1ª Lista de Exercícios - Mineração de Texto

Maria Fernanda Souza Andrade - fsandrade25@gmail.com
Saulo Lucas Gomes Ferreira - saulolucas.gm@gmail.com

2 de outubro de 2018

1 Quais as vantagens e desvantagens dos tipos de anotação de corpora inline vs. stand-off?

Uma das vantagens observadas na anotação *inline* comparada a anotação *stand-off*, é o metadado já está acompanhado no mesmo arquivo que o dado, assim a extração dessa informação torna-se mais rápida comparando com a outra anotação. No entanto, essa forma de estruturação também se mostra como uma desvantagem, pois é necessário realizar um pré-processamento com o intuito de separar o dado do metadado.

Em contra partida para a anotação *stand-off* a sua vantagem é exatamente o seus metadados já estarem separados em um arquivo diferente, entretanto, a necessidade de realizar referências entre esses dois arquivos torna a leitura custosa aparecendo como uma desvantagem para o método.

2 Ao seu ver, porque o formato híbrido, isto é, a anotação "em camadas"é o mais usado hoje em dia?

A organização sistemática feita dessa forma evita que se suje ou acumule informações demais no conteúdo principal. Separar dados dá mais legibilidade e, consequentemente, diminui os erros. Utilizando índices para ligar diferentes camadas mantém os dados divididos de acordo com a sua função na anotação e evita mistura de informações.

3 Quais tipos de anotação são realizadas pela ferramenta BRAT (http://brat.nlplab.org)? Para quais problemas de mineração de textos ela pode ser usada?

Essa ferramenta possibilita realizar marcações nas palavras presentes no texto informando o tipo de entidade que elas estão vinculadas, além disso, ela também realiza uma analise de dependência entre palavras(anotações) de uma sentença, normaliza palavras, permite classificar as frases como sendo substantiva, verbal ou preposicional. Sendo assim, essa ferramenta é mais utilizada para problemas do tipo de extração de informação e processamento de linguagem natural.

4 O que se entende por POS tagging? E qual a diferença entre POS taging e parsing?

POS tagging contém funções gramaticais simples que são utilizadas para marcar separadamente cada palavra que compõe um texto, enquanto que o *parsing* se baseia em dependências binárias entre palavras.

5 Discuta sobre as propriedades do parsing constituinte e o parsing de dependências.

São duas diferentes maneiras de se obter a constituição sintática de um texto, que se baseiam em diferentes conexões de papéis semânticos de uma palavra num determinado contexto.

O *parsing constituinte* faz uma divisão hierárquica do contexto geral até a separação atômica das palavras e a determinação do seu papel na frase. É, visualmente, mais prático e fácil de se analisar humanamente, tendo em vista que a pirâmide que se é obtida na separação gradativa é bem explícita nas camadas que separam determinadas classes. No entanto, se mostra incapaz de relacionar semanticamente duas palavras atômicas de forma direta, o faz em contexto de função de dois campos semânticos no

período, como a relação de sujeito e predicado, mas a relação direta entre palavras, na maior parte das vezes, fica implícita.

Se a explicitação dessa relação for necessária, o *parsing de dependências* se mostra mais eficaz para essa tarefa, pois tem foco em como as palavras se relacionam. Esse modelo conecta palavras com dependências semânticas numa oração. Todavia, esse modelo parece complexo ocasionalmente até para humanos, quem dirá máquinas.

6 Um algoritmo de tokenização para a língua portuguesa.

```
public static String getNextToken(){
        char ch;
        String token = new String();

        while(true){
            cursor = currentPosition;
            if(stream.length() >= cursor) ch = stream.charAt(cursor);
            else return null;

            while(cursor < stream.length()-1 && !isDelimiterSet(ch)){
                ch = stream.charAt(++cursor);
            }

            if(cursor >= stream.length()){
                cursor = currentPosition;
                return null;
            }

            if(isEmptySpace(ch)){
                if(cursor == currentPosition){
                    currentPosition+=1;
                    return null;
                }
                else{
                    token = stream.substring(currentPosition,cursor);
                    currentPosition = cursor+1;
                    return token;
                }
            }

            if(cursor == currentPosition){
                token = ch + "";
                currentPosition = cursor+1;
                }else{
                    token = stream.substring(currentPosition,cursor);
                currentPosition = cursor;
                return token;
            }

            return null;
        }
}

public static boolean isDelimiterSet(char ch){
        if(delimiterSet.contains(ch + "")) return true;
        return false;
}

public static boolean isEmptySpace(char ch){
        if(whiteSpace.contains(ch + "")) return true;
        return false;
}
```

7 Um algoritmo para segmentação de sentenças(orações) de textos em português.

```
public static String segmentation(){
        StringBuilder text = new StringBuilder(stream);

        char ch;
        for(int i = 0; i < text.length()-1; i++){
            ch = text.charAt(i);
            if(text.length() == '?' || text.length() == '!'){
                text.replace(i+1, i+2, "\n");
            }
```

```
            else if((ch == '\"' || ch == '\'') && text.charAt(i+1) == '.'){
                text.replace(i+1, i+2, "\n");

            }
            else if(text.charAt(i+1) != ' '){

            }
            else if(ch == ']' || ch == ')' || ch == '}'){
                if(text.charAt(i+1) == '.'){
                    text.replace(i+1, i+2, "\n");
                }
            }
            else if(ch == '.'){
                if(Character.isUpperCase(thisToken(i-1).charAt(0)) && thisToken(i-1).
    length() < 5 && Character.isUpperCase(text.charAt(i+2))){

                }else if(Character.isLowerCase(thisToken(i-1).charAt(0)) && Character.
    isUpperCase(text.charAt(i+2))){
                    text.replace(i+1, i+2, "\n");
                }else if(thisToken(i-1).length() < 2){

                }else{
                    text.replace(i+1, i+2, "\n");
                }
            }
            else if(ch == ' '){
                if(text.charAt(i+1) == '(' || text.charAt(i+1) == '{' || text.charAt(i+1)
     == '[' || text.charAt(i+1) == '\'' || text.charAt(i+1) == '\"' || text.charAt(i+1)
    == '$'){
                    text.replace(i, i+1, "\n");
                }
            }
        }
        return text.toString();
}

// Funcao que retorna o token daquela determinada posicao
public static String thisToken(int i){
        while(!whiteSpace.contains(stream.charAt(i) + "") && !delimiterSet.equals(stream.
    charAt(i) + "")){

            i--;

            //System.out.println("a");
        }
        int first = ++i;
        while(!whiteSpace.contains(stream.charAt(i) + "") && !delimiterSet.equals(stream.
    charAt(i) + "")){

            i++;
            //System.out.println("a");
        }
        return stream.substring(first,i);
}
```

8 Use o NLTK para criar um pipeline.

```
arq = open("texto_en.txt", "r")

texto = ""
texto = arq.read()

# ------ Tokenization --------
tokens = nltk.word_tokenize(texto)
print("[INFO] Tokenization:")
print(tokens)

# ------ Sentence Splitting ------
sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
sentences = sent_detector.tokenize(texto.strip())
print("[INFO] Sentence Splitting:")
print(sentences)

# ------ Lemmatization (Reduce inflections or variant forms to base form)-------
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
lema = {}
```

```
for w in tokens:
    if w not in lema:
        lema[w] = wordnet_lemmatizer.lemmatize(w)
print("[INFO] Lematization:")
print(lema)

# ----- Stemming (Reduce terms to their stems or roots)-----
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()
stem = {}
for w in tokens:
    if w != ',' and w != '.' and w != '!' and w != '?' and w != '"' and w != "'" and w
    != '``':
        if w not in stem:
            stem[w] = porter_stemmer.stem(w)
print("[INFO] Stemming:")
print(stem)

# ----- POS tagging -----
tagged = nltk.pos_tag(tokens)
print("[INFO] POS tagging:")
print(tagged)

arq.close()
```

```
# Respostas:
print("\n")
split = []
for w in tokens:
    if w != ',' and w != '.' and w != '!' and w != '?' and w != '"' and w != "'" and w
    != '``':
        split.append(w)
print("Letra A: ", len(split))  # Resposta 693
print("Letra B: ", len(stem.keys()))   # Resposta 326

tokens_per_sentence = []
for s in sentences:
    token = nltk.word_tokenize(s)
    tokens_per_sentence.append(len(token))

print("Letra C: {0} sentencas e {1:2.4f} media de tokens por sentenca".format(len(
    sentences), (sum(tokens_per_sentence)/len(tokens_per_sentence))))

print("Letra E: Grafico em anexo")

tag_fd = nltk.FreqDist(tag for (word, tag) in tagged)

freq = []
tags = []
for i in tag_fd.most_common():
    tags.append(i[0])
    freq.append(i[1])

#--------- Grafico questao 08 letra e ----------
plt.figure(0)
plt.bar(tags, freq)
plt.xlabel("TAGS")
plt.ylabel("Frequency")
plt.title("Question 8 Letter E")
plt.grid(True)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5, forward=True)
plt.savefig('letter_e.png')

print("Letra F: ")
stem_cont = {}
for w in tokens:
    if w != ',' and w != '.' and w != '!' and w != '?' and w != '"' and w != "'" and w
    != '``':
        w_aux = porter_stemmer.stem(w)
        if w_aux not in stem_cont:
            stem_cont[w_aux] = 1
        elif w_aux in stem_cont:
            cont = stem_cont[w_aux]
            stem_cont[w_aux] = cont + 1

from operator import itemgetter
stem_cont = dict(reversed(sorted(stem_cont.items(), key=itemgetter(1))))
```

```
51
52 freq = list(stem_cont.values())
53 stems = list(stem_cont.keys())
54 print(freq)
55 print(stems)
56
57 # ------- Grafico questao 08 letra F ----------
58 plt.figure(1)
59 plt.bar(stems[0:15], freq[0:15])
60 plt.xlabel("Steamming")
61 plt.ylabel("Frequency")
62 #plt.yticks(sorted(set(list(stem_cont.values()))))
63 plt.title("Question 8 Letter F")
64 plt.grid(True)
65 fig = plt.gcf()
66 fig.set_size_inches(18.5, 10.5, forward=True)
67 plt.savefig('letter_f.png')
68
```

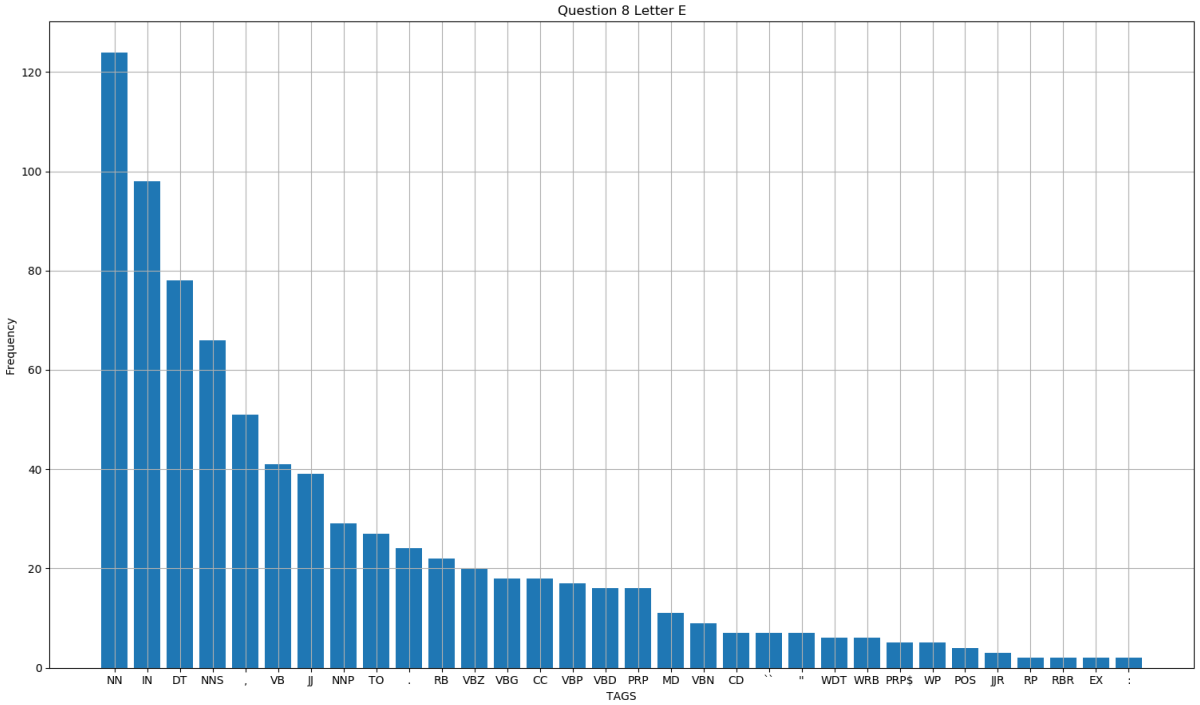

Figura 1: Gráfico referente a letra E da questão 8.

e) As classes que tem maior ocorrência são os substantivos, preposições e determinantes ou artigos.

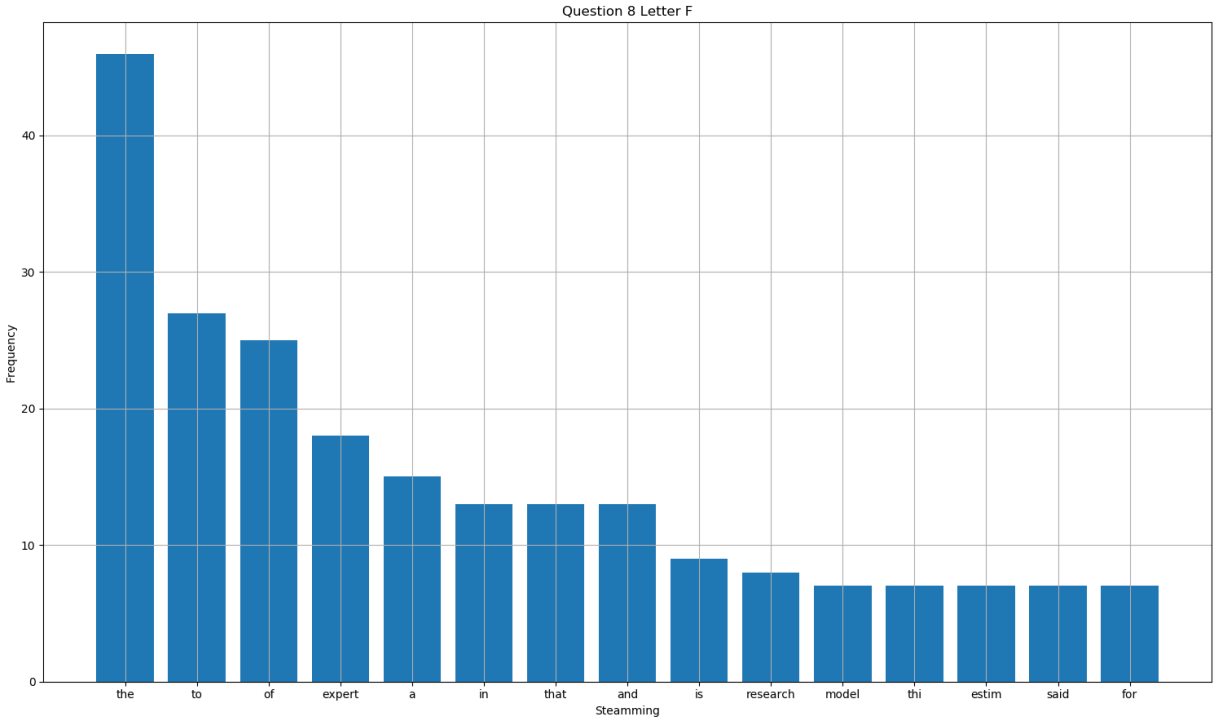

Figura 2: Gráfico referente a letra F da questão 8.

f) Os radicais mais frequentes são geralmente artigos e preposições. Por se tratarem de palavras padrões de ligação, é comum que se use com mais frequência no texto e, consequentemente, que se identifique mais.