TP2 - Classificadores não supervisionados

December 9, 2018

Yuri Diego Santos Niitsuma

Todo o código, inclusive o **Jupyter Notebook** está disponível em https://github.com/ignitz/datamining_tp2

1 Objetivo

O trabalho prático 2 consiste em utilizar técnicas de agrupamentos da segunda parte do cronograma do curso. A proposta de objetivo do trabalho é agrupar documentos com características similares, que assim espera-se ter como resultado uma classificação automática dos documentos.

2 Base de dados

A base de dados que será utilizada é a BBC News Articles, a mesma base de dados do trabalho prático 1 https://www.kaggle.com/pariza/bbc-news-summary/home

Nesta base, consiste em publicações de artigos de notícias divididos em 5 categorias:

- Negócios (business)
- Entretenimento (entertainment)
- Política (politics)
- Esporte (sport)
- Tecnologia (tech)

3 Metodologia

Os documentos serão misturados e após o processo de clusterização será verificada se com alguns parâmetros se aproximará da divisão inicial ou se encontrará uma característica nova da base de dados.

Em resumo iremos utilizar um classificador não supervisionado e verificaremos a acurácia do modelo comparando com a classe já rotulada.

```
In [1]: import gensim
        import numpy as np
        import collections
        from gensim.test.utils import common_texts
        from gensim.models.doc2vec import Doc2Vec, TaggedDocument
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
%pylab inline
# Para o PDF exportado conter as imagens vetorizadas
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png', 'pdf')

import os
import pandas as pd
```

```
c:\users\ignit\local\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; ali
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
# Leitura do dataset mantendo o texto original e separando em tokens

bbc_news_folder = 'BBC News Articles'

content = dict()

for article_class in os.listdir(bbc_news_folder):
    collections_articles = []
    for each_article in os.listdir(bbc_news_folder + '/' + article_class):
        article = ''
        with open(bbc_news_folder + '/' + article_class + '/' + each_article) as f:
            article = f.read()
            article_preprocess = gensim.utils.simple_preprocess(article, article_class)

        collections_articles.append((article, article_preprocess))

    content[article_class] = collections_articles

content.keys()
```

Out[2]:
```
dict_keys(['business', 'entertainment', 'politics', 'sport', 'tech'])
```

In [3]:
```
dfs = []
dfs += [pd.DataFrame({'class': 'business', 'texts': [x for x, y in content['business']]
dfs += [pd.DataFrame({'class': 'entertainment', 'texts': [x for x, y in content['enter
dfs += [pd.DataFrame({'class': 'politics', 'texts': [x for x, y in content['politics']]
dfs += [pd.DataFrame({'class': 'sport', 'texts': [x for x, y in content['sport']], 'to
dfs += [pd.DataFrame({'class': 'tech', 'texts': [x for x, y in content['tech']], 'toke
df = pd.concat(dfs)
df['class'].unique()
```

```
Out[3]: array(['business', 'entertainment', 'politics', 'sport', 'tech'],
              dtype=object)
```

Como pode ver o texto original com tokens separados.

```
In [4]: df.head()
```

```
Out[4]:       class                                              texts  \
        0  business  Ad sales boost Time Warner profit\n\nQuarterly...
        1  business  Dollar gains on Greenspan speech\n\nThe dollar...
        2  business  Yukos unit buyer faces loan claim\n\nThe owner...
        3  business  High fuel prices hit BA's profits\n\nBritish A...
        4  business  Pernod takeover talk lifts Domecq\n\nShares in...

                                                      tokens
        0  [ad, sales, boost, time, warner, profit, quart...
        1  [dollar, gains, on, greenspan, speech, the, do...
        2  [yukos, unit, buyer, faces, loan, claim, the, ...
        3  [high, fuel, prices, hit, ba, profits, british...
        4  [pernod, takeover, talk, lifts, domecq, shares...
```

```
In [5]: # Contagem do total de documentos
        df.drop(['texts', 'tokens'], axis=1).describe()
```

```
Out[5]:         class
        count    2225
        unique      5
        top     sport
        freq      511
```

```
In [6]: # Contagem dos documentos de cada classe
        df.groupby('class').count()
```

```
Out[6]:                texts  tokens
        class
        business         510     510
        entertainment    386     386
        politics         417     417
        sport            511     511
        tech             401     401
```

Aqui vamos converter no formato apropriado para o **Doc2Vec** do **gensim** e atrelando um identificador inteiro ao documento.

```
In [7]: common_texts = df.drop('class', axis=1).values[:,:].tolist()
        documents = [TaggedDocument(doc[1], [i]) for i, doc in enumerate(common_texts)]

        # Segue um exemplo
        documents[:1], len(documents)
```

```
Out[7]: ([TaggedDocument(words=['ad', 'sales', 'boost', 'time', 'warner', 'profit', 'quarterly
         2225)
```

3.1 Treinamento do Word2Vec/Doc2Vec

Aqui ocorre o treinamento, que é feito o dicionário, o treinamento Word2vec junto com umvetor de parágrafo a cada documento (o que origina o Doc2Vec).

O vetor criado será de 50 dimensões.

```
In [8]: model = Doc2Vec(documents, vector_size=50, window=5, min_count=1, epochs=30, workers=4)
```

3.2 Visualização do Word2vec

O Word2Vec também é treinado, por isso vamos verificar como as caracteríscas das palavras foram separadas

Vamos aplicar **Principal Component Analysis** (PCA) nos dados para termos uma melhor visualização em 2D.

```
In [9]: pd_pca = pd.DataFrame(model.wv.vectors)
        pd_pca.head()
```

```
Out[9]:          0         1         2         3         4         5         6  \
        0 -0.637268  0.030117  0.538063  0.281174 -0.379946  0.257558 -0.880657
        1  0.926567  0.289307  0.360031 -0.462692  0.770400 -0.036767 -0.469325
        2 -0.727616 -0.437768 -0.856574 -1.160780 -1.092559  1.332829 -2.051537
        3 -0.540567 -0.183050 -0.685313  0.334265 -0.305506  0.178646  0.105787
        4  0.610515  0.194876  0.367597  0.339840 -0.986072  1.206034 -1.914971

                 7         8         9    ...          40        41        42  \
        0 -0.097359 -0.111983  0.844684    ...   -0.579255  0.574445 -0.373330
        1 -0.970775  0.660744  0.303800    ...   -2.162195  0.657468 -0.993551
        2  0.159460  0.343384  0.604731    ...    0.728012  0.197751  0.226651
        3  0.886896  0.091103  0.530676    ...    0.230574 -0.440062  0.000045
        4  0.422496  0.837491  0.463325    ...   -0.512525 -0.214044 -1.188965

                43        44        45        46        47        48        49
        0 -0.186627  0.356217  1.041784 -1.336890 -0.841114 -2.055547  0.254072
        1 -0.473160 -0.706495 -1.486052 -2.697282  0.200836 -1.150358 -0.861136
        2 -0.093445  0.499904  1.903753 -1.808682 -0.298960 -1.709499  0.067094
        3 -0.984545 -1.001008  0.522968 -1.100183 -0.945127  0.109518  0.340796
        4 -0.876839  0.214407  1.487839  0.364629 -0.672114 -1.033079  0.833580

        [5 rows x 50 columns]
```

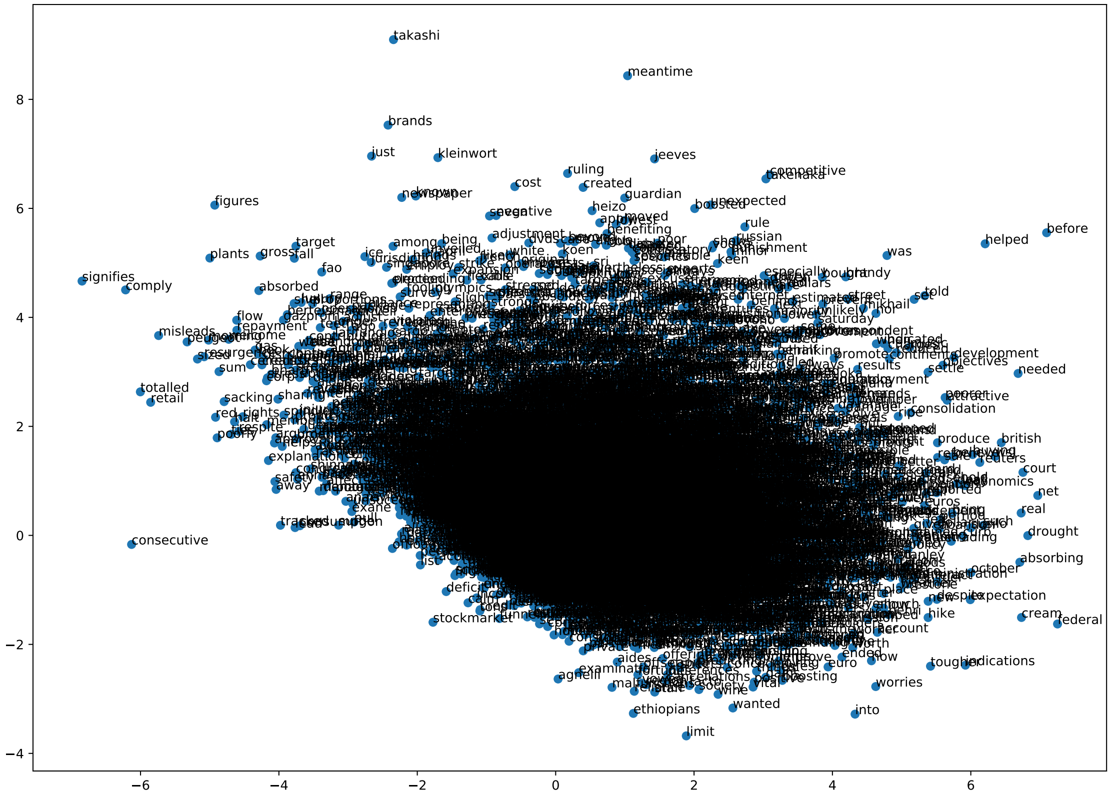
```
In [10]: pd_pca.describe()
```

```
Out[10]:                  0              1              2              3              4  \
         count  27820.000000   27820.000000   27820.000000   27820.000000   27820.000000
         mean       0.125524      -0.169628      -0.067795       0.120553       0.001522
         std        0.550139       0.476991       0.505541       0.508944       0.560462
         min       -4.154860      -6.404276      -4.949392      -4.925735      -5.870419
         25%       -0.082077      -0.267298      -0.187023      -0.045301      -0.208761
```

```
50%          0.050476      -0.101843      -0.035407       0.091097      -0.057569
75%          0.223501       0.015809       0.094506       0.248617       0.119810
max          6.200262       4.677252       6.145994       5.772127       7.172783

                    5              6              7              8              9   \
count  27820.000000   27820.000000   27820.000000   27820.000000   27820.000000
mean      -0.095893       0.002890       0.063026       0.185860      -0.249106
std        0.519468       0.508111       0.486796       0.536238       0.515400
min       -7.153666      -6.365167      -5.612720      -3.575420      -5.251027
25%       -0.253834      -0.139617      -0.089004      -0.032671      -0.372970
50%       -0.097138      -0.003427       0.044339       0.102022      -0.162185
75%        0.053901       0.145818       0.189228       0.287856      -0.024491
max        5.649132       5.340718       6.153919       6.863942       5.008322

             ...                40             41             42             43   \
count        ...      27820.000000   27820.000000   27820.000000   27820.000000
mean         ...         -0.152143      -0.088448       0.128279      -0.064979
std          ...          0.541836       0.537160       0.548088       0.474637
min          ...         -7.821931      -5.750125      -5.388204      -5.450548
25%          ...         -0.248100      -0.217153      -0.049167      -0.163617
50%          ...         -0.087370      -0.022593       0.085809      -0.011314
75%          ...          0.037388       0.119919       0.252745       0.106930
max          ...          6.557204       4.221970       5.968032       4.754703

                   44             45             46             47             48   \
count  27820.000000   27820.000000   27820.000000   27820.000000   27820.000000
mean      -0.186449       0.033639      -0.146283      -0.051561      -0.258583
std        0.533074       0.475167       0.502708       0.526275       0.520547
min       -5.100598      -5.088698      -5.662367      -5.621291      -5.798713
25%       -0.300172      -0.106059      -0.245580      -0.159858      -0.360391
50%       -0.113899       0.039580      -0.073597      -0.006742      -0.159692
75%        0.023998       0.179518       0.050471       0.122295      -0.022224
max        4.868486       5.038669       4.323182       5.331218       5.819730

                   49
count  27820.000000
mean       0.244275
std        0.592942
min       -4.717986
25%       -0.026753
50%        0.114209
75%        0.334777
max        5.879242

[8 rows x 50 columns]
```

In [11]:
```python
# Isso aqui demora um pouco
plt.figure(figsize=(15,11))
```

```
X_pca = PCA(n_components=2).fit_transform(model.wv.vectors)
plt.scatter(X_pca[:, 0], X_pca[:, 1]);

xv = list(zip(X_pca[:, :2].tolist(), model.wv.vocab))

count_left = [0, 0, 0, 0]
for X, word in xv:
    plt.text(X[0], X[1], word)
```



O PCA em duas dimensões não obtem um bom resultado mas já da para verificar alguns resultados que mostram algumas similaridades entre as palavras

```
In [12]: # Segue algumas palavras junto com suas palavras mais similares baseado no treinament
         for _ in range(5):
             word = np.random.choice(list(model.wv.vocab))
             print(word, model.most_similar(word))
             print()
```

```
decorated [('obese', 0.6022043824195862), ('digitally', 0.5962415933609009), ('venerable', 0.5

xlr [('coupes', 0.9377292394638062), ('srx', 0.8572789430618286), ('freestyle', 0.856921136379

expose [('overrule', 0.7195631265640259), ('delist', 0.7141237854957581), ('afflict', 0.71378
```

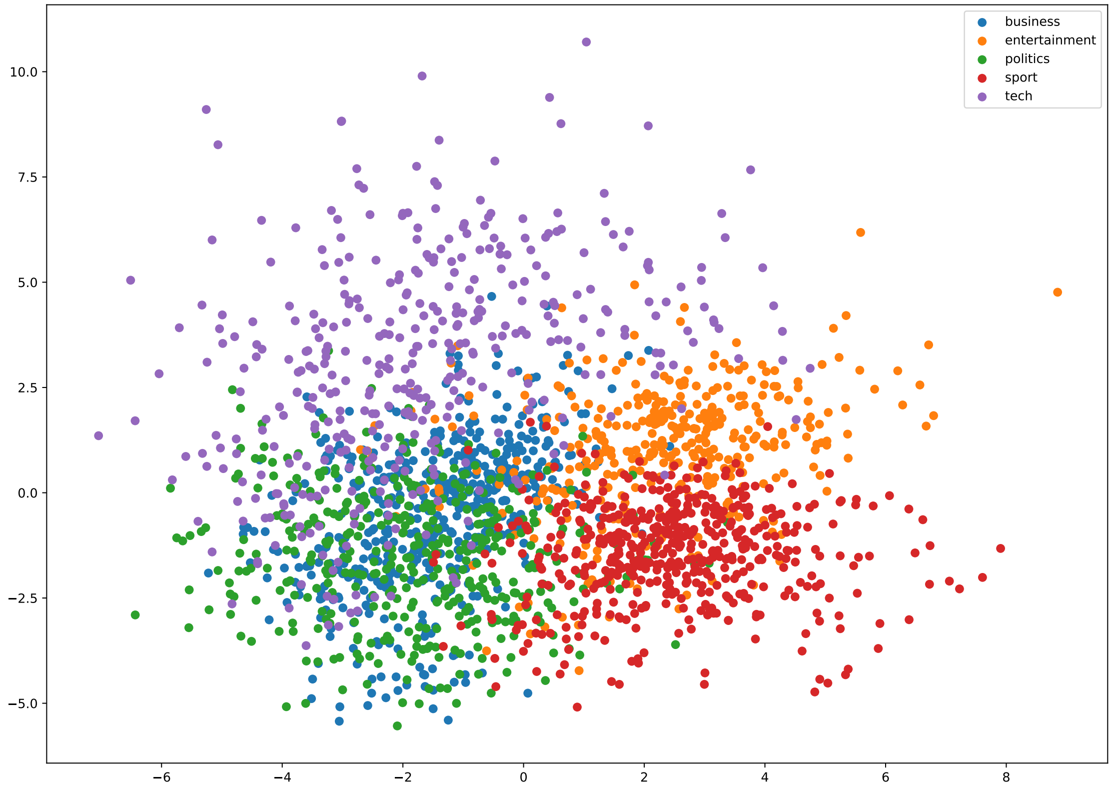
```
particuarly [('broom', 0.6206068992614746), ('zealander', 0.6149219274520874), ('listening', 0

convictions [('prima', 0.7080569863319397), ('exhibited', 0.6881054639816284), ('astronaut', 0
```

```
c:\users\ignit\local\lib\site-packages\ipykernel_launcher.py:4: DeprecationWarning: Call to dep
  after removing the cwd from sys.path.
c:\users\ignit\local\lib\site-packages\gensim\matutils.py:737: FutureWarning: Conversion of the
  if np.issubdtype(vec.dtype, np.int):
```

3.3 Doc2Vec

O modelo sendo uma extensão do Word2vec, faz com que cada documento possua um vetor de características também, quer dizer que posso plotar estes dados já certo?

```
In [12]: plt.figure(figsize=(15,11))
         X_pca = PCA(n_components=2).fit_transform(model.docvecs.vectors_docs)

         for cl in df['class'].unique():
             plt.scatter(X_pca[df['class'] == cl, 0], X_pca[df['class'] == cl, 1]);
         plt.legend(list(df['class'].unique()));
```



```
In [13]: # Exemplo de inferência utilizando 3 palavras
         # o resultado pode ser diferente em cada execução devido ao processo de escolhas alea
         model.infer_vector(['playstation', 'launchs', 'day'])
```

```
Out[13]: array([ 0.03649032, -0.02189194, -0.21670197,  0.10387941, -0.07442366,
                 0.38828552,  0.0548493 ,  0.05776116,  0.06041327,  0.05055714,
                -0.07257181, -0.19030558,  0.00718318, -0.05017371, -0.17033762,
                 0.22027378,  0.14903647,  0.04072108,  0.01473602,  0.17601186,
                 0.10048933,  0.11544574,  0.13109535,  0.10451017,  0.05056227,
                -0.03562302, -0.02262274,  0.21025397, -0.03123427,  0.01517741,
                 0.21585454, -0.23880677, -0.07298025, -0.42041168, -0.20753248,
                -0.06107889,  0.09495735, -0.14960605,  0.06073452,  0.06555965,
                -0.04054106, -0.01832818, -0.09735152, -0.07830091, -0.05992536,
                 0.07480311,  0.02125435, -0.09300853, -0.2033982 , -0.3173989 ],
               dtype=float32)
```

Esta função tem como objetivo colecionar os ranks dos documentos além de retornar os vetores dos documentos inferidos.

```
In [18]: def get_ranks():
             ranks = []
             second_ranks = []
             vectors = []
             for doc_id in range(len(documents)):
                 inferred_vector = model.infer_vector(documents[doc_id].words)
                 vectors.append(inferred_vector)
                 sims = model.docvecs.most_similar([inferred_vector], topn=len(model.docvecs))
                 rank = [docid for docid, sim in sims].index(doc_id)
                 ranks.append(rank)

                 second_ranks.append(sims[1])

             return ranks, second_ranks, vectors
```

4 Partindo para o lemonade

Vamos utilizar estes dados contendo os vetores inferidos dos documentos para utilizarmos um classificador na plataforma.

```
In [19]: columns_name = []
         for dim in range(model.vector_size):
             columns_name += ['vec' + str(dim)]

         ranks, second_ran, vectors = get_ranks()
```

```
vectors = np.array(vectors)

df_vec = pd.DataFrame(vectors, columns=columns_name)
df_vec.head()
```

Out[19]:

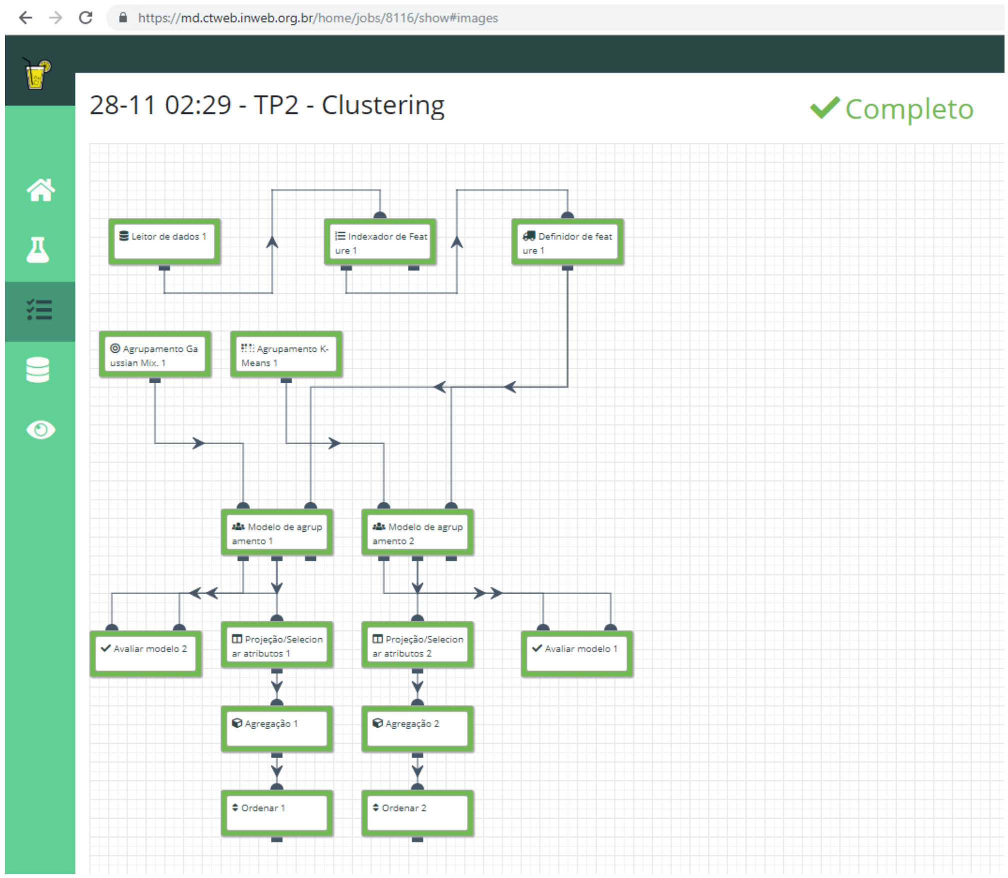
```
       vec0      vec1      vec2      vec3      vec4      vec5      vec6  \
0  1.858941 -2.175003 -0.860748 -0.215045  1.378530 -0.315213 -1.636518
1  1.570992 -1.304538  0.746336 -0.641785 -2.830541 -0.326982 -0.047434
2  0.848429 -1.960781 -1.300094 -0.624425  1.769062  1.167360  1.614454
3  3.158700  2.152319 -1.599548 -0.752534  0.494211 -2.105348 -0.999065
4 -0.178957 -0.854439 -0.611729 -1.097837 -0.386107  2.416610  0.359226

       vec7      vec8      vec9    ...        vec40     vec41     vec42  \
0  2.533383 -1.316651  1.742312    ...     1.992256 -0.732948  1.204113
1 -0.174483  3.324303  0.243610    ...     0.772442 -1.068378  0.822466
2  0.708561 -0.169156 -2.343792    ...    -2.094990 -0.036672  0.859912
3  0.671154 -0.670825 -1.369363    ...     0.392010 -0.945897 -0.270733
4  1.209108  1.373217 -0.529249    ...     0.703905 -1.906525  0.769721

      vec43     vec44     vec45     vec46     vec47     vec48     vec49
0  1.005686  1.989221  0.802130 -2.041804 -2.542200 -2.846420  0.775310
1  1.299721  1.436806 -0.840429  0.006554 -1.228581 -3.059551 -0.845000
2  1.560098  0.539960 -0.839072 -1.087391 -1.019731 -1.437251 -0.306309
3  1.301290  3.639134  1.762413 -1.953189 -0.657869 -0.690287  1.083300
4  2.589010 -0.299332  0.556422 -1.882543  2.112225 -1.616394 -0.649231

[5 rows x 50 columns]
```

In [20]:

```
# Gravar os vetores para importar no Lemonade
output_data = pd.concat([pd.DataFrame(df['class'].values, columns=['class']), df_vec]
output_data.to_csv('BBCNewsdoc2vecinfer.csv', index=False)
output_data.head()
```

Out[20]:

```
      class      vec0      vec1      vec2      vec3      vec4      vec5  \
0  business  1.858941 -2.175003 -0.860748 -0.215045  1.378530 -0.315213
1  business  1.570992 -1.304538  0.746336 -0.641785 -2.830541 -0.326982
2  business  0.848429 -1.960781 -1.300094 -0.624425  1.769062  1.167360
3  business  3.158700  2.152319 -1.599548 -0.752534  0.494211 -2.105348
4  business -0.178957 -0.854439 -0.611729 -1.097837 -0.386107  2.416610

       vec6      vec7      vec8    ...        vec40     vec41     vec42  \
0 -1.636518  2.533383 -1.316651    ...     1.992256 -0.732948  1.204113
1 -0.047434 -0.174483  3.324303    ...     0.772442 -1.068378  0.822466
2  1.614454  0.708561 -0.169156    ...    -2.094990 -0.036672  0.859912
3 -0.999065  0.671154 -0.670825    ...     0.392010 -0.945897 -0.270733
4  0.359226  1.209108  1.373217    ...     0.703905 -1.906525  0.769721

      vec43     vec44     vec45     vec46     vec47     vec48     vec49
```



Lemonade Gaussian Example

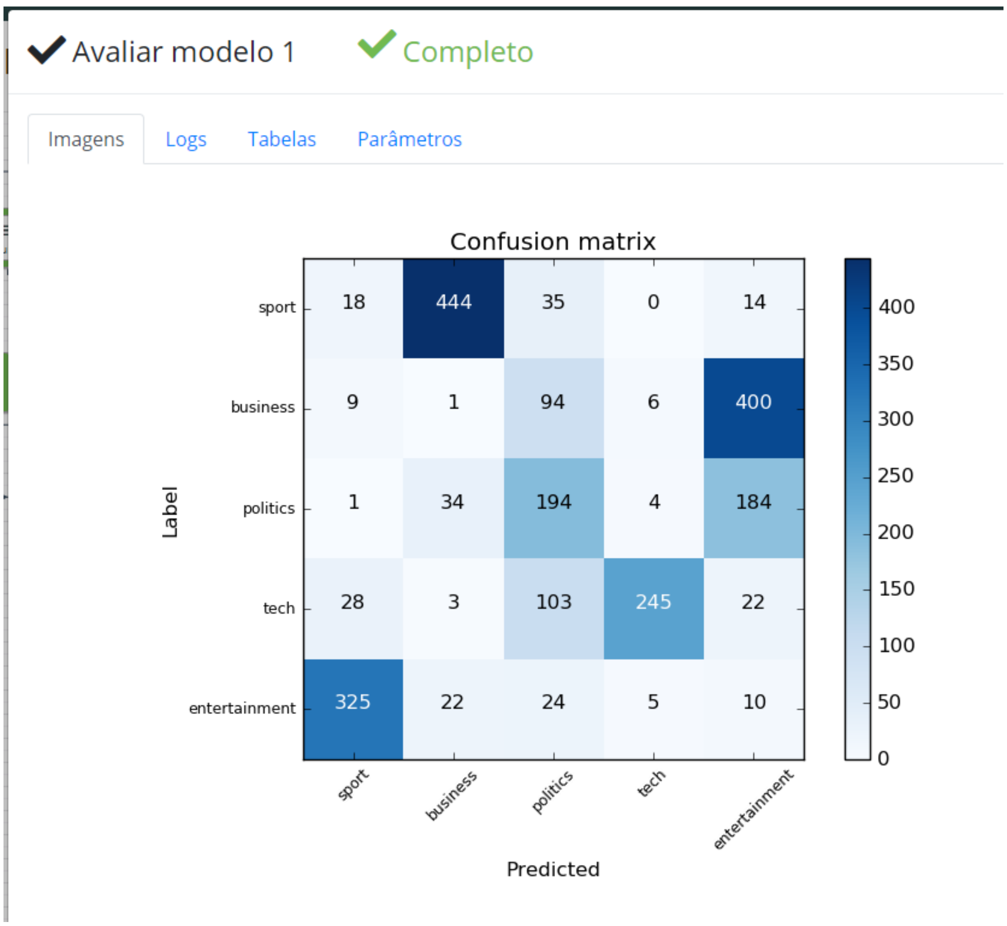
```
0   1.005686   1.989221   0.802130 -2.041804 -2.542200 -2.846420   0.775310
1   1.299721   1.436806 -0.840429  0.006554 -1.228581 -3.059551 -0.845000
2   1.560098   0.539960 -0.839072 -1.087391 -1.019731 -1.437251 -0.306309
3   1.301290   3.639134   1.762413 -1.953189 -0.657869 -0.690287   1.083300
4   2.589010 -0.299332   0.556422 -1.882543  2.112225 -1.616394 -0.649231

[5 rows x 51 columns]
```

O Job **8116** no lemonade nos da os seguintes resultados:

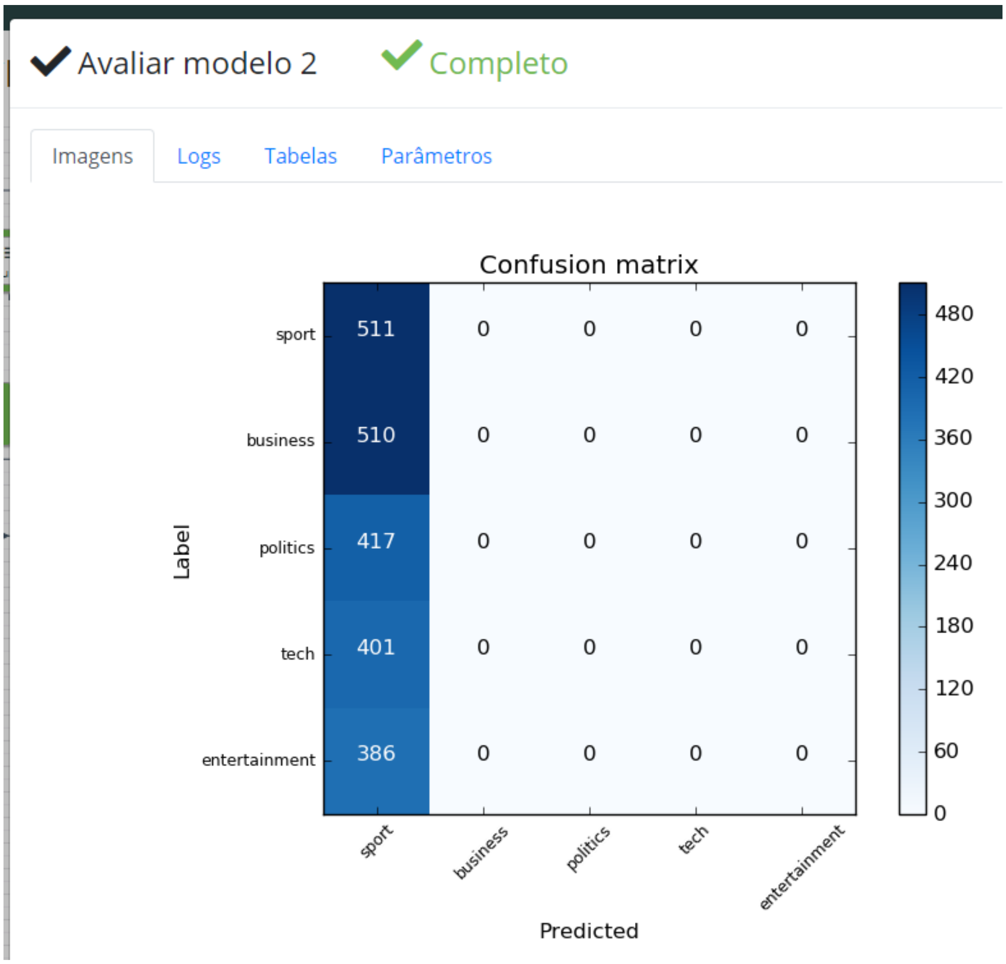
K-Means Ignore os labels pois tenho que indexar por números as classes.

O K-means se saiu bem a mais do que o esperado apesar que errou muito. Talvex o conjuntos dos vetores possuem um manifold e distâncias euclidianas não funcionam muito bem para K-means.

Gaussian-Mix O Gaussian mix não consegui fazer funcionar para 50 dimensões, a maldição da dimensionalidade faz com que as distâncias muito longas não influênciam muito para o movimento dos centroides das gaussianas.


Avaliar modelo 1
Completo
Imagens
Logs
Tabelas
Parâmetros
Confusion matrix
sport
business
politics
tech
entertainment
Label
18
444
35
0
14
9
1
94
6
400
1
34
194
4
184
28
3
103
245
22
325
22
24
5
10
sport
business
politics
tech
entertainment
Predicted
400
350
300
250
200
150
100
50
0

K-Means



Gaussian-Mix

5 Conclusão

Nesse pequeno dataset perbemos uma boa eficácia do K-Means sobre o Gaussian-Mix neste dados que são bem complicados de transformar em um vetor para agrupá-los. O método do Doc2Vec não se mostrou tão eficaz para o agrupamento pela natureza dos dados apresentados.