trabalho recebe um arquivo texto anotado a ser processado e uma árvore de decisão. O algoritmo deve garantir uma execução correta e em um tempo satisfatório.

5.1 – Avaliação crítica a respeito do desenvolvimento na linguagem Python

O Python cumpriu a sua promessa quando disse ser uma linguagem de alto nível, fácil de ser manipulada, deixa os programas com um número menor de linhas quando comparado a outras linguagens tais como o Java e o C, por exemplo.

Particularmente, a experiência foi muito boa, aprendi muitos conceitos que não conhecia tais como a metaprogramação, tipagem de variáveis e execução de código dinamicamente. Alguns outros recursos estudados e que não foram ditos neste trabalho, tais como funções lambda e passagem de funções inteiras como argumentos para outras funções, enfim, consegui ampliar o meu conhecimento com novas técnicas e com certeza foi muito bom.

5.2 – Avaliação crítica à respeito do kit NLTK

A minha experiência com o kit de desenvolvimento NLTK foi muito boa, o kit realmente possui muitas funções que sem dúvida ajudam bastante no desenvolvimento de aplicações de PLN. O que mais me chamou a atenção com relação ao kit foi a didática do livro disponível no site dos autores, é muito fácil aprender e o conteúdo é bastante interessante, não se prende só ao kit, mas ensina diversos recursos da linguagem Python e o livro indica diversas outras bibliotecas que podem ser úteis, de acordo com o tema em questão.

A aplicação prática desenvolvida não utilizou muitos recursos do NLTK, foram utilizados recursos para processamento de Xml, Html e algumas funções para buscas com expressões regulares. Muitos dos recursos estudados no kit não puderam ser aplicados devido à natureza do problema proposto.

Não tenho nenhuma crítica para fazer do kit, gostei bastante e recomendo para que outras pessoas desenvolvam aplicações PLN com ele. Foi uma experiência muito agradável e prazerosa.

5.3 – Contribuições deste trabalho para a área de PLN

Este trabalho contribui com a área de PLN nos seguintes pontos:

- ✓ Apresenta um novo recurso para o desenvolvimento de aplicações PLN, o kit NLTK, que possui muitas funções rotineiras de algoritmos PLN já implementadas e que poupam tempo aos usuários.
- ✓ Deixa uma ferramenta de resolução de co-referência de pronomes de terceira pessoa como exemplo da utilização da linguagem Python e do kit NLTK para que pessoas interessadas possam beneficiar-se com os recursos desenvolvidos.

5.4 – Experiências Adquiridas

- ✓ Aprendizado de uma nova linguagem de programação, o Python, que eu não conhecia. Em especial esta linguagem me trouxe um aprendizado muito grande pelo fato de ela possuir tipagem de variáveis dinamicamente, ser interpretada e bastante flexível, diferentemente da linguagem Java ao qual estou habituado.
- ✓ Aprendi o conceito de metaprogramação, utilizando-me das rotinas que o Python fornece para programar de forma dinâmica e não declarativa, é uma nova filosofia e bastante interessante.

- ✓ Aprendi técnicas de PLN, tais como a resolução anafórica, o conceito de texto anotado, programas que transformam textos (tagger e parser) para que se consiga realizar uma análise sobre eles.

5.5 – Sugestões de Trabalhos Futuros

Existe uma seção disponível no site oficial do NLTK, http://nltk.sourceforge.net/index.php/Projects, que apresenta diversas idéias para trabalhos futuros utilizando-se a biblioteca NLTK, abaixo seguem algumas idéias extraídas do site e algumas opiniões pessoais sobre trabalhos futuros:

- ✓ Desenvolver um analisador morfológico para um idioma qualquer, que classificaria cada frase gramaticalmente e produziria uma espécie de texto anotado, como os textos utilizados neste trabalho por exemplo.
- ✓ Desenvolver um sistema de classificação textual entre dois ou mais idiomas próximos, tal como o Português e Espanhol, por exemplo, para uma análise de similaridade entre eles.
- ✓ Desenvolver uma ferramenta que incorpore as principais funções básicas que os lingüistas utilizam em seu dia a dia, esta ferramenta poderia ser utilizada por eles, sem que os mesmos precisassem aprender programação para utilizá-lo.
- ✓ Desenvolver um sistema de buscas semânticas em textos que permitiria a extração de trechos de textos baseando-se em perguntas formuladas por usuários.

Referências Bibliográficas

BIRD, Steven; KLEIN, Ewan; LOPER, Edward. Natural Language Processing in Python.
Livro gratuito disponível em <http://nltk.sourceforge.net/index.php/Book>
Acesso em: 07 junho 2008.

TOOLKIT, Natural Language. Projeto Open Source desenvolvido em Python.
Página Web do projeto disponível em
<http://nltk.sourceforge.net/index.php/Main_Page>.
Acesso em: 07 junho 2008.

TKINTER, Interface gráfica na linguagem Python. Tutorial Oficial
Página Web disponível em
<http://www.Pythonware.com/library/tkinter/introduction/>.
Acesso em: 07 junho 2008.

PYTHON, Language Programming. Site Oficial da Linguagem Python.
Página Web com documentação e tutoriais disponível em: <http://www.Python.org/doc/>.
Acesso em: 07 junho 2008.

FELDMAN, Ronen; SANGER, James. (2006) The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Primeira Edição. New York. Cambridge University Press.

WITTEN, I. H. & E. Frank (2005) Data Mining: Practical machine learning tools and techniques. 2nd edition Morgan Kaufmann, San Francisco.

CHAVES, Amanda Rocha. A resolução de anáforas pronominais da língua portuguesa com base no algoritmo de Mitkov. Agosto – 2007. 119f – Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de São Carlos – UFSCAR, 2007.

CUEVAS, Ramon Re Moya; PARABONI, Ivandré (2008) A Machine Learning Approach to Portuguese Pronoun Resolution. Submetido. Universidade de São Paulo – Escola de Artes, Ciências e Humanidades – USP EACH, 2008.

Anexo A

Neste anexo, estão disponíveis imagens das telas da interface gráfica e todos os arquivos que constituem o código-fonte da aplicação desenvolvida. Cada arquivo é um script da linguagem Python, e possui a extensão ".py". para executar esta aplicação, coloque todos os arquivos em um mesmo diretório, pode ser o próprio diretório source ou "src", gerado automaticamente pelo Eclipse, quando se inicia uma nova aplicação Python.

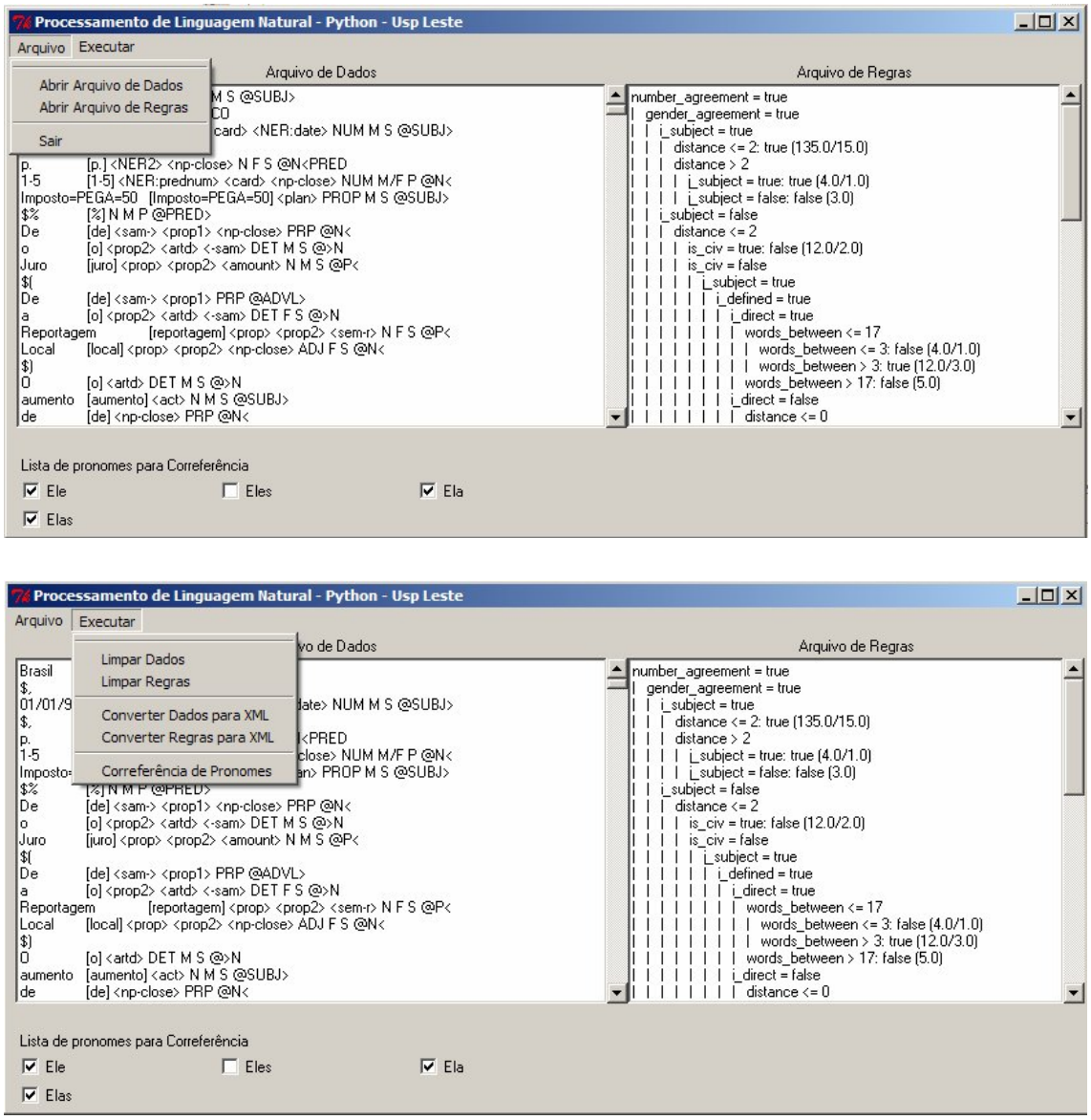

Figura 9 – Interface Gráfica Desenvolvida

Arquivo DadosEstrutura.py

```
# coding: iso-8859-15
from os import SEEK_SET
from NoFrase import *
from NoPalavra import *

class DadosEstrutura:

   def geraEstrutura(self, arquivoDados):
       finalDeFrase = '</s>'
       pontuacao = '$'
       rootFrase = NoFrase()
       ultimoNoFrase = rootFrase
       ultimoNoPalavra = None
       listaPalavras = []
       arquivoDados.seek(SEEK_SET)
       for linha in arquivoDados:
           listaPalavras.append(linha.split())
       noFrase = NoFrase()
       tamanhoLista = len(listaPalavras)
       contador = 0
       for linhaPalavra in listaPalavras:
           contador += 1
           noPalavra = NoPalavra(noFrase)
           palavra = linhaPalavra[0]
           letraInicial = palavra[0]
           if palavra == finalDeFrase or contador == tamanhoLista:
               if ultimoNoFrase.getTexto() != '':
                   ultimoNoFrase.addProximaFrase(noFrase)
               else:
                   rootFrase = noFrase
               ultimoNoFrase = noFrase
               noFrase = NoFrase()
               continue
           elif letraInicial == pontuacao:
               noPalavra.setTexto(palavra[1:])
           else:
               noPalavra.setTexto(palavra)
           if noFrase.getRootPalavra() != None:
               ultimoNoPalavra.addProximaPalavra(noPalavra)
           else:
               noFrase.setRootPalavra(noPalavra)
           noPalavra.setAtributos(linhaPalavra[1:])
           ultimoNoPalavra = noPalavra
       return rootFrase
```

Arquivo RegrasEstrutura.py

```
# coding: iso-8859-15
from os import SEEK_SET
from NoCondicao import *

class RegrasEstrutura:

    def geraRegrasLista(self, arquivoRegras):
        listaRegras = []
        arquivoRegras.seek(SEEK_SET)
        for linha in arquivoRegras:
```

```
        listaTemp = self.formataLinha(linha).split('|')
        listaRegras.append(listaTemp)
    return listaRegras

def geraEstrutura(self, arquivoRegras):
    listaRegras = self.geraRegrasLista(arquivoRegras)
    filaNoCondicao = []
    root = NoCondicao()
    while len(listaRegras) != 0:
        noCondicao = NoCondicao()
        if len(filaNoCondicao) == 0:
            noCondicao = root
        regra = listaRegras.pop(0)
        noCondicao.setCodigoNo(regra[0])
        noCondicao.setTexto(regra[1])
        self.setItemsFilaCondicao(filaNoCondicao, noCondicao)
        filaNoCondicao.append(noCondicao)
        if len(listaRegras) == 0:
            self.setarRespostaNo(noCondicao)
    return root

def setItemsFilaCondicao(self, fila, noAtual):
    if len(fila) > 0:
        noFila = fila.pop()
        codigoAnalisado = noAtual.getCodigoNo()
        if noAtual.getCodigoNo() > noFila.getCodigoNo():
            noFila.setIfTrue(noAtual)
            fila.append(noFila)
        if noAtual.getCodigoNo() == noFila.getCodigoNo():
            noFila.setIfFalse(noAtual)
            self.setarRespostaNo(noFila)
        if noAtual.getCodigoNo() < noFila.getCodigoNo():
            self.setarRespostaNo(noFila)
            self.deletaNosRestantes(fila, noAtual)
            noFila = fila.pop()
            noFila.setIfFalse(noAtual)

def deletaNosRestantes(self, fila, noAtual):
    filaTemp = []
    while len(fila) != 0:
        noFila = fila.pop()
        if noFila.getCodigoNo() <= noAtual.getCodigoNo():
            filaTemp.insert(0, noFila)
            break
    fila.extend(filaTemp)

def setarRespostaNo(self, no):
    codigo = no.getCodigoNo()
    textoCondicao = no.getTexto()
    listaResposta = textoCondicao.split(':')
    textoCondicao = listaResposta[0]
    listaResposta[1] = listaResposta[1].replace(' ','')
    retornoResposta = None
    if listaResposta[1][0] == 'f':
        retornoResposta = 'false'
    if listaResposta[1][0] == 't':
        retornoResposta = 'true'
    no.setTexto(textoCondicao)
    noResposta = NoCondicao()
    noResposta.setTexto(retornoResposta)
    no.setIfTrue(noResposta)

def formataLinha(self, linha):
    contador = 0
    linha = linha.replace('|','')
    linha = linha.replace('\n','')
    for char in linha:
```

```
            if char == ' ':
                contador+=1
            else:
                linha = linha[contador:]
                break
        return str(contador) + '|' + linha
```

Arquivo NoFrase.py

```
# coding: iso-8859-15
from nltk.etree.ElementTree import Element
from NoPalavra import *
from Util import *

class NoFrase:

    def __init__(self):
        self.fraseProx = None
        self.fraseAnt = None
        self.rootPalavra = None
        self.ultimaPalavra = None
        self.util = Util()

    def getTexto(self):
        texto = ''
        palavra = self.rootPalavra
        while palavra != None:
            texto += palavra.getTexto() + ' '
            palavra = palavra.getProximaPalavra()
        return texto

    def getFraseAnterior(self):
        return self.fraseAnt

    def setFraseAnterior(self, frase):
        self.fraseAnt = frase

    def getProximaFrase(self):
        return self.fraseProx

    def setProximaFrase(self, frase):
        self.fraseProx = frase

    def getRootPalavra(self):
        return self.rootPalavra

    def setRootPalavra(self, palavra):
        self.rootPalavra = palavra

    def getUltimaPalavra(self):
        return self.ultimaPalavra

    def setUltimaPalavra(self, ultimaPalavra):
        self.ultimaPalavra = ultimaPalavra

    def addProximaFrase(self, novaFrase):
        novaFrase.setFraseAnterior(self)
        self.setProximaFrase(novaFrase)

    def parseXml(dadosEstrutura):
        rootFrase = dadosEstrutura
        rootDados = Element('dadosXml')
        while rootFrase != None:
```

```
            frase = Element('frase')
            frase.text = rootFrase.getTexto()
            rootDados.append(frase)
            rootPalavra = rootFrase.getRootPalavra()
            while rootPalavra != None:
                palavra = Element('palavra')
                palavra.text = rootPalavra.getTexto()
                contador = 1
                for atributo in rootPalavra.getAtributos():
                    palavra.attrib['atr'+str(contador)] = atributo
                    contador += 1
                frase.append(palavra)
                rootPalavra = rootPalavra.getProximaPalavra()
            rootFrase = rootFrase.getProximaFrase()
        return rootDados

    def contemPalavra(self, palavra):
        noPalavra = self.getRootPalavra()
        while noPalavra != None:
            if self.util.comparaStringsInsensitive(palavra, noPalavra.getTexto()):
                return True
            noPalavra = noPalavra.getProximaPalavra()
        return False
```

Arquivo NoPalavra.py

```
# coding: iso-8859-15
from nltk.etree.ElementTree import Element

class NoPalavra:

    def __init__(self, noFrasePai, textoPalavra=''):
        self.texto = textoPalavra
        self.atributos = None
        self.palavraProx = None
        self.palavraAnt = None
        self.rootFrasePai = noFrasePai

    def getTexto(self):
       return self.texto

    def setTexto(self, textoPalavra):
       self.texto = textoPalavra

    def getAtributos(self):
        return self.atributos

    def setAtributos(self, atributos):
        self.atributos = atributos

    def getPalavraAnterior(self):
        return self.palavraAnt

    def setPalavraAnterior(self, palavra):
        self.palavraAnt = palavra

    def getProximaPalavra(self):
        return self.palavraProx

    def setProximaPalavra(self, palavra):
        self.palavraProx = palavra

    def getNoFrasePai(self):
        return self.rootFrasePai
```

```
    def addProximaPalavra(self, novaPalavra):
        novaPalavra.setPalavraAnterior(self)
        self.setProximaPalavra(novaPalavra)
        self.rootFrasePai.setUltimaPalavra(novaPalavra)
```

Arquivo NoCondicao.py

```
# coding: iso-8859-15

from nltk.etree.ElementTree import Element

class NoCondicao:

    def __init__(self, textoCondicao=''):
        self.texto = textoCondicao
        self.codigoNo = -1
        self.ifTrue = None
        self.ifFalse = None

    def getTexto(self):
       return self.texto

    def getCodigoNo(self):
        return int(self.codigoNo)

    def getIfTrue(self):
        return self.ifTrue

    def setIfTrue(self, noTrue):
        self.ifTrue = noTrue

    def getIfFalse(self):
        return self.ifFalse

    def setIfFalse(self, noFalse):
        self.ifFalse = noFalse

    def setTexto(self, texto):
        self.texto = texto

    def setCodigoNo(self, codigo):
        self.codigoNo = codigo

    def parseXml(regrasEstrutura):
        rootCondicao = regrasEstrutura
        rootRegras = Element('regrasXml')
        condicaoPai = Element('condicao')
        condicaoPai.text = rootCondicao.getTexto()
        rootRegras.append(condicaoPai)
        pilhaRegras = []
        pilhaRegrasXml = []
        pilhaRegras.append(rootCondicao)
        pilhaRegrasXml.append(condicaoPai)
        while len(pilhaRegras) != 0:
            noRegra = pilhaRegras.pop()
            noRegraXml = pilhaRegrasXml.pop()

            if noRegra.getIfTrue() != None:
                noIfTrue = Element('ifTrue')
                noRegraXml.append(noIfTrue)
                condicaoTrue = Element('condicao')
                condicaoTrue.text = noRegra.getIfTrue().getTexto()
                noIfTrue.append(condicaoTrue)
                pilhaRegras.append(noRegra.getIfTrue())
                pilhaRegrasXml.append(condicaoTrue)
```

```
            if noRegra.getIfFalse() != None:
                noIfFalse = Element('ifFalse')
                noRegraXml.append(noIfFalse)
                condicaoFalse = Element('condicao')
                condicaoFalse.text = noRegra.getIfFalse().getTexto()
                noIfFalse.append(condicaoFalse)
                pilhaRegras.append(noRegra.getIfFalse())
                pilhaRegrasXml.append(condicaoFalse)

        return rootRegras
```

Arquivo CorreferenciaPronome.py

```
# coding: iso-8859-15
from nltk.etree.ElementTree import Element

class CorreferenciaPronome:

    def __init__(self):
        self.html = Element('html')
        self.head = Element('head')
        self.title = Element('title')
        self.body = Element('body')
        self.html.append(self.head)
        self.html.append(self.body)
        self.head.append(self.title)

    def getTitulo(self):
       return self.title.text

    def setTitulo(self, titulo):
        self.title.text = titulo

    def setConteudo(self, conteudo):
        self.body.text = conteudo

    def getHtml(self):
       return self.html
```

Arquivo Executor.py

```
# coding: iso-8859-15

from Util import *
from Regras import *
from DadosEstrutura import *
from RegrasEstrutura import *
from CorreferenciaPronome import *

class Executor:

    def __init__(self):
        self.regras = Regras()
        self.dadosEstrutura = DadosEstrutura()
        self.regrasEstrutura = RegrasEstrutura()
        self.util = Util()
        self.listaPronomesValidos = ['Ele','Eles','Ela','Elas']
        self.listaCoresPronomes = ['blue','green','red','yellow']
        self.dicionarioCoresPronomes = {}
        self.listaPronomesSelecionados = []
```

```
    def executaCorreferenciaPronomes(self, dadosEstrutura,
regrasEstrutura):
        contador = 0
        resultadoEstrutura = dadosEstrutura
        self.dicionarioCoresPronomes =
self.util.geraDicionario(self.listaPronomesValidos,
self.listaCoresPronomes)
        while dadosEstrutura != None:
            for pronome in self.listaPronomesSelecionados:
                if dadosEstrutura.contemPalavra(pronome):
                    dadosPalavra = dadosEstrutura.getRootPalavra()
                    while dadosPalavra != None:
                        if
self.util.comparaStringsInsensitive(dadosPalavra.getTexto(), pronome):
                            fraseAtual = dadosEstrutura
                            distFrases = 0
                            distPalavras = 1
                            while fraseAtual != None:
                                if fraseAtual == dadosEstrutura:
                                    palavraAtual =
dadosPalavra.getPalavraAnterior()
                                else:
                                    palavraAtual =
fraseAtual.getUltimaPalavra()
                                while palavraAtual != None:
                                    iPalavraCandidato = palavraAtual
                                    jPronomeCandidato = dadosPalavra
                                    resultado =
self.aplicarRegras(iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, regrasEstrutura)
                                    if resultado == 'true':
                                        cor =
self.dicionarioCoresPronomes.get(self.util.retornaPronomeValido(jPronomeCa
ndidato.getTexto(), self.listaPronomesValidos))

palavraAtual.setTexto(self.util.aplicaMarcadorHtml(iPalavraCandidato.getTe
xto(), contador, cor))

jPronomeCandidato.setTexto(self.util.aplicaMarcadorHtml(jPronomeCandidato.
getTexto(), contador, cor))
                                        palavraAtual = None
                                        fraseAtual = None
                                        contador += 1
                                        continue
                                    palavraAtual =
palavraAtual.getPalavraAnterior()
                                    distPalavras += 1
                                if fraseAtual != None:
                                    fraseAtual =
fraseAtual.getFraseAnterior()
                                    distFrases += 1
                        dadosPalavra = dadosPalavra.getProximaPalavra()
            dadosEstrutura = dadosEstrutura.getProximaFrase()
        return resultadoEstrutura

    def aplicarRegras(self, iPalavraCandidato, jPronomeCandidato,
distFrases, distPalavras, regrasEstrutura):
        if self.regras.i_candidate(iPalavraCandidato):
            noCondicao = regrasEstrutura
            listaCondicao = noCondicao.getTexto().split(" ")
            argumento1 = listaCondicao[0]
            while argumento1 not in ('true', 'false'):
                operador = listaCondicao[1]
                argumento2 = listaCondicao[2]
                if eval('self.regras.'+argumento1+'(iPalavraCandidato,
jPronomeCandidato, distFrases, distPalavras, operador, argumento2)'):
                    noCondicao = noCondicao.getIfTrue()
```

```
                else:
                    noCondicao = noCondicao.getIfFalse()
                listaCondicao = noCondicao.getTexto().split(" ")
                argumento1 = listaCondicao[0]
            return argumento1
        else:
            return 'false'

    def transformaDadosXml(self, arquivoDados):
        dados = self.dadosEstrutura.geraEstrutura(arquivoDados)
        return dados.parseXml()

    def transformaRegrasXml(self, arquivoRegras):
        regras = self.regrasEstrutura.geraEstrutura(arquivoRegras)
        return regras.parseXml()

    def geraArquivoResultadoHtml(self, resultadoEstrutura):
        texto = ''
        noFrase = resultadoEstrutura
        while noFrase != None:
            texto += noFrase.getTexto()
            noFrase = noFrase.getProximaFrase()
        return texto

    def getPronomesValidos(self):
        return self.listaPronomesValidos

    def setListaPronomesSelecionados(self, lista):
        self.listaPronomesSelecionados = lista

    def getListaPronomesSelecionados(self):
        return self.listaPronomesSelecionados
```

Arquivo InterfaceGrafica.py

```
# coding: iso-8859-15

from Tkinter import *
from tkFileDialog import *
from tkMessageBox import *
from Executor import *
from DadosEstrutura import *
from RegrasEstrutura import *
from Util import *
from nltk.etree import ElementTree as ET

class InterfaceGrafica():
    def __init__(self):
        self.root = Tk()
        self.root.title('Processamento de Linguagem Natural - Python - Usp
Leste')
        self.listaCheckboxes = []
        self.util = Util()
        self.dadosEstrutura = DadosEstrutura()
        self.regrasEstrutura = RegrasEstrutura()
        self.correferenciaPronome = CorreferenciaPronome()
        self.executor = Executor()
        self.arquivoDados = None
        self.arquivoRegras = None
        self.criaTextRegras()
        self.criaTextDados()
        self.criaMenu()
        self.criaCheckboxes()
```

```
        self.renderizaInterface()

    def criaMenu(self):
        self.menu = Menu(self.root)
        filemenu = Menu(self.menu, tearoff=0)
        self.menu.add_cascade(label='Arquivo', menu=filemenu)
        filemenu.add_separator()
        filemenu.add_command(label='Abrir Arquivo de Dados',
command=self.abrirArquivoDados)
        filemenu.add_command(label='Abrir Arquivo de Regras',
command=self.abrirArquivoRegras)
        filemenu.add_separator()
        filemenu.add_command(label='Sair', command=self.sair)
        execmenu = Menu(self.menu, tearoff=0)
        self.menu.add_cascade(label='Executar', menu=execmenu)
        execmenu.add_separator()
        execmenu.add_command(label='Limpar Dados',
command=self.limparDados)
        execmenu.add_command(label='Limpar Regras',
command=self.limparRegras)
        execmenu.add_separator()
        execmenu.add_command(label='Converter Dados para XML',
command=self.salvarArquivoDados)
        execmenu.add_command(label='Converter Regras para XML',
command=self.salvarArquivoRegras)
        execmenu.add_separator()
        execmenu.add_command(label='Correferência de Pronomes',
command=self.salvarArquivoResultado)

    def criaTextDados(self):
        self.labelDados = Label(self.root, text='Arquivo de Dados',
width=75)
        self.scrollbarDados = Scrollbar(self.root, orient=VERTICAL)
        self.textDados = Text(self.root, width=75, height=20,
state=DISABLED, yscrollcommand=self.scrollbarDados.set)
        self.scrollbarDados.config(command=self.setScrollDados)

    def criaTextRegras(self):
        self.labelRegras = Label(self.root, text='Arquivo de Regras',
width=55)
        self.scrollbarRegras = Scrollbar(self.root, orient=VERTICAL)
        self.textRegras = Text(self.root, width=55, height=20,
state=DISABLED, yscrollcommand=self.scrollbarRegras.set)
        self.scrollbarRegras.config(command=self.setScrollRegras)

    def criaCheckboxes(self):
        self.labelCheckboxes = Label(self.root, text='Lista de pronomes
para Correferência', width=75, anchor=W, justify=LEFT)
        contador = 1
        listaPronomesValidos = self.executor.getPronomesValidos()
        for pronome in listaPronomesValidos:
            globals()['statusCheckbox' + str(contador)] =
StringVar(self.root)
            eval('statusCheckbox' + str(contador) +
'.set("naoSelecionado")')
            checkbox = eval('Checkbutton(self.root, text="'+pronome+'",
var=statusCheckbox' + str(contador) + ', onvalue="'+pronome+'",
offvalue="naoSelecionado",
command=self.atualizaListaPronomesSelecionados)')
            self.listaCheckboxes.append(checkbox)
            contador += 1

    def atualizaListaPronomesSelecionados(self):
        lista = []
        listaPronomesValidos = self.executor.getPronomesValidos()
        for item in range(len(listaPronomesValidos)):
            pronome = eval('statusCheckbox' + str(item+1) + '.get()')
```

```
            if pronome != 'naoSelecionado':
                lista.append(pronome)
        self.executor.setListaPronomesSelecionados(lista)

    def renderizaInterface(self):
        self.root.config(menu=self.menu)
        self.labelDados.grid(columnspan=4, row=0, column=1, sticky=W+E)
        self.labelRegras.grid(columnspan=4, row=0, column=5, sticky=W+E)
        self.textDados.grid(columnspan=3, row=1, column=1)
        self.scrollbarDados.grid(row=1, column=4, sticky=N+S)
        self.textRegras.grid(columnspan=3, row=1, column=5)
        self.scrollbarRegras.grid(row=1, column=8, sticky=N+S)
        self.renderizaCheckboxes(linhaInicial=2)

    def renderizaCheckboxes(self, linhaInicial):
        Label(text='').grid(row=linhaInicial)
        self.labelCheckboxes.grid(columnspan=3, row=linhaInicial+1,
column=1)
        numLinhas = linhaInicial+2
        contador = 1
        for checkbox in self.listaCheckboxes:
            checkbox.grid(row=numLinhas, column=contador, sticky=W)
            if contador % 3 == 0:
                numLinhas += 1
                contador = 0
            contador += 1

    def setScrollDados(self, *args):
        apply(self.textDados.yview, args)

    def setScrollRegras(self, *args):
        apply(self.textRegras.yview, args)

    def abrirArquivoDados(self):
        initial_dir = 'C:/textos'
        mask = [('Arquivos de Texto','*.txt')]
        fileDados = askopenfile(initialdir=initial_dir, filetypes=mask,
mode='r')
        if fileDados != None:
            text = fileDados.read()
            if text != None:
                self.arquivoDados = fileDados
                self.textDados.config(state=NORMAL)
                self.textDados.delete(0.0, END)
                self.textDados.insert(END,text)
                self.textDados.config(state=DISABLED)

    def abrirArquivoRegras(self):
        initial_dir = 'C:/textos'
        mask = [('Arquivos de Texto','*.txt')]
        fileRegras = askopenfile(initialdir=initial_dir, filetypes=mask,
mode='r')
        if fileRegras != None:
            text = fileRegras.read()
            if text != None:
                self.arquivoRegras = fileRegras
                self.textRegras.config(state=NORMAL)
                self.textRegras.delete(0.0, END)
                self.textRegras.insert(END,text)
                self.textRegras.config(state=DISABLED)

    def salvarArquivoDados(self):
        if self.arquivoDados != None:
            initial_dir = 'C:/'
            mask = [('Arquivos Xml','*.xml')]
            fileDadosXml = asksaveasfile(initialdir=initial_dir,
filetypes=mask, defaultextension='.xml', mode='w')
```

```
            dadosXml = ET.tostring(self.transformaDadosXml())
            fileDadosXml.write(dadosXml)
            showinfo('Transformação Dados XML','Processo executado com
SUCESSO !')
        else:
            showwarning('Transformar Dados para XML','Para transformar é
preciso abrir o arquivo de dados !')

    def salvarArquivoRegras(self):
        if self.arquivoRegras != None:
            initial_dir = 'C:/'
            mask = [('Arquivos Xml','*.xml')]
            fileRegrasXml = asksaveasfile(initialdir=initial_dir,
filetypes=mask, defaultextension='.xml', mode='w')
            regrasXml = ET.tostring(self.transformaRegrasXml())
            fileRegrasXml.write(regrasXml)
            showinfo('Transformação Regras XML','Processo executado com
SUCESSO !')
        else:
            showwarning('Transformar Regras para XML','Para transformar é
preciso abrir o arquivo de regras !')

    def salvarArquivoResultado(self):
        if self.arquivoDados == None or self.arquivoRegras == None:
            showinfo('Executar Correferência de Pronomes','Para executar é
preciso abrir um Arquivo de Dados e um Arquivo de Regras')
        elif len(self.executor.getListaPronomesSelecionados()) == 0:
            showinfo('Executar Correferência de Pronomes','Para executar é
preciso selecionar um Pronome na Lista de Correferência')
        else:
            initial_dir = 'C:/'
            mask = [('Arquivos Html','*.html')]
            fileResultadoHtml = asksaveasfile(initialdir=initial_dir,
filetypes=mask, defaultextension='.xml', mode='w')
            dados = self.dadosEstrutura.geraEstrutura(self.arquivoDados)
            regras =
self.regrasEstrutura.geraEstrutura(self.arquivoRegras)
            resultadoEstrutura =
self.executor.executaCorreferenciaPronomes(dados, regras)
            conteudo =
self.executor.geraArquivoResultadoHtml(resultadoEstrutura)
            self.correferenciaPronome.setTitulo('Resultado de
Correferência de Pronomes utilizando o arquivo de Dados =
'+self.arquivoDados.name+' e o arquivo de Regras =
'+self.arquivoRegras.name)
            self.correferenciaPronome.setConteudo(conteudo)
            textoHtml = ET.tostring(self.correferenciaPronome.getHtml())
            textoHtml = self.util.converteAsciiParaTexto(textoHtml)
            fileResultadoHtml.write(textoHtml)
            showinfo('Executar Correferência','Processo executado com
SUCESSO !')

    def limparDados(self):
        self.textDados.config(state=NORMAL)
        self.arquivoDados = None
        self.textDados.delete(0.0, END)
        self.textDados.config(state=DISABLED)

    def limparRegras(self):
        self.textRegras.config(state=NORMAL)
        self.arquivoRegras = None
        self.textRegras.delete(0.0, END)
        self.textRegras.config(state=DISABLED)

    def transformaDadosXml(self):
        dadosXml = self.executor.transformaDadosXml(self.arquivoDados)
        return dadosXml
```

```
    def transformaRegrasXml(self):
        regrasXml = self.executor.transformaRegrasXml(self.arquivoRegras)
        return regrasXml

    def sair(self):
        self.root.destroy()

    def iniciaInterface(self):
        self.root.mainloop()
```

Arquivo Util.py

```
# coding: iso-8859-15
import re

class Util():

    def converteAsciiParaTexto(self, texto):
        p = re.compile('&#\d+;')
        listaAscii = list(set(p.findall(texto)))
        for item in listaAscii:
            codAscii = item[2:-1]
            texto = texto.replace(item, chr(int(codAscii)))
        texto = texto.replace('&amp;', '&')
        texto = texto.replace('&lt;', '<')
        texto = texto.replace('&gt;', '>')
        return texto

    def aplicaMarcadorHtml(self, texto, contador, cor):
        return '<font size=6 color='+cor+'>' + texto + '(' +
str(contador) + ')' + '</font>'

    def geraDicionario(self, lista1, lista2):
        dicionario = {}
        if len(lista1) == len(lista2):
            for indice in range(len(lista1)):
                item1 = lista1[indice]
                item2 = lista2[indice]
                dicionario[item1] = item2
        return dicionario


    def retornaPronomeValido(self, pronome, listaPronomes):
        for item in listaPronomes:
            if self.comparaStringsInsensitive(pronome, item):
                return item
        return None

    def comparaStringsInsensitive(self, str1, str2):
        if str1.lower() == str2.lower():
            return True
        else:
            return False
```

Arquivo Main.py

```
# coding: iso-8859-15
from InterfaceGrafica import *

interface = InterfaceGrafica()
interface.iniciaInterface()
```

Arquivo Regras.py

```
# coding: iso-8859-15
class Regras:

    def comparaResultados(self, resultado, esperado):
        if resultado.lower() == esperado.lower():
            return True
        else:
            return False

    def number_agreement(self, iPalavraCandidato, jPronomeCandidato,
distFrases, distPalavras, operador, esperado):
        if '3S' in jPronomeCandidato.getAtributos() and 'S' in
iPalavraCandidato.getAtributos():
            resultado = 'true'
        elif '3P' in jPronomeCandidato.getAtributos() and 'P' in
iPalavraCandidato.getAtributos():
            resultado = 'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)

    def gender_agreement(self, iPalavraCandidato, jPronomeCandidato,
distFrases, distPalavras, operador, esperado):
        if 'M' in jPronomeCandidato.getAtributos() and 'M' in
iPalavraCandidato.getAtributos():
            resultado = 'true'
        elif 'F' in jPronomeCandidato.getAtributos() and 'F' in
iPalavraCandidato.getAtributos():
            resultado = 'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)

    def i_subject(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if '@SUBJ>' in iPalavraCandidato.getAtributos():
            resultado = 'true'
        elif '@<SUBJ' in iPalavraCandidato.getAtributos():
            resultado = 'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)

    def j_subject(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if '@SUBJ>' in jPronomeCandidato.getAtributos():
            resultado = 'true'
        elif '@<SUBJ' in jPronomeCandidato.getAtributos():
            resultado = 'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)
```

```
    def i_direct(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if '@<ACC' in iPalavraCandidato.getAtributos():
            resultado = 'true'
        elif '@ACC>' in iPalavraCandidato.getAtributos():
            resultado = 'true'
        elif '@ACC>>' in iPalavraCandidato.getAtributos():
            resultado = 'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)

    def i_defined(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if iPalavraCandidato.getPalavraAnterior() == None:
            resultado = 'false'
        else:
            if 'artd' in
iPalavraCandidato.getPalavraAnterior().getAtributos():
                resultado = 'true'
            else:
                resultado = 'false'
        return self.comparaResultados(resultado, esperado)

    def i_candidate(self, iPalavraCandidato):
        if 'N' in iPalavraCandidato.getAtributos():
            return True
        elif 'PROP' in iPalavraCandidato.getAtributos():
            return True
        else:
            return False

    def distance(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if eval('distFrases ' + operador + ' ' +esperado):
            return True
        else:
            return False

    def same_sentence(self, iPalavraCandidato, jPronomeCandidato,
distFrases, distPalavras, operador, esperado):
        if distFrases == 0:
            return True
        else:
            return False

    def pronoun_type(self, iPalavraCandidato, jPronomeCandidato,
distFrases, distPalavras, operador, esperado):
        if iPalavraCandidato.getPalavraAnterior() == None:
            resultado = '0'
        else:
            if 'de' == iPalavraCandidato.getPalavraAnterior().getTexto():
                resultado = '2'
            elif 'em' ==
iPalavraCandidato.getPalavraAnterior().getTexto():
                resultado = '3'
            else:
                resultado = '1'
        return self.comparaResultados(resultado, esperado)

    def words_between(self, iPalavraCandidato, jPronomeCandidato,
distFrases, distPalavras, operador, esperado):
        if eval('distPalavras ' + operador + ' ' +esperado):
            return True
        else:
            return False
```

```
    def is_civ(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if '<civ>' in iPalavraCandidato.getAtributos():
            resultado =  'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)

    def is_hh(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if '<hh>' in iPalavraCandidato.getAtributos():
            resultado =  'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)

    def is_inst(self, iPalavraCandidato, jPronomeCandidato, distFrases,
distPalavras, operador, esperado):
        if '<inst>' in iPalavraCandidato.getAtributos():
            resultado =  'true'
        else:
            resultado = 'false'
        return self.comparaResultados(resultado, esperado)
```