

Projeto 2: Analisando dados com Spark

Gustavo Molina e Guilherme Batista

1. Explicando o código

Nesta seção será analisado um pouco do funcionamento do código dentro do projeto, célula a célula do jupyter em questão.

Primeira célula:

Todas as libs utilizadas dentro do projeto são importadas para futura utilização.

```
import pyspark
import nltk
import string
from nltk.tokenize import word_tokenize
import pandas as pd
import math
```

Segunda célula:

O cluster chamado Projeto2_bs é criado.

```
sc = pyspark.SparkContext(appName="Projeto2_bs")
```

Terceira célula:

São feitos o broadcast das variáveis de estudo, nesse caso as strings relativas às marcas de carros Mercedes e Fiat.

```
used_terms = sc.broadcast(['Mercedes','Fiat'])
```

Quarta célula:

A pasta contendo todos os arquivos que serão analisados são lidos através do comando `pickleFile` do spark que carrega um “resilient distributed dataset” (RDD) em uma variável local.

```
rdd_ = sc.pickleFile('web-brasil-small')
```

Quinta célula:

Através do comando `count` checamos quantos documentos existem dentro do rdd

```
rdd_.count()
```

Sexta célula:

Nesta célula fazemos o broadcast(disponibilizamos a variável para ser usada globalmente em todos os nodes) que é relativa ao count de documentos do rdd.

Sétima célula:

Inspier

Megadados - 2021

Nesta célula é feita uma função que checa se no documento em questão há alguma ocorrência de um dos termos do broadcast.

```
def detecta_termos(texto):
    termos_ = used_terms.value
    exists = [i in texto for i in termos_]

    return (texto, exists)
```

Oitava célula:

Nesta função (filter_and_count) são filtrados termos indesejados - stopwords - as palavras são colocadas em minúsculo e padronizadas a fim de contarmos quantas ocorrências da mesma existem no documento em questão.

```
def filter_and_count_txt(item):
    texto, unico = item
    stop_words = nltk.corpus.stopwords.words('portuguese')
    stop_words2 = nltk.corpus.stopwords.words('english')
    palavras_separadas = texto.lower().strip().split()

    return [(palavra, 1) for palavra in palavras_separadas if not palavra in stop_words
            and palavra not in stop_words2]
```

Nona célula:

Nesta célula temos 3 funções sendo suas funcionalidades as seguintes:

- 1.filter_unique_text_elements_0 -> Checamos se no documento em questão existe a ocorrência do primeiro termo do broadcast e não existe a do segundo.
- 2.filter_unique_text_elements_1 ->Checamos se no documento em questão existe a ocorrência do segundo termo do broadcast e não existe a do primeiro.
- 3.filter_unique_text_elements_2 ->Checamos se no documento em questão existe a ocorrência do segundo termo do broadcast e existe a do primeiro também.

```
def filter_unique_text_elements_0(item):
    if item[1][0] and not item[1][1]:

        return item[0]
```

```
def filter_unique_text_elements_1(item):
    if item[1][1] and not item[1][0]:

        return item[0]
```

```
def filter_unique_text_elements_2(item):
    if item[1][1] and item[1][0]:
```

```
return item[0]
```

Décima célula:

Nesta célula temos 1 função responsável por realizar o cálculo do IDF das palavras, tal ferramenta é utilizada para mensurarmos a importância “inversa” de uma palavra em um conjunto de documentos. O mesmo é considerado como uma importância inversa dado que quanto maior o valor de idf para uma palavra, mais “esquisita” ou menos frequente no documento ela estará, já caso a palavra tenha um IDF baixo, a mesma tem relevância no contexto e aparece diversas vezes. Por isso neste caso é extremamente importante realizarmos filtros de stopwords e padronizarmos as palavras para que seja possível termos não somente mais acurácia no resultado, como também palavras mais “coerentes”.

```
def df_to_idf(item):  
    palavra, df = item  
    idf = math.log10(total_texts.value / df)  
  
    return (palavra, idf)
```

Décima-primeira célula:

Nesta célula é feita a função que calcula a frequência de cada termo (tf) nos documentos da web-brasil-small. Desta forma conseguimos computar uma forma mais direta de relevância para a palavra no dado contexto, dado que quanto maior a frequência, necessariamente maior a relevância SE a palavra não estiver dentro dos casos previamente filtrados.

```
def calculate_freq(item):  
    palavra, df = item  
    idf = math.log10(1 + df)  
  
    return (palavra, idf)
```

Décima-segunda célula:

Nesta célula é feita a função que irá transformar os outputs dos cálculos em DataFrames para melhor visualização dos dados.

```
def gera_df(lista_items):  
    info = pd.DataFrame(list(lista_items.items()), columns = ["palavra", "frequencia"])  
    info.index = info.palavra  
    info = info.drop(columns=["palavra"])  
  
    info.sort_values(by=["frequencia"], inplace=True, ascending=False)  
    return info.head(100)
```

Décima terceira célula:

Nesta célula temos uma função que desempenha diversas tarefas, sendo elas:

1. Aplica a função detecta termos para os itens do rdd e separa os itens desejados.
2. Aplica a função filter_and_count_txt no resultado do item 1 , como também agrupa o resultado pelas keys.
3. Aplica a função df_to_idf no resultado do 2 para calcular o idf.
4. Pega todos os 100 resultados com maior IDF e ordena.

5. Pega todos os 100 resultados com menor IDF e ordena.

```
def generate_count_take(rdd_first):
    result = rdd_first.map(lambda x: x[1]).map(detecta_termos)

    filtered_count_words = result.flatMap(filter_and_count_txt).reduceByKey(lambda x,
y : x + y).cache()
    filtered_count_words = filtered_count_words.map(df_to_idf)

    take_files_ordered_idf_not_comum = filtered_count_words.takeOrdered(100, key =
lambda x: -x[1])
    take_files_ordered_idf_comum = filtered_count_words.takeOrdered(100, key =
lambda x: x[1])

    return take_files_ordered_idf_not_comum, take_files_ordered_idf_comum
```

Décima-quarta célula:

Nesta célula temos a função que irá a frequência das palavras para todos os casos amostrados (contém a primeira e não contém a segunda palavra, etc)

Ele irá repetir processos mudando somente no passo em que aplica a função filter_unique_text_elements_x que irá variar a caso amostral.

No primeiro caso temos: Existe a primeira palavra e não existe a segunda, no segundo caso temos Existe a segunda palavra e não existe a primeira, no terceiro caso existem ambas as palavras.

Para todos são feitos os passos:

1. É aplicada a função no rdd para o filtro previamente citado neste tópico.
2. Aplica a função filter_and_count_txt no resultado do item 1, como também agrupa o resultado pelas keys somando as ocorrências.
3. Aplica a função de calcular a frequência das palavras no resultado no item 2.
4. Agrupa as 100 palavras com maior frequência em ordem decrescente.

Repete o processo para todas as análises.

```
def generate_count_take2(rdd_first):
    result = rdd_first.map(lambda x: x[1]).map(detecta_termos)

    filtered_mercedes = result.filter(filter_unique_text_elements_0)
    filtered_mercedes =
filtered_mercedes.flatMap(filter_and_count_txt).reduceByKey(lambda x, y : x +
y).cache()
    filtered_mercedes = filtered_mercedes.map(calculate_freq)
    take_files_ordered_idf_mercedes = filtered_mercedes.takeOrdered(100, key =
lambda x: -x[1])

    filtered_fiat = result.filter(filter_unique_text_elements_1)
    filtered_fiat = filtered_fiat.flatMap(filter_and_count_txt).reduceByKey(lambda x, y :
x + y).cache()
    filtered_neymar = filtered_fiat.map(calculate_freq)
    take_files_ordered_idf_fiat = filtered_fiat.takeOrdered(100, key = lambda x: -x[1])

    filtered_mercedes_fiat = result.filter(filter_unique_text_elements_2)
```

Inspier

Megadados - 2021

```
filtered_mercedes_fiat =  
filtered_mercedes_fiat.flatMap(filter_and_count_txt).reduceByKey(lambda x, y : x +  
y).cache()  
filtered_mercedes_fiat = filtered_mercedes_fiat.map(calculate_freq)  
take_files_ordered_idf_mercedes_fiat = filtered_mercedes_fiat.takeOrdered(100,  
key = lambda x: -x[1])  
  
return take_files_ordered_idf_fiat,take_files_ordered_idf_mercedes,  
take_files_ordered_idf_mercedes_fiat
```

Nas células subsequentes são gerados os df's contendo os outputs de todo tratamento, como também é aplicada a técnica de wordCloud para melhor visualização dos dados que saíram.

```
list_idf_not_comum, list_idf_comum = generate_count_take(rdd_)
```

```
df_high_idf = gera_df(dict(list_idf_not_comum))  
df_high_idf
```

```
df_low_idf = gera_df(dict(list_idf_comum))  
df_low_idf
```

```
mercedes, fiat, mercedes_fiat = generate_count_take2(rdd_)
```

```
df_mercedes = gera_df(dict(mercedes))  
df_fiat = gera_df(dict(fiat))  
df_mercedes_fiat = gera_df(dict(mercedes_fiat))
```

```
df_mercedes
```

```
df_fiat
```

```
df_mercedes_fiat
```

```
from wordcloud import WordCloud  
import matplotlib.pyplot as plt
```

```
wordcloud = WordCloud()  
wordcloud.generate_from_frequencies(frequencies=dict(mercedes))  
plt.figure(figsize=(40, 6))  
plt.imshow(wordcloud, interpolation="bilinear")  
plt.axis("off")  
plt.show()
```

Inspier
Megadados - 2021

```
wordcloud = WordCloud()  
wordcloud.generate_from_frequencies(frequencies=dict(fiat))  
plt.figure(figsize=(40, 6))  
plt.imshow(wordcloud, interpolation="bilinear")  
plt.show()
```

```
wordcloud = WordCloud()  
wordcloud.generate_from_frequencies(frequencies=dict(mercedes_fiat))  
plt.figure(figsize=(40, 6))  
plt.imshow(wordcloud, interpolation="bilinear")  
plt.show()
```

```
sc.stop()
```

2. Analisando os resultados

In [21]: df_mercedes

Out[21]:

palavra	frequencia
-	52872
gasolina	5488
serie	5216
jardim	4988
mecanico	4458
...	...
3.0	379
maria	376
c	374
dia	370
3	368

100 rows × 1 columns

In [23]: df_mercedes_fiat

Out[23]:

palavra	frequencia
diesel	3.853211
-	3.826981
revisão	3.407561
2p	3.369772
jardim	3.351796
...	...
6x2	2.376577
oficina	2.376577
iveco	2.369216
(2)	2.363612
jeep	2.357935

100 rows × 1 columns

In [22]: df_fiat

Out[22]:

palavra	frequencia
-	4.639954
(1)	3.525563
=	3.438701
jardim	3.402433
tv	3.366423
...	...
lopes	2.556303
(5)	2.553883
la	2.551450
câmera	2.551450
3	2.549003

100 rows × 1 columns

Análise e visualização dos resultados



Insper
Megadados - 2021



Img: WorldCloud Mercedes e Fiat

Com base nos resultados obtidos nessa worldCloud é razoável pensar que as palavras com frequência maior entre Mercedes e Fiat se deve a uma comparação que as pessoas estão fazendo entre os dois carros, com fatores como revisão e diesel como as palavras com maior frequência encontrada. Outras palavras que fortalecem essa hipótese são outras palavras com frequência considerável encontradas, como chevrolet, volkswagen, audi, etc, que indicam algum tipo de benchmark.

Para a Fiat, no WorldCloud algumas hipóteses interessantes também podem ser feitas: a palavra Santa, João e Maria poderiam indicar que algum personagem ou pessoa famosa utiliza o carro em algum vídeo no Youtube e/ou ficou famosa por algo relacionada a marca. Pode-se deduzir também que estes são nomes comuns entre pessoas que anunciam carros da marca na internet.

Já no Mercedes, a frequência de palavras mais alta está relacionada a estados como SP, MG, Belo e Horizonte (separadas). Seus usos indicam ser mais relacionados ao cotidiano do povo brasileiro, que tem uma frequência de palavras buscadas muito maior que a da Mercedes. (como pode ser visto nas tabelas acima)

Link Código github: <https://github.com/gubenites/projetoSpark/blob/main/Projeto2.ipynb>