```
__author__      = "Jeferson S. Pazze"
__description__ = "This code has the algorithms of data preprocessing and inference"
__github__      = "jeff-pazze"
__data__        = "December/2022"
__credits__     = "Docket"
__version__     = "1.1"
__maintainer__  = "Jeferson S. Pazze"
__email__       = "jeff.pazze@gmail.com"
__status__      = "engineer"
```

Tip:tips and notes.
Example: Typically also used to display warning messages.
Success: This alert box indicates a successful or positive action.
Danger: This alert box indicates a dangerous or potentially negative action.

Observações: 🎯

No decorrer do documento explico a metodologia utilizada, as técnicas em cada etapa entre outros dados que julgo importante para o desenvolvimento de modelos de ML.

Para a administração do meu tempo utilizei o Azure Devops onde tenho um board com atividades. Distribui cada atividade no periodo de 3 dias (Seg, ter e qua) com um tempo máximo de 2h30m dia.

Reservei 80 % do tempo para o dataprep e modelagem.

Com o tempo que sobrou executei um XAI para explicar o porque que o modelo toma tal decisão, como por exemplo: porquê ele disse que tal review era negativa ou neutra.

Executei técnicas de balanceamento dos dados, mas como estamos trabalhando em uma base pequena se utilizaremos undersample reduziriamos demais o dataset, já a tecnica de oversample e SMOTE não trouxeram melhoras nos resultados, assim como, pioraram a capacidade do modelo generalizar.

Link para acesso ao drive com o modelo, dataset e jupyter notebook: https://drive.google.com/drive/folders/1N6WYR7ZcW5CD8259okteQS-WxVxCCvj2?usp=share_link

Link para acesso ao github com o modelo, dataset e jupyter notebook: https://github.com/jeff-pazze/Docket-AI-Test

PROPOSTA

Neste teste para a área de machine learning, a partir do briefing e requisito apresentados, a Docket propõe a você classificar amostras sobre o mercado de ações.

IMPORTANTE

- Crie um projeto no Google Colab. Permita ele ser acessado por qualquer um com o link;
- O modelo treinado pode ser salvo em alguma plataforma como Google Drive.Permita também ele ser acessado por qualquer um com o link.

BRIEFING

A Docket tem o objetivo de desburocratizar os serviços cartorários para nossos clientes B2B, realizando a busca, gestão e pré-análise dos documentos, assim reduzindo o tempo de entrega e acompanhando o andamento de seu pedido através do nosso produto.

O QUE FAZEMOS

- Buscamos documentos em todo o Brasil;
- Entregamos os documentos aos clientes em até 15 dias;
- Pré-analisamos documentos de forma automática.

REQUISITOS

CLASSIFICAÇÃO DE AMOSTRAS DE TEXTO

Atualmente, nossos clientes solicitam ou nos enviam vários tipos de documentos. A partir desses documentos, extraímos e selecionamos diversas informações importantes para eles. Para que essas informações sejam corretamente extraídas, um processo inteligente de classificação do documento é necessário. O desafio que a Docket propõe é um problema de NLP, para classificação de notícias sobre o mercado de ações, a partir do dataset Stock Market News.

REQUISITOS OBRIGATÓRIOS

● Utilizar o dataset Stock Market News em versão portuguesa;

● Treinar um modelo de machine learning capaz de classificar cada amostra do dataset como uma das seguintes labels:

- positiva;
- negativa;
- neutra.

TECNOLOGIA

USAR AS SEGUINTES TECNOLOGIAS

- Python;
- Google Colab;
- Pacotes de sua preferência.

PLANEJAMENTO

Nos conte como irá se planejar para executar o projeto, como por exemplo: como transformou os requisitos em tarefas, se utilizou alguma ferramenta para se organizar, se desenhou algum diagrama, o processo de criação do dataset, porque escolheu certo modelo, etc. Essa é uma questão livre!

Metodologia Utilizada

Foi selecionado a metodologia **CRISP-DM** (Cross Industry Standard Process for Data Mining) para a execução deste trabalho.

O objetivo dessa metodologia é desenvolver modelos a partir da análise de informações e dados de um negócio para prever futuras falhas e soluções. Está metodologia está dividida em seis etapas, como é apresentado na sequência.

1. Entendimento do negócio
2. Entendimento dos dados
3. Preparação dos dados
4. Modelagem dos dados
5. Avaliação do Modelo
6. Deployment

1. **Entendimento do negócio:** A primeira atividade nesta metodologia é entender de fato qual o problema a ser resolvido.

Caso ela não seja feita da maneira correta, todo o resto do projeto pode ser invalidado futuramente.

2. **Entendimento dos dados:**

   Quantas fontes de dados serão utilizadas?

   Quais serão os formatos dos dados?

   Os dados estão estruturados?.

   A partir destes questionamento é realizado a coleta dos dados, tomando cuidado para que nenhuma informação importante fique de fora.

   defina → colete → explore

3. **Preparação dos dados:** Após a coleta dos dados, é necessário organizá-los de modo que seja possivel identificar o que os mesmos contam.

   Como os valores nulos devem ser tratados?

   Os atributos estão nos formatos corretos?

   Será necessário fazer alguma fusão com outros dados?

   Quais variáveis serão utilizadas na modelagem?

   Geralmente está etapa consome de 70 à 90% de todo o tempo proposto na atividades da CRISP-DM.

   Se esta etapa passar para próxima fase com erros, seu modelo inteiro tem que ser refeito.

4. **Modelagem dos dados:** Nesta etapa é realizada a criação do modelo.

   Nesta etapa o modelo começa a tomar forma, ou seja, é possivel ver os primeiros resultados.

   O tipo de modelagem a ser utilizada normalmente é definida de acordo com a necessidade do negócio e com o tipo de variável a ser analisada.

   selecione um método → separe um conjunto de dados para teste → construa o modelo → valide em todas as possibilidades

5. **Avaliação do Modelo:** Com a etapa de modelagem finalizada é possivel avaliar se o se o resultado corresponde à expectativa do projeto.

6. **Deployment (Implementação):** Aqui, o modelo deve ser colocado em produção, de modo a agregar valor para o negócio.

Referência:

https://www.knowsolution.com.br/voce-sabe-o-que-e-metodologia-crisp-dm-descubra-aqui/

https://blog.mbauspesalq.com/2022/04/12/crisp-dm-as-6-etapas-da-metodologia-do-futuro/

https://www.escoladnc.com.br/blog/data-science/metodologia-crisp-dm/

https://www.linkedin.com/pulse/crisp-dm-o-que-%C3%A9-e-como-usar-rodrigo-ribeiro/?originalSubdomain=pt

Install Dependencies

(Remember to choose GPU in Runtime if not already selected. Runtime --> Change Runtime Type --> Hardware accelerator --> GPU)

```
!pip install -q kaggle
!pip install scikit-plot
!pip install shap
!pip install lime
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting scikit-plot
  Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.8/dist-packages (
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: matplotlib>=1.4.0 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (fr
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-
Installing collected packages: scikit-plot
Successfully installed scikit-plot-0.3.7
```

```
# Utility
import re
```

```
import numpy as np
import os
from collections import Counter
import logging
import time
import pickle
import itertools
import numpy as np
import pandas as pd
from wordcloud import WordCloud
from tqdm.auto import tqdm
from scipy.stats import uniform, randint
from numpy import argmax

# nltk
import nltk
from nltk.corpus import stopwords
from  nltk.stem import SnowballStemmer
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.util import ngrams

# Word2vec
import gensim

##Mount Google Drive
import os
from google.colab import drive

# Vizualitation
import matplotlib.pyplot as plt
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#model building
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, auc
from scikitplot.metrics import plot_roc
from scikitplot.metrics import plot_precision_recall
from scikitplot.metrics import plot_cumulative_gain
from scikitplot.metrics import plot_lift_curve
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.utils.validation import check_is_fitted
from sklearn.exceptions import NotFittedError
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingC
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import plot_confusion_matrix

# SMOTE
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE

# Keras
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, Embedding, Flatten, Conv1D, MaxPoolin
from keras import utils
from keras.callbacks import ReduceLROnPlateau, EarlyStopping

#XAI
import shap
import lime
from lime import lime_text
from lime.lime_text import LimeTextExplainer
```

Declaração das funções utilizadas

```
def train_cv(model, X_train, y_train, params, n_splits=5, scoring='f1_weighted'):
    kf = KFold(n_splits=n_splits, random_state=0, shuffle=True)

    cv = RandomizedSearchCV(model,
                        params,
                        cv=kf,
                        scoring=scoring,
                        return_train_score=True,
                        n_jobs=-1,
                        verbose=2,
                        random_state=1
                        )
    cv.fit(X_train, y_train)

    print('Best params', cv.best_params_)
```

```
    return cv

#Function to fit and apply a model
def model_apply(model):
    #train the model
    model.fit(X_train,y_train)
    #make predictions
    pred=model.predict(X_val)
    #model evaluation
    print(model)
    print('Accuracy score: ',accuracy_score(pred,y_val))
    print('Weighted F1 score: ',f1_score(y_pred=pred,y_true=y_val,average='weighted'))
    print('Confusion Matrix: \n',confusion_matrix(pred,y_val))


def build_and_test(X_tr, X_te, y_tr, y_te, class_weight=None, threshold=False):
    # Build and Plot PCA
    pca = PCA(n_components=2)
    pca.fit(X_tr.toarray())
    X_pca = pca.transform(X_tr.toarray())
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_tr, cmap=plt.cm.prism, edgecolor='k', alpha=
    plt.show()

    # Build and fit the model
    if class_weight:
        model = DecisionTreeClassifier(class_weight=class_weight)
    else:
        model = DecisionTreeClassifier()
    model.fit(X_tr, y_tr)

    # Test the model
    y_pred = model.predict(X_te)
    print('Precision score %s' % precision_score(y_te, y_pred, average='macro'))
    print('Recall score %s' % recall_score(y_te, y_pred, average='macro'))
    print('F1-score score %s' % f1_score(y_te, y_pred, average='macro'))
    print('Accuracy score %s' % accuracy_score(y_te, y_pred))

    # Print a classification report
    print(classification_report(y_te,y_pred))
```

Configurações

```
nltk.download('stopwords')
stop = set(stopwords.words('portuguese'))
```

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

```
"""
GPU utilizada Tesla T4 com 16GB
"""
```

```
!nvidia-smi
```

```
Wed Dec 28 12:01:13 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   53C    P0    26W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
## Montar o drive no Google Drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

```
os.chdir("/content/gdrive/MyDrive/Docket/") ## Acessar o diretório onde o dataset foi salv

df = pd.read_csv("financial_phrase_bank_pt_br.csv") ## Consumir o arquivo e transforma-lo

## Chamar as informações basicas como quantidade de linhas e colunas e o tipo de cada dado
print("Summary\n")
print("Row Size : ", df.shape[0])
print("Column Size : ", df.shape[1])

print("\nColumn Type Information")
df.info()
```

```
Summary

Row Size :  4845
Column Size :  3

Column Type Information
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4845 entries, 0 to 4844
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   y        4845 non-null   object
 1   text     4845 non-null   object
 2   text_pt  4845 non-null   object
dtypes: object(3)
memory usage: 113.7+ KB
```

```
df
```

	y	text	text_pt
0	neutral	Technopolis plans to develop in stages an area...	A Technopolis planeja desenvolver em etapas um...
1	negative	The international electronic industry company ...	A Elcoteq, empresa internacional da indústria ...
2	positive	With the new production plant the company woul...	Com a nova planta de produção a empresa aument...
3	positive	According to the company 's updated strategy f...	De acordo com a estratégia atualizada da empre...
4	positive	FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is ag...	FINANCIAMENTO DO CRESCIMENTO DA ASPOCOMP A Asp...
...	...	...	...
4840	negative	LONDON MarketWatch -- Share prices ended lower...	LONDRES MarketWatch - Os preços das ações term...
4841	neutral	Rinkuskiai 's beer sales fell by 6.5 per cent ...	As vendas de cerveja da Rinkuskiai caíram 6,5 ...

About Datase:

Stock Market News Data in Portuguese The Financial Phrase Bank is a dataset originally developed for the paper Good Debt or Bad Debt: Detecting Semantic Orientations in Economic Texts, made available by researchers from Aalto University and the Indian Institute of Management. The dataset allows for a useful benchmark for fine-tuning Language Models on Sentiment Analysis Tasks.

As the amount of annotated text data (especially about the financial market) in Portuguese, I went ahead and translated the entire dataset for people to try out Sentiment Analysis tasks in Portuguese.

Content The dataset originally contains about 4840 manually annotated financial news in English and consists of three columns:

```
# y: the annotated label for the sentiment of the news text (neutral, positive, negative);

text: the original text for each record;

text_pt: the translated and that I manually validated version of the original record;
```

Acknowledgments [1] Malo, P., Sinha, A., Korhonen, P., Wallenius, J., & Takala, P. (2014). Good debt or bad debt: Detecting semantic orientations in economic texts. Journal of the Association for Information Science and Technology, 65(4), 782-796.

Photo by Markus Winkler on Unsplash

1. Entendimento do negócio: ✅

Como o dataset foi disponibilizado pelo negócio, assume-se que o entendimento do mesmo já foi realizado.

Em suma, o negocio busca um modelo de ML que seja capaz de classificar cada amostra do dataset como uma das seguintes labels: positiva, negativa ou neutra.

2. Entendimento dos dados: ✅

Quantas fontes de dados serão utilizadas?

Quais serão os formatos dos dados?

Os dados estão estruturados?

Qual a distribuição dos dados?

Existe dados faltantes, se sim, qual a distribuição?

Os dados estão balanceados?

Exploratory Data Analysis (EDA)

EDA é a investigação de um conjunto de dados para encontrar padrões e anomalias (outliers), bem como para criar hipóteses com base em nosso conhecimento do conjunto de dados.

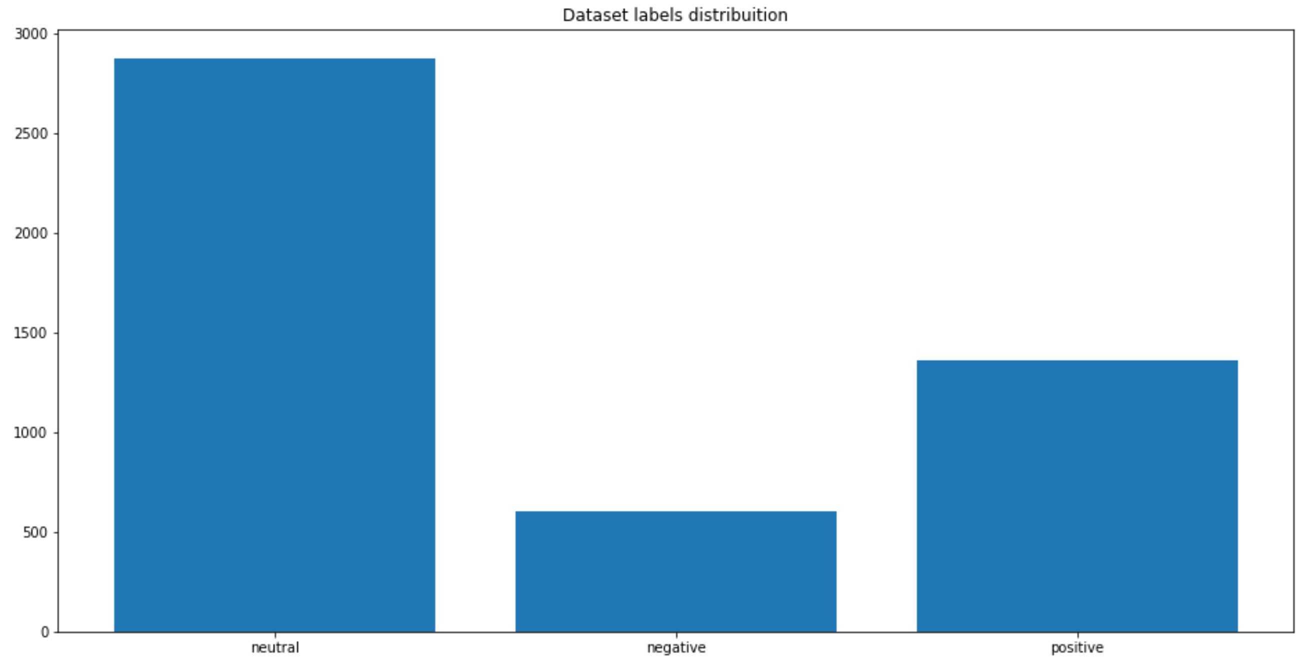
```
#Summary of the dataset
df.describe()
```

	target
count	4845.000000
mean	1.156656

Analise da distribuição do target (y)

```
target_cnt = Counter(df.y)

plt.figure(figsize=(16,8))
plt.bar(target_cnt.keys(), target_cnt.values())
plt.title("Dataset labels distribuition")
```

```
Text(0.5, 1.0, 'Dataset labels distribuition')
```



Nesta análise é possivel identificar que o target está desbalanceado, o que por sua vez, pode enviesar o modelo de ML.

É possivel empregar diversas tecnicas para dataset não balanceados como: Over-sampling (Up Sampling)(SMOTE), Under-sampling (Down Sampling) ou ambos.

Dado o tamanho do dataset, realizar o under-sampling reduziria em mais de 60% o tamanho do mesmo, o que por sua vez, faz com que não seja o mais indicado dado o tamanho do dataset.

```
## label encoding on sentiment
```

```
df['target'] = df.y.map({'negative':0, 'neutral':1, 'positive':2})
```

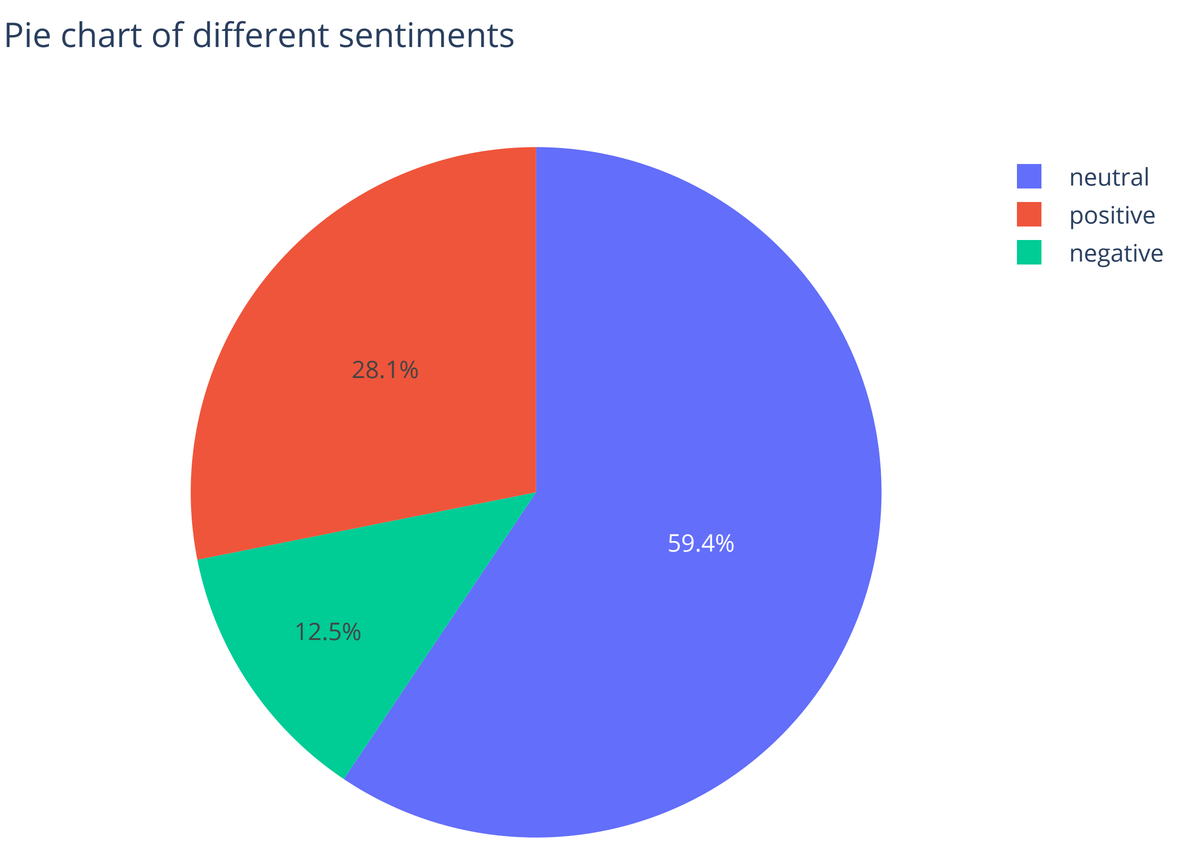
```
df['target'].value_counts()
```

```
1    2878
2    1363
0     604
Name: target, dtype: int64
```

```
temp = df.groupby('y').count()['text_pt'].reset_index().sort_values(by='text_pt',ascending
temp.style.background_gradient(cmap='Purples')
```

	y	text_pt
1	neutral	2878
2	positive	1363
0	negative	604

```
fig = px.pie(df, names='y', title ='Pie chart of different sentiments')
fig.show()
```



GARBAGE IN, GARBAGE OUT (Charles Babbage)

GARBAGE IN, GARBAGE OUT (Lixo dentro, lixo fora) é uma frase usada para expressar a ideia de que a inserção de dados de baixa qualidade ou sem sentido em um sistema produzirá resultados de baixa qualidade ou sem sentido.

Não importa o quão poderoso ou inteligentemente projetado seja um sistema, ele só pode operar com base nas informações fornecidas

A fim de verificar se o target foi imputado corretamente, é feita uma analise amostral com n=10 para cada classe e apresentado através de uma wordcloud.

Durante está análise, não foi encontrado nenhum grande desvio nos targets.

```
Negative_sent = df[df['target']==0]
Neutral_sent  = df[df['target']==1]
Positive_sent = df[df['target']==2]
```

```
df[df['target']==0].sample(n=10)
```

	y	text	text_pt	target
4203	negative	In Sweden , there is an oversupply of pharmaci...	Na Suécia, há um excesso de oferta de farmácias.	0
4704	negative	Pretax profit decreased by 37 % to EUR 193.1 m...	O lucro antes dos impostos diminuiu 37% para E...	0
4445	negative	Kone shares dropped 4.1 percent to x20ac 43 U...	As ações da Kone caíram 4,1 por cento para x20...	0
499	negative	One of the challenges in the oil production in...	Um dos desafios na produção de petróleo no Mar...	0
697	negative	ADP News - Apr 22 , 2009 - Finnish business in...	ADP News - 22 de abril de 2009 - O desenvolved...	0
4436	negative	`` We can say that the number of deals has bec...	`` Podemos dizer que o número de negócios se n...	0
4537	negative	stores 16 March 2010 - Finnish	lojas 16 de março de 2010 - O	0

```
text = ' '.join(df[df['target']==0]['text_pt'].str.lower())#.values[-1000000:])
wordcloud = WordCloud(max_font_size=None, stopwords=stop, background_color='white',
width=1200, height=1000).generate(text)

plt.figure(figsize=(12, 8))
plt.imshow(wordcloud)
plt.title('negative sentiment')
plt.axis("off")
plt.show()
```

negative sentiment


```
df[df['target']==1].sample(n=10)
```

	y	text	text_pt	target
3958	neutral	TVO 's two-unit 1,740 MW Olkiluoto plant gener...	A usina de Olkiluoto de 1.740 MW de duas unida...	1
3239	neutral	The engine has an electrical output of 18,321 ...	O motor tem potência elétrica de 18.321 kW, se...	1
1296	neutral	Oka specialises in new construction , renovati...	A Oka é especialista em construção nova, renov...	1
4581	neutral	Operating profit for the 12-month period decre...	O lucro operacional para o período de 12 meses...	1
2587	neutral	The bridge will be 1.2 km long and is located ...	A ponte terá 1,2 km de comprimento e está loca...	1
2737	neutral	A total of 131000 Talvivaara Mining Company Pl...	Um total de 131.000 novas ações da Talvivaara ...	1
4134	neutral	The issue came up in connection	A questão surgiu em conexão com a	1

```
text = ' '.join(df[df['target']==1]['text_pt'].str.lower())#.values[-1000000:])
wordcloud = WordCloud(max_font_size=None, stopwords=stop, background_color='white',
width=1200, height=1000).generate(text)

plt.figure(figsize=(12, 8))
plt.imshow(wordcloud)
plt.title('Neutral sentiment')
```

```
plt.axis("off")
plt.show()
```


```
df[df['target']==2].sample(n=10)
```

	y	text	text_pt	target
690	positive	Profit after taxes for the period was up to EU...	O lucro após os impostos para o período foi de...	2
717	positive	Operating profit totalled EUR 7.0 mn , up from...	O lucro operacional totalizou 7,0 milhões de e...	2
311	positive	With the acquisition , the company will expand...	Com a aquisição, a empresa vai expandir sua of...	2
1219	positive	Finnish automation solutions developer Cencorp...	A desenvolvedora finlandesa de soluções de aut...	2
769	positive	Managing Director Kari Inkinen says that Spond...	O diretor administrativo Kari Inkinen diz que ...	2
4082	positive	The talks are aimed at restructuring operation...	As negociações visam à reestruturação das oper...	2
776	positive	Ponsse projects the forest machine	A Ponsse projeta que os mercados	2

```
text = ' '.join(df[df['target']==2]['text_pt'].str.lower())#.values[-1000000:])
wordcloud = WordCloud(max_font_size=None, stopwords=stop, background_color='white',
```

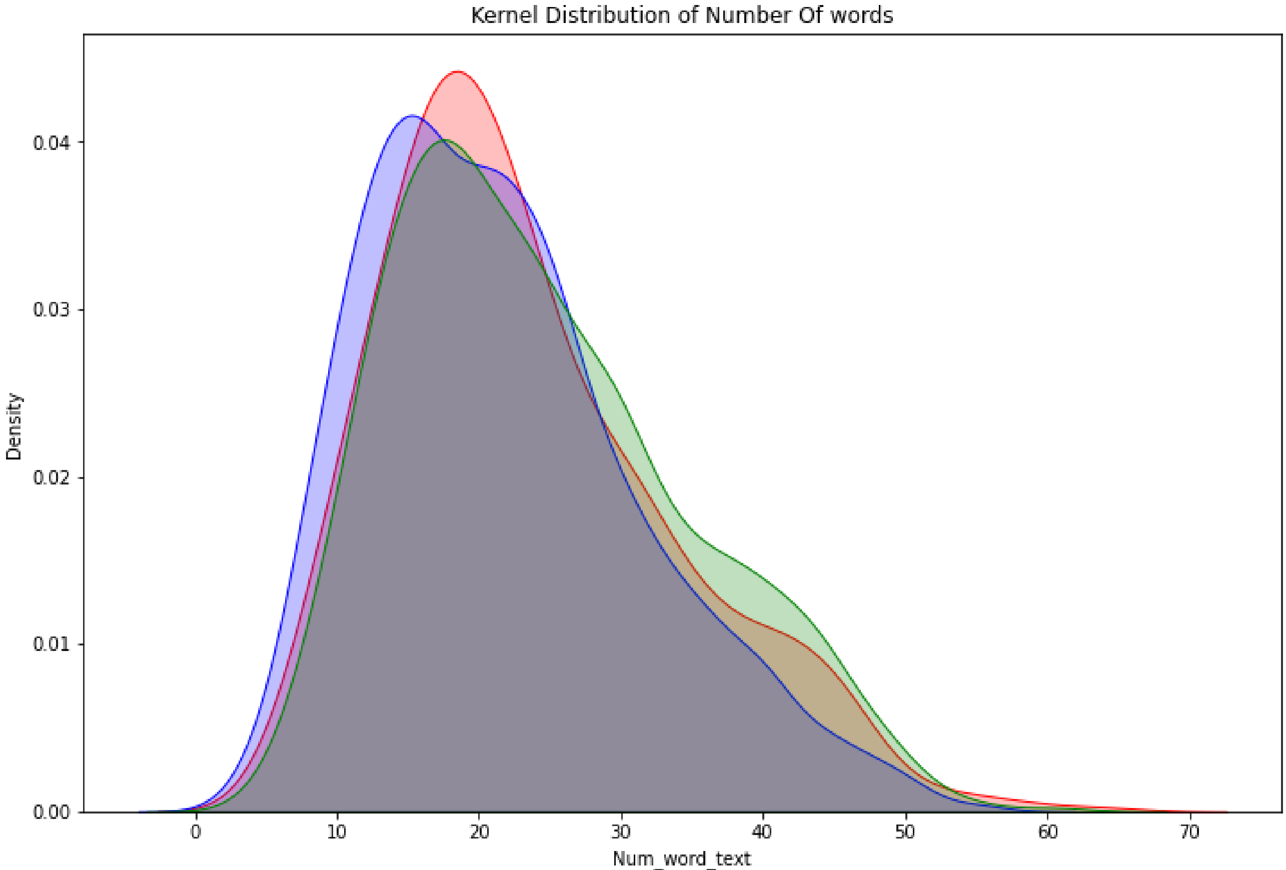
```
width=1200, height=1000).generate(text)

plt.figure(figsize=(12, 8))
plt.imshow(wordcloud)
plt.title('positive sentiment')
plt.axis("off")
plt.show()
```

Criação da feature do tamanho das palavras nos comentários

```
df['Num_word_text'] = df['text_pt'].apply(lambda x:len(str(x).split())) #Number Of words i
df
```

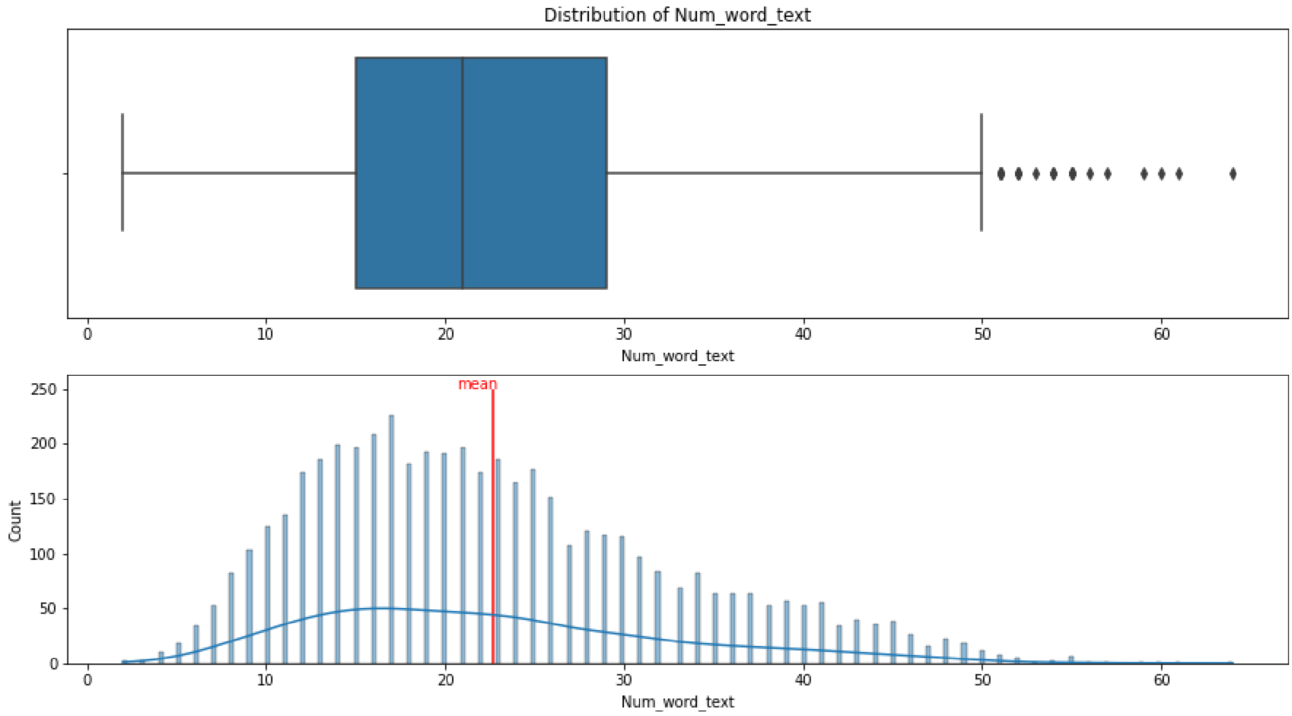
	y	text	text_pt	target	Num_word_text
0	neutral	Technopolis plans to develop in stages an area...	A Technopolis planeja desenvolver em etapas um...	1	29
1	negative	The international electronic industry	A Elcoteq, empresa internacional da indústria	0	33

```
## Analise da distribuição do tamanho das avaliação por clsse.
plt.figure(figsize=(12,8))
p1=sns.kdeplot(df[df['target']==0]['Num_word_text'], shade=True, color="r").set_title('Ker
p1=sns.kdeplot(df[df['target']==1]['Num_word_text'], shade=True, color="b")
p1=sns.kdeplot(df[df['target']==2]['Num_word_text'], shade=True, color="green")
```




```
len_mean = np.mean(df.Num_word_text)

fig, axes = plt.subplots(2, 1, figsize=(15, 8))
axes[0].set_title('Distribution of Num_word_text')
sns.boxplot(df.Num_word_text, ax=axes[0])
sns.histplot(df.Num_word_text, bins=250, kde=True, ax=axes[1])
axes[1].vlines(len_mean, 0, 250, color = 'r')
plt.annotate("mean", xy=(len_mean, 250), xytext=(len_mean-2, 250),color='r')
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid p
```



Após plotar a distribuição das palavras pelas classes, podemos desconsiderar as mesmas, pois estas não possuem uma representatividade do fenômeno, dado a semelhança do tamanho da avaliação pelas classes: neutro, negativo e posito.

Ou seja, não é possivel determinar que uma avaliação negativa tem uma média de 14 palavras e uma positiva 50 ou o inverso

```
print(df[df['target']==0]['Num_word_text'].mean())
print(df[df['target']==1]['Num_word_text'].mean())
print(df[df['target']==2]['Num_word_text'].mean())
```

```
23.455298013245034
21.633773453787352
24.399853264856933
```

3. Preparação dos dados: ✅

Data cleaning

Preprocessing the text data

Text transformation

Remove Additional Letter

Remove Stop Words

Stemming

TF-IDF

Data Cleaning

```
df['target'].isnull().sum()
```

```
0
```

```
df['text_pt'].isnull().sum()
```

```
0
```

```
df.dropna(axis=0, inplace=True)
```

```
print("Percentage null or na values in df")
((df.isnull() | df.isna()).sum() * 100 / df.index.size).round(2)
```

```
Percentage null or na values in df
y                0.0
text             0.0
text_pt          0.0
target           0.0
Num_word_text    0.0
dtype: float64
```

Preprocessing the text data

```
### Preprocessing the text data

def text_to_words(text):
    letters_only = re.sub("[^a-zA-Z]", " ",text)

    words = letters_only.lower().split()

    meaningful_words = [w for w in words if not w in stops]
    return( " ".join( meaningful_words ))
```

```
#nltk.download('stopwords')
stops = set(stopwords.words("portuguese"))

df['clean_text'] = df['text_pt'].apply(lambda x: text_to_words(x))
```

```
df
```

	y	text	text_pt	target	Num_word_text	clean_text
0	neutral	Technopolis plans to develop in stages an area...	A Technopolis planeja desenvolver em etapas um...	1	29	technopolis planeja desenvolver etapas rea n i...
1	negative	The international electronic industry company ...	A Elcoteq, empresa internacional da indústria ...	0	33	elcoteq empresa internacional ind stria eletr ...
2	positive	With the new production plant the company woul...	Com a nova planta de produção a empresa aument...	2	31	nova planta produ empresa aumentaria capacidad...
3	positive	According to the company 's updated strategy f...	De acordo com a estratégia atualizada da empre...	2	43	acordo estrat gia atualizada empresa anos basw...

Text transformation

```
#Text transformation
df['clean_text']=df.clean_text.str.lower() #lowercase
df['clean_text']=[str(data) for data in df.clean_text] #converting all to string
df['clean_text']=df.clean_text.apply(lambda x: re.sub('[^A-Za-z0-9 ]+', ' ', x)) #regex
```

Remove Additional Letter

```
## Remove Additional Letter
REPLACE_WITH_SPACE = re.compile("(@)")
SPACE = " "

def preprocess_reviews(reviews):
    reviews = [REPLACE_WITH_SPACE.sub(SPACE, line.lower()) for line in reviews]

    return reviews

reviews_train_clean = preprocess_reviews(df['clean_text'])
```

Remove Stop Words

```
##Remove Stop Words
def remove_stop_words(corpus):
    removed_stop_words = []
    for review in corpus:
        removed_stop_words.append(
            ' '.join([word for word in review.split()  if word not in stop]))
    return removed_stop_words

no_stop_words_train = remove_stop_words(reviews_train_clean)
```

Stemming

```
##Stemming
def get_stemmed_text(corpus):

    stemmer = PorterStemmer()

    return [' '.join([stemmer.stem(word) for word in review.split()]) for review in corpus

stemmed_reviews_train = get_stemmed_text(no_stop_words_train)
```

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF significa Frequência de Documentos Inversos de Frequência de Prazo de registros. Pode ser definido como o cálculo de quão relevante uma palavra em uma série ou corpus é para um texto. O significado aumenta proporcionalmente ao número de vezes no texto que uma palavra aparece, mas é compensado pela frequência da palavra no corpus (conjunto de dados).

TF-IDF é um cálculo estatístico adotado pelo algoritmo do Google para medir quais termos são mais relevantes para um tópico, analisando a frequência com que aparecem em uma página, em comparação à sua frequência em um conjunto maior de páginas.

```
##TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(stemmed_reviews_train)
X = tfidf_vectorizer.transform(stemmed_reviews_train)

features = X.toarray()
labels = df['target']

print("Each of the %d Text is represented by %d features (TF-IDF score of unigrams and big
```

```
Each of the 4845 Text is represented by 8907 features (TF-IDF score of unigrams and b
```

4. Modelagem dos dados: ✅

Balanceamento do dataset

Definição do baseline

Modelagem

Teste e validação

Balanceamento do dataset

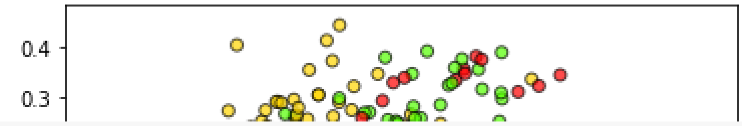
Na primeira interação com os dados e os possiveis modleos, eu sempre rodo em datasets pequenos (até 100k registros) diversos modelos para identificar qual classe de modelo melhor se ajusta aos dados.

Também utilizo o lazypredicit que tem como objetivo rodar sobre os dados uma gama de mais de 20 modelos.

Desta forma, posso selecionar apenas os modelos mais aderentes para a etapa de Gridsearch.

```
## Spliting Data
X_train, X_val, y_train, y_val = train_test_split(X, df['target'], train_size = 0.70)
```

```
## Modelo desbalanceado
build_and_test(X_train, X_val, y_train, y_val)
```

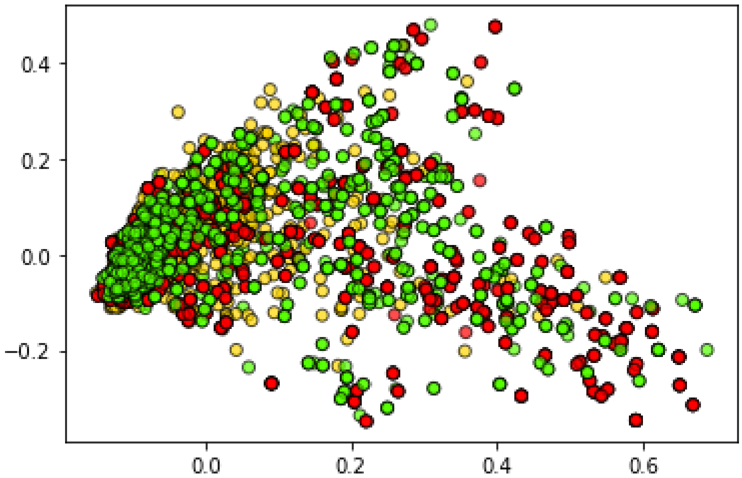



```
## Modelo balanceado (Oversample the smallest class)
over_sampler = RandomOverSampler(random_state=42)
X_res, y_res = over_sampler.fit_resample(X_train, y_train)
print(f"Training target statistics: {Counter(y_res)}")
print(f"Testing target statistics: {Counter(y_val)}")
```

```
Training target statistics: Counter({2: 2036, 1: 2036, 0: 2036})
Testing target statistics: Counter({1: 842, 2: 434, 0: 178})
```

0.0 0.2 0.4 0.6

```
build_and_test(X_res, X_val, y_res, y_val)
```

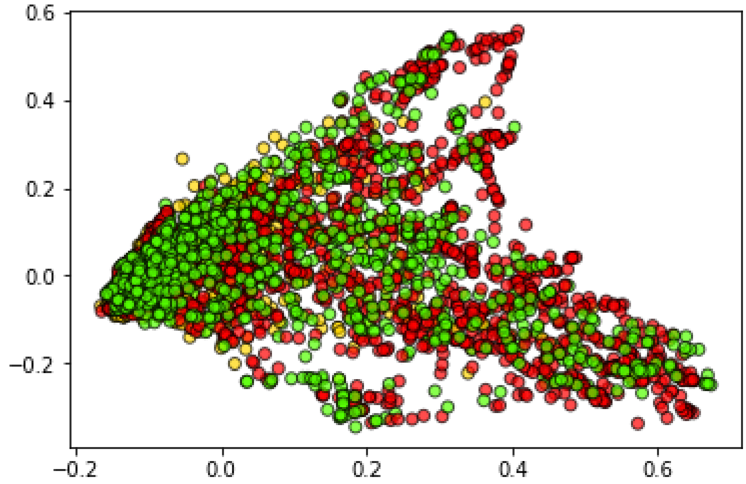



```
Precision score 0.5569284909170165
Recall score 0.555445732732505
F1-score score 0.5556168288389971
Accuracy score 0.6265474552957359
              precision    recall  f1-score   support

           0       0.44      0.46      0.45       178
           1       0.72      0.75      0.73       842
           2       0.52      0.47      0.49       434

    accuracy                           0.63      1454
   macro avg       0.56      0.56      0.56      1454
weighted avg       0.62      0.63      0.62      1454
```

```
##The same analysis can be done for the SMOTE technique.
over_sampler = SMOTE(k_neighbors=2)
X_res, y_res = over_sampler.fit_resample(X_train, y_train)
print(f"Training target statistics: {Counter(y_res)}")
print(f"Testing target statistics: {Counter(y_val)}")
```

```
Training target statistics: Counter({2: 2036, 1: 2036, 0: 2036})
Testing target statistics: Counter({1: 842, 2: 434, 0: 178})
```

```
build_and_test(X_res, X_val, y_res, y_test)
```




```
Precision score 0.5983173278549767
Recall score 0.5862149474184887
F1-score score 0.5909581992946206
Accuracy score 0.657496561210454
              precision    recall  f1-score   support

           0       0.50      0.49      0.50       178
           1       0.72      0.78      0.75       842
           2       0.57      0.49      0.53       434

    accuracy                           0.66      1454
   macro avg       0.60      0.59      0.59      1454
weighted avg       0.65      0.66      0.65      1454
```

Training target statistics: Counter({2: 2036, 1: 2036, 0: 2036})

Testing target statistics: Counter({1: 842, 2: 434, 0: 178})

Observações ⚡

É possivel verificar que com o balanceamento as métricas de avaliação ficaram piores.

Dito isso, seguiu com os dados desbalanceados!

Baseline (ML Based Approach)

Na primeira inteeação com os dados e os possiveis modleos, eu sempre rodo em datasets pequenos (até 100k registros) diversos modelos para identificar qual classe de modelo melhor se ajusta aos dados.

Também utilizo o lazypredicit que tem como objetivo rodar sobre os dados uma gama de mais de 20 modelos.

Desta forma, posso selecionar apenas os modelos mais aderentes para a etapa de Gridsearch.

```
##ML Based Approach

rf  = RandomForestClassifier(random_state=42)
gb  = GradientBoostingClassifier(random_state=42)
ada = AdaBoostClassifier(random_state=42)
lgb = LGBMClassifier(random_state=42)
xgb = XGBClassifier(eval_metric="mlogloss",random_state=42)
dt  = DecisionTreeClassifier(random_state=42)
svc = SVC(random_state=42)
nb  = MultinomialNB()
mlp = MLPClassifier(random_state=42)
lr  = LogisticRegression(random_state=10, max_iter=500)
ada = AdaBoostClassifier(random_state=42)

clfs = {
    "Random Forest": rf,
    "Gradient Boosting":gb,
    "AdaBoost": ada,
    "LightGBM": lgb,
    "XGBoost": xgb,
    "Decision Tree":dt,
    "Support Vector Machine":svc,
    "Naive Bayes": nb,
    "Multilayer Perceptron":mlp,
    "Logistic Regression":lr,
    "AdaBoost Classifier" : ada
    }

def fit_model(clf,x_train,y_train,x_test, y_test):
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    accuracy = accuracy_score(y_pred, y_test)
    return accuracy

accuracys = []

for name,clf in tqdm(clfs.items()):
    curr_acc = fit_model(clf,X_train,y_train,X_val,y_val)

    accuracys.append(curr_acc)
```

100% 11/11 [01:34<00:00, 10.02s/it]

```
models_df = pd.DataFrame({"Models":clfs.keys(),"Accuracy Scores":accuracys}).sort_values('
models_df
```

	Models	Accuracy Scores
1	Gradient Boosting	0.759285
3	LightGBM	0.748281
0	Random Forest	0.742779
9	Logistic Regression	0.740028
4	XGBoost	0.738652
6	Support Vector Machine	0.735901
8	Multilayer Perceptron	0.726272
2	AdaBoost	0.705640
10	AdaBoost Classifier	0.705640

Execução de Tuning no Hiperparâmetros (GridSearsh)

1) Multinomail Naive Bayes (Hyperparameter tuning best models)

```
#Multinomail Naive Bayes
#Model # Hyperparameter tuning best models

parameters = {'alpha': [0.001,0.01,0.1,0.2,0.3,0.5,0.7,1,1.5,1.6,1.8,10,100, 1000],
              'fit_prior': [True, False]
              }

grid_search = MultinomialNB()

grid_search = GridSearchCV(grid_search, parameters, cv=5, scoring = 'accuracy', n_jobs = -

grid_result = grid_search.fit(X_train, y_train)

print('Best params: ', grid_result.best_params_)
print('Best score: ', grid_result.best_score_)

# Best params:  {'alpha': 0.2, 'fit_prior': True}
# Best score:  0.6971452900109045
```

```
Best params:  {'alpha': 0.2, 'fit_prior': True}
Best score:  0.6971452900109045
```

2) SVM (Hyperparameter tuning best models)

```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
```

```
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid_result = grid.fit(X_train, y_train)



#grid_result = grid_search.fit(X_train, y_train)

print('Best params: ', grid_result.best_params_)
print('Best score: ', grid_result.best_score_)

# Best params:  {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
# Best score:  0.7487533723461102
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.608 total time=   1.6s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.608 total time=   1.6s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.608 total time=   1.6s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.611 total time=   1.7s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.614 total time=   1.6s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.586 total time=   1.3s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.587 total time=   1.3s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.587 total time=   1.2s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.587 total time=   1.3s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.587 total time=   1.3s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.586 total time=   1.1s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.1s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.1s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.1s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.1s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.586 total time=   1.0s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 1/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.586 total time=   1.0s
[CV 2/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 3/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.694 total time=   1.7s
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.706 total time=   1.7s
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.687 total time=   1.7s
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.730 total time=   1.7s
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.715 total time=   1.7s
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.644 total time=   1.3s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.653 total time=   1.3s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.640 total time=   1.3s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.664 total time=   1.3s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.650 total time=   1.3s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.586 total time=   2.1s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.3s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.3s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.3s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.587 total time=   1.3s
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.586 total time=   1.2s
```

```
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.1s
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.1s
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.1s
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.587 total time=   1.1s
[CV 1/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.586 total time=   1.5s
[CV 2/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.8s
[CV 3/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.3s
[CV 4/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 5/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.587 total time=   1.0s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.717 total time=   1.8s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.737 total time=   1.8s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.730 total time=   2.7s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.760 total time=   2.6s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.748 total time=   1.7s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.726 total time=   1.4s
```

```
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)

grid_predictions = grid.predict(X_val)

# print classification report
print(classification_report(y_valid, grid_predictions))
```

```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
SVC(C=10, gamma=0.1)
              precision    recall  f1-score   support

           0       0.72      0.62      0.67       165
           1       0.79      0.91      0.85       888
           2       0.75      0.54      0.63       401

    accuracy                           0.78      1454
   macro avg       0.76      0.69      0.72      1454
weighted avg       0.77      0.78      0.77      1454
```

3) Random Forest (Hyperparameter tuning best models)

```
rfc=RandomForestClassifier()

param_grid = {
    'n_estimators': [1, 10, 100, 500],
    'max_features': ['auto'],#, 'sqrt', 'log2'],
    'max_depth' : [1, 4, 7,8],
    'criterion' :['gini', 'entropy'],
    'min_samples_split': [2, 4, 9],
    "min_samples_leaf": [1, 3, 9]

}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5, refit = True, verbose =
```

```
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5, refit = True, verbose
grid_result =  CV_rfc.fit(X_train, y_train)

print('Best params: ', grid_result.best_params_)
print('Best score: ', grid_result.best_score_)

# Best params:  {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'min_samp
# Best score:  0.6198778352687666
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
```

```
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 2/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 3/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 4/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 5/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
[CV 1/5] END criterion=gini, max_depth=1, max_features=auto, min_samples_leaf=1, m
```

4) Gradient Boosting Classifier (Hyperparameter tuning best models)

```
## Boosting is an ensemble method to aggregate all the weak models to make them better and

gbc = GradientBoostingClassifier()

parameters = {
    "n_estimators":[5,50,500,1000],
    "max_depth":[1,3,9],
    "learning_rate":[0.01,0.1,1,10]
}

CV_rfc = GridSearchCV(estimator=gbc, param_grid=parameters, cv= 3, verbose = 3)#, n_jobs =
grid_result =  CV_rfc.fit(X_train, y_train)

print('Best params: ', grid_result.best_params_)
print('Best score: ', grid_result.best_score_)

# Best params:  {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
# Best score:  0.7393144135896653
```

```
Fitting 3 folds for each of 64 candidates, totalling 192 fits
[CV 1/3] END learning_rate=0.01, max_depth=1, n_estimators=5;, score=0.591 total t
[CV 2/3] END learning_rate=0.01, max_depth=1, n_estimators=5;, score=0.590 total t
[CV 3/3] END learning_rate=0.01, max_depth=1, n_estimators=5;, score=0.590 total t
[CV 1/3] END learning_rate=0.01, max_depth=1, n_estimators=50;, score=0.606 total
[CV 2/3] END learning_rate=0.01, max_depth=1, n_estimators=50;, score=0.605 total
[CV 3/3] END learning_rate=0.01, max_depth=1, n_estimators=50;, score=0.604 total
[CV 1/3] END learning_rate=0.01, max_depth=1, n_estimators=250;, score=0.656 total
[CV 2/3] END learning_rate=0.01, max_depth=1, n_estimators=250;, score=0.642 total
[CV 3/3] END learning_rate=0.01, max_depth=1, n_estimators=250;, score=0.658 total
[CV 1/3] END learning_rate=0.01, max_depth=1, n_estimators=500;, score=0.681 total
[CV 2/3] END learning_rate=0.01, max_depth=1, n_estimators=500;, score=0.665 total
[CV 3/3] END learning_rate=0.01, max_depth=1, n_estimators=500;, score=0.681 total
[CV 1/3] END learning_rate=0.01, max_depth=3, n_estimators=5;, score=0.591 total t
[CV 2/3] END learning_rate=0.01, max_depth=3, n_estimators=5;, score=0.590 total t
[CV 3/3] END learning_rate=0.01, max_depth=3, n_estimators=5;, score=0.590 total t
[CV 1/3] END learning_rate=0.01, max_depth=3, n_estimators=50;, score=0.633 total
[CV 2/3] END learning_rate=0.01, max_depth=3, n_estimators=50;, score=0.630 total
[CV 3/3] END learning_rate=0.01, max_depth=3, n_estimators=50;, score=0.620 total
[CV 1/3] END learning_rate=0.01, max_depth=3, n_estimators=250;, score=0.696 total
[CV 2/3] END learning_rate=0.01, max_depth=3, n_estimators=250;, score=0.694 total
[CV 3/3] END learning_rate=0.01, max_depth=3, n_estimators=250;, score=0.696 total
[CV 1/3] END learning_rate=0.01, max_depth=3, n_estimators=500;, score=0.709 total
[CV 2/3] END learning_rate=0.01, max_depth=3, n_estimators=500;, score=0.712 total
[CV 3/3] END learning_rate=0.01, max_depth=3, n_estimators=500;, score=0.720 total
[CV 1/3] END learning_rate=0.01, max_depth=5, n_estimators=5;, score=0.591 total t
[CV 2/3] END learning_rate=0.01, max_depth=5, n_estimators=5;, score=0.590 total t
```

```
[CV 3/3] END learning_rate=0.01, max_depth=5, n_estimators=5;, score=0.590 total t
[CV 1/3] END learning_rate=0.01, max_depth=5, n_estimators=50;, score=0.647 total
[CV 2/3] END learning_rate=0.01, max_depth=5, n_estimators=50;, score=0.652 total
[CV 3/3] END learning_rate=0.01, max_depth=5, n_estimators=50;, score=0.650 total
[CV 1/3] END learning_rate=0.01, max_depth=5, n_estimators=250;, score=0.704 total
[CV 2/3] END learning_rate=0.01, max_depth=5, n_estimators=250;, score=0.706 total
[CV 3/3] END learning_rate=0.01, max_depth=5, n_estimators=250;, score=0.727 total
[CV 1/3] END learning_rate=0.01, max_depth=5, n_estimators=500;, score=0.714 total
[CV 2/3] END learning_rate=0.01, max_depth=5, n_estimators=500;, score=0.716 total
[CV 3/3] END learning_rate=0.01, max_depth=5, n_estimators=500;, score=0.739 total
[CV 1/3] END learning_rate=0.01, max_depth=9, n_estimators=5;, score=0.591 total t
[CV 2/3] END learning_rate=0.01, max_depth=9, n_estimators=5;, score=0.590 total t
[CV 3/3] END learning_rate=0.01, max_depth=9, n_estimators=5;, score=0.590 total t
[CV 1/3] END learning_rate=0.01, max_depth=9, n_estimators=50;, score=0.664 total
[CV 2/3] END learning_rate=0.01, max_depth=9, n_estimators=50;, score=0.678 total
[CV 3/3] END learning_rate=0.01, max_depth=9, n_estimators=50;, score=0.674 total
[CV 1/3] END learning_rate=0.01, max_depth=9, n_estimators=250;, score=0.714 total
[CV 2/3] END learning_rate=0.01, max_depth=9, n_estimators=250;, score=0.714 total
[CV 3/3] END learning_rate=0.01, max_depth=9, n_estimators=250;, score=0.735 total
[CV 1/3] END learning_rate=0.01, max_depth=9, n_estimators=500;, score=0.715 total
[CV 2/3] END learning_rate=0.01, max_depth=9, n_estimators=500;, score=0.727 total
[CV 3/3] END learning_rate=0.01, max_depth=9, n_estimators=500;, score=0.749 total
[CV 1/3] END learning_rate=0.1, max_depth=1, n_estimators=5;, score=0.606 total ti
[CV 2/3] END learning_rate=0.1, max_depth=1, n_estimators=5;, score=0.609 total ti
[CV 3/3] END learning_rate=0.1, max_depth=1, n_estimators=5;, score=0.604 total ti
[CV 1/3] END learning_rate=0.1, max_depth=1, n_estimators=50;, score=0.687 total t
[CV 2/3] END learning_rate=0.1, max_depth=1, n_estimators=50;, score=0.665 total t
[CV 3/3] END learning_rate=0.1, max_depth=1, n_estimators=50;, score=0.681 total t
[CV 1/3] END learning_rate=0.1, max_depth=1, n_estimators=250;, score=0.708 total
[CV 2/3] END learning_rate=0.1, max_depth=1, n_estimators=250;, score=0.712 total
```

5) Logistic Regression (Hyperparameter tuning best models)

```
lr = LogisticRegression()

parameters = {
    "C": np.logspace(-4, 4, 50), #np.logspace(-3,3,7)
    "penalty":['l1', 'l2'] # l1 lasso l2 ridge
    }

CV_rfc = GridSearchCV(estimator=lr, param_grid=parameters, cv= 5, refit = True, verbose =
grid_result =  CV_rfc.fit(X_train, y_train)

print('Best params: ', grid_result.best_params_)
print('Best score: ', grid_result.best_score_)

# Best params:  {'C': 7.9060432109076855, 'penalty': 'l2'}
# Best score:  0.7496352870132635
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
[CV 1/5] END ................C=0.0001, penalty=l1;, score=nan total time=   0.0s
[CV 2/5] END ................C=0.0001, penalty=l1;, score=nan total time=   0.0s
[CV 3/5] END ................C=0.0001, penalty=l1;, score=nan total time=   0.0s
[CV 4/5] END ................C=0.0001, penalty=l1;, score=nan total time=   0.0s
[CV 5/5] END ................C=0.0001, penalty=l1;, score=nan total time=   0.0s
```

```
[CV 1/5] END .............C=0.0001, penalty=l2;, score=0.591 total time=   0.1s
[CV 2/5] END .............C=0.0001, penalty=l2;, score=0.591 total time=   0.2s
[CV 3/5] END .............C=0.0001, penalty=l2;, score=0.590 total time=   0.2s
[CV 4/5] END .............C=0.0001, penalty=l2;, score=0.590 total time=   0.4s
[CV 5/5] END .............C=0.0001, penalty=l2;, score=0.590 total time=   0.2s
[CV 1/5] END C=0.00014563484775012445, penalty=l1;, score=nan total time=   0.0s
[CV 2/5] END C=0.00014563484775012445, penalty=l1;, score=nan total time=   0.0s
[CV 3/5] END C=0.00014563484775012445, penalty=l1;, score=nan total time=   0.0s
[CV 4/5] END C=0.00014563484775012445, penalty=l1;, score=nan total time=   0.0s
[CV 5/5] END C=0.00014563484775012445, penalty=l1;, score=nan total time=   0.0s
[CV 1/5] END C=0.00014563484775012445, penalty=l2;, score=0.591 total time=   0.2s
[CV 2/5] END C=0.00014563484775012445, penalty=l2;, score=0.591 total time=   0.1s
[CV 3/5] END C=0.00014563484775012445, penalty=l2;, score=0.590 total time=   0.2s
[CV 4/5] END C=0.00014563484775012445, penalty=l2;, score=0.590 total time=   0.4s
[CV 5/5] END C=0.00014563484775012445, penalty=l2;, score=0.590 total time=   0.2s
[CV 1/5] END C=0.00021209508879201905, penalty=l1;, score=nan total time=   0.0s
[CV 2/5] END C=0.00021209508879201905, penalty=l1;, score=nan total time=   0.0s
[CV 3/5] END C=0.00021209508879201905, penalty=l1;, score=nan total time=   0.0s
[CV 4/5] END C=0.00021209508879201905, penalty=l1;, score=nan total time=   0.0s
[CV 5/5] END C=0.00021209508879201905, penalty=l1;, score=nan total time=   0.0s
[CV 1/5] END C=0.00021209508879201905, penalty=l2;, score=0.591 total time=   0.2s
[CV 2/5] END C=0.00021209508879201905, penalty=l2;, score=0.591 total time=   0.2s
[CV 3/5] END C=0.00021209508879201905, penalty=l2;, score=0.590 total time=   0.2s
[CV 4/5] END C=0.00021209508879201905, penalty=l2;, score=0.590 total time=   0.4s
[CV 5/5] END C=0.00021209508879201905, penalty=l2;, score=0.590 total time=   0.1s
[CV 1/5] END C=0.00030888435964774815, penalty=l1;, score=nan total time=   0.0s
[CV 2/5] END C=0.00030888435964774815, penalty=l1;, score=nan total time=   0.0s
[CV 3/5] END C=0.00030888435964774815, penalty=l1;, score=nan total time=   0.0s
[CV 4/5] END C=0.00030888435964774815, penalty=l1;, score=nan total time=   0.0s
[CV 5/5] END C=0.00030888435964774815, penalty=l1;, score=nan total time=   0.0s
[CV 1/5] END C=0.00030888435964774815, penalty=l2;, score=0.591 total time=   0.2s
[CV 2/5] END C=0.00030888435964774815, penalty=l2;, score=0.591 total time=   0.2s
[CV 3/5] END C=0.00030888435964774815, penalty=l2;, score=0.590 total time=   0.2s
[CV 4/5] END C=0.00030888435964774815, penalty=l2;, score=0.590 total time=   0.2s
[CV 5/5] END C=0.00030888435964774815, penalty=l2;, score=0.590 total time=   0.4s
[CV 1/5] END .C=0.0004498432668969444, penalty=l1;, score=nan total time=   0.0s
[CV 2/5] END .C=0.0004498432668969444, penalty=l1;, score=nan total time=   0.0s
[CV 3/5] END .C=0.0004498432668969444, penalty=l1;, score=nan total time=   0.0s
[CV 4/5] END .C=0.0004498432668969444, penalty=l1;, score=nan total time=   0.0s
[CV 5/5] END .C=0.0004498432668969444, penalty=l1;, score=nan total time=   0.0s
[CV 1/5] END C=0.0004498432668969444, penalty=l2;, score=0.591 total time=   0.1s
[CV 2/5] END C=0.0004498432668969444, penalty=l2;, score=0.591 total time=   0.2s
[CV 3/5] END C=0.0004498432668969444, penalty=l2;, score=0.590 total time=   0.1s
[CV 4/5] END C=0.0004498432668969444, penalty=l2;, score=0.590 total time=   0.2s
[CV 5/5] END C=0.0004498432668969444, penalty=l2;, score=0.590 total time=   0.2s
[CV 1/5] END .C=0.0006551285568595509, penalty=l1;, score=nan total time=   0.0s
[CV 2/5] END .C=0.0006551285568595509, penalty=l1;, score=nan total time=   0.0s
[CV 3/5] END .C=0.0006551285568595509, penalty=l1;, score=nan total time=   0.0s
[CV 4/5] END .C=0.0006551285568595509, penalty=l1;, score=nan total time=   0.0s
[CV 5/5] END .C=0.0006551285568595509, penalty=l1;, score=nan total time=   0.0s
```

6) MLP Classifier (Hyperparameter tuning best models)

```
ml = MLPClassifier()

parameters = {
```

```
              'solver': ['lbfgs'],
              'max_iter': [1500, 2000],
              'alpha': [0.1, 1],
              'random_state':[1,3,9]
              }

ml_grid = GridSearchCV(estimator=ml, param_grid=parameters, cv= 3, verbose = 3)
grid_result =  ml_grid.fit(X_train, y_train)

print('Best params: ', grid_result.best_params_)
print('Best score: ', grid_result.best_score_)

#MLPClassifier()
#Best params:  {'alpha': 1, 'max_iter': 1500, 'random_state': 9, 'solver': 'lbfgs'}
#Best score:  0.7366548516075522
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
[CV 1/3] END alpha=0.1, max_iter=1500, random_state=1, solver=lbfgs;, score=0.739 tot
[CV 2/3] END alpha=0.1, max_iter=1500, random_state=1, solver=lbfgs;, score=0.721 tot
[CV 3/3] END alpha=0.1, max_iter=1500, random_state=1, solver=lbfgs;, score=0.732 tot
[CV 1/3] END alpha=0.1, max_iter=1500, random_state=3, solver=lbfgs;, score=0.737 tot
[CV 2/3] END alpha=0.1, max_iter=1500, random_state=3, solver=lbfgs;, score=0.712 tot
[CV 3/3] END alpha=0.1, max_iter=1500, random_state=3, solver=lbfgs;, score=0.734 tot
[CV 1/3] END alpha=0.1, max_iter=1500, random_state=9, solver=lbfgs;, score=0.729 tot
[CV 2/3] END alpha=0.1, max_iter=1500, random_state=9, solver=lbfgs;, score=0.726 tot
[CV 3/3] END alpha=0.1, max_iter=1500, random_state=9, solver=lbfgs;, score=0.735 tot
[CV 1/3] END alpha=0.1, max_iter=2000, random_state=1, solver=lbfgs;, score=0.739 tot
[CV 2/3] END alpha=0.1, max_iter=2000, random_state=1, solver=lbfgs;, score=0.721 tot
[CV 3/3] END alpha=0.1, max_iter=2000, random_state=1, solver=lbfgs;, score=0.732 tot
[CV 1/3] END alpha=0.1, max_iter=2000, random_state=3, solver=lbfgs;, score=0.737 tot
[CV 2/3] END alpha=0.1, max_iter=2000, random_state=3, solver=lbfgs;, score=0.712 tot
[CV 3/3] END alpha=0.1, max_iter=2000, random_state=3, solver=lbfgs;, score=0.734 tot
[CV 1/3] END alpha=0.1, max_iter=2000, random_state=9, solver=lbfgs;, score=0.729 tot
[CV 2/3] END alpha=0.1, max_iter=2000, random_state=9, solver=lbfgs;, score=0.726 tot
[CV 3/3] END alpha=0.1, max_iter=2000, random_state=9, solver=lbfgs;, score=0.735 tot
[CV 1/3] END alpha=1, max_iter=1500, random_state=1, solver=lbfgs;, score=0.739 total
[CV 2/3] END alpha=1, max_iter=1500, random_state=1, solver=lbfgs;, score=0.732 total
[CV 3/3] END alpha=1, max_iter=1500, random_state=1, solver=lbfgs;, score=0.735 total
[CV 1/3] END alpha=1, max_iter=1500, random_state=3, solver=lbfgs;, score=0.741 total
[CV 2/3] END alpha=1, max_iter=1500, random_state=3, solver=lbfgs;, score=0.727 total
[CV 3/3] END alpha=1, max_iter=1500, random_state=3, solver=lbfgs;, score=0.737 total
[CV 1/3] END alpha=1, max_iter=1500, random_state=9, solver=lbfgs;, score=0.740 total
[CV 2/3] END alpha=1, max_iter=1500, random_state=9, solver=lbfgs;, score=0.730 total
[CV 3/3] END alpha=1, max_iter=1500, random_state=9, solver=lbfgs;, score=0.740 total
[CV 1/3] END alpha=1, max_iter=2000, random_state=1, solver=lbfgs;, score=0.739 total
[CV 2/3] END alpha=1, max_iter=2000, random_state=1, solver=lbfgs;, score=0.732 total
[CV 3/3] END alpha=1, max_iter=2000, random_state=1, solver=lbfgs;, score=0.735 total
[CV 1/3] END alpha=1, max_iter=2000, random_state=3, solver=lbfgs;, score=0.741 total
[CV 2/3] END alpha=1, max_iter=2000, random_state=3, solver=lbfgs;, score=0.727 total
[CV 3/3] END alpha=1, max_iter=2000, random_state=3, solver=lbfgs;, score=0.737 total
[CV 1/3] END alpha=1, max_iter=2000, random_state=9, solver=lbfgs;, score=0.740 total
[CV 2/3] END alpha=1, max_iter=2000, random_state=9, solver=lbfgs;, score=0.730 total
[CV 3/3] END alpha=1, max_iter=2000, random_state=9, solver=lbfgs;, score=0.740 total
Best params:  {'alpha': 1, 'max_iter': 1500, 'random_state': 9, 'solver': 'lbfgs'}
Best score:  0.7366548516075522
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron

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
```

7) XGB Classifier (Hyperparameter tuning best models)

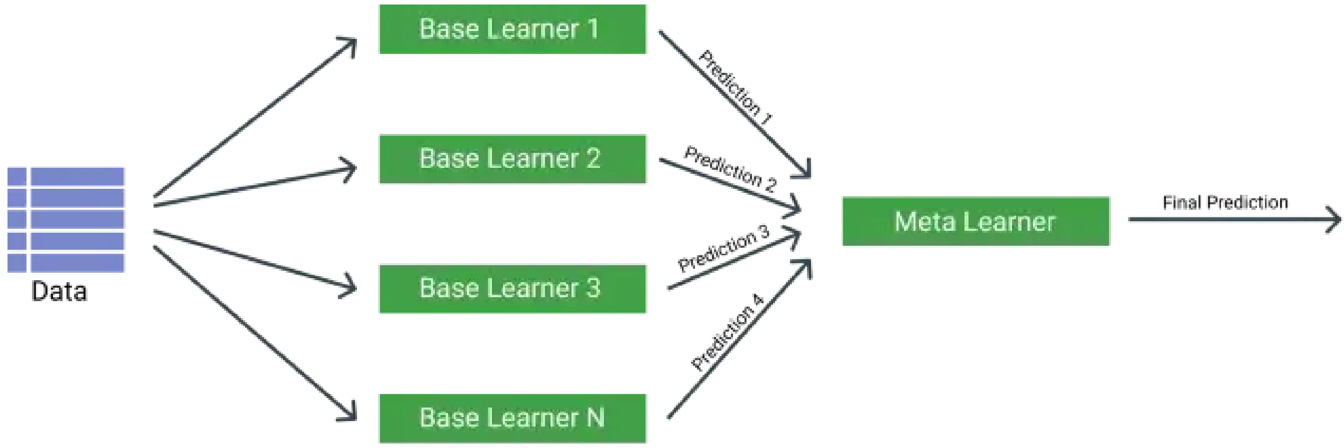
```
xgb = XGBClassifier(objective= 'multi:softprob')

parameters = {
    'max_depth': [1, 5 ,10],
    'n_estimators': [10, 100, 200],
    'learning_rate': [0.1, 0.05],
    'subsample': [0.6, 1.0],
}

"""
parameters =    {
            'max_depth': [3, 4, 5],
            'learning_rate': [0.1, 0.2, 0.3],
            'n_estimators': [50, 100, 150],
            'gamma': [0, 0.1, 0.2],
            'min_child_weight': [0, 0.5, 1],
            'max_delta_step': [0],
            'subsample': [0.7, 0.8, 0.9, 1],
            'colsample_bytree': [0.6, 0.8, 1],
            'colsample_bylevel': [1],
            'reg_alpha': [0, 1e-2, 1, 1e1],
            'reg_lambda': [0, 1e-2, 1, 1e1],
            'base_score': [0.5]
            }
"""

xgb_grid = GridSearchCV(estimator=xgb, param_grid=parameters , cv= 5, refit = True, verbos
grid_result =  xgb_grid.fit(X_train, y_train)

print('Best params: ', grid_result.best_params_)
print('Best score: ', grid_result.best_score_)

#XGBClassifier
#Best params:  {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 200, 'subsample': 1
#Best score:  0.739309499915284
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV 1/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=0.6;, scor
[CV 2/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=0.6;, scor
[CV 3/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=0.6;, scor
[CV 4/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=0.6;, scor
[CV 5/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=0.6;, scor
[CV 1/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=1.0;, scor
[CV 2/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=1.0;, scor
[CV 3/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=1.0;, scor
```

```
[CV 4/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=1.0;, scor
[CV 5/5] END learning_rate=0.1, max_depth=1, n_estimators=10, subsample=1.0;, scor
[CV 1/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=0.6;, sco
[CV 2/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=0.6;, sco
[CV 3/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=0.6;, sco
[CV 4/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=0.6;, sco
[CV 5/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=0.6;, sco
[CV 1/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=1.0;, sco
[CV 2/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=1.0;, sco
[CV 3/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=1.0;, sco
[CV 4/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=1.0;, sco
[CV 5/5] END learning_rate=0.1, max_depth=1, n_estimators=100, subsample=1.0;, sco
[CV 1/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=0.6;, sco
[CV 2/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=0.6;, sco
[CV 3/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=0.6;, sco
[CV 4/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=0.6;, sco
[CV 5/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=0.6;, sco
[CV 1/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=1.0;, sco
[CV 2/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=1.0;, sco
[CV 3/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=1.0;, sco
[CV 4/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=1.0;, sco
[CV 5/5] END learning_rate=0.1, max_depth=1, n_estimators=200, subsample=1.0;, sco
[CV 1/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=0.6;, scor
[CV 2/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=0.6;, scor
[CV 3/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=0.6;, scor
[CV 4/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=0.6;, scor
[CV 5/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=0.6;, scor
[CV 1/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=1.0;, scor
[CV 2/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=1.0;, scor
[CV 3/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=1.0;, scor
[CV 4/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=1.0;, scor
[CV 5/5] END learning_rate=0.1, max_depth=5, n_estimators=10, subsample=1.0;, scor
[CV 1/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=0.6;, sco
[CV 2/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=0.6;, sco
[CV 3/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=0.6;, sco
[CV 4/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=0.6;, sco
[CV 5/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=0.6;, sco
[CV 1/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=1.0;, sco
[CV 2/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=1.0;, sco
[CV 3/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=1.0;, sco
[CV 4/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=1.0;, sco
[CV 5/5] END learning_rate=0.1, max_depth=5, n_estimators=100, subsample=1.0;, sco
[CV 1/5] END learning_rate=0.1, max_depth=5, n_estimators=200, subsample=0.6;, sco
[CV 2/5] END learning_rate=0.1, max_depth=5, n_estimators=200, subsample=0.6;, sco
[CV 3/5] END learning_rate=0.1, max_depth=5, n_estimators=200, subsample=0.6;, sco
[CV 4/5] END learning_rate=0.1, max_depth=5, n_estimators=200, subsample=0.6;, sco
[CV 5/5] END learning_rate=0.1, max_depth=5, n_estimators=200, subsample=0.6;, sco
[CV 1/5] END learning_rate=0.1, max_depth=5, n_estimators=200, subsample=1.0;, sco
```

8) Stacking (Hyperparameter tuning best models)

O Stacking (empilhamento) é uma técnica que usa vários modelos de regressão ou classificação e usa sua saída como entrada para o meta-classificador/regressor.




```
#Stacking
# Building and fitting
from sklearn.ensemble import StackingClassifier


##LogisticRegression('C': 7.9060432109076855, 'penalty': 'l2')
# Best params:  {'C': 7.9060432109076855, 'penalty': 'l2'}
# Best score:  0.7496352870132635

# GradientBoostingClassifier()
# Best params:  {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
# Best score:  0.7393144135896653

# RandomForestClassifier()
# Best params:  {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'min_samp
# Best score:  0.6198778352687666


#MLPClassifier()
#Best params:  {'alpha': 1, 'max_iter': 1500, 'random_state': 9, 'solver': 'lbfgs'}
#Best score:  0.7366548516075522

#XGBClassifier
#Best params:  {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 200, 'subsample': 1
#Best score:  0.739309499915284

lr = LogisticRegression()

"""estimators = [
    ('rf', RandomForestClassifier(random_state=42)),
    ('GBC', GradientBoostingClassifier(random_state=42)),
    ('lr', LogisticRegression(random_state=10, max_iter=500)),
    ('MLP', MLPClassifier(random_state=42)),
    ('xgb', XGBClassifier(eval_metric="mlogloss",random_state=42))
    ]"""

estimators = [
    ('rf', RandomForestClassifier(criterion='entropy', max_depth= 8, max_features= 'auto',
    ('GBC', GradientBoostingClassifier(learning_rate= 0.1, max_depth= 3, n_estimators= 500
    ('lr', LogisticRegression(C= 7.9060432109076855, penalty= 'l2')),
```

```
    ('MLP', MLPClassifier(alpha = 1, max_iter= 1500, random_state= 9, solver= 'lbfgs')),
    ('xgb', XGBClassifier(learning_rate= 0.1, max_depth= 10, n_estimators= 200, subsample=
    ]


clf = StackingClassifier(estimators=estimators, final_estimator=lr, verbose = 1)#, n_jobs=

clf.fit(X_train, y_train).score(X_val, y_val)
```

```
/usr/local/lib/python3.8/dist-packages/joblib/externals/loky/process_executor.py:700

A worker stopped while some jobs were given to the executor. This can be caused by a

/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: Converg

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

0.7799174690508941
```

5. Avaliação do Modelo (Modelo selecionado - Stacking): ✅

Com a etapa de modelagem finalizada é possivel avaliar se o se o resultado corresponde à expectativa do projeto.

```
print(classification_report(y_val, clf.predict(X_val)))
```

```
              precision    recall  f1-score   support

           0       0.79      0.62      0.69       180
           1       0.79      0.91      0.84       876
           2       0.74      0.58      0.65       398

    accuracy                           0.78      1454
   macro avg       0.77      0.70      0.73      1454
weighted avg       0.78      0.78      0.77      1454
```

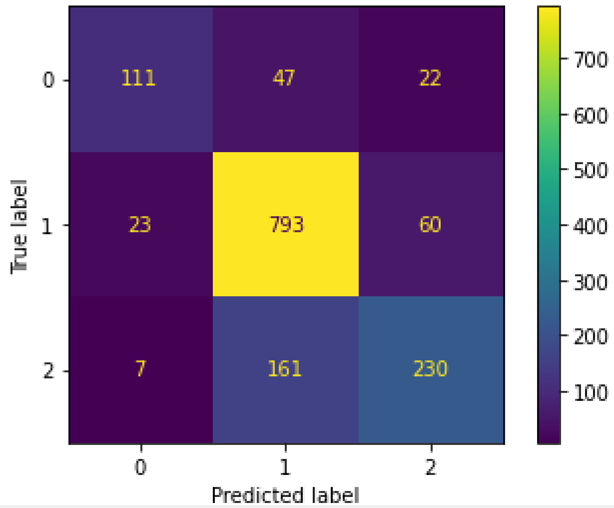
```
confusion_matrix(y_val, clf.predict(X_val))
```

```
array([[111,  47,  22],
       [ 23, 793,  60],
       [  7, 161, 230]])
```

```
plt.figure(figsize=(16,8))
plot_confusion_matrix(clf, X_val, y_val)
```

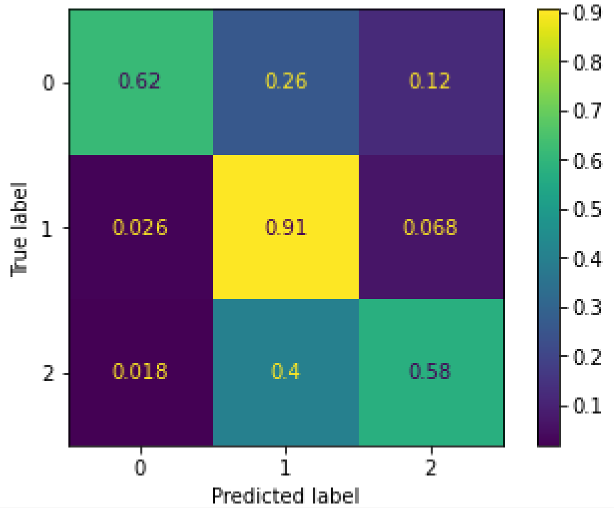
```
plot_confusion_matrix(clf, X_val, y_val)
plt.show()
```

```
Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is dep
<Figure size 1152x576 with 0 Axes>
```




```
plot_confusion_matrix(clf, X_val, y_val, normalize = 'true')
```

```
Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is dep
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f62235bce50>
```



Observações ⚡

O modelo utilizando Stacking foi o que obteve as melhores metricas de avaliação chegando ao F-score de 0.78 global.

Desta forma foi possivel chegar a um modelo eficiente e robusto.

XAI

Explain Model Predictions Using SHAP Partition Explainer

O Partition Explainer calcula valores SHAP tentando recursivamente uma hierarquia diferente de recursos de dados. Tentaremos explicar as previsões corretas e incorretas feitas pelo nosso modelo usando este explicador.

```
print("SHAP Version : {}".format(shap.__version__))
```

```
SHAP Version : 0.41.0
```

```
shap.initjs()
```

```
explainer = shap.KernelExplainer(clf.predict, X_train[0:15])
shap_values = explainer.shap_values(X_train[0:15])
shap.summary_plot(shap_values, X_train[0:15], plot_type="bar")
```

100% 15/15 [01:21<00:00, 5.56s/it]

```
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())
```

```
If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessi

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLarsIC())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parame

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
The default of 'normalize' will be set to False in version 1.2 and deprecated in v
```