Apêndice 3 - Notebook webscraping repositórios institucionais

Script para buscar resumos na BDTD, testar se eles são relevantes para o domínio de óleo e gás e baixar o documento original no repositório institucional.

[1]:
```
import requests
from bs4 import BeautifulSoup as bs
import pandas as pd
import numpy as np
import json
import nltk
from nltk.tokenize import word_tokenize
from langdetect import detect
from langdetect import detect_langs
from keras.models import load_model
import gensim
from gensim.models import Word2Vec
import csv
import re
```

```
Using TensorFlow backend.
```

[2]:
```
# Definindo configurações globais de proxy para realizar a extração dentro da␣
 ↪rede Petrobras
chave = 'XXXX'
pwd = 'XXXXXXXXXX'
proxy_url = 'http://'+chave+':'+pwd+'@inet-sys.gnet.petrobras.com.br:804/'
proxies = {
  'http' : proxy_url ,
  'https' : proxy_url ,
}
```

Inicialmente entraremos no site da BDTD e buscaremos os links de todas as teses de uma determinada intituição.

[3]:
```
#função para coletar os links das tese

def get_links(page):

    #preparar a url
```

```
    url = ('http://bdtd.ibict.br/vufind/Search/Results?
↪filter%5B%5D=institution%3A%22UFBA%22&type=AllFields&page=' +
          str(page))

    #Fazer requisição e parsear o arquivo html
    f = requests.get(url, proxies = proxies).text
    soup = bs(f, "html.parser")

    #Coletando link para as teses
    links = []
    for doc in soup.find_all('a', href=True):
        if 'title' in doc.get('class', []):
            links.append(doc['href'])
    return links
```

[4]:
```
#Coletar o link de todas as teses
start_page = 1
n_pages = 500 # Cada página retorna 20 teses

links = []

for p in range(start_page, n_pages):
    link = get_links(p)
    if link != []:
        links = links + link
    else:
        break

    if p % 100 == 0:
        print (p*20, ' links capturados, ', p, ' páginas')
        with open('links_ufba', "w") as output:
            writer = csv.writer(output, lineterminator='\n')
            for val in links:
                writer.writerow([val])

with open('links_ufba', "w") as output:
    writer = csv.writer(output, lineterminator='\n')
    for val in links:
        writer.writerow([val])
print (p*20, ' links capturados, ', p, ' páginas')
```

```
2000  links capturados,  100  páginas
4000  links capturados,  200  páginas
6000  links capturados,  300  páginas
8000  links capturados,  400  páginas
9940  links capturados,  497  páginas
```

[5]:
```
# Abrindo arquivo gravado anteriormente

#links = []
#with open('links_ufba', 'r') as f:
#    reader = csv.reader(f)
#    for link in reader:
#        links.append(link[0])
```

Em seguida vamos recuperar os metadados de cada link coletado anteriormente.

[6]:
```
#função para buscar os metadados das teses no BDTD
def tese_link(link):
    #definir url
    url = 'http://bdtd.ibict.br' + link

    #Requisitar html e fazer o parser
    f = requests.get(url, proxies = proxies).text
    soup = bs(f, "html.parser")

    #Dicionário para armazenar as informações da tese
    tese = {}

    #Adicionar título
    tese['Title'] = soup.find('h3').get_text()
    for doc in soup.find_all('tr'):
        #Identificar atributo
        try:
            atributo = doc.find('th').get_text()
        except:
            pass
        #Verificar se o atributo possui mais de um dado
        for row in doc.find_all('td'):
            #Adicionar o atributo no dicionário
            if row.find('div') == None:
                try:
                    tese[atributo] = doc.find('td').get_text()
                except:
                    pass
            else:
                element = []
                #No dicionário, adicionar todos os dados ao seu respectivo␣
↪atributo
                for e in doc.find_all('div'):
                    try:
                        sub_e = []
                        for sub_element in e.find_all('a'):
                            element.append(sub_element.get_text())
                        #element.append(sub_e)
```

```
                    except:
                        pass
                tese[atributo] = element

    return(tese)
```

Como em alguns casos o resumo português e inglês se misturaram, foi implementado uma função para separar os textos misturados

[7]:
```
# Função para separar resumos português e inglês
def separacao_port_engl(abstract):

    mix_sent = nltk.sent_tokenize(abstract)

    new_mix = []
    for sent in mix_sent:
        position = sent.find('.')
        if position != len(sent)-1:
            sent_1 = sent[:position+1]
            sent_2 = sent[position+1:]
            new_mix.append(sent_1)
            new_mix.append(sent_2)
        else:
            new_mix.append(sent)

    mix_sent = new_mix

    port = []
    engl = []

    for sent in mix_sent:
        try:
            if detect (sent) == 'pt':
                port.append(sent)
            else:
                engl.append (sent)
        except:
            pass

    port = " ".join(port)
    engl = " ".join(engl)

    return(port, engl)
```

Até esse momento estamos recuperando as informações de todas as teses de uma determinada instituição. No entanto o objetivo é gravar os metadados e salvar o arquivo apenas das teses relacionadas a O&G. Portanto, vamos carregar os algoritmos de classificação e de vetorização de palavras treinados previamente.

[8]:
```
# Carregando modelo Word2Vec
BDTD_word2vec_50 = Word2Vec.load("..\..\..\Embeddings\BDTD_word2vec_50")
# Carregando modelo keras
model_keras = load_model('..\..\..\model_cnn.h5')
model_keras.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 400)               0
_________________________________________________________________
embedding_6 (Embedding)      (None, 400, 50)           9289150
_________________________________________________________________
spatial_dropout1d_6 (Spatial (None, 400, 50)           0
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 396, 128)          32128
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, 198, 128)          0
_________________________________________________________________
dropout_18 (Dropout)         (None, 198, 128)          0
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 194, 128)          82048
_________________________________________________________________
max_pooling1d_10 (MaxPooling (None, 97, 128)           0
_________________________________________________________________
dropout_19 (Dropout)         (None, 97, 128)           0
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 93, 128)           82048
_________________________________________________________________
global_max_pooling1d_5 (Glob (None, 128)               0
_________________________________________________________________
dropout_20 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_30 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_31 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_21 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_32 (Dense)             (None, 1)                 513
=================================================================
Total params: 9,814,591
Trainable params: 525,441
Non-trainable params: 9,289,150
_________________________________________________________________
```

[9]:
```
# dicionário proveniente do modelo de word embedding para converter palavras em␣
 ↪índices
word2index = {}
for index, word in enumerate(BDTD_word2vec_50.wv.index2word):
    word2index[word] = index

# Função para converter texto em sequência de índices
def index_pad_text(text, maxlen, word2index):
    maxlen = 400
    new_text = []

    for word in word_tokenize(text):
        try:
            new_text.append(word2index[word])
        except:
            pass
    # Add the padding for each sentence. Here I am padding with 0
    if len(new_text) > maxlen:
        new_text = new_text[:400]
    else:
        new_text += [0] * (maxlen - len(new_text))

    return np.array(new_text)

maxlen = 400
```

Para cada link coletado será fita as seguintes tarefas: * verificar se o texto português e inglês estão misturados; * transformar o texto em sequência de índices; * classificar quanto a relevância ao domínio de O&G; * se for relevante, gravar os metadados

[10]:
```
# Dicionário para agrupar os metadados
metadados = {}
# Contadores te links testados e classificados como O&G
n_test = 0
n_pet = 0
# Testando cada link de links
for link in links:
    n_test += 1
    try:
        # Recuperar o metadados de uma tese
        metadado = tese_link(link)
        # Verificar se existe resumo em inglês, separar texto português/inglês e␣
 ↪realocar
        # os textos separados nas respectivas colunas
        if 'Resumo inglês:' not in metadado:
            metadado['Resumo inglês:'] = separacao_port_engl(metadado['Resumo␣
 ↪Português:'])[1]
```

```
        metadado['Resumo Português:'] = separacao_port_engl(metadado['Resumo␣
↪Português:'])[0]
        # Colocando o texto em minúscula
        text = metadado['Resumo Português:'].lower()
        # Convertendo as palavras em sequencias de acordo com o modelo word2vec
        text_seq = index_pad_text(text, maxlen, word2index)
        text_seq = text_seq.reshape((1, 400))
        # Usando o algoritmo classificador para prever se a tese é relevante
        pred = model_keras.predict(text_seq)[0]
        #Se a classificação for menor do que 0.2 manter os metadados
        if (pred < 0.2 and len(text) > 100):
            metadado['Classificador'] = pred[0]
            texto_completo = metadado['Download Texto Completo:']
            metadados[texto_completo] = metadado
            n_pet += 1
            # Gravando os resultados em JSON
            metadados_ufba = pd.DataFrame.from_dict(metadados, orient='index')
            metadados_ufba.to_json('metadados_ufba.json', orient = 'index')
            print(n_test, " teses avaliadas e ", n_pet, " teses relacionadas a␣
↪O&G encontradas.")

    except:
        pass
```

```
9531  teses avaliadas e  187  teses relacionadas a O&G encontradas.
```

[11]:
```
# Incluindo um ID para cada tese
universidade = 'UFBA'
metadados_ufba['PDF_ID'] = metadados_ufba['Download Texto Completo:'].
↪apply(lambda x: universidade +
                                                                    '_' +
                                                                    re.
↪sub('/', '_', x[-6:]))
```

[12]:
```
metadados_ufba.to_json('metadados_ufba.json', orient = 'index')
```

[13]:
```
# Carregando arquivos já gravados
metadados_ufba = pd.read_json('metadados_ufba.json', orient = 'index')
```

A próxima etapa será fazer o download das teses classificadas como relevante para o domínio de O&G

[14]:
```
for tese in metadados_ufba.iterrows():
    print(tese[1]['PDF_ID'])
    try:
        #preparar a url
        url = tese[1]['Download Texto Completo:']
```

```
        #Fazer requisição e parsear o arquivo html
        f = requests.get(url, proxies = proxies).text
        soup = bs(f, "html.parser")

        #Coletando link para arquivo das teses
        links = []
        for doc in soup.find_all('a', href=True):
            if doc.get_text() == 'View/Open':
                links.append(doc['href'])

        #Recuperando e gravando arquivo PDF
        url = 'http://repositorio.ufba.br' + links[0]
        pdf = requests.get(url, proxies = proxies)
        filename = tese[1]['PDF_ID'] + '.pdf'
        with open(filename, 'wb') as f:
            f.write(pdf.content)
    except:
        pass
```