prova2topicos-romulo

March 21, 2024

1 PROVA 2 - TÓPICOS ESPECIAIS EM ESTATÍSTICA (RÔMULO MENEZES DE SANTANA)

1.1 Sobre a base de dados:

- É uma base de dados do Twitter para análise de sentimento. Dada uma mensagem e uma entidade, o obejtivo é julgar o sentimento da mensagem sobre a entidade. As classes dos dados são: Positivo, Negativo e Neutro. As mensagens que não são relevantes à entidade são classificadas como neutras.
- Link para a base de dados: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis/data

1.1.1 Importações

[55]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import plotly.express as px
import re
import nltk
from wordcloud import WordCloud
import plotly.express as px
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.ensemble import RandomForestClassifier,␣
↪GradientBoostingClassifier, AdaBoostClassifier, StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score
```

1.1.2 Leitura do arquivo

[2]:
```
tweets_treino = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Tópicos␣
 ↪especiais em estatística/twitter_training.csv')
tweets_valid = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Tópicos␣
 ↪especiais em estatística/twitter_validation.csv')
```

1.1.3 Análise dos dados

[3]:
```
# Nomeando as colunas
nomes_colunas = ['tweetID', 'entidade', 'sentimento', 'conteudo_tweet']
tweets_treino.columns = nomes_colunas
tweets_valid.columns = nomes_colunas
```

[4]:
```
# Juntando os dados de treino e de validação
tweets = pd.concat([tweets_treino, tweets_valid], ignore_index=False)
tweets.head()
```

[4]:

	tweetID	entidade	sentimento	conteudo_tweet
0	2401	Borderlands	Positive	I am coming to the borders and I will kill you…
1	2401	Borderlands	Positive	im getting on borderlands and i will kill you …
2	2401	Borderlands	Positive	im coming on borderlands and i will murder you…
3	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder …
4	2401	Borderlands	Positive	im getting into borderlands and i can murder y…

[5]:
```
# conferindo os nomes das colunas
tweets.columns.tolist()
```

[5]:
```
['tweetID', 'entidade', 'sentimento', 'conteudo_tweet']
```

[6]:
```
# informações das colunas
tweets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75680 entries, 0 to 998
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   tweetID         75680 non-null  int64
 1   entidade        75680 non-null  object
 2   sentimento      75680 non-null  object
```

```
 3   conteudo_tweet  74994 non-null  object
dtypes: int64(1), object(3)
memory usage: 2.9+ MB
```

[7]:
```
# procurando valores nulos
tweets.isnull().sum()
```

[7]:
```
tweetID             0
entidade            0
sentimento          0
conteudo_tweet    686
dtype: int64
```

[8]:
```
tweets[tweets['conteudo_tweet'].isnull() == False]
```

[8]:

	tweetID	entidade	sentimento	\
0	2401	Borderlands	Positive	
1	2401	Borderlands	Positive	
2	2401	Borderlands	Positive	
3	2401	Borderlands	Positive	
4	2401	Borderlands	Positive	
..	…	…	…	
994	4891	GrandTheftAuto(GTA)	Irrelevant	
995	4359	CS-GO	Irrelevant	
996	2652	Borderlands	Positive	
997	8069	Microsoft	Positive	
998	6960	johnson&johnson	Neutral	

	conteudo_tweet
0	I am coming to the borders and I will kill you…
1	im getting on borderlands and i will kill you …
2	im coming on borderlands and i will murder you…
3	im getting on borderlands 2 and i will murder …
4	im getting into borderlands and i can murder y…
..	…
994	Toronto is the arts and culture capital of …
995	tHIS IS ACTUALLY A GOOD MOVE TOT BRING MORE VI…
996	Today sucked so it’s time to drink wine n play…
997	Bought a fraction of Microsoft today. Small wins.
998	Johnson & Johnson to stop selling talc baby po…

[74994 rows x 4 columns]

[9]:
```
tweets.duplicated().sum()
```

[9]:
```
3216
```

[10]:
```
# Removendo os valores nulos e duplicados e checando em seguida
tweets.dropna(inplace=True)
tweets.drop_duplicates(inplace=True)
print("valores nulos:\n", tweets.isnull().sum())
print("valores duplicados:", tweets.duplicated().sum())
```

```
valores nulos:
 tweetID           0
entidade           0
sentimento         0
conteudo_tweet     0
dtype: int64
valores duplicados: 0
```

[11]:
```
# Removendo colunas para simplificar a análise inicial dos dados
tts = tweets.drop(columns=['tweetID', 'conteudo_tweet'], inplace=False)
tts.head()
```

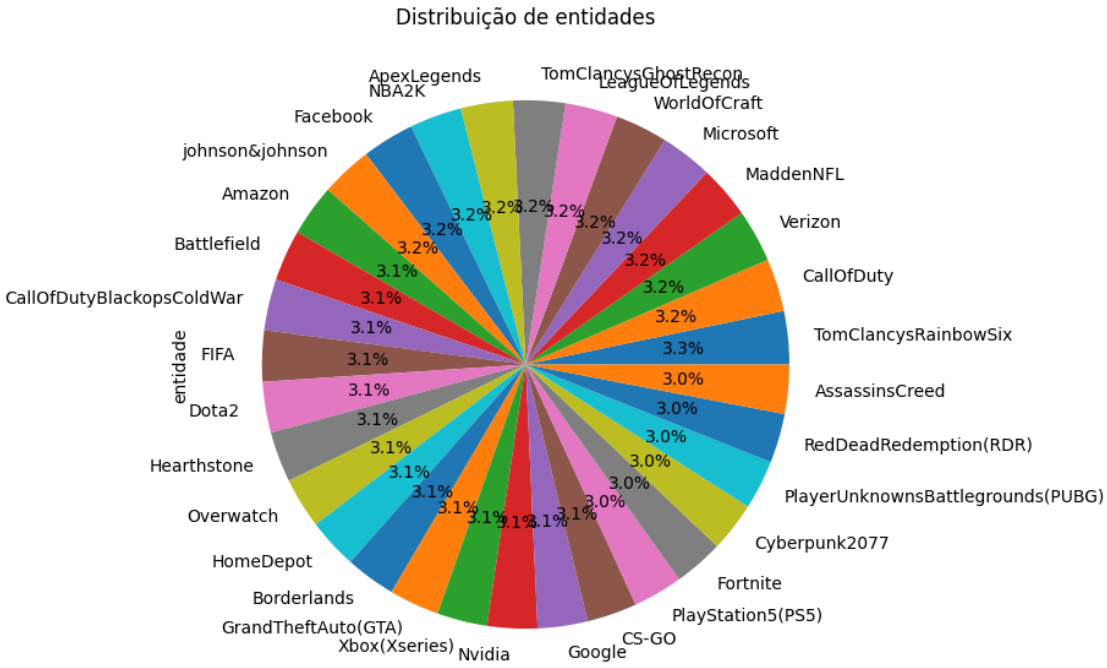
[11]:

	entidade	sentimento
0	Borderlands	Positive
1	Borderlands	Positive
2	Borderlands	Positive
3	Borderlands	Positive
4	Borderlands	Positive

[12]:
```
tts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 72138 entries, 0 to 995
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   entidade    72138 non-null  object
 1   sentimento  72138 non-null  object
dtypes: object(2)
memory usage: 1.7+ MB
```

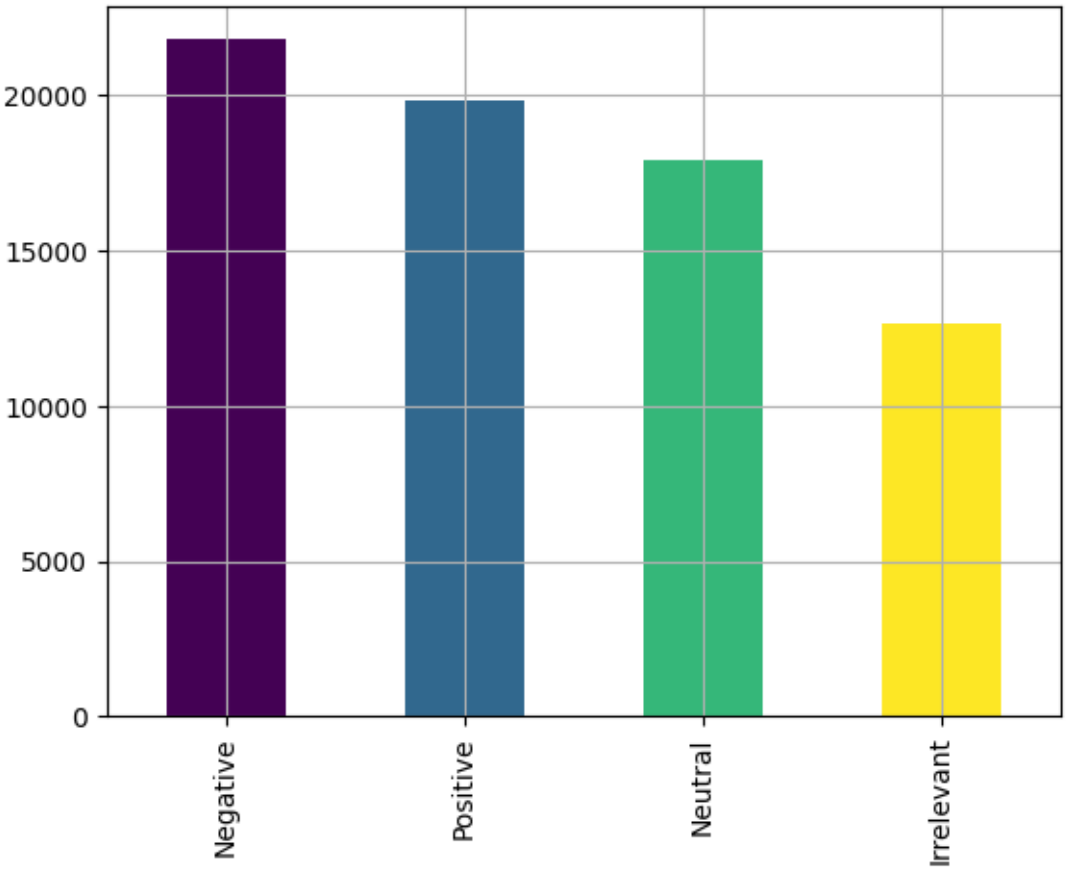
[13]:
```
conteudo_entidade = tts['entidade'].value_counts()
conteudo_entidade.plot(kind='pie', autopct='%1.1f%%', figsize=(7, 7))
plt.title('Distribuição de entidades')

plt.show()
```



Podemos perceber que os valores são bem próximos, ou seja, as entidades aparecem um número parecido de vezes nos tweets.

[14]:
```
conteudo_sentimentos = tts['sentimento'].value_counts()
color = plt.get_cmap('viridis')
colors = [color(i) for i in np.linspace(0, 1, len(conteudo_sentimentos))]
conteudo_sentimentos.plot(kind='bar', color=colors, grid=True)
```

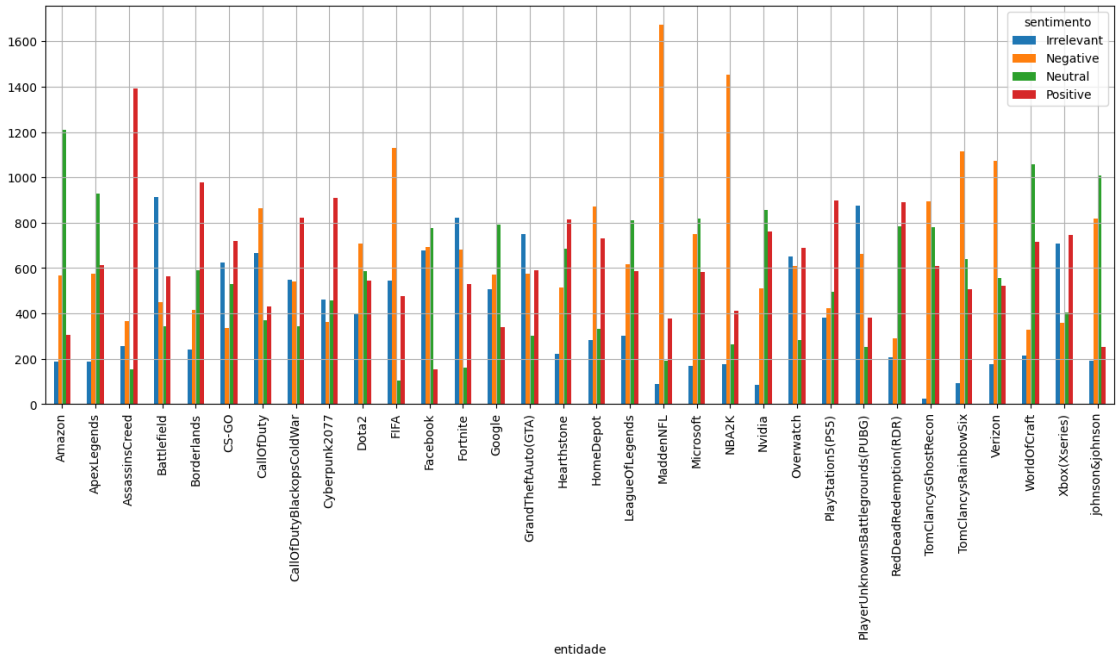
[14]: <Axes: >



A partir do gráfico anterior, podemos observar as quantias de cada classificação.

[15]:
```
reacoes_entidades = pd.crosstab(tts['entidade'], tts['sentimento'])
reacoes_entidades.plot(kind='bar', figsize=(16, 6), grid=True)
```

[15]:
```
<Axes: xlabel='entidade'>
```



Com esse gráfico que mescla todas as entidades e seus sentimentos, podemos notar os seguintes pontos: - a entidade MaddenNFL possui um número maior de reações negativas - a entidade Battlefield possui mais reações irrelevantes - a entidade Amazon possui mais reações neutras - e a entidade AssassinsCreed possui mais reações positivas

1.2 Projeção de sentimento usando Machine Learning

[16]:
```
tweets.head()
```

[16]:
```
   tweetID     entidade sentimento  \
0     2401  Borderlands   Positive
1     2401  Borderlands   Positive
2     2401  Borderlands   Positive
3     2401  Borderlands   Positive
4     2401  Borderlands   Positive

                                      conteudo_tweet
0  I am coming to the borders and I will kill you…
1  im getting on borderlands and i will kill you …
2  im coming on borderlands and i will murder you…
3  im getting on borderlands 2 and i will murder …
4  im getting into borderlands and i can murder y…
```

[17]:
```
dados = tweets
dados.shape
```

[17]:
```
(72138, 4)
```

[18]:
```
dados.dtypes
```

[18]:
```
tweetID            int64
entidade          object
sentimento        object
conteudo_tweet    object
dtype: object
```

[19]:
```
# Gráfico em pizza mostrando as porcentagens
fig = px.pie(dados['sentimento'].value_counts(),
             values='sentimento',
             names=dados['sentimento'].value_counts().index,
             title='Distribuição de emoções',
             hole=0.1)
fig.update_traces(textinfo='percent+label')
fig.update_layout(template='plotly_dark')

fig.show()
```

[20]:
```
ex_neg= dados.conteudo_tweet[50].lower()
ex_neg
```

[20]:
```
"@pubg_support hiya! so when'll u be fixing your f'n console game? been asking a
lot and have been very satisfied with your lack of responses  just wondering
if lagouts are a part of the game like red zones and i'm just not aware or if
you just can't fix a 2 year + problem. thx!"
```

[21]:
```
# Removendo caracteres especiais
ex = re.sub("[^a-zA-Z]",' ', ex_neg)
ex
```

[21]:
```
' pubg support hiya  so when ll u be fixing your f n console game  been asking a
lot and have been very satisfied with your lack of responses    just wondering
if lagouts are a part of the game like red zones and i m just not aware or if
you just can t fix a   year   problem  thx '
```

Agora é necessário remover palavras de pouco impacto para que o modelo consiga processar as palavras-chave com mais eficiência

[22]:
```
ex = ex.split()
ex
```

[22]:
```
['pubg',
 'support',
 'hiya',
```

```
'so',
'when',
'll',
'u',
'be',
'fixing',
'your',
'f',
'n',
'console',
'game',
'been',
'asking',
'a',
'lot',
'and',
'have',
'been',
'very',
'satisfied',
'with',
'your',
'lack',
'of',
'responses',
'just',
'wondering',
'if',
'lagouts',
'are',
'a',
'part',
'of',
'the',
'game',
'like',
'red',
'zones',
'and',
'i',
'm',
'just',
'not',
'aware',
'or',
'if',
'you',
```

```
 'just',
 'can',
 't',
 'fix',
 'a',
 'year',
 'problem',
 'thx']
```

[23]:
```
# Utilizando a biblioteca NLTK
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[23]:
```
True
```

[24]:
```
palavras = set(nltk.corpus.stopwords.words('english'))
ex = [palavra for palavra in ex if palavra not in palavras]
ex
```

[24]:
```
['pubg',
 'support',
 'hiya',
 'u',
 'fixing',
 'f',
 'n',
 'console',
 'game',
 'asking',
 'lot',
 'satisfied',
 'lack',
 'responses',
 'wondering',
 'lagouts',
 'part',
 'game',
 'like',
 'red',
 'zones',
 'aware',
 'fix',
 'year',
 'problem',
 'thx']
```

Passaremos essas informações para o dataframe

[25]:
```
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 72138 entries, 0 to 995
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   tweetID         72138 non-null  int64 
 1   entidade        72138 non-null  object
 2   sentimento      72138 non-null  object
 3   conteudo_tweet  72138 non-null  object
dtypes: int64(1), object(3)
memory usage: 4.8+ MB
```

[26]:
```
# dados['conteudo_tweet'][1].info()
# dados.loc[168]
```

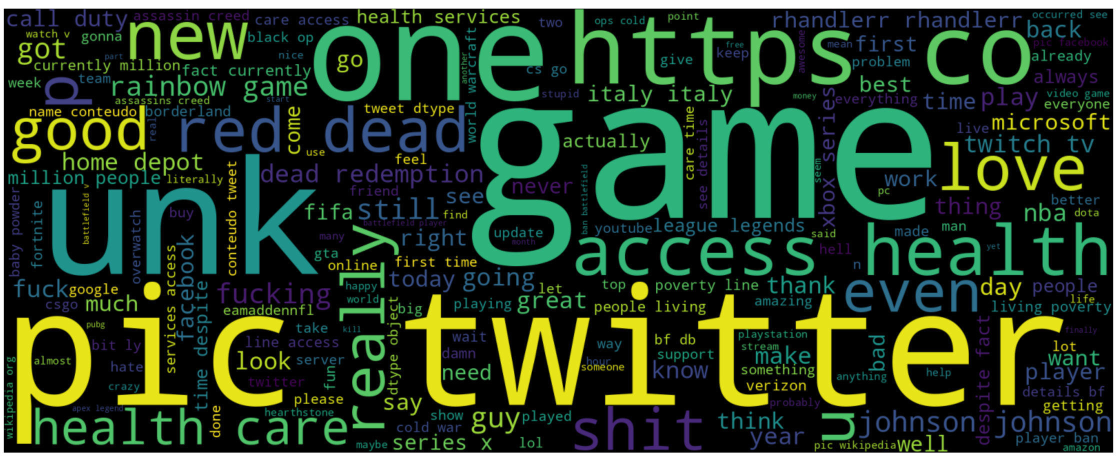
[27]:
```
def processo(tweet):
  # removendo pontuação e números do tweet
  tweet = re.sub("[^a-zA-Z]", ' ', tweet)
  # convertendo para minúsculas e dividindo para eliminar palavras irrelevantes
  tweet = tweet.lower()
  tweet = tweet.split()
  # removendo stopwords
  stop_words = set(nltk.corpus.stopwords.words('english'))
  tweet = [ccc for ccc in tweet if ccc not in stop_words]
  # juntando as palavras em um parágrafo e retornando
  return " ".join(tweet)
```

[28]:
```
train_data = []
for i in range(len(dados['conteudo_tweet'])):
  try:
    conteudo_tweet = dados.conteudo_tweet[i]
  except:
    i += 1
  if not isinstance(conteudo_tweet, str):
      conteudo_tweet = str(conteudo_tweet)
  if (i+1) % 10000 == 0:
    print('valor =', i+1)

  train_data.append(processo(conteudo_tweet))
```

```
valor = 10000
valor = 30000
valor = 40000
valor = 50000
```

```
valor = 60000
valor = 70000
```

[29]:
```python
texto = ' '.join(train_data)

# criando uma instância no WordCloud
wordcloud = WordCloud(width=2000, height=800,
                      background_color='black',
                      min_font_size=10).generate(texto)
# mostrando a imagem criada
plt.figure(figsize=(18, 6), facecolor=None)
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



[33]:
```python
text = ' '.join(train_data)

# frequência de cada palavra
freq_palavra = Counter(text.split())

# ordenando pelos maiores valores
palavras_ord = sorted(freq_palavra.items(), key=lambda x: x[1], reverse=True)

# palavras mais comuns e suas frequências
recorrentes = [par[0] for par in palavras_ord[:20]]
recorrentes_cont = [par[1] for par in palavras_ord[:20]]

# criando o gráfico
fig = px.bar(x=recorrentes,
             y=recorrentes_cont,
             text=recorrentes_cont,
```

```
            title='Palavras mais recorrentes',
            labels={'x': 'Palavras', 'y': 'Contagem'})
fig.update_layout(plot_bgcolor='gray', paper_bgcolor='gray', font_color='white')
fig.update_traces(marker_color='black')
fig.show()
```

1.2.1 Split dos dados

[36]:
```
y = train_data
x = np.array(dados['sentimento'])

train_x, test_x, y_train, y_test = train_test_split(y, x, stratify=x,␣
 ↪test_size=0.3, shuffle=True, random_state=1)
```

[39]:
```
vectorizer = CountVectorizer(max_features=1000)

train_x = vectorizer.fit_transform(train_x)
```

[40]:
```
train_x = train_x.toarray()
train_y = y_train
```

[41]:
```
print('Dados treinados totais: ', train_x.shape[0])
```

```
Dados treinados totais:  50496
```

1.2.2 Validando os modelos

Explicando cada modelo: - LogisticRegression: Regressão logística é um modelo estatístico que é usado para classificação binária. Ele modela a probabilidade de uma variável dependente com distribuição de Bernoulli (0 ou 1) em função de uma ou mais variáveis independentes. A saída é transformada usando a função logística (sigmoid), resultando em valores entre 0 e 1, que são interpretados como probabilidades de pertencer a uma das classes. É útil para casos onde a relação entre a variável dependente e as variáveis independentes é aproximadamente linear. - RidgeClassifier: O Ridge Classifier é uma variação do modelo de regressão linear regularizado, que usa a penalidade L2 para restringir a magnitude dos coeficientes do modelo, ajudando a evitar o sobreajuste (overfitting). Ele transforma a tarefa de classificação em um problema de regressão, mas, ao final, usa um limiar para classificar as instâncias em classes. É particularmente útil quando se espera multicolinearidade entre as características ou quando o número de características supera o número de observações. - RandomForestClassifier: O Classificador Random Forest é um modelo de ensemble que utiliza múltiplas árvores de decisão para fazer suas previsões, agregando os resultados (por exemplo, por votação majoritária para classificação). Ele introduz aleatoriedade adicional ao treinamento de árvores individuais, o que ajuda a reduzir o sobreajuste e geralmente resulta em um modelo mais robusto e preciso. É adequado para uma ampla gama de problemas de classificação e é conhecido por sua alta precisão e capacidade de lidar com variáveis categóricas e numéricas sem necessidade de escala.

- DecisionTreeClassifier: O Classificador de Árvore de Decisão utiliza uma estrutura de árvore onde cada nó representa uma característica, cada ramificação representa uma regra de decisão,

e cada folha representa um resultado (classe). As árvores de decisão são fáceis de interpretar e podem lidar com dados categóricos e numéricos, mas são propensas ao sobreajuste, especialmente com árvores muito profundas. Este modelo é útil para problemas de classificação e regressão e oferece uma representação visual clara de como as decisões são tomadas.

- O Classificador AdaBoost (Adaptive Boosting) é um modelo de ensemble que combina múltiplos classificadores fracos (tipicamente árvores de decisão de um único nível) para criar um classificador forte. O AdaBoost ajusta iterativamente os pesos das instâncias no conjunto de dados, dando mais peso às instâncias mal classificadas em iterações anteriores, e ajusta os pesos dos classificadores baseados em sua precisão. Finalmente, combina os classificadores fracos para formar um modelo mais robusto. É útil para aumentar a precisão de modelos simples e pode ser eficaz mesmo em casos onde os dados são um pouco ruidosos.

[49]:
```
models = []

models.append(('Logistic Regression', LogisticRegression()))
models.append(('Ridge Classifier', RidgeClassifier()))
models.append(('Random Forest Classifier', RandomForestClassifier()))
models.append(('Decision Tree Classifier', DecisionTreeClassifier()))
models.append(('Ada Boost Classifier', AdaBoostClassifier()))
```

[51]:
```
result_teste = vectorizer.transform(test_x)

result_teste = result_teste.toarray()

result_teste.shape
```

[51]:
```
(21642, 1000)
```

Explicando o trecho abaixo: - `for name, model in models:` Esta linha inicia um loop for que itera sobre a coleção models. Cada elemento em models é esperado ser uma tupla contendo dois elementos: o nome do modelo (name) e a instância do modelo (model).

- `model.fit(train_x, train_y)` Ajusta (treina) o modelo atual aos dados de treinamento representados por train_x (características/variáveis independentes) e train_y (variável dependente ou rótulo). Este passo é fundamental para que o modelo aprenda a relação entre os dados de entrada e a saída esperada.

- `test_pred = model.predict(result_teste)` Depois de treinar o modelo, esta linha usa o modelo treinado para fazer previsões sobre um novo conjunto de dados chamado result_teste. As previsões são armazenadas na variável test_pred.

- `print(name, 'acurácia: ', accuracy_score(y_test, test_pred))` Calcula a acurácia do modelo, que é a fração de previsões corretas entre todas as previsões feitas, usando os rótulos reais y_test e as previsões test_pred. Então, imprime o nome do modelo seguido pela acurácia calculada. A função accuracy_score é utilizada para esse cálculo.

- `print(name, 'precisão: ', precision_score(y_test, test_pred, average='weighted'))` Calcula a precisão do modelo, que é a fração de previsões corretas positivas em relação a todas as previsões positivas feitas pelo modelo, ponderada pelo número de casos em cada classe. Imprime o nome do modelo seguido pela precisão

calculada.

- `print(name, 'recall: ', recall_score(y_test, test_pred, average='weighted'))` Calcula o recall (sensibilidade) do modelo, que é a fração de verdadeiros positivos identificados corretamente pelo modelo, ponderada pelo número de casos em cada classe. Imprime o nome do modelo seguido pelo recall calculado.
- `print(name, 'F1 score: ', f1_score(y_test, test_pred, average='weighted'))` Calcula o F1 score do modelo, que é a média harmônica da precisão e recall, oferecendo um balanço entre essas duas métricas, ponderada pelo número de casos em cada classe. Imprime o nome do modelo seguido pelo F1 score calculado.

[54]:
```
for name, model in models:
  print('-----------------------------------------------------------------------')
  model.fit(train_x, train_y)
  test_pred = model.predict(result_teste)
  print(name, 'acurácia: ', accuracy_score(y_test, test_pred))
  print(name, 'precisão: ', precision_score(y_test, test_pred,␣
 ↪average='weighted'))
  print(name, 'recall: ', recall_score(y_test, test_pred, average='weighted'))
  print(name, 'F1 score: ', f1_score(y_test, test_pred, average='weighted'))
```

```
-----------------------------------------------------------------------

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression


Logistic Regression acurácia:  0.3715460678310692
Logistic Regression precisão:  0.3690257455516584
Logistic Regression recall:  0.3715460678310692
Logistic Regression F1 score:  0.36037829128220106
-----------------------------------------------------------------------
Ridge Classifier acurácia:  0.3714074484798078
Ridge Classifier precisão:  0.37052102030116374
Ridge Classifier recall:  0.3714074484798078
Ridge Classifier F1 score:  0.3565412385093654
-----------------------------------------------------------------------
Random Forest Classifier acurácia:  0.6096017003973755
Random Forest Classifier precisão:  0.6118742775689299
Random Forest Classifier recall:  0.6096017003973755
```

```
Random Forest Classifier F1 score:  0.6091240875921546
----------------------------------------------------------------------
Decision Tree Classifier acurácia:  0.5471767858793087
Decision Tree Classifier precisão:  0.5480359956553726
Decision Tree Classifier recall:  0.5471767858793087
Decision Tree Classifier F1 score:  0.5464443776734149
----------------------------------------------------------------------
Ada Boost Classifier acurácia:  0.3407725718510304
Ada Boost Classifier precisão:  0.35695499924380714
Ada Boost Classifier recall:  0.3407725718510304
Ada Boost Classifier F1 score:  0.2890884234638743
```

[82]:
```
# Substituindo CountVectorizer por TfidfVectorizer para melhorar a precisão dos␣
 ↪modelos

x = dados['conteudo_tweet']
y = dados['sentimento']

train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.3,␣
 ↪stratify=y, random_state=1)

vectorizer = TfidfVectorizer(max_features=1000)
train_x = vectorizer.fit_transform(train_x)
test_x = vectorizer.transform(test_x)

train_x = train_x.toarray()
test_x = test_x.toarray()

print(train_x.shape)
print(len(train_y))
```

```
(50496, 1000)
50496
```

[83]:
```
models = []
models.append(('Random Forest Classifier',␣
 ↪RandomForestClassifier(n_estimators=100, max_depth=10, random_state=1)))
models.append(('Decision Tree Classifier', DecisionTreeClassifier(max_depth=5,␣
 ↪random_state=1)))
```

[77]:
```
models.append(('Logistic Regression', LogisticRegression(C=0.5)))
models.append(('Ridge Classifier', RidgeClassifier(alpha=1.0)))
```

[78]:
```
# from sklearn.model_selection import GridSearchCV

# # Definindo o modelo e os hiperparâmetros para testar
# model = RandomForestClassifier(random_state=1)
```

```
# param_grid = {
#     'n_estimators': [100, 200],
#     'max_depth': [10, 20, 30],
# }

# # Configurando GridSearchCV
# grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
 ↪scoring='accuracy')

# grid_search.fit(train_x, train_y)

# # Melhor modelo
# best_model = grid_search.best_estimator_

# test_pred = best_model.predict(result_teste)

# print("Melhores parâmetros: ", grid_search.best_params_)
```

O código anteriror encontrou `Melhores parâmetros: max_depth: 30, n_estimators: 200`.

[79]:
```
optimized_model = RandomForestClassifier(max_depth=30, n_estimators=200,␣
 ↪random_state=1)

optimized_model.fit(train_x, train_y)
```

[79]:
```
RandomForestClassifier(max_depth=30, n_estimators=200, random_state=1)
```

[81]:
```
test_pred = optimized_model.predict(test_x)

# Calculando e imprimindo as métricas de avaliação
print('Acurácia: ', accuracy_score(y_test, test_pred))
print('Precisão: ', precision_score(y_test, test_pred, average='weighted'))
print('Recall: ', recall_score(y_test, test_pred, average='weighted'))
print('F1 score: ', f1_score(y_test, test_pred, average='weighted'))
```

```
Acurácia:  0.6357545513353664
Precisão:  0.7069057253554207
Recall:  0.6357545513353664
F1 score:  0.6087514624675281
```

Após o refatoramento do código, obtivemos uma melhoria significativa na qualidade do modelo. Os valores anteriores indicam a eficiência do modelo em prever o sentimento de futuros conteúdos de tweets caso seja alimentado com tal base de dados adicional.