NLP - Classificador de Artistas

February 26, 2021

1 Classificador de Artistas

O objetivo desse desafio é classificar com base no trecho de uma musica, qual artista é o cantor.

Etapas

- Analise exploratoria
  - Explorar dados faltantes
  - Verificar balanceamento do dataset
  - Identificar Stopwords mais comuns
- Preprocessamento de dados
  - Remoção de stopwords
  - Aplicação do TF-IDF
  - Criação do Bag of Words
- Analise de modelos
  - Escolha da melhor metrica
  - Testes usando os diferentes tipos de dados processados
  - Testes com SVC, MLP, XGB e outros
  - Escolha dos melhores modelos
- Otimização de modelos
  - Tuning do modelo
  - Criação de um ensemble
  - Seleção do melhor resultado

[56]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import entropy
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import VotingClassifier
import xgboost as xgb
import re
import random
```

[57]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/andresacilotti/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[57]:
```
True
```

Para garantir a replicabilidade do projeto é necessario definir as seeds de valores random

[58]:
```
random.seed = 42
np.random.seed(42)
```

1.1 Extração de dados

[3]:
```
data = pd.read_excel("Data/dataset.xls", sheet_name='NLP')
```

[4]:
```
data
```

[4]:

	letra	artista
0	Jay-z Uh-uh-uh You ready b? Let's go get 'em. …	Beyoncé
1	Your challengers are a young group from Housto…	Beyoncé
2	Dum-da-de-da Do, do, do, do, do, do (Coming do…	Beyoncé
3	If I ain't got nothing I got you If I ain't go…	Beyoncé
4	Six inch heels She walked in the club like nob…	Beyoncé
..	…	…
513	Yeah yeah Yeah yeah I ain't trying to think a…	Rihanna
514	You the one that I dream about all day You the…	Rihanna
515	No, no, no You don't love me and I know now No…	Rihanna
516	You should be mine Oh baby, oh baby, oh baby, …	Rihanna
517	[Rihanna] I remember when the world was just m…	Rihanna

[518 rows x 2 columns]

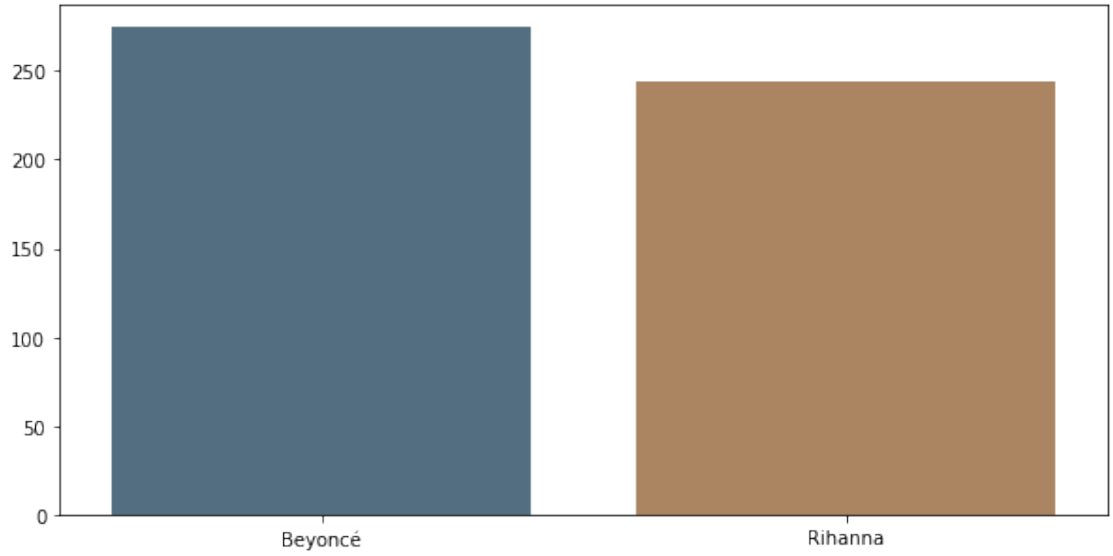
1.2 Analise Exploratoria

1.2.1 Distribuiçao das classes

Um ponto inicial interessante é analisar a distribuição das classes e verificar se há dados desbalanceados ou não, para que assim possa inferido a necessidade de tratamentos especiais como data agumentation ou não.

[5]:
```
y = list(data['artista'].value_counts())
x = list(np.unique(data['artista']))

plt.figure(figsize=(10, 5))
sns.barplot(x, y, saturation=0.3)
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58294e0ed0>
```



Temos uma informação super importante, trata-se de uma classificação binaria, pois há somente duas classes.

Além disso, os dados não estão tão desbalanceados, podemos checar a entropia para ter certeza. Lembrando que, pela entropia de Shannon:

0 - Completamente desbalanceado;

1 - Completamente Balanceado

[6]:
```
print("Entropia:", entropy([new_val/sum(y) for new_val in y], base=2))
```

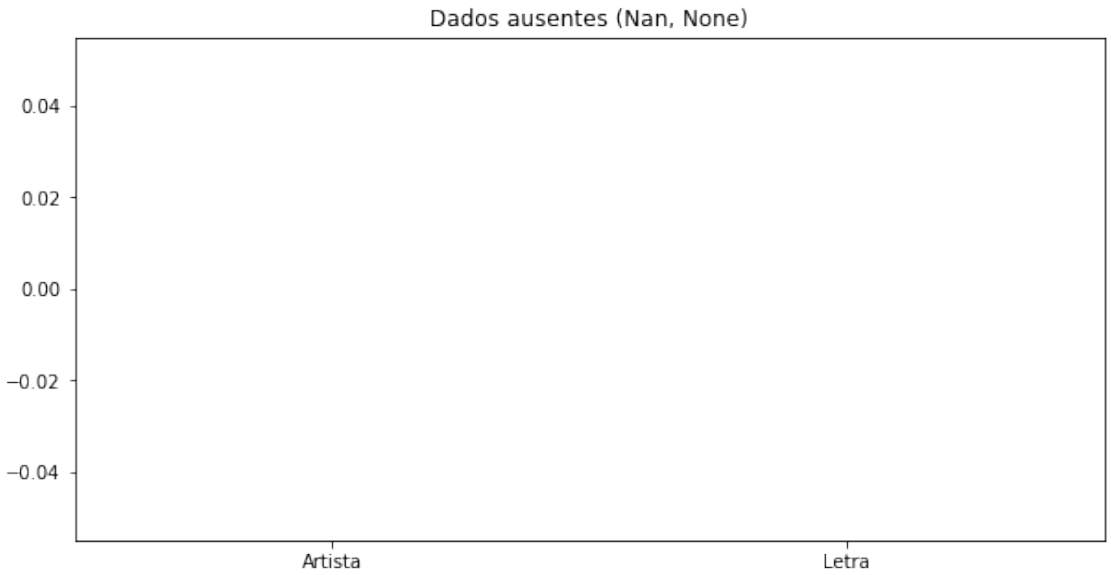
```
Entropia: 0.9975791349905527
```

1.2.2 Dados Faltantes

Há alguns tratamentos especiais para tal caso e não podemos ignorar dados ausentes.

[7]:
```
x = ["Artista", "Letra"]
y = [data['artista'].isna().sum(), data['letra'].isna().sum()]


plt.figure(figsize=(10, 5))
plt.title("Dados ausentes (Nan, None)")
sns.barplot(x, y, saturation=0.3)

print("Dados Faltantes: {} -> {}".format(x, y))
```

```
Dados Faltantes: ['Artista', 'Letra'] -> [0, 0]
```



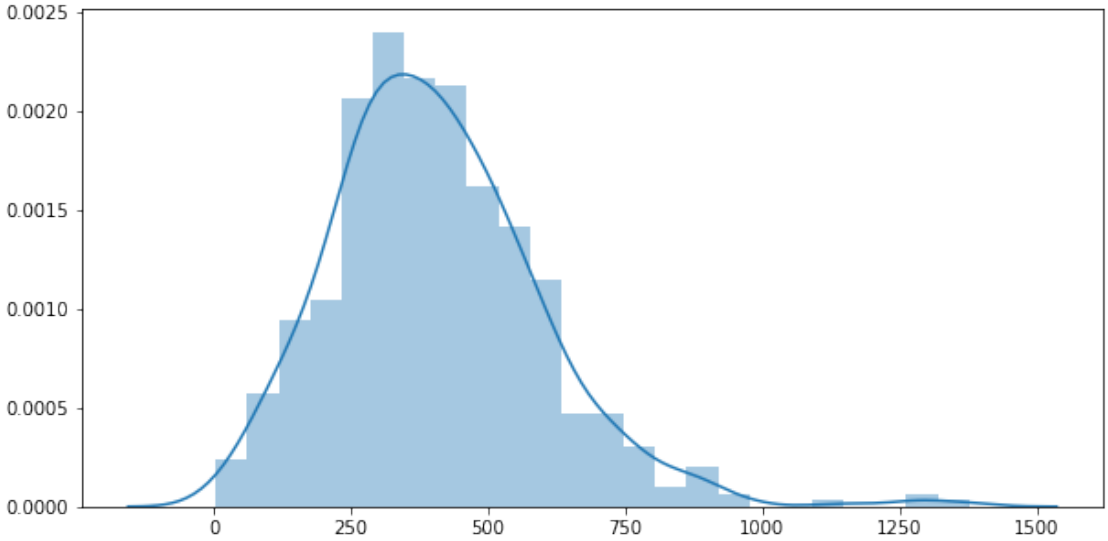
No entanto, não há dados ausentes nessa base de dados

1.2.3 Quantidade de palavras em cada letra

Alguns algoritmos como o BERT, possuem uma limitação de tokens (Palavras) a serem usadas no embedding, portanto, devemos checar se a quantidade de palavras e inferir uma media.

[8]:
```
words = data['letra'].str.split(" ").replace(" ", "")
song_size = [len(_words) for _words in words]

plt.figure(figsize=(10, 5))
sns.distplot(song_size)
```

[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5827f80790>



[9]:
```
print("Média: ", sum(song_size)/len(song_size))
print("Mediana: ", np.median(song_size))
```

```
Média:  405.8050193050193
Mediana:  386.0
```

Temos o valor médio da quantidade palavras, em alguns casos ultrapassa o limite de tokens do BERT, no entanto, ainda teremos que decidir qual modelos usar mais a frente

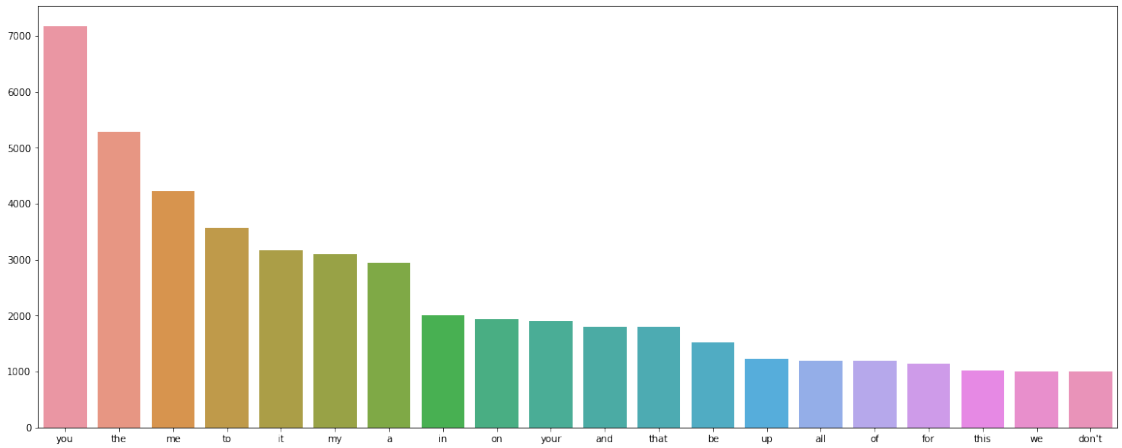
1.2.4 Presença de stopswords

Stopwords são palavras que em alguns contextos podem ser irrelevantes, por isso, precisamos analisar a presença das mesmas para que possamos julgar se há necessidade um processamento futuro

[10]:
```
stop_words = set(stopwords.words('english'))
stop_words


stop_words_in_songs = {}

for rows in words:
    for _word in rows:
        if _word in stop_words:
            try:
                stop_words_in_songs[_word] += 1
            except:
                stop_words_in_songs[_word] = 1
```

[11]:
```
stop_words_in_songs = dict(sorted(stop_words_in_songs.items(), key=lambda item:␣
 ↪item[1], reverse=True))

y = list(stop_words_in_songs.values())[:20]
x = list(stop_words_in_songs.keys())[:20]

plt.figure(figsize=(20, 8))
sns.barplot(x=x, y=y)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5827e5fb50>
```



Observa-se que há uma grande prevalencia de stopwords, palavras como you, the, me, to, it provavelmente não nos ajudarão a saber quem é o cantor, no entanto, faremos testes para confirmar nossa hipotese futuramente

1.3 Processamento dos dados

No nosso processo de treinamento de dados, vamos testar os seguintes processamentos

- Dados cru
- Dados sem stopwords e caracteres especiais

Além disso, usaremos os seguintes algoritmos para transformar os textos em dados estruturados:

- TF-IDF
- Bag of Words

OBSERVAÇÃO IMPORTANTE: não podemos usar o TF-IDF diretamente no dataset inteiro, pois assim estaremos passando informações adiante no modelo, aumentando o score de testes para além de um valor real

[12]:
```
x_raw = pd.Series(data['letra'])
y_raw = preprocessing.LabelEncoder().fit_transform(data['artista'])
```

1.3.1 Dados sem stopwords

[13]:
```
x_without_stopword = []

for song in data['letra']:
    song_letra = ""
    for letra in song.split(" "):
        if letra.replace(" ", "").lower() not in stop_words:
            song_letra += re.sub('\\/.,;?!^/[/]', "", letra) + " "
    x_without_stopword.append(song_letra)

x_without_stopword = pd.Series(x_without_stopword)
y_without_stopword = preprocessing.LabelEncoder().fit_transform(data['artista'])
```

1.4 Divisão de treino e testes

Para dividir entre teste e treino, usaremos 40% para testes e 60% para treino, estratificando os dados e setando uma seed no random, além de aplicar o fit do TF-IDF/WordEmbedding nos dados de treino e transformar os dados de teste

1.4.1 TF-IDF - without stopword data

Utilizando o método TF-IDF e os dados sem as stopwords, criaremos um conjunto de dados processados

[14]:
```
(x_train_without_stopword,
 x_test_without_stopword,
 y_train_without_stopword,
 y_test_without_stopword) = train_test_split(x_without_stopword,
                                             y_without_stopword,
                                             test_size=0.3,
                                             random_state=42,
                                             stratify=y_without_stopword)

x_test_without_stopword_raw = x_test_without_stopword
x_train_without_stopword_raw = x_train_without_stopword
y_test_without_stopword_raw = y_test_without_stopword
y_train_without_stopword_raw = y_train_without_stopword

without_stopword_vectorizer = TfidfVectorizer()

x_train_without_stopword_tfidf = without_stopword_vectorizer.
 ↪fit_transform(x_train_without_stopword)
x_test_without_stopword_tfidf = without_stopword_vectorizer.
 ↪transform(x_test_without_stopword)
```

1.4.2 TF-IDF - raw data

Utilizando o método TF-IDF e os dados brutos, criaremos um conjunto de dados processados

[15]:
```
(x_train_raw,
 x_test_raw,
 y_train_raw,
 y_test_raw) = train_test_split(x_raw,
                                y_raw,
                                test_size=0.3,
                                random_state=42,
                                stratify=y_raw)
x_test_raw_raw = x_test_raw
x_train_raw_raw = x_train_raw
y_test_raw_raw = y_test_raw
y_train_raw_raw = y_train_raw


raw_vectorizer = TfidfVectorizer(max_features=5000)

x_train_raw_tfidf = raw_vectorizer.fit_transform(x_train_raw)
x_test_raw_tfidf = raw_vectorizer.transform(x_test_raw)
```

1.4.3 Bag of Words - raw data

Utilizando o método BoW e os dados brutos, criaremos um conjunto de dados processados

[16]:
```
(x_train_raw,
 x_test_raw,
 y_train_raw,
 y_test_raw) = train_test_split(x_raw,
                                y_raw,
                                test_size=0.3,
                                random_state=42,
                                stratify=y_raw)
x_test_raw_raw = x_test_raw
x_train_raw_raw = x_train_raw
y_test_raw_raw = y_test_raw
y_train_raw_raw = y_train_raw


raw_vectorizer = CountVectorizer(stop_words='english')

x_train_raw_bow = raw_vectorizer.fit_transform(x_train_raw)
x_test_raw_bow = raw_vectorizer.transform(x_test_raw)
```

1.4.4 Bag of Words - without stopword data

Utilizando o método BoW e os dados sem as stopwords, criaremos um conjunto de dados processados

[17]:
```
(x_train_without_stopword,
 x_test_without_stopword,
 y_train_without_stopword,
 y_test_without_stopword) = train_test_split(x_without_stopword,
                                             y_without_stopword,
                                             test_size=0.3,
                                             random_state=42,
                                             stratify=y_without_stopword)

x_test_without_stopword_raw = x_test_without_stopword
x_train_without_stopword_raw = x_train_without_stopword
y_test_without_stopword_raw = y_test_without_stopword
y_train_without_stopword_raw = y_train_without_stopword


without_stopword_vectorizer = CountVectorizer(stop_words='english')

x_train_without_stopword_bow = without_stopword_vectorizer.
 ↪fit_transform(x_train_without_stopword)
x_test_without_stopword_bow = without_stopword_vectorizer.
 ↪transform(x_test_without_stopword)
```

1.5 Modelo de Aprendizado de Maquina

A escolha da metrica é um ponto importante, a principio, por se tratar de uma classificação binaria pouco desbalanceado, poderiamos utilizar a acuracia, no entanto, optarei por utilizar o F1-Score, garantindo uma certeza maior na inferencia do score. Além disso, utilizaremos a media MACRO, garantindo que não há vies devido ao desbalanceamento dos dados

1.5.1 SVC - Dados Cru - TF-IDF

[18]:
```
svc = SVC(random_state=42).fit(x_train_raw_tfidf, y_train_raw)

y_new = svc.predict(x_test_raw_tfidf)

f1_score(y_new, y_test_raw, average='macro')
```

[18]:
```
0.6907764328792543
```

1.5.2 SVC - Dados sem stopwords - TF-IDF

[19]:
```
svc = SVC(random_state=42).fit(x_train_without_stopword_tfidf,␣
↪y_train_without_stopword)

y_new = svc.predict(x_test_without_stopword_tfidf)

f1_score(y_new, y_test_without_stopword, average='macro')
```

[19]: 0.7218070652173914

1.5.3 SVC - Dados Cru - BoW

[20]:
```
svc = SVC(random_state=42).fit(x_train_raw_bow, y_train_raw)

y_new = svc.predict(x_test_raw_bow)

f1_score(y_new, y_test_raw, average='macro')
```

[20]: 0.6343749999999999

1.5.4 SVC - Dados sem stopwords - BoW

[21]:
```
svc = SVC(random_state=42).fit(x_train_without_stopword_bow,␣
↪y_train_without_stopword)

y_new = svc.predict(x_test_without_stopword_bow)

f1_score(y_new, y_test_without_stopword, average='macro')
```

[21]: 0.6287482151356496

1.5.5 NB - Dados Cru - TF-IDF

[22]:
```
nb = MultinomialNB().fit(x_train_raw_tfidf, y_train_raw)

y_new = nb.predict(x_test_raw_tfidf)

f1_score(y_new, y_test_raw, average='macro')
```

[22]: 0.6713728654027162

1.5.6 NB - Dados sem Stopwords - TF-IDF

[23]:
```
nb = MultinomialNB().fit(x_train_without_stopword_tfidf,␣
↪y_train_without_stopword)

y_new = nb.predict(x_test_without_stopword_tfidf)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
```

```
0.6728988292853398
```

1.5.7 NB - Dados Cru - BoW

[24]:
```
nb = MultinomialNB().fit(x_train_raw_bow, y_train_raw)

y_new = nb.predict(x_test_raw_bow)

f1_score(y_new, y_test_raw, average='macro')
```

[24]:
```
0.7112418246884127
```

1.5.8 NB - Dados sem Stopwords - BoW

[25]:
```
nb = MultinomialNB().fit(x_train_without_stopword_bow, y_train_without_stopword)

y_new = nb.predict(x_test_without_stopword_bow)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
```

```
0.7305921052631579
```

1.5.9 MLP - Dados Cru - TF-IDF

[26]:
```
mlp = MLPClassifier(random_state=42).fit(x_train_raw_tfidf, y_train_raw)

y_new = mlp.predict(x_test_raw_tfidf)

f1_score(y_new, y_test_raw, average='macro')
```

[26]:
```
0.6843278959322734
```

1.5.10 MLP - Dados sem Stopwords - TF-IDF

[27]:
```
mlp = MLPClassifier(random_state=42).fit(x_train_without_stopword_tfidf,␣
↪y_train_without_stopword)

y_new = mlp.predict(x_test_without_stopword_tfidf)
```

```
print(f1_score(y_new, y_test_without_stopword, average='macro'))
```

0.7305921052631579

1.5.11 MLP - Dados Cru - BoW

[28]:
```
mlp = MLPClassifier(random_state=42).fit(x_train_raw_bow, y_train_raw)

y_new = mlp.predict(x_test_raw_bow)

f1_score(y_new, y_test_raw, average='macro')
```

[28]: 0.7167849480112229

1.5.12 MLP - Dados sem Stopwords - BoW

[29]:
```
mlp = MLPClassifier(random_state=42).fit(x_train_without_stopword_bow,␣
↪y_train_without_stopword)

y_new = mlp.predict(x_test_without_stopword_bow)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
```

0.7100970472847409

1.5.13 SGDClassifier - Dados cru - TF-IDF

[30]:
```
sgd = SGDClassifier(random_state=42).fit(x_train_raw_tfidf, y_train_raw)

y_new = sgd.predict(x_test_raw_tfidf)

f1_score(y_new, y_test_raw, average='macro')
```

[30]: 0.6852637213324

1.5.14 SGDClassifier - Dados sem Stopwords - TF-IDF

[31]:
```
sgd = SGDClassifier(random_state=42).fit(x_train_without_stopword_tfidf,␣
↪y_train_without_stopword)

y_new = sgd.predict(x_test_without_stopword_tfidf)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
```

0.663960231980116

1.5.15 SGDClassifier - Dados cru - BoW

[32]:
```
sgd = SGDClassifier(random_state=42).fit(x_train_raw_bow, y_train_raw)

y_new = sgd.predict(x_test_raw_bow)

f1_score(y_new, y_test_raw, average='macro')
```

[32]:
```
0.6537892487259576
```

1.5.16 SGDClassifier - Dados sem Stopwords - BoW

[33]:
```
sgd = SGDClassifier(random_state=42).fit(x_train_without_stopword_bow,␣
↪y_train_without_stopword)

y_new = sgd.predict(x_test_without_stopword_bow)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
```

```
0.7046913580246913
```

1.5.17 XGB - Dados cru - TF-IDF

[34]:
```
_xgb = xgb.XGBClassifier(random_state=42).fit(x_train_raw_tfidf, y_train_raw)

y_new = _xgb.predict(x_test_raw_tfidf)

f1_score(y_new, y_test_raw, average='macro')
```

[34]:
```
0.7425317709192936
```

1.5.18 XGB - Dados sem stopwords - TF-IDF

[35]:
```
_xgb = xgb.XGBClassifier(random_state=42).fit(x_train_without_stopword_tfidf,␣
↪y_train_without_stopword)

y_new = _xgb.predict(x_test_without_stopword_tfidf)

f1_score(y_new, y_test_without_stopword, average='macro')
```

[35]:
```
0.7686222808174028
```

1.5.19 XGB - Dados cru - BoW

[36]:
```
_xgb = xgb.XGBClassifier(random_state=42).fit(x_train_raw_bow, y_train_raw)

y_new = _xgb.predict(x_test_raw_bow)

f1_score(y_new, y_test_raw, average='macro')
```

[36]:
```
0.7347268881423417
```

1.5.20 XGB - Dados sem stopwords - BoW

[37]:
```
_xgb = xgb.XGBClassifier(random_state=42).fit(x_train_without_stopword_bow,␣
↪y_train_without_stopword)

y_new = _xgb.predict(x_test_without_stopword_bow)

f1_score(y_new, y_test_without_stopword, average='macro')
```

[37]:
```
0.7305921052631579
```

1.6 Otimização de modelos

Alguns modelos se destacaram, dentre eles XGB e SVC, portanto, aplicaremos um gridsearch para otimiziar cada modelo. Além disso, como na maioria dos modelos o TF-IDF obteve o melhor resultado, usaremos tal processo durante a otimização

1.6.1 XGB - Dados Cru

[48]:
```
x_train_raw.reset_index(drop=True)
```

[48]:
```
0      I can do a lot of shit But I can't make you lo…
1      Miss Third Wards, your first question What is …
2      I love you.. Baby I love you You are my life M…
3      I lick the gun When I'm done Cause I know That…
4      It's hot in this club It's time to set it off …
                            …
357    She was lost In so many different ways Out in …
358    [Intro: Beyoncé] Who wants that perfect love s…
359    Shine bright like a diamond (C'mon) Shine brig…
360    Kiss it, kiss it better, baby Kiss it, kiss it…
361    Oh! Oooooh I love to see you walking into the …
Name: letra, Length: 362, dtype: object
```

[49]:
```
parameters = {
    'learning_rate': [0.3, 0.1, 0.15],
    'gamma': [0, 100],
```

```
    'max_depth': [6, 3, 10, 25],
    'subsample': [1, 0.5, 0.1],
    'scale_pos_weight': [0, 1],
    'random_state': [42]
}

_xgb = xgb.XGBClassifier()
gs = GridSearchCV(_xgb, parameters)

gs.fit(x_train_raw_tfidf, y_train_raw)

y_new = gs.predict(x_test_raw_tfidf)

print(f1_score(y_new, y_test_raw, average='macro'))
print(gs.best_estimator_)
```

```
0.7363017934446505
XGBClassifier(learning_rate=0.3, max_depth=6, random_state=42)
```

1.6.2 XGB - Dados sem stopwords

[50]:
```
parameters = {
    'learning_rate': [0.3, 0.1, 0.15],
    'gamma': [0, 100],
    'max_depth': [6, 3, 10, 25],
    'subsample': [1, 0.5, 0.1],
    'scale_pos_weight': [0, 1],
    'random_state': [42]
}

_xgb = xgb.XGBClassifier()
gs = GridSearchCV(_xgb, parameters)

gs.fit(x_train_without_stopword_tfidf, y_train_without_stopword)

y_new = gs.predict(x_test_without_stopword_tfidf)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
print(gs.best_estimator_)
```

```
0.7814765985497693
XGBClassifier(max_depth=10, random_state=42)
```

1.6.3 SVC - Dados cru

[51]:
```
parameters = {
    'C': [0.001, 0.01, 1, 10, 1000],
    'kernel': ['linear','poly', 'rbf', 'sigmoid'],
    'degree': [2, 3, 4, 5, 10],
    'coef0': [0.001, 0, 1000],
    'decision_function_shape': ['ovo', 'ovr'],
    'random_state': [42]
}

svc = SVC()
gs = GridSearchCV(svc, parameters)

gs.fit(x_train_raw_tfidf, y_train_raw)

y_new = gs.predict(x_test_raw_tfidf)

print(f1_score(y_new, y_test_raw, average='macro'))
print(gs.best_estimator_)
```

```
0.655140771637122
SVC(C=0.001, coef0=1000, decision_function_shape='ovo', degree=2, kernel='poly',
    random_state=42)
```

1.6.4 SVC - Dados sem stopwords

[52]:
```
parameters = {
    'C': [0.001, 0.01, 1, 10, 1000],
    'kernel': ['linear','poly', 'rbf', 'sigmoid'],
    'degree': [2, 3, 4, 5, 10],
    'coef0': [0.001, 0, 1000],
    'decision_function_shape': ['ovo', 'ovr'],
    'random_state': [42]
}

svc = SVC()
gs = GridSearchCV(svc, parameters)

gs.fit(x_train_without_stopword_tfidf, y_train_without_stopword)

y_new = gs.predict(x_test_without_stopword_tfidf)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
print(gs.best_estimator_)
```

```
0.7122253899044106
SVC(C=10, coef0=0.001, decision_function_shape='ovo', degree=2, random_state=42)
```

1.7 Ensemble

Um outro conceito importante para testar, é aplicar um ensemble com os melhores modelos.

[53]:
```
_xgb = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
              max_depth=10, min_child_weight=1, missing=None, n_estimators=100,
              n_jobs=1, nthread=None, objective='binary:logistic',
              random_state=42, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=None, silent=True, subsample=1)

svc = SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.
↪001,
    decision_function_shape='ovo', degree=2, gamma='scale', kernel='rbf',
    max_iter=-1, probability=True, random_state=42, shrinking=True, tol=0.001,
    verbose=False)
```

[54]:
```
eclf = VotingClassifier(
        estimators=[('lr', svc), ('rf', _xgb)],
        voting='soft')

eclf.fit(x_train_without_stopword_tfidf, y_train_without_stopword)

y_new = eclf.predict(x_test_without_stopword_tfidf)

print(f1_score(y_new, y_test_without_stopword, average='macro'))
```

```
0.8134817563388992
```

1.8 Conclusão

O melhor score obtido foi 0.814, utilizando um ensemble do SVC e do XGB, conforme descrito acima. Para casos com poucos dados, tem-se uma medida de f1 muito boa, podendo ser melhorada se for ampliado o dataset, extraindo mais dados, criando dados sinteticos e afins.

1.9 Proximos Passos

Definido o melhor modelo, os proximos passos serão, criar uma pipeline de transformação de dados, no nosso caso, a pipeline com melhores resultados é aquela que exclui stopwords e caracteres especiais. E expor uma API para requisição da predição de uma letra.