2ª Lista de Exercícios - Mineração de Texto

Maria Fernanda Souza Andrade, Saulo Lucas Ferreira

01/10/2018

1 Usando o NLTK

1.1 Gerar um word cloud contendo os 20 lemas de substantivos mais frequentes.

```
arq = open('NLP.txt','r')
text = arq.read()
arq.close()

lemmatizer = WordNetLemmatizer()
# tokenizing
tokens = nltk.word_tokenize(text)
# lemmatizing
tokens = [lemmatizer.lemmatize(word) for word in tokens]
# pos-tagging
pos_tags = nltk.pos_tag(tokens)

nouns = ""
for i in pos_tags:
    if i[1] == 'NN' or i[1] == 'NNP' or i[1] == 'NNS' or i[1] == 'NNPS':
        nouns = nouns + " " + i[0]

wc = WordCloud(max_words=20).generate(nouns)
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

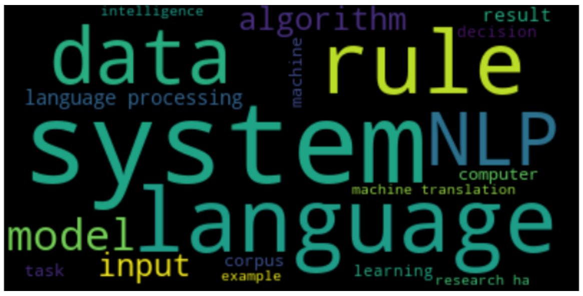

Figura 1: Word Cloud da questão 1.

1.2 Gerar um word cloud contendo os 20 lemas de verbos mais frequentes.

```
verbs = " ";
for i in pos_tags:
    if i[1] == 'VB':
        verbs = verbs + " " + i[0]

wc = WordCloud(max_words=20).generate(verbs)
```

```

plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

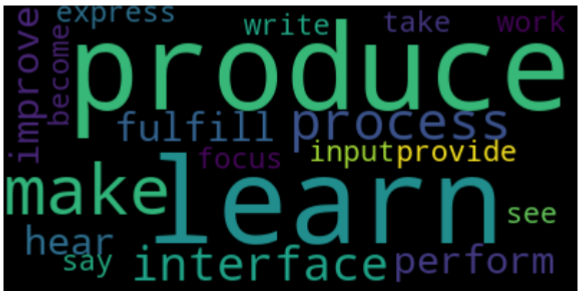

Figura 2: Word Cloud da questão 2.

1.3 Refazer os itens 1 e 2 usando a lista de stop words.

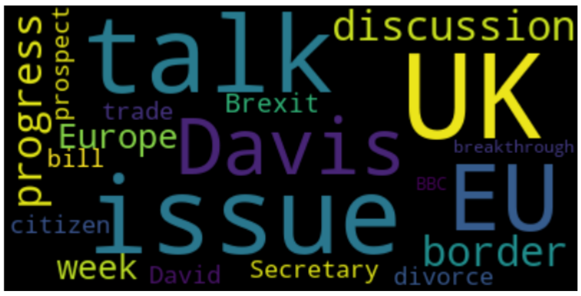

Figura 3: Word Cloud de substantivos da questão 3.

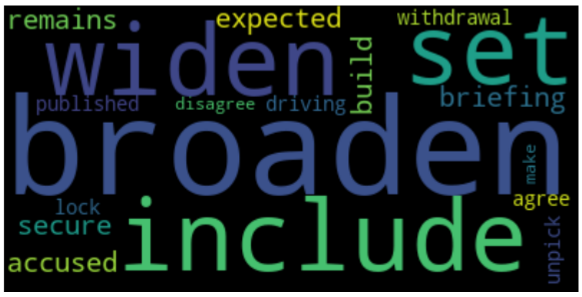

Figura 4: Word Cloud de verbos da questão 3.

```
arq = open('Corpus_en_NER.txt','r')
text = arq.read()
arq.close()

lemmatizer = WordNetLemmatizer()
# tokenizing
tokens = nltk.word_tokenize(text)
print(tokens)
# lemmatizing
tokens = [lemmatizer.lemmatize(word) for word in tokens]
print(tokens)
# pos-tagging
pos_tags = nltk.pos_tag(tokens)
sw = open('stopwords.txt','r')
```

```python
stopwords = sw.read()

nouns = ""

for i in pos_tags:
    if i[1] == 'NN' or i[1] == 'NNP' or i[1] == 'NNS' or i[1] == 'NNPS':
        if i[0] not in stopwords:
            nouns = nouns + " " + i[0]

wc = WordCloud(max_words=20).generate(nouns)
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()

verbs = "";
for i in pos_tags:
    if i[1] == 'VB' or i[1] == 'VBD' or i[1] == 'VBG' or i[1] == 'VBN' or i[1] == 'VBP' or i[1] == 'VBZ':
        if i[0] not in stopwords:
            verbs = verbs + " " + i[0]

wc = WordCloud(max_words=20).generate(verbs)
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

1.4 Gerar uma árvore de parsing da frase.

```python
from nltk import Tree
from pycorenlp import StanfordCoreNLP

nlp = StanfordCoreNLP('http://localhost:9000')

text = "The last love letter I wrote was probably about 10 years ago."

output = nlp.annotate(text, properties={
  'annotators': 'parse',
  'outputFormat': 'json'
})

tree1 = output['sentences'][0]['parse'] + ""
tree2 = Tree.fromstring(tree1)
tree2.draw()
```

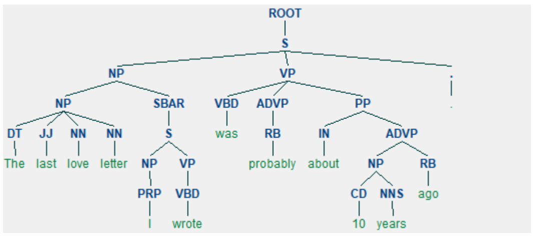

Figura 5: Parse tree da questão 4.

2 Usando o CoreNLP

```python
from stanfordcorenlp import StanfordCoreNLP
import json

nlp = StanfordCoreNLP("C:\stanford-corenlp-full-2018-02-27")
arq = open("Corpus_en_NER.txt", "r")
```

```
sentence = arq.read()

print("——— PIPELINE ———")
print('[INFO] Tokenize:', nlp.word_tokenize(sentence))

props = {'annotators': 'tokenize,ssplit', 'pipelineLanguage': 'en', 'outputFormat': 'json'}
sentence_splitting = nlp.annotate(sentence, properties=props)
jsonToPython = json.loads(sentence_splitting)
sentences = []
for sent in jsonToPython["sentences"]:
    start_offset = sent['tokens'][0]['characterOffsetBegin']
    end_offset = sent['tokens'][-1]['characterOffsetEnd']
    sent_str = sentence[start_offset:end_offset]
    sentences.append(sent_str)
print('[INFO] Sentence Splitting:', sentences)

pos = nlp.pos_tag(sentence)
print('[INFO] Part of Speech:', pos)

props = {'annotators': 'tokenize,lemma', 'pipelineLanguage': 'en', 'outputFormat': 'json'}
lemma = nlp.annotate(sentence, properties=props)
jsonToPython = json.loads(lemma)
lemmas = {}
for sent in jsonToPython["sentences"]:
    for token in sent['tokens']:
        word_original = token['word']
        lemma_original = token['lemma']
        if word_original not in lemmas:
            lemmas[word_original] = lemma_original
print('[INFO] Lemmatization:', lemmas)

entities = nlp.ner(sentence)
print('[INFO] Named Entities:', entities)
print('[INFO] Dependency Parsing:', nlp.dependency_parse(sentence))
```

2.1 Os verbos identificados e os seus respectivos lemmas

```
question1 = []
for world in pos:
    if 'VB' in world[1]:
        world_aux = list(world)
        world_aux.append(lemmas[world[0]])
        question1.append(world_aux)
print("Verbos e seus lemas: ", question1)

past_form = []
for world in pos:
    if 'VBD' in world[1] or 'VBN' in world[1]:
        world_aux = list(world)
        world_aux.append(lemmas[world[0]])
        past_form.append(world_aux)
print("Verbos no passado e seus lemas: ", past_form)
print("O lemma   o verbo no presente.")
```

2.2 As entidades nomeadas encontradas

```
question2 = []
for entiti in entities:
    if 'O' not in entiti[1]:
        question2.append(entiti)
print(question2)
arq.close()
```

2.3 Os tipos distintos de dependências gramaticais encontradas no arquivo NLP.txt

```python
arq = open("NLP.txt", "r")
sentence = arq.read()
dependency = nlp.dependency_parse(sentence)
types = []
for dependences in dependency:
    if dependences[0] not in types:
        types.append(dependences[0])
print(types)
arq.close()
```

3 Implementando um sumarizador simples de notícias

```python
import nltk
from glob import glob

def pipeline(text):
    sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
    sentences = sent_detector.tokenize(text.strip())
    qtt = []
    for sentence in sentences:
        tokens = nltk.word_tokenize(sentence)
        postag = nltk.tag.pos_tag(tokens)
        parse_tree = nltk.ne_chunk(nltk.tag.pos_tag(tokens), binary=True)  # POS tagging
 before chunking!

        named_entities = []

        soma = []

        for t in parse_tree.subtrees():
            if t.label() == 'NE':
                named_entities.append(list(t))  # if you want to save a list of tagged
    words instead of a tree
                soma.append(list(t).__len__())


        named_entities.clear()
        qtt.append(sum(soma))
    return sentences, qtt


def getScore(sentences,qtt):
    scoreSi = []
    i = 0
    n = qtt.__len__()
    while i < qtt.__len__():
        score = 1- ((i+1)/n)
        scoreGlobal = 1 + ((2*qtt[i])/(n+score))
        scoreSi.append([sentences[i], scoreGlobal])
        i = i+1
    return scoreSi


def summarized(score):
    score.sort(key=lambda x: x[1], reverse = True)
    n = score.__len__()
    k = 0.3*n
    k = int(float(k))
    for i in range(0,n-k):
        score.pop()

    summarize = ""
    for sentence in score:
        summarize = summarize + sentence[0]

    return summarize



```

```python
def main():
    for filepath in glob('News\\**'):
        arq = open(filepath,'r')
        text = arq.read()
        (sentences, qtt) = pipeline(text)
        score = getScore(sentences,qtt)
        print(summarized(score), "\n")

if __name__ == "__main__":
    main()
```

3.1 A sumarização é razoável? Apresentam problema de coerência? Qual sua opnião?

O resumo trás, de fato, uma certa parcela das informações mais importantes no texto. Mas, apesar disso, sua informação fica muito dispersa pela falta de coesão entre as sentenças. As informações parecem jogadas, bagunçadas e não se complementam. De fato, resume razoavelmente bem o que o texto possui, porém, não resume bem o que ele quer dizer.