

Classificação de Textos com Algoritmos Tradicionais e BoW Features

▼ Ambiente

```
!pip install unicode
```

```
Collecting unicode
  Downloading Unicode-1.3.8-py3-none-any.whl.metadata (13 kB)
  Downloading Unicode-1.3.8-py3-none-any.whl (235 kB)
    235.5/235.5 kB 5.0 MB/s eta 0:00:00
Installing collected packages: unicode
Successfully installed unicode-1.3.8
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
from unicode import unicode
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import svm, naive_bayes
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

▼ Dataset AmericanasBR

https://github.com/americanas-tech/b2w-reviews01/blob/main/b2wreviews01_stil2019.pdf

O dataset original foi simplificado, o campo 'overall_rating' foi usado como 'label':

- 4 e 5 representam positivo (label=1)
- 1 e 2 representam negativo (label=0)
- rating 3 não foi usado

O dataset resultante originou os dois conjuntos de dados:

- Treino: usando 5 mil instâncias de cada classe (positivo e negativo).
- Teste: usando 3 mil instâncias de cada classe excluindo-se instâncias de treino

Treino e Teste são disjuntos.

```
#baixando os dados de treino e teste
!curl https://www.inf.ufrgs.br/~viviane/DS/B2W-Reviews01_binario5000_TRAIN.csv > B2W-Reviews01_binario5000_TRAIN.csv
!curl https://www.inf.ufrgs.br/~viviane/DS/B2W-Reviews01_binario_TEST.csv > B2W-Reviews01_binario_TEST.csv
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
             Dload  Upload   Total       Spent    Left  Speed
100 1657k    100 1657k    0     0   710k      0  0:00:02  0:00:02 --:--:--  710k
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
             Dload  Upload   Total       Spent    Left  Speed
100  981k    100  981k    0     0   594k      0  0:00:01  0:00:01 --:--:--  594k
```

```
df_train = pd.read_csv('B2W-Reviews01_binario5000_TRAIN.csv')
df_test = pd.read_csv('B2W-Reviews01_binario_TEST.csv')
```

▼ Explorando o dataset

```
df_train.sample(n=5)
```

	label	text	label_descr
4789	0	tampo de vidro com medida quadrada, e a mesa é...	negativo
8069	1	Conforme a utilização do produto vou poder ter...	positivo
1097	0	Comprei em Dezembro e até hoje Não recebi. O l...	negativo
761	0	Entrega super atrasada, paguei um valor a mais...	negativo
4875	0	O produto diz ser compatível com iOS mas quand...	negativo

```
#está balanceado e possui duas classes:
df_train.groupby('label_descr').count()
```

	label	text
label_descr		
negativo	5000	5000
positivo	5000	5000

```
df_test.sample(n=5)
```

	label	text	label_descr
1541	1	Atende todas as expectativas, realmente dá vol...	positivo
418	1	Atende as minhas necessidades. Da para baixar ...	positivo
2564	1	Produto muito bom. Superou minhas expectativas...	positivo
2497	1	Linda tv atendeu todas minhas expectativas, mo...	positivo
3584	0	Recebi o produto aberto e faltando duas peças,...	negativo

```
#dataset de teste também está balanceado e possui duas classes:
df_test.groupby('label_descr').count()
```

	label	text
label_descr		
negativo	3000	3000
positivo	3000	3000

✓ Pré-processando

```
#função de pré-processamento
special_chars = "'!#$%&()*+,-./:;<=>@[\\]^_`{|}~"
stop_words = stopwords.words('portuguese')

stop_words.remove('não') # mantém o não na lista de stopwords

def preprocess(x):
    new_x = x.replace(' ', ' ')
    for c in special_chars:
        new_x = new_x.replace(c, ' ')
    new_x = ' '.join([word for word in nltk.word_tokenize(new_x.lower(), language='portuguese') if word not in stop_words]) #removendo s
    new_x = re.sub(r'[\W\s]', ' ', new_x) #removendo pontuação do texto
    new_x = re.sub("http\S+", ' ', new_x) # remove links
    new_x = re.sub("@\w+", ' ', new_x) # remove contas com @
    new_x = re.sub('#\S+', ' ', new_x) # hashtags
    new_x = re.sub('[0-9]+', ' ', new_x) # remove numeros e palavras com numeros
    new_x = unicode(new_x) #acentos
    new_x = re.sub("\s+", ' ', new_x) # espaços
    new_x = new_x.strip()
    return new_x

#pré-processar datasets de treino e teste
df_train['text_original'] = df_train['text']
df_train['text'] = df_train['text'].apply(preprocess)

df_test['text_original'] = df_test['text']
df_test['text'] = df_test['text'].apply(preprocess)
```

```
df_train = df_train.sample(n=len(df_train)).copy() # embaralha treino
df_train.reset_index(drop=True, inplace=True)
len(df_train)
```

↗ 10000

```
# remove instâncias com texto com comprimento zero
df_train = df_train[df_train['text']!='']
df_train = df_train[~df_train['text'].isna()]
df_train.reset_index(drop=True, inplace=True) # reindexa dataframe
len(df_train)
```

↗ 9996

✓ Tarefa 1

- 1. Verificação de Duplicidades Internas:** Verifique se existem sentenças duplicadas dentro dos conjuntos de treino e teste separadamente.
- 2. Verificação de Duplicidades Entre Conjuntos:** Verifique se há sentenças duplicadas presentes tanto no conjunto de treino quanto no de teste.
- 3. Remoção de Duplicidades:** Caso sejam encontradas duplicidades, remova-as tanto dentro de cada conjunto quanto entre os conjuntos de treino e teste.
- 4. Análise de Impacto:** Apresente o impacto das remoções nos conjuntos de dados, destacando as alterações no tamanho e na distribuição das sentenças.

```
aux_train = df_train.copy()
aux_test = df_test.copy()
tamanho_treino = int(len(aux_train))
tamanho_teste = int(len(aux_test))
print(tamanho_teste, tamanho_treino)
df_train = df_train.drop_duplicates(subset=['text'], keep='first')
df_test = df_test.drop_duplicates(subset=['text'], keep='first')

print(f'Duplicadas entre df_train {tamanho_treino - int(len(df_train))}')
print(f'Duplicadas entre df_test {tamanho_teste - int(len(df_test))}')
```

```
# Remove duplicatas entre si, mas deixa 1 referencia no conjunto de teste
duplicados = df_test[df_test['text'].isin(df_train['text'])]
df_test = df_test[~df_test['text'].isin(df_train['text'])]
```

↗ 6000 9996
Duplicadas entre df_train 72
Duplicadas entre df_test 35

Se não removemos duplicadas tanto dos conjuntos de treino e teste quando entre os dois conjuntos, podemos ter um treinamento ruim pois ele verá dados repetidos e no treinamento podemos ter uma super valorização do nosso modelo já que ele vai acertar dados repetidos no conjunto de teste que possui referencia do conjunto de teste.

✓ Gerando Representação BoW com pesos TFIDF

Mais informações em https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

# calcula features e os valores tfidf gerando os vetores dos documentos:
tfidf_vectorizer = TfidfVectorizer()

vetores_docs = tfidf_vectorizer.fit_transform(df_train['text'].values)
features = tfidf_vectorizer.get_feature_names_out()
print(features[1:100])
vetores_docs.shape, features.shape
```

↗ ['aaa' 'aaaaaak' 'aaprelho' 'aaptador' 'abacaxi' 'abafado' 'abaixa'
'abaixando' 'abaixar' 'abaixo' 'abaixou' 'abajur' 'abalada' 'abalar'
'abandonei' 'abandonou' 'abc' 'abdomem' 'abelhas' 'abencoe' 'aberta']

```
'abertas' 'aberto' 'abertos' 'abertura' 'abhp' 'abi' 'abisurdo' 'abnt'
'abolir' 'aborrecer' 'aborreceu' 'aborrecida' 'aborrecido'
'aborrecimento' 'abr' 'abra' 'abraca' 'abracadabra' 'abracadeira'
'abraco' 'abracos' 'abram' 'abrange' 'abre' 'abrem' 'abreu' 'abreviado'
'abri' 'abria' 'abrigada' 'abril' 'abrimos' 'abrindo' 'abrir' 'abriram'
'abrimos' 'abriu' 'abro' 'abs' 'absolut' 'absolutamente' 'absorcao'
'absorto' 'absorve' 'absorvendo' 'absorvicao' 'absurda' 'absurdamente'
'absurdo' 'absurdoooo' 'absurdooooo' 'absurdos' 'abusivo' 'abuso'
'abusurdo' 'acaba' 'acabada' 'acabadas' 'acabado' 'acabam' 'acabamente'
'acabamento' 'acabamentos' 'acabamos' 'acabando' 'acabar' 'acabaram'
'acabe' 'acabei' 'acabem' 'acabo' 'acabou' 'academia' 'academico'
'academicos' 'acampamento' 'acampamentos' 'acampar']
((9924, 13286), (13286,))
```

```
# # opções de préprocessamento que sobrescrevem funcoes do tfidftokenizer:
```

```
# tfidf_vectorizer = TfidfVectorizer(input='filename', max_features=200,
#                                   token_pattern='(?u)\b[a-zA-Z]\w{2,}\b',
#                                   max_df=0.05,
#                                   stop_words='english',
#                                   ngram_range=(1, 3))
```

```
# def meu_preprocessamento(doc):
#     # tokeniza com algum tokenizador
#     # adiciona funcoes de préprocessamento
#     return doc
#
# tfidf = TfidfVectorizer(
#     analyzer='word',
#     tokenizer=meu_tokenizador,
#     preprocessor=meu_tokenizador,
#     token_pattern=None)
```

▼ Analisando os vetores gerados

```
doc_id = 0
vetores_docs[doc_id]
```

```
↳ <Compressed Sparse Row sparse matrix of dtype 'float64'
   with 9 stored elements and shape (1, 13286)>
```

```
#features e valores tfidf pertencentes ao 1o documento (indice 0)
print(vetores_docs[doc_id])
```

```
↳ <Compressed Sparse Row sparse matrix of dtype 'float64'
   with 9 stored elements and shape (1, 13286)>
  Coords      Values
(0, 12934)    0.4200425978327417
(0, 8566)     0.12381730728034006
(0, 2787)     0.21547387886718128
(0, 8221)     0.36438155720997806
(0, 9566)     0.4200425978327417
(0, 12732)    0.4200425978327417
(0, 4334)     0.3889520221383027
(0, 1746)     0.20535097108781147
(0, 9976)     0.2875870544280312
```

```
# outra forma de lidar com esses vetores comprimidos é expandir os mesmos
# verificando o documento (d1)
print('Documento1: ', df_train.at[doc_id, 'text'])
d1 = vetores_docs[doc_id].toarray()[0]
print('tamanho do vetor d1:', len(d1))
print(d1)
```

```
↳ Documento1:  verdade nao comprei mim pessoa usando diz bom preco
tamanho do vetor d1: 13286
[0. 0. 0. ... 0. 0. 0.]
```

```
# no vetor expandido do documento d1 as features que realmente existem no vetor são:
ids = np.argwhere(d1>0)
ids = np.transpose(ids)[0]
print(ids)
```

```
↳ [ 1746  2787  4334  8221  8566  9566  9976 12732 12934]
```

```
d1[ids[0]], features[ids[0]]
```

```
(np.float64(0.20535097108781147), 'bom')

print('Documento d1: ',df_train.at[doc_id,'text'])
print('>>> as features que existem em d1 são:')
print(features[ids])
print('>>> tfidf das features de d1 são: ')
print(d1[ids])

Documento d1:  verdade nao comprei mim pessoa usando diz bom preco
>>> as features que existem em d1 são:
['bom' 'comprei' 'diz' 'mim' 'nao' 'pessoa' 'preco' 'usando' 'verdade']
>>> tfidf das features de d1 são:
[0.20535097 0.21547388 0.38895202 0.36438156 0.12381731 0.4200426
 0.28758705 0.4200426 0.4200426 ]

# podemos expandir toda a matriz tfidf:
m = vetores_docs.toarray()
print(f'Matriz tfidf (linhas=docs, colunas=features): {m.shape}')
# visualizando a matriz esparsa tfidf: LINHAS = DOCUMENTOS, COLUNAS = FEATURES = TOKENS
dfM = pd.DataFrame(m)
dfM

Matriz tfidf (linhas=docs, colunas=features): (9924, 13286)
   0  1  2  3  4  5  6  7  8  9  ...  13276  13277  13278  13279  13280  13281  13282  13283  13284  13285
0  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
9919 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
9920 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
9921 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
9922 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
9923 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

9924 rows x 13286 columns

# imprimindo as palavras e seus índices
#for w in dict(enumerate(features)):
#    print(w, dict(enumerate(features))[w])

# obtendo o token a partir do id (nr da coluna)
id_palavra = 7178 #palavra "inutil" 7178
featuresId=dict(enumerate(features))
featuresId[id_palavra]

'inutil'

# Em quantos documentos (document frequency) esse token está presente e qual o valor do score
print(f"O token <{features[id_palavra]}> aparece em {len(dfM[dfM[id_palavra]>0][id_palavra])} documento(s):")
dfM[dfM[id_palavra]>0][id_palavra]

O token <inutil> aparece em 6 documento(s):

7178
2618 0.218399
3047 0.239357
5750 0.235929
6418 0.284066
8133 0.526516
9859 0.310409

dtype: float64
```

Visualizando TFIDF

```
vetores_docs.shape, tfidf_vectorizer.idf_.shape

((9924, 13286), (13286,))

idf_scores = tfidf_vectorizer.idf_

dfidf = pd.DataFrame({'idf':idf_scores,'words':tfidf_vectorizer.get_feature_names_out()})

dfidf.sort_values(by=['idf'],ascending=False, inplace=True)

# importância das palavras no corpus
dfidf
```

	idf	words
5	9.509665	abacaxi
13277	9.509665	zeus
13278	9.509665	zinco
13280	9.509665	zona
13282	9.509665	zumbi
...
1746	3.028854	bom
4900	2.832581	entrega
10731	2.787637	recomendo
8566	1.826261	nao
10175	1.750264	produto

13286 rows × 2 columns

Próximas etapas:

Gerar código com dfidf

☒ Ver gráficos recomendados

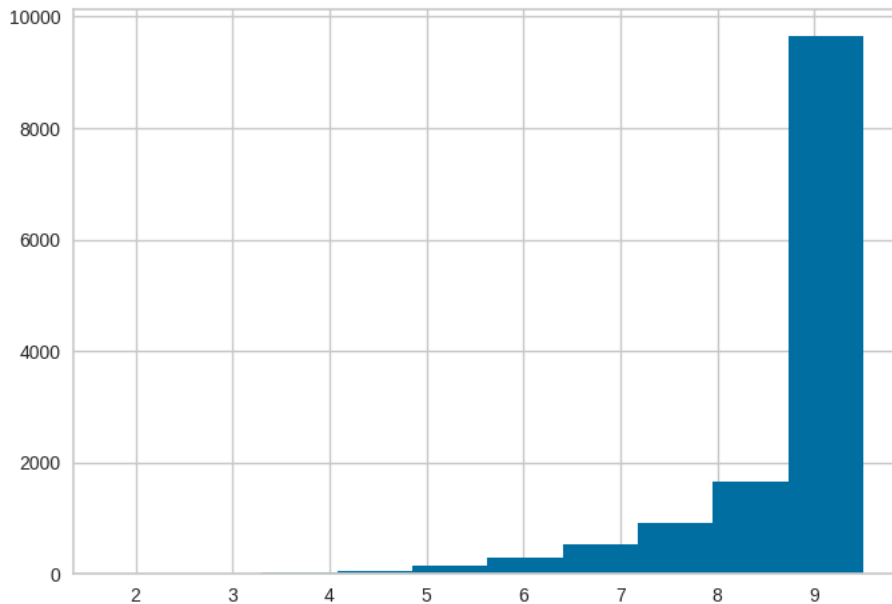
New interactive sheet

```
# valores de idf e quantidade de palavras
dfidf.groupby(['idf']).count().sort_values(by=['idf'],ascending=False).head(10)
```

	words
idf	
9.509665	6922
9.104200	1879
8.816518	851
8.593374	586
8.411053	374
8.256902	262
8.123371	248
8.005588	178
7.900227	160
7.804917	133

```
dfidf['idf'].hist()
```

<Axes: >



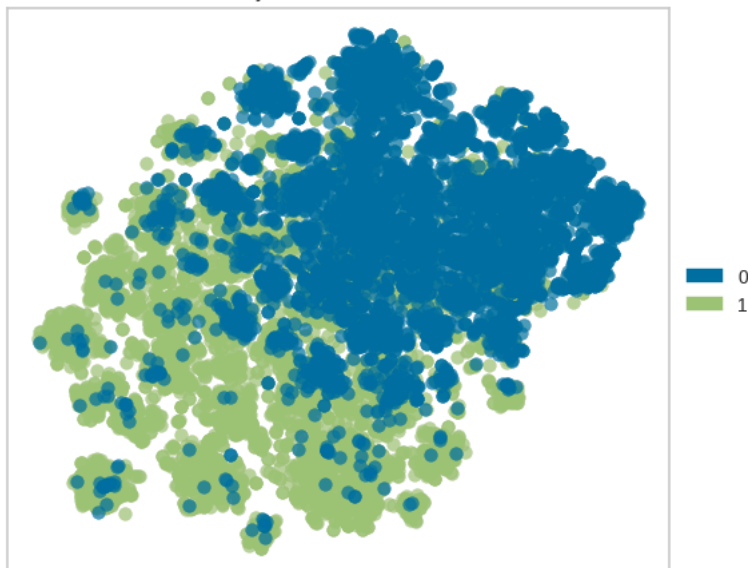
```
%%time
#visualizando espacialmente os vetores tfidf reduzidos a duas dimensoes (demora para executar):
from yellowbrick.text import TSNEVisualizer

tfidf = TfidfVectorizer()
X = tfidf.fit_transform(df_train['text'].values)
y = df_train['label'].values

# Create the visualizer and draw the vectors
tsne = TSNEVisualizer(random_state=42) # random_state para reprodutibilidade das projeções em 2D
tsne.fit(X, y)
tsne.show()
```

/usr/local/lib/python3.11/dist-packages/yellowbrick/text/tsne.py:401: UserWarning: *c* argument looks like a single numeric RGB or F
self.ax.scatter()


TSNE Projection of 9924 Documents




CPU times: user 2min 23s, sys: 4.83 s, total: 2min 28s
Wall time: 2min 39s
<Axes: title={'center': 'TSNE Projection of 9924 Documents'}>

✓ Treinando Modelo de Classificação

```
df_train.groupby(['label_descr', 'label']).count()
```




		text	text_original	
label_descr	label			
negativo	0	4944	4944	
positivo	1	4980	4980	



```
# Gerando a representação vetorial para os textos da base de treino
vectorizer = TfidfVectorizer()
#fit_transform ajusta o vetorizador tfidf à base de treino e também transforma o texto em X
X = vectorizer.fit_transform(list(df_train['text']))
y = np.array(df_train['label'])
```

```
#efetuar o treinamento usando parâmetros predeterminados
#clf = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
#efetuar o treinamento usando parâmetros default
clf = svm.SVC()
clf.fit(X, y)
```



▼ SVC ⓘ ?

SVC()

```
# efetuar o treinamento fazendo refit com a melhor configuração:
#model = svm.SVC()
#parameters = {'kernel':['linear','rbf'], 'C':[1, 5]} # neste caso estaremos variando o parâmetro 'C' e 'Kernel' do algoritmo svm
#clf = GridSearchCV(estimator=model, param_grid=parameters, cv=5, verbose=4, scoring=('accuracy','f1_macro'), refit='accuracy')
#clf.fit(X, y)
#print(f"Para {model} melhor score {clf.best_score_:.3f} para os seguintes parâmetros: {clf.best_params_}")
```

▼ Prevendo a classe das instâncias de teste

```
# Gerando a representação vetorial para os textos da base de teste
X_true = vectorizer.transform(df_test['text'].values) #somente transform usando vocabulario do treino
y_true = df_test['label'].values
# gera as predições para os dados de teste:
y_pred = clf.predict(X_true)
```

```
y_pred[0:15]
```

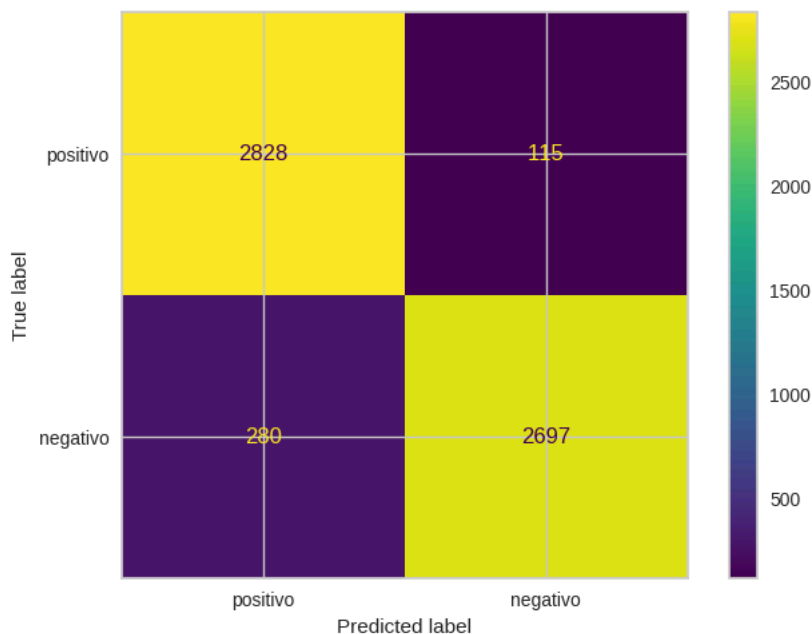


```
array([1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1])
```

▼ Avaliando a Qualidade do Modelo

```
print(f"Acurácia: {accuracy_score(y_true, y_pred):.4f}")
print(f"F1-macro: {f1_score(y_true, y_pred, average='macro'):.4f}")
cm = confusion_matrix(y_true, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = list(df_train.label_descr.unique()))
cm_display.plot()
plt.show()
```


↗ Acurácia: 0.9333
F1-macro: 0.9332



✓ Tarefa 2

- Treinamento com Naive Bayes:** Realize o treinamento do corpus utilizando o algoritmo Naive Bayes.
- Comparação com SVM:** Compare os resultados obtidos com o classificador SVM treinado no mesmo corpus.
- Análise de Desempenho:** Avalie as diferenças de desempenho entre os dois classificadores com base nas seguintes análises:
 - Matriz de confusão
 - Métricas de F1-score
 - Acurácia
- Discussão das Diferenças:** Identifique e discuta as principais diferenças observadas nos resultados.

```
import pandas as pd
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report

x_train = df_train['text']
y_train = df_train['label']
x_test = df_test['text']
y_test = df_test['label']

vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(x_train)
X_test_vec = vectorizer.transform(x_test)

model = MultinomialNB(alpha=1.0)
model.fit(X_train_vec, y_train)
y_predict = model.predict(X_test_vec)

df_results = pd.DataFrame({
    'text': x_test,
    'true_label': y_test,
    'predicted_label': y_predict
})

print(classification_report(y_test, y_predict))

df_errors = df_results[df_results['true_label'] != df_results['predicted_label']]

df_errors = df_errors.reset_index(drop=True)

# Mostrar os primeiros erros
print(f"Total de erros: {len(df_errors)}")
print(df_errors.head())
```

↗

	precision	recall	f1-score	support
0	0.90	0.93	0.92	2943

	1	0.93	0.90	0.91	2977
accuracy				0.92	5920
macro avg	0.92	0.92	0.92	0.92	5920
weighted avg	0.92	0.92	0.92	0.92	5920

Total de erros: 498

	text	true_label	\
0	comprei entrega jato entregar dia util comprei...	1	
1	recomendo botao agua temperatura ambiente fres...	1	
2	melhor tons po mandam avaliar criam dificuldade...	1	
3	nao liguem valor material outro mundo	1	
4	desses filmes assistimos vez percebemos novos ...	1	

	predicted_label
0	0
1	0
2	0
3	0
4	0

O F1 score e a acurácia foi um pouco menor do que a SVM

✓ Examinando os as instâncias mal classificadas

```

erros = list(zip(df_test['text'].values,df_test['text_original'].values,y_true,y_pred)) #criando lista com os erros
erros = [item for item in erros if item[2] != item[3]] #removendo as instâncias corretas
df_erros = pd.DataFrame(erros,columns =['Texto','Original','True','Pred']) #gerando um dataframe para ficar mais fácil de trabalhar

```

```

#acrescentando colunas FP e FN no dataframe com os erros
df_erros['FP'] = df_erros.apply(lambda x: 1 if ((x['Pred']==1) & (x['True']==0)) else 0, axis=1)
df_erros['FN'] = df_erros.apply(lambda x: 1 if ((x['Pred']==0) & (x['True']==1)) else 0, axis=1)

```

```
print('Há ', len(df_erros),' instâncias mal classificadas.')
```

➡ Há 498 instâncias mal classificadas.

```
df_erros[df_erros['FN']==1]
```

➡

	Texto	Original	True	Pred	FP	FN
0	comprei entrega jato entregar dia util comprei...	Comprei entrega a jato para entregar em 1 dia ...	1	0	0	1
1	recomendo botao agua temperatura ambiente fres...	Recomendo! Botão água com temperatura ambiente...	1	0	0	1
2	melhor tons po mandam avaliar criam dificuldade...	melhor que 50 tons , pô vocês mandam avaliar m...	1	0	0	1
3	nao liguem valor material outro mundo	Não liguem para o valor, o material é de outro...	1	0	0	1
4	desses filmes assistimos vez percebemos novos ...	É um desses filmes que quando o assistimos mai...	1	0	0	1
...
299	bom pulgas saem mesma hora algum tempo ainda p...	bom, as pulgas saem na mesma hora, mas depois ...	1	0	0	1
300	bom material acabamento deveriam informar melh...	bom material e acabamento. só deveriam informa...	1	0	0	1
301	bicicleta boa porem nao pega cambio traseiro r...	A bicicleta é muito boa porém não pega câmbio ...	1	0	0	1
302	melhor pc ever compreee oque ta esperando vai ...	melhor pc ever compreee.Oque se ta esperando v...	1	0	0	1
303	produto qualidade bom porem atendimento devolu...	O produto é de qualidade e muito bom , porém o...	1	0	0	1

304 rows × 6 columns

```
df_erros[df_erros['FP']==1]
```

	Texto	Original	True	Pred	FP	FN
304	chapinha nao boa progressiva nao chega graus p...	A chapinha não é boa para progressiva,não cheg...	0	1	1	0
305	atencao professor procura livro ajude pratica ...	Atenção professor, se estiver à procura de um ...	0	1	1	0
306	imagem ilustrativa nao condiz produto real qua...	A imagem ilustrativa não condiz com o produto ...	0	1	1	0
307	chapinha nao alisa cabelo conforme diz anuncio	Chapinha não alisa o cabelo conforme diz o anú...	0	1	1	0
308	americanas cumpriu papel divulgacao entrega pr...	A americanas cumpriu seu papel de divulgação/e...	0	1	1	0
...
493	nao aquece suficiente alisar perfeitamente inf...	Ele não aquece o suficiente para alisar perfei...	0	1	1	0
494	prnsei vidro qualidade bem ruim quer algo simp...	Prnsei que fosse de vidro e qualidade bem ruim...	0	1	1	0
495	nao aprovei produto questoes qualidade cor ach...	Não aprovei o produto pelas questões da qualid...	0	1	1	0
496	tamanho g segundo vendedor indicado crianas a...	O tamanho G, segundo o vendedor , é indicado ...	0	1	1	0
497	bordados descritos produto nao bordados colcha...	Os bordados descritos no produto não são os bo...	0	1	1	0

194 rows × 6 columns

```
#salvando csv com as instâncias mal classificadas
from google.colab import drive
drive.mount('/content/drive')
path = '/content/drive/MyDrive/Colab Notebooks/erros_americanas.csv'
df_erros.to_csv(path)
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

✓ Tarefa 3

- Análise de Erros do Naive Bayes:** Realize uma análise detalhada dos erros cometidos pelo classificador Naive Bayes, seguindo o mesmo procedimento adotado anteriormente para o SVM.
- Verificação de Interseção de Erros:** Verifique se há interseção entre as classificações incorretas de ambos os classificadores (Naive Bayes e SVM), use o diagram de Venn como apoio visual. Identifique quais frases foram classificadas de forma errada por ambos.
- Comparação de Resultados:** Analise se há alguma característica distinta nos erros de cada classificador. Discuta se o tipo de erro cometido pelo Naive Bayes difere dos erros cometidos pelo SVM e, se sim, explore as possíveis causas dessas diferenças.

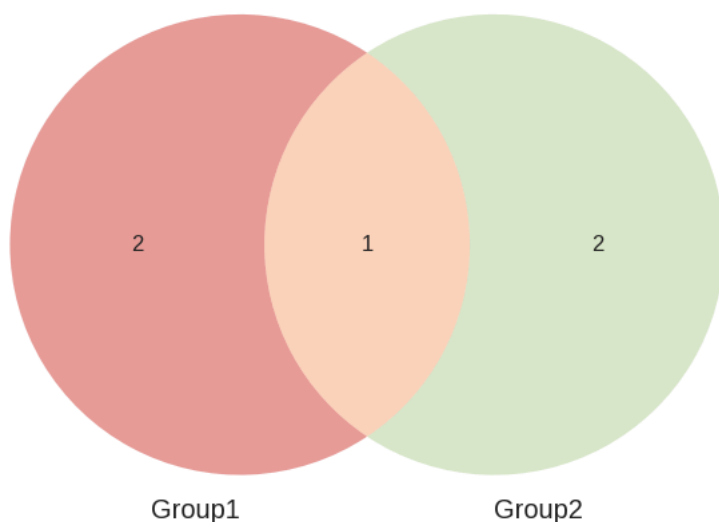
Exemplo de código para a criação do diagram de Venn

```
import matplotlib.pyplot as plt
from matplotlib_venn import venn2

set1 = set(['A', 'B', 'C'])
set2 = set(['A', 'E', 'F'])

venn2([set1, set2], ('Group1', 'Group2'))

plt.show()
```



```

from matplotlib_venn import venn2
import matplotlib.pyplot as plt

print('Há', len(df_errors), 'instâncias mal classificadas.')

# Criando conjuntos de erros formatados
erros_set1 = set(
    f"Texto: {row['Texto']}, True: {row['True']}, Pred: {row['Pred']}"
    for _, row in df_erros.iterrows()
)

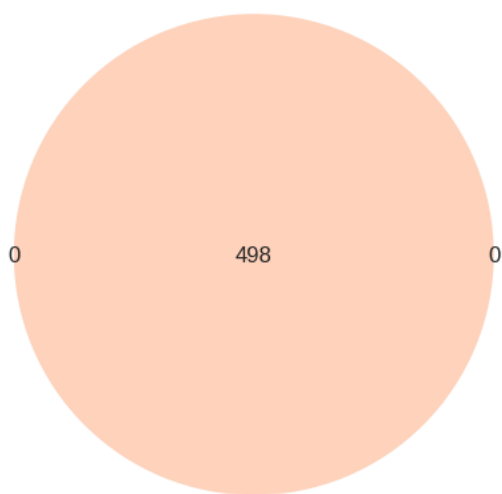
erros_set2 = set(
    f"Texto: {row['text']}, True: {row['true_label']}, Pred: {row['predicted_label']}"
    for _, row in df_errors.iterrows()
)

# Plotar diagrama de Venn
venn2([erros_set1, erros_set2], ('Erros Modelo 1', 'Erros Modelo 2'))
plt.title("Comparação de Erros entre Modelos")
plt.show()

```

→ Há 498 instâncias mal classificadas.

Comparação de Erros entre Modelos



Erros Modelo 1 Erros Modelo 2

✓ O problema das palavras fora do vocabulário OOV

Não consegui identificar diferenças de erros entre os dois modelos

```

#retorna o vocabulário do dataset
def vocabulario (texto):
    tokens_nltk = []
    for t in texto.values:
        tokens_nltk.extend(nltk.word_tokenize(t, language='portuguese'))
    tokens_distintos = set(tokens_nltk)
    return tokens_distintos

```

```

vocab_train = vocabulario(df_train['text'])
vocab_test = vocabulario(df_test['text'])

```

```

#tamanho do vocabulário de treino e teste
print (len(vocab_train))
print (len(vocab_test))

```

→ 13311
10213

```
oov = vocab_test.difference(vocab_train) #oov tem as palavras que aparecem no teste mas não no treino
```

```
print('Número de palavras que estão no teste e não no treino (OOV):',len(oov))
```

→ Número de palavras que estão no teste e não no treino (OOV): 3604

```
oov = list(oov) #transformando o conjunto em lista  
oov.sort() #ordenando a lista
```

```
#imprimindo as oovs  
oov[400:600]
```

```
↵ ['barbar',  
  'barbas',  
  'barber',  
  'barelhenta',  
  'barra',  
  'barulhentas',  
  'barulhos',  
  'basa',  
  'baseada',  
  'bastantes',  
  'bastaria',  
  'bastemp',  
  'bata',  
  'batalhas',  
  'batch',  
  'bateira',  
  'hatons']
```