Mineração de dados com Stack

Extração de tópicos com LDA

Tarefas:

- Extrair tópicos usando LDA

Pacotes e configurações

In [1]:

```
#%matplotlib inline

import pandas as pd
import numpy as np
import pickle
import operator
import os

from elasticsearch import Elasticsearch
from elasticsearch.helpers import bulk
from elasticsearch import client

from random import random

# html render para o notebook
from IPython.core.display import display, HTML

# https://scikit-learn.org/stable/install.html
from sklearn.feature_extraction.text import TfidfVectorizer
```

Lendo o arquivo

- Os dados utilizados foram pré-processados e indexados em uma base do Elastic.
- Para agilizar as consultas e testes, um dataset foi gerado com um sub-conjunto de dados.

In [2]:

```
# Recuperando os dados
df_dados = pd.read_pickle("results_elastic.pkl")
```

In [10]:

```
# Alguns registros recuperados.
# O conteúdo que será usado é a coluna Body
df_dados[["Id","CreationDate", "Body"]].head(5)
```

Out[10]:

	Id	CreationDate	Body
0	20489633	2013-12-10T08:39:25.207	<p>I had the same problem. Installing the root...
1	3888726	2010-10-08T08:02:58.167	<p>By enabling virtualization from the BIOS se...
2	46169264	2017-09-12T06:21:45.053	<p>See this link. This resolved this issue, th...
3	8745880	2012-01-05T16:21:24.913	<p>The version of ImageMagick in the CentOS yu...
4	11134318	2012-06-21T08:29:00.500	<p>Umm, I would say try again. The file stated...

Extraindo categorias dos Dados

Os códigos a seguir foram baseados nos materiais publicados abaixo.

- **Material de ajuda**
  - https://towardsdatascience.com/nlp-extracting-the-main-topics-from-your-dataset-using-lda-in-minutes-21486f5aa925 (https://towardsdatascience.com/nlp-extracting-the-main-topics-from-your-dataset-using-lda-in-minutes-21486f5aa925)
  - https://github.com/priya-dwivedi/Deep-Learning/blob/master/topic_modeling/LDA_Newsgroup.ipynb (https://github.com/priya-dwivedi/Deep-Learning/blob/master/topic_modeling/LDA_Newsgroup.ipynb)

In [11]:

```
# Gensim e NLTK são duas bibliotecas para trabalhar com
# NLP
import gensim
import nltk

from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *

# O pacode wordnet é necessário para trabalhar com o dicionário
# em inglês. Caso não esteja instalado, descomentar a linha abaixo.
#nltk.download('wordnet')
```

Processando os textos que serão utilizados

In [14]:

```
'''
Stemming é uma redução de palavras para a sua formação raíz.
Dessa forma o sistema consegue identificar palavras relacionadas.
'''
stemmer = SnowballStemmer("english")

def nlp_lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

def nlp_preprocess(text):
    result=[]
    for token in gensim.utils.simple_preprocess(text) :
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) >
3:
            result.append(nlp_lemmatize_stemming(token))
    return result
```

Exemplo de uso

In [16]:

```
# O primeiro registro no dataset contém o seguinte texto
display(df_dados["Body"][1])
```

```
'<p>By enabling virtualization from the BIOS setup of the main machi
ne it worked!\n<a href="https://stackoverflow.com/questions/744211/p
roblem-installing-x64-guest-os-with-vmware-server">Problem installin
g x64 guest OS with vmware Server</a></p>\n'
```

In [18]:

```
"""
Fazendo processamento com as fuções de stemming:
- O texto original é separado em palavras únicas.
- Cada palavra é reduzida (via stemming) à sua base.
"""
print(nlp_preprocess(df_dados["Body"][1]))
```

```
['enabl', 'virtual', 'bio', 'setup', 'main', 'machin', 'work', 'hre
f', 'https', 'stackoverflow', 'question', 'problem', 'instal', 'gues
t', 'vmware', 'server', 'problem', 'instal', 'guest', 'vmware', 'ser
ver']
```

Processando todo o dataset (resultado da pesquisa)

In [19]:

```
# processa todos os documentos (coluna Body)
# do dataset
nlp_processed_docs = []

for item in df_dados["Body"]:
    nlp_processed_docs.append(nlp_preprocess(item))

# 300 documentos
len(nlp_processed_docs)
```

Out[19]:

```
300
```

Criando uma Bag of Words

In [32]:

```
'''
Uma vez que todas as palavras dos registros foram processados
é preciso criar uma bag of words. A biblioteca gensim será utilizada.
'''
nlp_dictionary = gensim.corpora.Dictionary(nlp_processed_docs)
```

In [33]:

```
# quantas palavras no dicionário?
len(nlp_dictionary)
```

Out[33]:

```
1333
```

In [36]:

```
'''
Uma vez criado a bag of words, é possível apliar um filtro
para remover palavras raras e/ou muito comuns (opcional)

- palavras com ocorrência inferior a 15 vezes (raras).
- palavras com ocorrência em mais de 10% do total de documentos.
'''
nlp_dictionary.filter_extremes(no_below=5, no_above=0.1, keep_n= 100000)
```

In [37]:

```
# quantas palavras no dicionário filtrado?
len(nlp_dictionary)
```

Out[37]:

```
184
```

In [38]:

```
'''
Agora é possível criar um modelo de bag of words para os documentos.
Pra cada documento será criado um dicionário com a contagem de frequência
de cada palavra no documento.
'''
nlp_bow_corpus = [nlp_dictionary.doc2bow(doc) for doc in nlp_processed_docs]
```

In [39]:

```
'''
Uma vez que o modelo é criado, é possível consultar a frequência de palavras
em cada documento.
'''
# 20o documento do registro.
nlp_bow_corpus_doc = nlp_bow_corpus[20]

for i in range(len(nlp_bow_corpus_doc)):
    print("Palavra {} (\"{}\") = {} ocorrências.".format(nlp_bow_corpus_doc[i][0
],
                                                         nlp_dictionary[nlp_bow_
corpus_doc[i][0]],
                                                         nlp_bow_corpus_doc[i][1
]))
```

```
Palavra 27 ("blockquot") = 6 ocorrências.
Palavra 30 ("strong") = 2 ocorrências.
Palavra 31 ("support") = 1 ocorrências.
Palavra 58 ("django") = 3 ocorrências.
Palavra 59 ("final") = 1 ocorrências.
Palavra 60 ("fixtur") = 6 ocorrências.
Palavra 61 ("initial_data") = 4 ocorrências.
Palavra 62 ("json") = 1 ocorrências.
Palavra 63 ("know") = 2 ocorrências.
Palavra 64 ("loaddata") = 2 ocorrências.
Palavra 65 ("manag") = 2 ocorrências.
```

Executando o LDA

- O LDA (latent Dirichlet allocation) é um modelo estatístico que pode ser aplicado a um conjunto de dados.
- Pode ser utilizado para descoberta de tópicos em documentos.
- No dataset utilizado, ainda não existem os registros do tópicos relacionados a cada um dos documentos.
- Nesse caso, o LDA será utilizado para fazer uma descoberta simples de palavras que podem ser consideradas relacionadas e pertencentes a um tópico em particular.

- **TODO**:
  - Ampliar o dataset utilizado com as identificações das categorias.

In [40]:

```
"""
Referência:

LDA mono-core -- fallback code in case LdaMulticore throws an error on your mach
ine
lda_model = gensim.models.LdaModel(bow_corpus,
                                   num_topics = 10,
                                   id2word = dictionary,
                                   passes = 50)

https://radimrehurek.com/gensim/models/ldamodel.html
"""

# LDA multicore
lda_model =  gensim.models.LdaMulticore(nlp_bow_corpus,
                                        num_topics = 8,
                                        id2word = nlp_dictionary,
                                        passes = 10,
                                        workers = 2)
```

In [42]:

```
lda_model.get_topic_terms(0)
```

Out[42]:

```
[(60, 0.10404698),
 (62, 0.06982548),
 (143, 0.039006505),
 (65, 0.03753074),
 (61, 0.036499448),
 (58, 0.03621276),
 (161, 0.03180927),
 (27, 0.03164836),
 (105, 0.03138134),
 (18, 0.020724475)]
```

In [45]:

```
"""
Uma vez que o modelo do LDA é criado, podemos consultar os resultados:

-  Cada tópico encontrado pelo LDA, será exibido:
    - palavra * probabilidade da palavra no documento de acordo com o LDA

Documentação do gensim:
num_topics (int, optional) – The number of topics to be selected, if -1 - all to
pics will be in result (ordered by significance).
num_words (int, optional) – The number of words to be included per topics (order
ed by significance).

https://radimrehurek.com/gensim/models/ldamodel.html
"""
for idx, topic in lda_model.print_topics(-1):
    print("Tópico: {} \nPalavras: {}".format(idx, topic ))
    print("\n")
```

```
Tópico: 0
Palavras: 0.104*"fixtur" + 0.070*"json" + 0.039*"load" + 0.038*"mana
g" + 0.036*"initial_data" + 0.036*"django" + 0.032*"model" + 0.032
*"blockquot" + 0.031*"test" + 0.021*"updat"


Tópico: 1
Palavras: 0.067*"question" + 0.065*"eclips" + 0.053*"sudo" + 0.038
*"local" + 0.031*"applic" + 0.029*"stackoverflow" + 0.026*"chang" +
0.026*"run" + 0.026*"directori" + 0.020*"like"


Tópico: 2
Palavras: 0.083*"visual" + 0.064*"studio" + 0.051*"compos" + 0.047
*"creat" + 0.044*"requir" + 0.035*"project" + 0.035*"nuget" + 0.031
*"microsoft" + 0.030*"resolv" + 0.021*"download"


Tópico: 3
Palavras: 0.217*"strong" + 0.060*"server" + 0.038*"fix" + 0.035*"mic
rosoft" + 0.029*"data" + 0.029*"librari" + 0.027*"need" + 0.026*"dow
nload" + 0.025*"tool" + 0.018*"support"


Tópico: 4
Palavras: 0.054*"command" + 0.050*"rubi" + 0.043*"brew" + 0.038*"ubu
ntu" + 0.037*"build" + 0.028*"sudo" + 0.027*"line" + 0.026*"tool" +
0.025*"mingw" + 0.023*"modul"


Tópico: 5
Palavras: 0.073*"stack" + 0.069*"imgur" + 0.061*"updat" + 0.044*"ima
g" + 0.038*"extens" + 0.032*"enter" + 0.032*"descript" + 0.029*"hav
e" + 0.026*"rstudio" + 0.025*"face"


Tópico: 6
Palavras: 0.062*"android" + 0.038*"strong" + 0.036*"download" + 0.03
4*"blockquot" + 0.029*"java" + 0.028*"question" + 0.024*"eclips" +
0.024*"django" + 0.023*"librari" + 0.022*"ubuntu"


Tópico: 7
Palavras: 0.082*"node" + 0.072*"github" + 0.059*"issu" + 0.035*"fram
ework" + 0.033*"have" + 0.032*"answer" + 0.025*"fix" + 0.021*"hope"
+ 0.021*"googl" + 0.021*"fine"
```

In [46]:

```
"""
Da documentação do gensim:

show_topic(topicid, topn=10)

Parameters:
 topicid (int) – The ID of the topic to be returned
 topn (int, optional) – Number of the most significant words that are associated
with the topic.

Returns:
  Word - probability pairs for the most relevant words generated by the topic.
  Return type:
       list of (str, float)

"""
# Exibindo tópico individual
lda_model.show_topic(0)
```

Out[46]:

```
[('fixtur', 0.10404698),
 ('json', 0.06982548),
 ('load', 0.039006505),
 ('manag', 0.03753074),
 ('initial_data', 0.036499448),
 ('django', 0.03621276),
 ('model', 0.03180927),
 ('blockquot', 0.03164836),
 ('test', 0.03138134),
 ('updat', 0.020724475)]
```

Continuação

- Os resultados obtidos definem:
  - Para cada tópico encontrado, temos a lista de palavras (+ pesos ou probabilidades) das palavras mais representativas para o determinado tópico.
  - Se no dataset original a informação de cada tópico já existir, a partir daqui é possível criar um classificador de dados onde:
  - Para cada conjunto de palavras (dicionário) de um documento, o LDA identifica qual dos tópicos é o mais adequado para o documento.