Ministério da Educação
Universidade Tecnológica Federal do Paraná
Pró-Reitoria de Pesquisa e Pós-Graduação

RELATÓRIO FINAL DE ATIVIDADES CIENTÍFICA

vinculado ao projeto

Final da Disciplina de Tópicos Avançados em Bioinformática

José Luiz Vilas Boas

Externo

PPGAB

Data de ingresso no programa: NA

Prof. Dr. Alexandre Rossi Paschoal

1.03.01.01-1 — Área do CNPq

CAMPUS CORNÉLIO PROCÓPCIO, 2021

JOSÉ LUIZ VILAS BOAS
ALEXANDRE ROSSI PASCHOAL

CORNÉLIO PROCÓPCIO, 2021

SUMÁRIO

PROBLEMA A SER SOLUCIONADO 4

DESCRIÇÃO DOS DADOS 4

ANÁLISE EXPLORATÓRIA DOS DADOS 5
Pandas 5
Gráfico de Scatter Matriz 5
Explorando os dados categóricos (qualitativos) 6
Atributo Gender 6
Atributo Vehicle_Age 6
Atributo Vehicle_Damage 7
Atributo Driving_License 8
Atributo Region_Code 8
Atributo Previously_Insured 9
Atributo Policy_Sales_Channel 9
Atributo o atributo alvo - Response 9
Explorando os dados quantitativos. 10
Atributo Age 11
Atributo Annual_Premium 11
Atributo Vintage 13
Finalizando a análise exploratória. Verificando a existência de valores nulos(NAN). . . 14

CORREÇÃO DOS DADOS E EXECUÇÃO DOS MODELOS DE MACHINE LEARNING 15
Instalação de Pacotes e Importação das Bibliotecas 15
Abrindo o arquivo em um Data Frame 16
Separar os atributos previsores da classe 16
Atributo Gender 16
Atributo Vehicle_Age 16
Atributo Vehicle_Damage 17
Resultado Final da Conversão 17
Criando os previsores definitivos 17
Balanceamento dos Dados 18
Função converteVetor(m) 18

MÉTRICAS DE AVALIAÇÃO EM MACHINE LEARNING 19

EXECUÇÃO DOS MODELOS DE MACHINE LEARNING 20
K-Nearest Neighbor(KNN) 20
Random Forest(RF) 22
Decision Tree (Árvores de Decisão) 25
Gradient Boosting 27
Support Vector Machine(SVM) ou Máquina Vetor de Suporte 29
XGBoost 32

RESULTADOS FINAIS 35

CONCLUSÕES **38**

ANEXO A - CÓDIGO-FONTE DA ANÁLISE EXPLORATÓRIA DOS DADOS. . . **40**

ANEXO B - CÓDIGO-FONTE DA CORREÇÃO DOS DADOS E EXECUÇÃO DOS ALGORITMOS K-NEAREST NEIGHBOR(KNN), RANDOM FOREST, DECISION TREE, GRADIENT BOOSTING, SUPPORT VECTOR MACHINE(SVM) E XGBOOST **45**

ANEXO C - CÓDIGO-FONTE DO ALGORITMO MÁQUINA VETOR DE SUPORTE(SVM) **59**

PROBLEMA A SER SOLUCIONADO

O enunciado e os dados selecionados para esse projeto foram retirados da plataforma Kaggle[6]. Portanto, o exemplo que será apresentado a seguir foi retirada dessa plataforma: "Our client is an Insurance company that has provided Health Insurance to its customers now they need your help in building a model to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the company.

An insurance policy is an arrangement by which a company undertakes to provide a guarantee of compensation for specified loss, damage, illness, or death in return for the payment of a specified premium. A premium is a sum of money that the customer needs to pay regularly to an insurance company for this guarantee.

For example, you may pay a premium of Rs. 5000 each year for a health insurance cover of Rs. 200,000/- so that if, God forbid, you fall ill and need to be hospitalised in that year, the insurance provider company will bear the cost of hospitalisation etc. for upto Rs. 200,000. Now if you are wondering how can company bear such high hospitalisation cost when it charges a premium of only Rs. 5000/-, that is where the concept of probabilities comes in picture. For example, like you, there may be 100 customers who would be paying a premium of Rs. 5000 every year, but only a few of them (say 2-3) would get hospitalised that year and not everyone. This way everyone shares the risk of everyone else.

Just like medical insurance, there is vehicle insurance where every year customer needs to pay a premium of certain amount to insurance provider company so that in case of unfortunate accident by the vehicle, the insurance provider company will provide a compensation (called ‘sum assured’) to the customer.

Building a model to predict whether a customer would be interested in Vehicle Insurance is extremely helpful for the company because it can then accordingly plan its communication strategy to reach out to those customers and optimise its business model and revenue.

Now, in order to predict, whether the customer would be interested in Vehicle insurance, you have information about demographics (gender, age, region code type), Vehicles (Vehicle Age, Damage), Policy (Premium, sourcing channel) etc."

DESCRIÇÃO DOS DADOS

A descrição das variáveis utilizadas encontram-se disponíveis a seguir:

- id: Unique ID for the customer;
- Gender: Gender of the customer;
- Age: Age of the customer;
- Driving_License: 0 = Customer does not have DL, 1 = Customer already has DL;
- Region_Code: Unique code for the region of the customer;
- Previously_Insured: 1 = Customer already has Vehicle Insurance, 0 = Customer doesn’t have Vehicle Insurance;
- Vehicle_Age: Age of the Vehicle;
- Vehicle_Damage: 1 = Customer got his or her vehicle damaged in the past. 0 = Customer didn’t get his or her vehicle damaged in the past;
- Annual_Premium: The amount customer needs to pay as premium in the year;
- PolicySalesChannel: Anonymized Code for the channel of outreaching to the customer ie. Different Agents, Over Mail, Over Phone, In Person, etc.
- Vintage: Number of Days, Customer has been associated with the company; : 1 = Customer is interested, 0 = Customer is not interested;

ANÁLISE EXPLORATÓRIA DOS DADOS

Pandas. Lendo o arquivo com pandas e gerando um DataFrame.
#abrindo o arquivo
>df_train = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/train.csv",sep=",")
#Listando os dados
>df_train.head()
#A Figura 1 apresenta o Data Frame gerado.

```
df_train = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/train.csv",sep=",")
df_train.head()
```

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0

Figura 1. Data Frame gerado.

Gráfico de Scatter Matriz. O Gráfico de Scatter Matriz será implementado nos dados brutos. Assim como o Scatter plots que consiste em um tipo de gráfico comumente utilizado para observar o comportamento entre duas variáveis, o Scatter Matriz utiliza todo o DataFrame, pois irá gerar os scatter plots para todos os pares possíveis de atributos. O gráfico pode ser observado na Figura 2 .

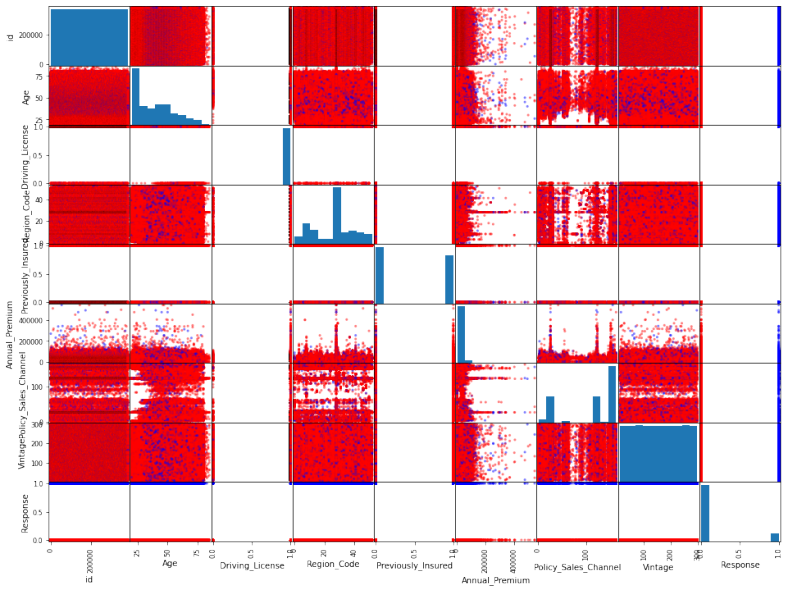

Figura 2. Gráfico de Scatter Matriz.

Os comandos para gerar o gráfico são listados a seguir:
#Scatter Matriz
#Criando um dicionario para mapear cada classe para uma cor
>classe_cor = {0 : 'red',1 : 'blue'}
#Criando uma lista com as cores de cada exemplo

>cores = [classe_cor[nome] for nome in df_train.Response]
#Gerando matriz de scatter plots
>pd.plotting.scatter_matrix(df_train, color=cores,figsize=(16,12))

Explorando os dados categóricos (qualitativos). Para esses utilizarei os gráficos de barras, pizza e linha para obter uma visão mais clara da distribuição dos dados. Não utilizei o histograma e boxplot, pois esses são para dados quantitativos.

Atributo Gender. Agrupando os dados de gênero dos clientes.
>group = df_train.groupby(['Gender']).size()
>group
Gender
Female 175020
Male 206089
dtype: int64
#Visualizar o dados de gênero dos clientes em um gráfico de barras(Figura 3).
>srn.countplot(df_train['Gender'])

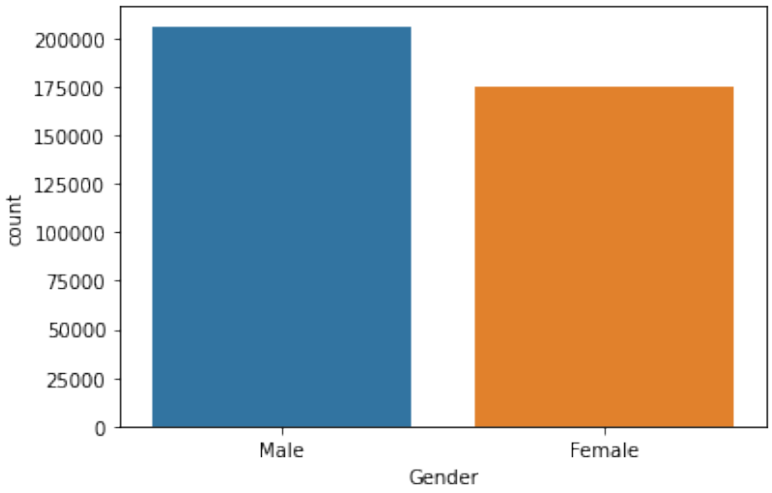

Figura 3. Gráfico de barras do atributo Gender.

Atributo Vehicle_Age. Agrupando a idade dos veículos.
>group = df_train.groupby(['Vehicle_Age']).size()
>group
Vehicle_Age
1-2 Year 200316
< 1 Year 164786
> 2 Years 16007
dtype: int64
#Visualizar a idade dos clientes em um gráfico de pizza (Figura 4).
>group.plot.pie(autopct='%.1f%%')

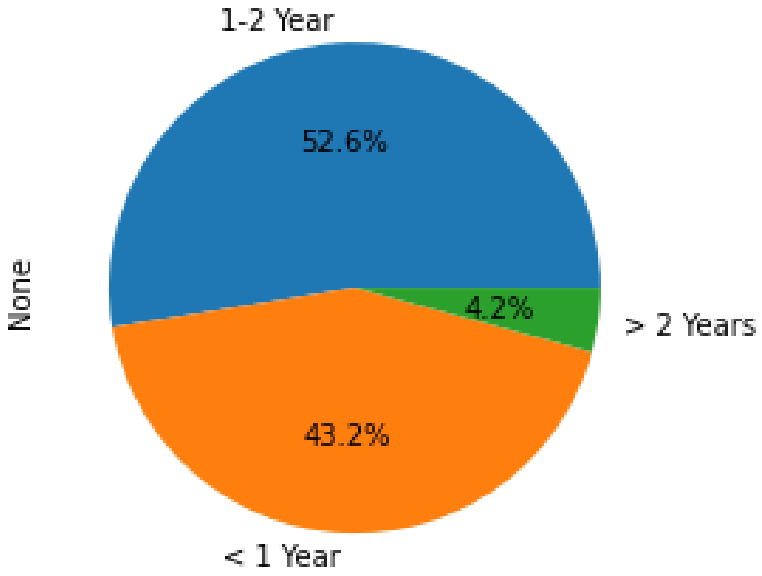

Figura 4. Gráfico de pizza do atributo Vehicle_Age.

Atributo Vehicle_Damage. Agrupando os dados de dano do veículo.
>group = df_train.groupby(['Vehicle_Damage']).size()
>group
Vehicle_Damage
No 188696
Yes 192413
dtype: int64
#Visualizar os dados de dano do veículo em um gráfico de barras (Figura 5).
>srn.countplot(df_train['Vehicle_Damage'])

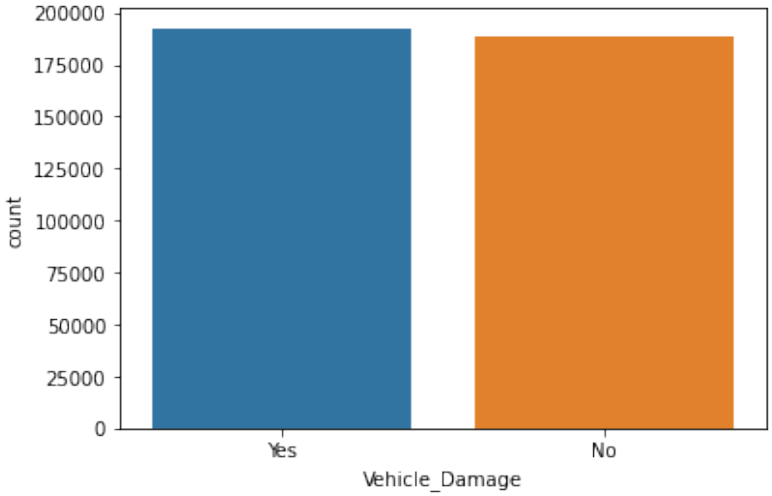

Figura 5. Gráfico de barras do atributo Vehicle_Damage.

Atributo Driving_License. Agrupando os dados de licença do veículo.
>group = df_train.groupby([’Driving_License’]).size()
>group
Driving_License
0 812
1 380297
dtype: int64
#Visualizar os dados de licença em um gráfico de pizza (Figura 6).
>group.plot.pie(autopct=’%.1f%%’)

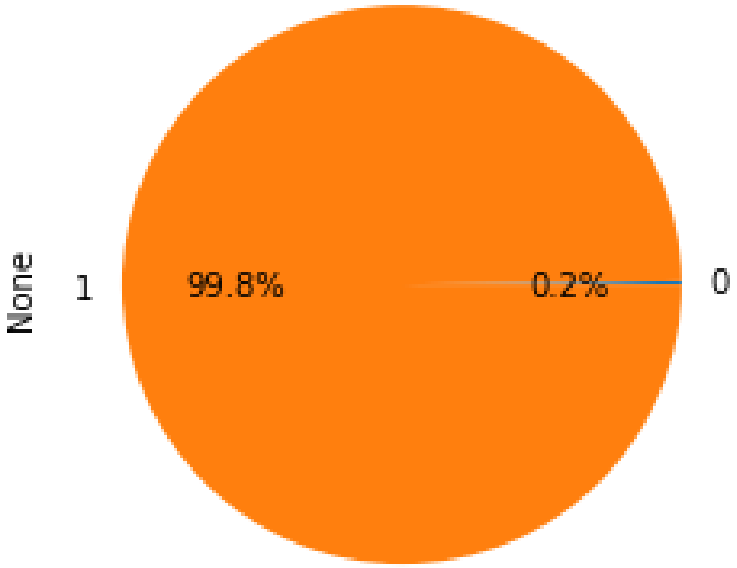

Figura 6. Gráfico de pizza do atributo Driving_License.

Atributo Region_Code. Agrupando os dados de código da região.
>group = df_train.groupby([’Region_Code’]).size()
>group
Region_Code
0.0 2021
1.0 1008
...
52.0 267
dtype: int64
#Visualização os dados de código da região em um gráfico de linha (Figura 7). Essa opção foi utilizada devido a quantidade enorme de agrupamentos.
>group.plot.line(color=’blue’)

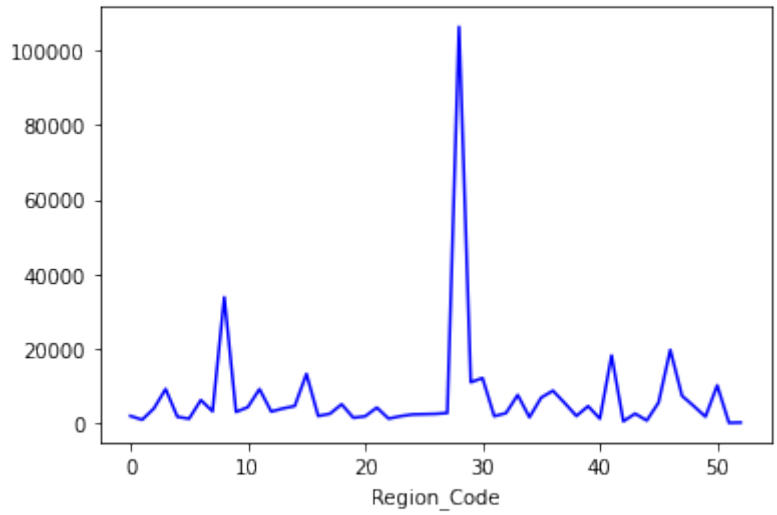

Figura 7. Gráfico de linha do atributo Region_Code.

Atributo Previously_Insured. Agrupando os dados de segurado anteriormente.
>group = df_train.groupby([’Previously_Insured’]).size()
>group
Previously_Insured
0 206481
1 174628
dtype: int64
#Analisando o campo Previously_Insured em um gráfico de pizza (Figura 8).
>group.plot.pie(autopct=’%.1f%%’)

Atributo Policy_Sales_Channel. Agrupando os dados de política de canal de vendas.
>group = df_train.groupby([’Policy_Sales_Channel’]).size()
>group
Policy_Sales_Channel
1.0 1074
2.0 4
3.0 523
...
163.0 2893
Length: 155, dtype: int64
#Visualizar os dados de código de política do canal de vendas em um gráfico de linha(Figura 9).
>group.plot.line(color=’red’)

Atributo o atributo alvo - Response. Agrupando os dados.
>group = df_train.groupby([’Response’]).size()
>group
Response
0 334399

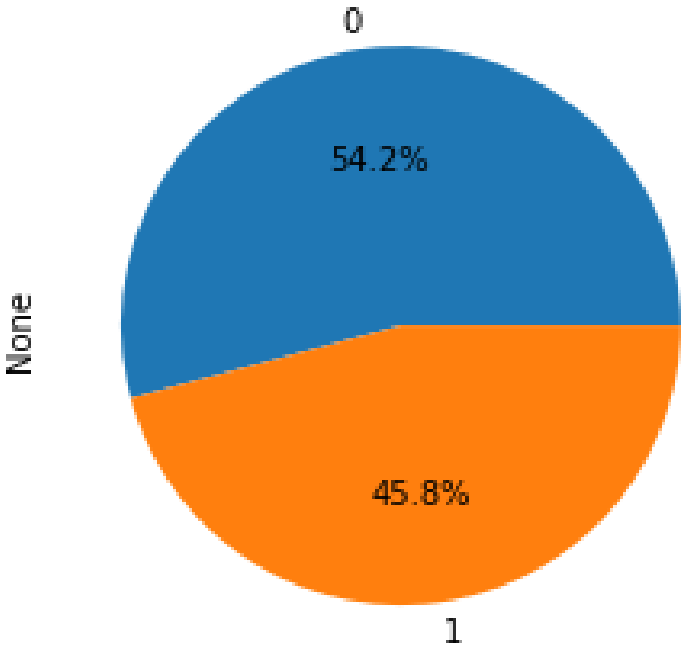

Figura 8. Gráfico de pizza do atributo Previously_Insured.

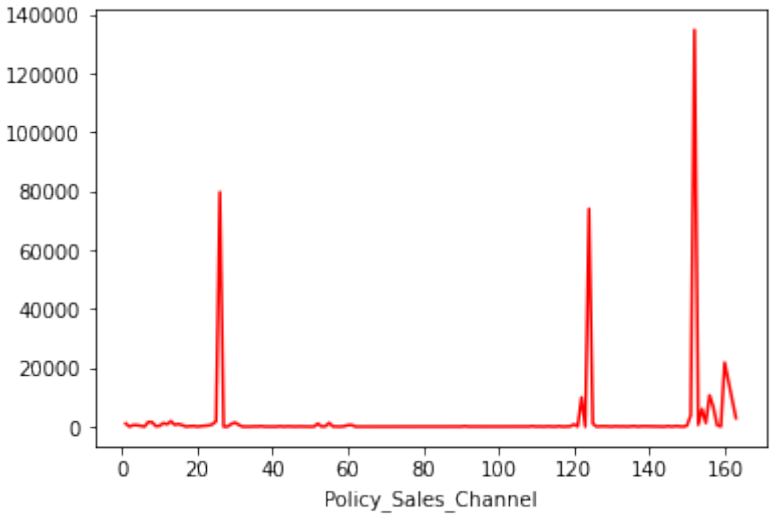

Figura 9. Gráfico de linha do atributo Policy_Sales_Channel.

1 46710
dtype: int64
#Visualizar os dados da classe um gráfico de barras(Figura 10).
>srn.countplot(df_train['Response'])

Ao observar, contata-se que os dados estão desbalanceados (Figura 10), portanto, um balanceamento antes de executar o modelo será necessário. Caso contrário, ele ficará enviesado.

Explorando os dados quantitativos. Para esses utilizarei o Histograma e Boxplot e, assim obter uma visão mais clara da distribuição dos dados

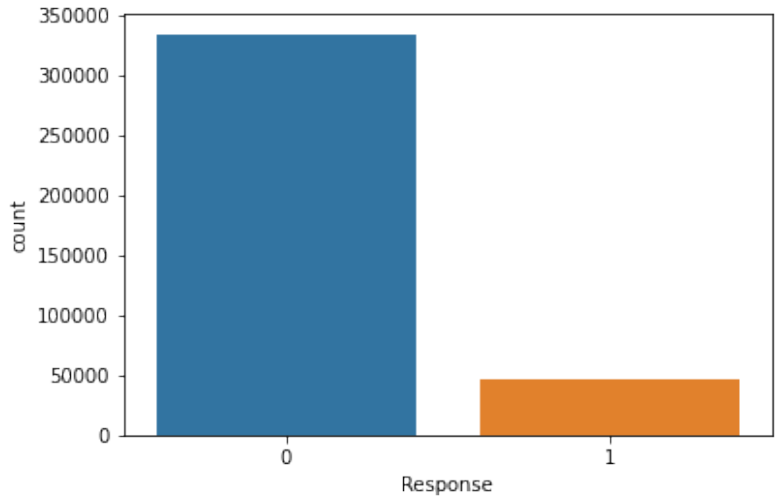

Figura 10. Gráfico de barra do atributo Response.

Atributo Age. Explorando a variável idade.
>df_train['Age'].describe()
count 381109.000000
mean 38.822584
std 15.511611
min 20.000000
25% 25.000000
50% 36.000000
75% 49.000000
max 85.000000
Name: Age, dtype: float64
Ao observar esses dados de idade, constata-se que a mediana e média estão próximas, portanto, isso indica que não exite grandes outliers, ou seja, existe uma distribuição regular dos dados. Também no valor mínimo é possível identificar que não existe valor igual a 0.Para ter uma visão completa da distribuição, será utilizado o histograma e o boxplot. Alguns valores importantes: mean(média), std(desvio padrão), 50% (mediana) serão explorados.

#Visualizar os dados de idade por um boxplot(Figura 11).
>srn.boxplot(df_train['Age']).set_title('Age')

O Boxplot(Figura 11) indica que a idade está entre 20 e 85 anos e não há outliers.
#Visualizar os dados de idade em um histograma (Figura 12).
>srn.distplot(df_train['Age']).set_title('Age')
Para a idade foi gerado um histograma assimétrico (Figura 12). A frequência decresce bruscamente em um dos lados de forma gradual no outro, produzindo uma calda mais longa em um dos lados. Assimetria à direita, mediana é inferior a média. Assimetria à esquerda, mediana é superior à média. O gráfico confirma o que foi calculado na função describe.

Atributo Annual_Premium. Explorando a variável prêmio anual.
>df_train['Annual_Premium'].describe()
count 381109.000000
mean 30564.389581

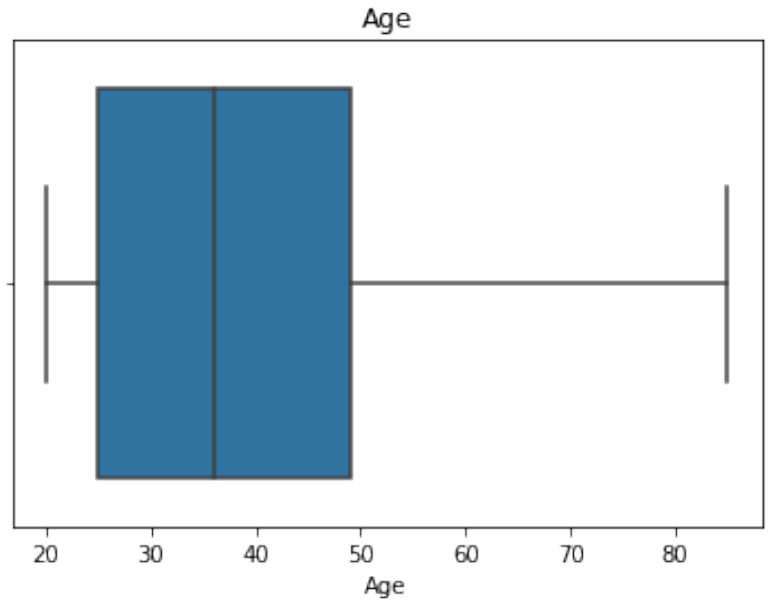

Figura 11. Gráfico boxplot do atributo Age.

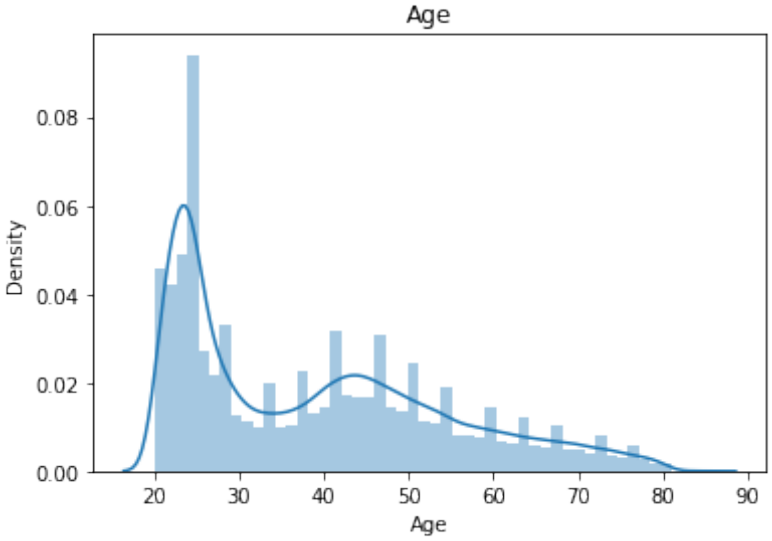

Figura 12. Gráfico histograma do atributo Age.

std 17213.155057
min 2630.000000
25% 24405.000000
50% 31669.000000
75% 39400.000000
max 540165.000000
Name: Annual_Premium, dtype: float64
#Visualizar os dados de prêmio anual por um boxplot(Figura 13).
>srn.boxplot(df_train['Annual_Premium']).set_title('Annual_Premium')

Analisando o boxplot (Figura 13) pode-se observar que há outliers. Nesse gráfico existe um desvio padrão à direita.

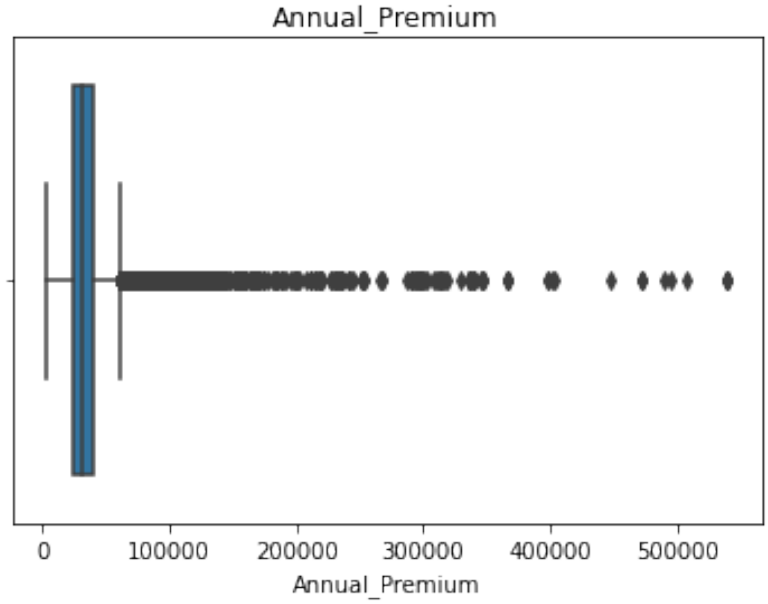

Figura 13. Gráfico boxplot do atributo Annual_Premium.

#Visualizar os dados de prêmio anual em um histograma (Figura 14).
>srn.distplot(df_train['Annual_Premium']).set_title('Annual_Premium')
Histograma com dois picos (Figura 14), indica que houve fusões de dados com médias, obtidos

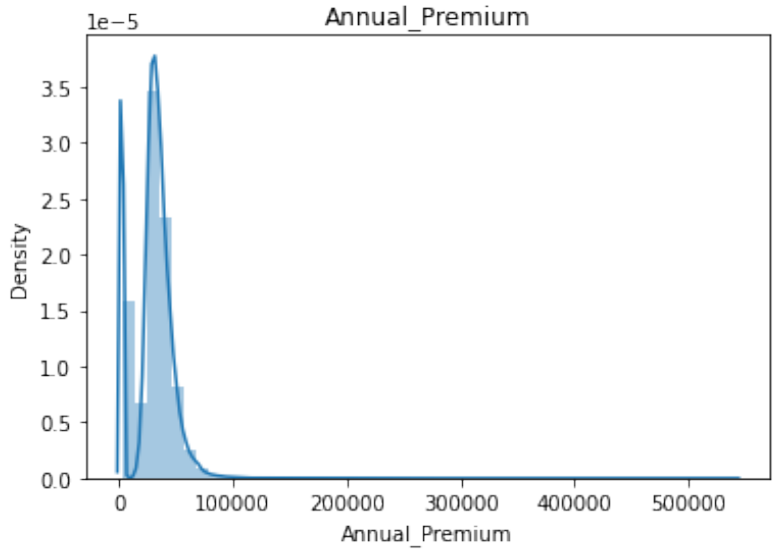

Figura 14. Gráfico histograma do atributo Annual_Premium.

em duas condições distintas. Esse gráfico responderia o desvio padrão do boxplot, contudo, para que não posso causar um enviesamento uma normalização será necessária. Isso será realizado na correção de dados.

Atributo Vintage. Explorando a variável vintage.
>df_train['Vintage'].describe()
count 381109.000000
mean 154.347397

std 83.671304
min 10.000000
25% 82.000000
50% 154.000000
75% 227.000000
max 299.000000
Name: Vintage, dtype: float64
#Visualizar os dados de vintage por um boxplot(Figura 15).
>srn.boxplot(df_train['Vintage']).set_title('Vintage')

Analisando o boxplot (Figura 15) pode-se observar que não há outliers.

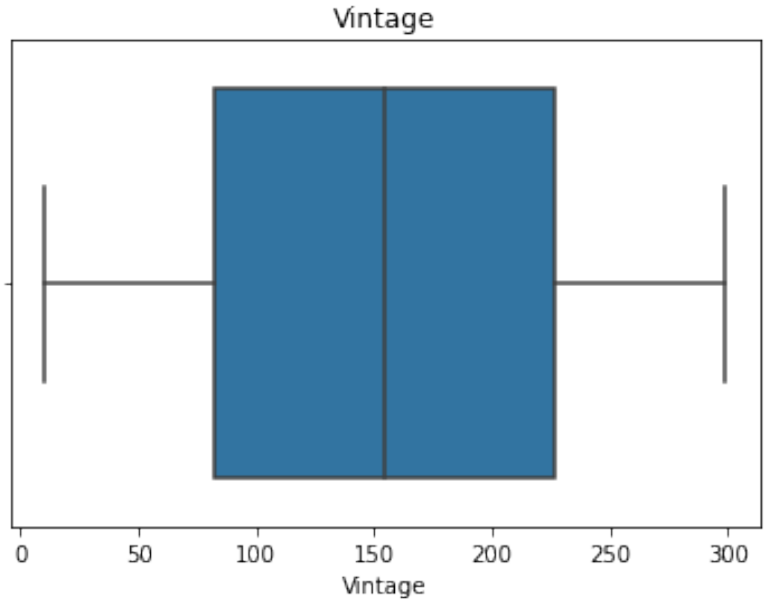

Figura 15. Gráfico boxplot do atributo Vintage.

#Visualizar os dados de vintage em um histograma (Figura 16).
>srn.distplot(df_train['Vintage']).set_title('Vintage')
Histograma do tipo platô ou achatado (Figura 16), indica que os dados são combinados de várias distribuições com médias diferentes.

Finalizando a análise exploratória. Verificando a existência de valores nulos(NAN).. #Verificar os valores nulos
#Eliminar a valores nulos
>df_train.isnull().sum()
id 0
Gender 0
Age 0
Driving_License 0
Region_Code 0
Previously_Insured 0
Vehicle_Age 0
Vehicle_Damage 0
Annual_Premium 0

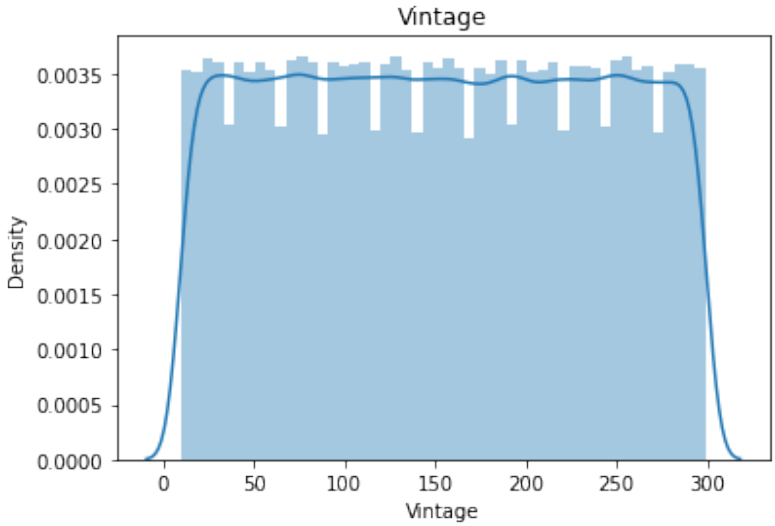

Figura 16. Gráfico histograma do atributo Vintage.

Policy_Sales_Channel 0
Vintage 0
Response 0
dtype: int64

Não há nenhum campo nulo.

CORREÇÃO DOS DADOS E EXECUÇÃO DOS MODELOS DE MACHINE LEARNING

Instalação de Pacotes e Importação das Bibliotecas. Correção dos dados por meio do levantamento das necessidades tratadas na AED.

```
#Instalando classificador o XGBoost
>!pip install xgboost
#Importando as bibliotecas
>import numpy as np
>import pandas as pd
>import seaborn as srn
>import statistics as sts
>import matplotlib.pyplot as plt
>from google.colab import files
>from sklearn.preprocessing import LabelEncoder, OneHotEncoder
>from sklearn.model_selection import train_test_split
>from sklearn.metrics import cohen_kappa_score,confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,roc_curve,auc,r2_score
>from sklearn.neighbors import KNeighborsClassifier
>from scipy import stats
>from sklearn.model_selection import cross_val_score
>from imblearn.metrics import specificity_score
>import matplotlib.pyplot as pyplot
>from sklearn.decomposition import PCA
>from sklearn.preprocessing import StandardScaler
```

>from xgboost import XGBClassifier
>from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
>from sklearn.tree import DecisionTreeClassifier
>from sklearn.svm import SVC
>from imblearn.over_sampling import SMOTE

Abrindo o arquivo em um Data Frame. Não foi necessário realizar o upload do arquivo, pois o mesmo estava disponível no GoogleDrive.
#Abrindo o arquivo em um Data Frame
>df_train = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/train.csv",sep=",")
#Listando os dados
>df_train.head()
#A Figura 17 apresenta o Data Frame gerado.

```
df_train = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/train.csv",sep=",")
df_train.head()
```

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0

Figura 17. Data Frame gerado.

Separar os atributos previsores da classe. Todos os passos realizados a seguir visam a correção de inconsistências que podem influenciar o aprendizado do modelo. Para evitar o overffiting, esses serão demonstrados na sequência e o resultado final será visualizado em uma imagem no término de todo esse processo. #Separando os previsores em um temporário
>previsores_df = df_train.iloc[:,1:11]
#Aqui quer dizer que irei percorrer todas as linas inserir em classe a coluna 12
>classe = df_train.iloc[:,11].values

Atributo Gender. Converter para número com o One-Hot Encoder. O One-Hot Encoder foi selecionado pois ele atribui valores numéricos sem dar um peso ou hierarquia. Nessa estratégia, cada valor de categoria é convertido em uma nova coluna e atribuído um valor 1 ou 0 (notação para verdadeiro / falso) à coluna.
#OneHotEncoder com a função get_dummies do pandas.
#Agora tentando remodelar com (-1, 1). Fornecemos coluna como 1, mas linhas como desconhecidas. Dua novas colunas serão geradas
>previsores_df = pd.concat([previsores_df,pd.get_dummies(previsores_df['Gender'], prefix='Gender')],axis=1)
#Deletendo a coluna antiga Gender
>previsores_df.drop(['Gender'],axis=1, inplace=True)

Atributo Vehicle_Age. Converter para número o Vehicle_Age. Usarei o encoder.
#Transformação dos atributos categóricos em atributos numéricos..

```
>labelencoder = LabelEncoder()
>previsores_df['Vehicle_Age'] = labelencoder.fit_transform(previsores_df['Vehicle_Age'])
#Vizualizar o resultado
>group = previsores_df.groupby(['Vehicle_Age']).size()
>group
Vehicle_Age
0 200316
1 164786
2 16007
dtype: int64
```

Atributo Vehicle_Damage. Será realizada a conversão para número manualmente.

```
>previsores_df.loc[previsores_df['Vehicle_Damage']=='No','Vehicle_Damage'] = '0'
>previsores_df.loc[previsores_df['Vehicle_Damage']=='Yes','Vehicle_Damage'] = '1'
>group = previsores_df.groupby(previsores_df['Vehicle_Damage']).size()
>group
Vehicle_Damage
0 188696
1 192413
dtype: int64
```

Resultado Final da Conversão. A Figura 18 exibe o resultado final da conversão para números.

	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Gender_Female	Gender_Male
0	44	1	28.0	0	2	1	40454.0	26.0	217	0	1
1	76	1	3.0	0	0	0	33536.0	26.0	183	0	1
2	47	1	28.0	0	2	1	38294.0	26.0	27	0	1
3	21	1	11.0	1	1	0	28619.0	152.0	203	0	1
4	29	1	41.0	1	1	0	27496.0	152.0	39	1	0

Figura 18. Resultado final da conversão.

Criando os previsores definitivos. Percorrendo todas as linhas inserir em previsores_df nas colunas de 0 a 10.

```
>previsores = previsores_df.iloc[:,0:11].values #Imprimindo a classe e os previsores
>print(previsores)
>print(classe)
[[44128.0...21701]
[76 1 3.0 ... 183 0 1]
[47128.0...2701]
...
[21130.0...16101]
[68 1 14.0 ... 74 1 0]
[46129.0...23701]]
[1 0 1 ... 0 0 0]
```

Balanceamento dos Dados. Como visualizado na AED, os dados encontram-se desbalanceados, portanto, uma correção será realizada.

#Executando o método SMOTE
>smt = SMOTE()
>previsores,classe = smt.fit_sample(previsores,classe)
#Fazendo a contagem para analisar como ficou e visualizando no gráfico(Figura 19)
>yi = classe.astype(int)
>np.bincount(yi)
array([334399, 334399])
#visualização em um gráfico
>ax = srn.countplot(x=yi)

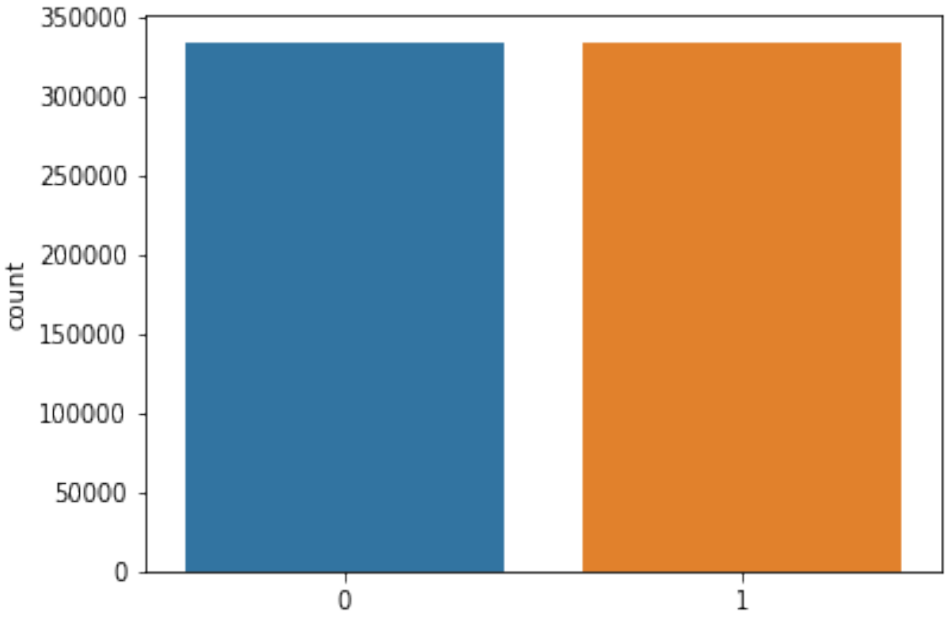

Figura 19. Dados balanceados.

Função converteVetor(m). Criando uma função para usar nos gráficos de radar plot e barras com os resultados da matriz de confusão(Figura 20).

```
[27] def converteVetor(m):
         vetor = []
         for i in range(len(m)):
             for j in range(len(m[i])):
                 vetor.append(m[i][j])
         return vetor
```

Figura 20. Funcao converteVetor(m).

MÉTRICAS DE AVALIAÇÃO EM MACHINE LEARNING

Essa seção dedica-se a descrever as principais métricas de avaliação dos modelos de ML. Essas, podem ser obtidas por meio da Matriz de Confusão(Figura 21) e, avaliam o quão bom o seu modelo está ao ser treinado[1].

Matriz de confusão		Classe predita	
		Positiva	Negativa
Classe original	Positiva	VP	FN
	Negativa	FP	VN

Figura 21. Matriz de Confusão. Fonte adaptado Mariano D.[4]

A matriz de confusão indica os números de ocorrências que o programa teve para cada uma das quatro categorias[4]. Nesse exemplo (Figura 21), tem-se nas colunas os valores preditos e nas linhas os valores reais. Também pode-se observar os resultados de uma classificação binária e, segundo Ferrari e Silva (2017) [3] podem ser:

- Verdadeiro positivo VP ou TP: usa-se quando a classe é positiva e, ao verificar a resposta, vê-se que é realmente positiva[1];
- Verdadeiro negativo VN ou TN: usa-se quando a classe é negativa e, ao verificar a resposta, vê-se que é realmente negativa[1];
- Falso positivo FP: o método diz que a classe é positiva, mas ao verificar a resposta, vê-se que a classe é negativa[4];
- Falso negativo FN: o método diz que a classe é negativo, mas ao verificar a resposta, vê-se que a classe é positivo[4];

As métricas são utilizadas para avaliar o modelo e, estas, serão explicadas a seguir::

- Acurácia(accuracy ou ACC): obtida pela divisão de total de acertos(VP e VN) dividido por total de itens. É uma das métricas mais simples e importantes. Exemplo: Acurácia = (VP + VN) / (VP+FN+VP+FP)[4];
- Sensibilidade ou Recall: avalia a capacidade do método de detectar com sucesso resultados classificados como positivos[4], ou seja, descreve a capacidade de um modelo prever a classe positiva[1]. Essa é obtida por: Recall = VP/(VP+FN);
- Precisão: avalia a quantidade de verdadeiros positivos sobre a soma de todos os valores positivos. Exemplo: Precisão = VP/(VP + FP);
- Especificidade: avalia a capacidade do método de detectar com sucesso resultados classificados como negativos[4], ou seja, descreve a capacidade de um modelo prever a classe negativa[1]. Essa é obtida por: Especificidade = VN/(VN+FP);
- F-score,F1-score ou F–measure: é média harmônica calculada com base na precisão e na sensibilidade(recall). Exemplo: 2 *((precisão + sensibilidade)/(precisão + sensibilidade))[4];
- Kappa: A pontuação kappa é um número entre -1 e 1. Pontuações acima de 0,8 são geralmente consideradas de boa concordância; zero ou menos significa que os resultados foram obtidos praticamente ao acaso(aleatório)[5].

Outro forma de avaliar o seu modelo é a curva Roc, no inglês Receiver Operating Characteristic Curve [4]. É um gráfico que permite avaliar um classificador binário e, traz a informação sobre a relação entre a Taxa de Falso Positivo(1- Especificidade) e a Taxa de Verdadeiro Positivo (Sensibilidade)[1]. A Figura 22 exemplifica a Curva ROC.

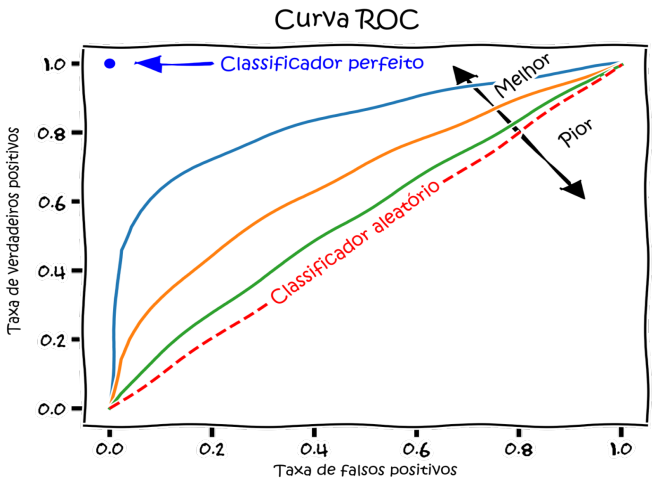

Figura 22. Exemplo de Curva ROC. Fonte adaptado Mariano D.[4]

A Curva ROC(Figura 22) é avaliada pela métrica AUC (Area Under the Curve ou "área sob a curva")[4]. Essa avalia a área abaixo da curva e os valores podem variar entre 0 a 1[1]. Quanto mais próximo de 1, melhor é o seu classificador. Caso obtenha uma curva de 0,50 ou 50%, o seu classificador não foi capaz de classificar corretamente. Um valor considerado adequado pode iniciar em 0,60 ou 60%[1].

EXECUÇÃO DOS MODELOS DE MACHINE LEARNING

Nesssa seção são apresentados as sequências de passos utilizados para a execução dos modelos de Machine Learning: K-Nearest Neighbor(KNN), Random Forest, Decision Tree, Gradient Boosting, Support Vector Machine (SVM) e XGBoost.

K-Nearest Neighbor(KNN). Executando o algoritmo KNN.

```
#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e 70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for executado
>X_treinamento_knn, X_teste_knn, y_treinamento_knn, y_teste_knn = train_test_split(previsores, classe,test_size = 0.3,random_state = 0)
>print(len(X_treinamento_knn))
>print(len(X_teste_knn))
468158
200640

#Realizando a parte de pré-processamento no conjunto de treinamento, teste e ajustar a escala padrão.
>sc = StandardScaler()
>X_treinamento_knn= sc.fit_transform(X_treinamento_knn)
>X_teste_knn = sc.transform(X_teste_knn)

#Criação do modelo, treinamento,
>knn = KNeighborsClassifier(n_neighbors = 5)
>knn.fit(X_treinamento_knn, y_treinamento_knn)
```

#Avalia o modelo com os dados de teste
>knn.score(X_teste_knn,y_teste_knn)
0.8536682615629984

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados.
>cv_scores = cross_val_score(knn, previsores, classe, cv=5, scoring='accuracy')
>print(f'Scores = cv_scores')
>print(f'Média dos Scores = cv_scores.mean()')
Scores = [0.814712920.814974580.814212020.813590110.81719361]
Média dos Scores = 0.8149366488991074

#Obtenção das previsões
>previsoes_knn = knn.predict(X_teste_knn)
>previsoes_knn

#Visualização da matriz de confusão
>print(confusion_matrix(y_teste_knn,previsoes_knn))
>print(pd.crosstab(y_teste_knn,previsoes_knn,rownames=['Real'],colnames=['Predito'],margins=True))
A Figura 23 exibe a matriz de confusão.

```
[[83126 17202]
 [12158 88154]]
Predito      0       1     All
Real
0        83126   17202  100328
1        12158   88154  100312
All      95284  105356  200640
```

Figura 23. Matriz de Confusão KNN.

#Gerando o vetor para ser usado no gráfico de barras e radar plot
>knn_vetor = converteVetor(confusion_matrix(y_teste_knn,previsoes_knn))
>knn_vetor
[83126, 17202, 12158, 88154]

#Calculando e exibindo as métricas: Acurácia, Sensibilidade, Especificidade, Precisão, Recall, F1-Score, Kappa e curva ROC
>acuracia_knn = accuracy_score(y_teste_knn,previsoes_knn)
>especificidade_knn = specificity_score(y_teste_knn,previsoes_knn)
>precisao_knn = precision_score(y_teste_knn,previsoes_knn)
>recall_knn = recall_score(y_teste_knn,previsoes_knn)
>f1Score_knn = f1_score(y_teste_knn,previsoes_knn)
>curva_roc_escore_knn = roc_auc_score(y_teste_knn,previsoes_knn)
>kappa_knn = cohen_kappa_score(y_teste_knn,previsoes_knn)
>print(f'Acurácia:round(acuracia_knn,2)')
>print(f'Especificidade:round(especificidade_knn,2)')
>print(f'Precisão:round(precisao_knn,2)')

```
>print(f'Recall ou Sensibilidade:round(recall_knn,2)')
>print(f'F1-Score:round(f1Score_knn,2)')
>print(f'Kappa:round(kappa_knn,2)')
>print(f'Curva ROC:round(curva_roc_escore_knn,2)')
Acurácia:0.85
Especificidade:0.83
Precisão:0.84
Recall ou Sensibilidade:0.88
F1-Score:0.86
Kappa:0.71
Curva ROC:0.85

#Plotando a Curva ROC do KNN
>rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
>pyplot.plot(rfp_knn, rvp_knn, marker='.', label='KNN='+str(round(curva_roc_escore_knn,2)),
color='orange')
>pyplot.plot([0, 1], [0, 1], color='black', linestyle='–')

#alterando o nome dos eixos
>pyplot.xlabel('1- Especificidade')
>pyplot.ylabel('Sensibilidade')

#Legenda
>pyplot.legend()

#Mostrando o gráfico
>pyplot.show()
```

O resultado desse sequência de comandos pode ser visualizada na Figura 24.

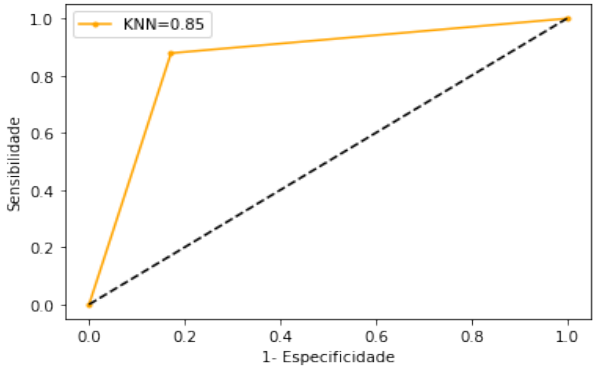

Figura 24. Curva ROC do KNN.

Random Forest(RF). Executando o algoritmo Random Forest.

```
#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e 70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for executado
>X_treinamento_rf, X_teste_rf, y_treinamento_rf, y_teste_rf = train_test_split(previsores, classe,test_size
```

= 0.3,random_state = 0)
>print(len(X_treinamento_rf))
>print(len(X_teste_rf))
468158
200640

#Realizando a parte de pré-processamento no conjunto de treinamento, teste e ajustar a escala padrão.
>sc = StandardScaler()
>X_treinamento_rf= sc.fit_transform(X_treinamento_rf)
>X_teste_rf = sc.transform(X_teste_rf)

#Criação do modelo, treinamento,
#n_estimators = 10, vou gerar 10 classificadores de árvore de decisão.
>floresta = RandomForestClassifier(n_estimators = 10)
>floresta.fit(X_treinamento_rf, y_treinamento_rf)

#Avalia o modelo com os dados de teste
>floresta.score(X_teste_rf,y_teste_rf)
0.9089264354066986

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados.
>cv_scores = cross_val_score(floresta, previsores, classe, cv=5,scoring='accuracy')
>print(f'Scores = cv_scores')
>print(f'Média dos Scores = cv_scores.mean()')
Scores = [0.682894740.952511960.952766150.953147080.95278822]
Média dos Scores = 0.8988216296879509

#Obtenção das previsões
>previsoes_rf = floresta.predict(X_teste_rf)
>previsoes_rf

#Visualização da matriz de confusão
>print(confusion_matrix(y_teste_rf,previsoes_rf))
>print(pd.crosstab(y_teste_rf,previsoes_rf,rownames=['Real'],colnames=['Predito'],margins= True))
A Figura 25 exibe a matriz de confusão.

```
[[93749  6579]
 [11694 88618]]
Predito       0      1     All
Real
0         93749   6579  100328
1         11694  88618  100312
All      105443  95197  200640
```

Figura 25. Matriz de Confusão Random Forest.

#Gerando o vetor para ser usado no gráfico de barras e radar plot
>floresta_vetor = converteVetor(confusion_matrix(y_teste_rf,previsoes_rf))

```
>floresta_vetor
[93749, 6579, 11694, 88618]

#Calculando e exibindo as métricas: Acurácia, Sensibilidade, Especificidade, Precisão, Recall, F1-Score, Kappa e curva ROC
>acuracia_rf = accuracy_score(y_teste_rf,previsoes_rf)
>especificidade_rf = specificity_score(y_teste_rf,previsoes_rf)
>precisao_rf = precision_score(y_teste_rf,previsoes_rf)
>recall_rf = recall_score(y_teste_rf,previsoes_rf)
>f1Score_rf = f1_score(y_teste_rf,previsoes_rf)
>curva_roc_escore_rf = roc_auc_score(y_teste_rf,previsoes_rf)
>kappa_rf = cohen_kappa_score(y_teste_rf,previsoes_rf)
>print(f'Acurácia:round(acuracia_rf,2)')
>print(f'Especificidade:round(especificidade_rf,2)')
>print(f'Precisão:round(precisao_rf,2)')
>print(f'Recall ou Sensibilidade:round(recall_rf,2)')
>print(f'F1-Score:round(f1Score_rf,2)')
>print(f'Kappa:round(kappa_rf,2)')
>print(f'Curva ROC:round(curva_roc_escore_rf,2)')
Acurácia:0.91
Especificidade:0.93
Precisão:0.93
Recall ou Sensibilidade:0.88
F1-Score:0.91
Kappa:0.82
Curva ROC:0.91

#Plotando a Curva ROC do KNN,RF e Decision Tree
>rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
>rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
>pyplot.plot(rfp_knn, rvp_knn, marker='.', label='KNN='+str(round(curva_roc_escore_knn,2)),
color="orange")
>pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random Forest='+str(round(curva_roc_escore_rf,2)),
color="blue")
>pyplot.plot([0, 1], [0, 1], color='black', linestyle='–')

#alterando o nome dos eixos
>pyplot.xlabel('1- Especificidade')
>pyplot.ylabel('Sensibilidade')

#Legenda
>pyplot.legend()

#Mostrando o gráfico
>pyplot.show()
```

O resultado desse sequência de comandos pode ser visualizada na Figura 26.

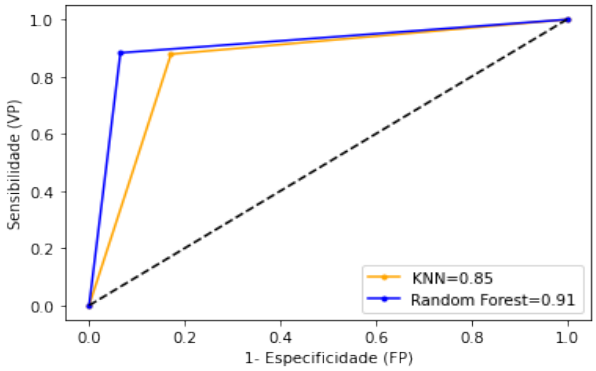

Figura 26. Curva ROC do KNN e RF.

Decision Tree (Árvores de Decisão). Executando o algoritmo Árvores de Decisão.
#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e 70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for executado
>X_treinamento_ar, X_teste_ar, y_treinamento_ar, y_teste_ar = train_test_split(previsores, classe,test_size = 0.3,random_state = 0)
>print(len(X_treinamento_ar))
>print(len(X_teste_ar))
468158
200640

#Realizando a parte de pré-processamento no conjunto de treinamento, teste e ajustar a escala padrão.
>sc = StandardScaler()
>X_treinamento_ar= sc.fit_transform(X_treinamento_ar)
>X_teste_ar = sc.transform(X_teste_ar)

#Criação do modelo, treinamento,
>arvore = DecisionTreeClassifier()
>arvore.fit(X_treinamento_ar, y_treinamento_ar)

#Avalia o modelo com os dados de teste
>arvore.score(X_teste_ar,y_teste_ar)
0.8953498803827751

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados.
>cv_scores = cross_val_score(arvore, previsores, classe, cv=5,scoring='accuracy')
>print(f'Scores = cv_scores')
>print(f'Média dos Scores = cv_scores.mean()')
Scores = [0.675388760.933545160.933193780.935144550.93421003]
Média dos Scores = 0.8822964553746793

#Obtenção das previsões
>previsoes_ar = arvore.predict(X_teste_ar)
>previsoes_ar

#Visualização da matriz de confusão
>print(confusion_matrix(y_teste_ar,previsoes_ar))
>print(pd.crosstab(y_teste_ar,previsoes_ar,rownames=['Real'],colnames=['Predito'],margins= True))
A Figura 27 exibe a matriz de confusão.

```
[[89451 10877]
 [10120 90192]]
Predito      0       1     All
Real
0        89451   10877  100328
1        10120   90192  100312
All      99571  101069  200640
```

Figura 27. Matriz de Confusão Decision Tree.

#Gerando o vetor para ser usado no gráfico de barras e radar plot
>arvore_vetor = converteVetor(confusion_matrix(y_teste_ar,previsoes_ar))
>arvore_vetor
[89451, 10877, 10120, 90192]

#Calculando e exibindo as métricas: Acurácia, Sensibilidade, Especificidade, Precisão, Recall, F1-Score, Kappa e curva ROC
>acuracia_ar = accuracy_score(y_teste_ar,previsoes_ar)
>especificidade_ar = specificity_score(y_teste_ar,previsoes_ar)
>precisao_ar = precision_score(y_teste_ar,previsoes_ar)
>recall_ar = recall_score(y_teste_ar,previsoes_ar)
>f1Score_ar = f1_score(y_teste_ar,previsoes_ar)
>curva_roc_escore_ar = roc_auc_score(y_teste_ar,previsoes_ar)
>kappa_ar = cohen_kappa_score(y_teste_ar,previsoes_ar)
>print(f'Acurácia:round(acuracia_ar,2)')
>print(f'Especificidade:round(especificidade_ar,2)')
>print(f'Precisão:round(precisao_ar,2)')
>print(f'Recall ou Sensibilidade:round(recall_ar,2)')
>print(f'F1-Score:round(f1Score_ar,2)')
>print(f'Kappa:round(kappa_ar,2)')
>print(f'Curva ROC:round(curva_roc_escore_ar,2)')
Acurácia:0.9
Especificidade:0.89
Precisão:0.89
Recall ou Sensibilidade:0.9
F1-Score:0.9
Kappa:0.79
Curva ROC:0.9

#Plotando a Curva ROC do KNN, RF e Árvore de Decisão
>rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
>rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
>rfp_ar, rvp_ar,lim3 = roc_curve(y_teste_ar,previsoes_ar)
>pyplot.plot(rfp_knn, rvp_knn, marker='.', label='KNN='+str(round(curva_roc_escore_knn,2)), color="orange")

```
>pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random Forest='+str(round(curva_roc_escore_rf,2)),
color="blue")
pyplot.plot(rfp_ar, rvp_ar, marker='.', label='Árvore Decisão='+str(round(curva_roc_escore_ar,2)),
color="yellow")
>pyplot.plot([0, 1], [0, 1], color='black', linestyle='–')

#alterando o nome dos eixos
>pyplot.xlabel('1- Especificidade')
>pyplot.ylabel('Sensibilidade')

#Legenda
>pyplot.legend()

#Mostrando o gráfico
>pyplot.show()
```

O resultado desse sequência de comandos pode ser visualizada na Figura 28.

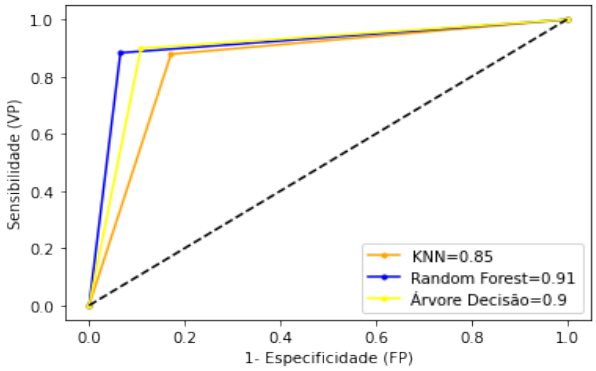

Figura 28. Curva ROC do KNN, RF e Árvore de Decisão.

Gradient Boosting. Executando o algoritmo Gradient Boosting.

```
#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e 70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for executado
>X_treinamento_gra, X_teste_gra, y_treinamento_gra, y_teste_gra = train_test_split(previsores,
classe,test_size = 0.3,random_state = 0)
>print(len(X_treinamento_gra))
>print(len(X_teste_gra))
468158
200640

#Realizando a parte de pré-processamento no conjunto de treinamento, teste e ajustar a escala
padrão.
>sc = StandardScaler()
>X_treinamento_gra= sc.fit_transform(X_treinamento_gra)
>X_teste_gra = sc.transform(X_teste_gra)
```

#Criação do modelo, treinamento,
>gradient = GradientBoostingClassifier()
>gradient.fit(X_treinamento_gra, y_treinamento_gra)

#Avalia o modelo com os dados de teste
>gradient.score(X_teste_gra,y_teste_gra)
0.8662828947368421

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados.
>cv_scores = cross_val_score(gradient, previsores, classe, cv=5,scoring='accuracy')
>print(f'Scores = cv_scores')
>print(f'Média dos Scores = cv_scores.mean()')
Scores = [0.712163580.883298440.883216210.8836340.88312562]
Média dos Scores = 0.8490875699156997

#Obtenção das previsões
>previsoes_gra = gradient.predict(X_teste_gra)
>previsoes_gra

#Visualização da matriz de confusão
>print(confusion_matrix(y_teste_gra,previsoes_gra))
>print(pd.crosstab(y_teste_gra,previsoes_gra,rownames=['Real'],colnames=['Predito'],margins=True))
A Figura 29 exibe a matriz de confusão.

```
[[81263 19065]
 [ 7764 92548]]
Predito      0       1     All
Real
0        81263   19065  100328
1         7764   92548  100312
All      89027  111613  200640
```

Figura 29. Matriz de Confusão Gradient Boosting.

#Gerando o vetor para ser usado no gráfico de barras e radar plot
>gradient_vetor = converteVetor(confusion_matrix(y_teste_ar,previsoes_ar))
>gradient_vetor
[81263, 19065, 7764, 92548]

#Calculando e exibindo as métricas: Acurácia, Sensibilidade, Especificidade, Precisão, Recall, F1-Score, Kappa e curva ROC
>acuracia_gra = accuracy_score(y_teste_gra,previsoes_gra)
>especificidade_gra = specificity_score(y_teste_gra,previsoes_gra)
>precisao_gra = precision_score(y_teste_gra,previsoes_gra)
>recall_gra = recall_score(y_teste_gra,previsoes_gra)
>f1Score_gra = f1_score(y_teste_gra,previsoes_gra)
>curva_roc_escore_gra = roc_auc_score(y_teste_gra,previsoes_gra)
>kappa_gra = cohen_kappa_score(y_teste_gra,previsoes_gra)

```
>print(f'Acurácia:round(acuracia_gra,2)')
>print(f'Especificidade:round(especificidade_gra,2)')
>print(f'Precisão:round(precisao_gra,2)')
>print(f'Recall ou Sensibilidade:round(recall_gra,2)')
>print(f'F1-Score:round(f1Score_gra,2)')
>print(f'Kappa:round(kappa_gra,2)')
>print(f'Curva ROC:round(curva_roc_escore_gra,2)')
Acurácia:0.87
Especificidade:0.81
Precisão:0.83
Recall ou Sensibilidade:0.92
F1-Score:0.87
Kappa:0.73
Curva ROC:0.87

#Plotando a Curva ROC do KNN, RF, Árvore de Decisão e Gradient Boosting
>rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
>rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
>rfp_ar, rvp_ar,lim3 = roc_curve(y_teste_ar,previsoes_ar)
>rfp_gra, rvp_gra,lim4 = roc_curve(y_teste_gra,previsoes_gra)
>pyplot.plot(rfp_knn, rvp_knn, marker='.', label='KNN='+str(round(curva_roc_escore_knn,2)),
color="orange")
>pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random Forest='+str(round(curva_roc_escore_rf,2)),
color="blue")
pyplot.plot(rfp_ar, rvp_ar, marker='.', label='Árvore Decisão='+str(round(curva_roc_escore_ar,2)),
color="yellow")
>pyplot.plot(rfp_gra, rvp_gra, marker='.', label='Gradient Boosting='+str(round(curva_roc_escore_gra,2)),
color="purple")
>pyplot.plot([0, 1], [0, 1], color='black', linestyle='–')

#alterando o nome dos eixos
>pyplot.xlabel('1- Especificidade')
>pyplot.ylabel('Sensibilidade')

#Legenda
>pyplot.legend()

#Mostrando o gráfico
>pyplot.show()
```

O resultado desse sequência de comandos pode ser visualizada na Figura 30.

Support Vector Machine(SVM) ou Máquina Vetor de Suporte. A Execução da SVM com validação cruzada foi realizada localmente, devido ao tempo excedido de 8 horas quando executado no Colab e, ter gerado um erro após esse tempo. A SVM precisou mais ou menos três dias para realizar a validação. Outro recurso utilizado foi a redução da dimensão pela técnica de PCA #Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e 70% para treinar.

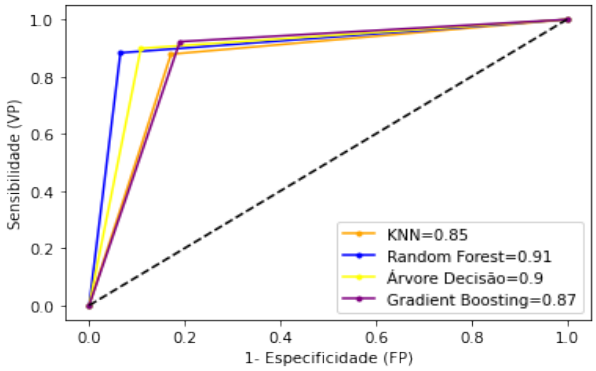

Figura 30. Curva ROC do KNN, RF, Árvore de Decisão e Gradient Boosting.

#Random_state = 0 para sempre obter a mesma divisão da base quando o código for executado
>X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(previsores, classe,test_size = 0.3,random_state = 0)
>print(len(X_treinamento))
>print(len(X_teste))
468158
200640

#Realizando a parte de pré-processamento no conjunto de treinamento, teste e ajustar a escala padrão.
>sc = StandardScaler()
>X_treinamentosc.fit_transform(X_treinamento)
>X_teste = sc.transform(X_teste)

#Aplicação da função PCA no treinamento e conjunto de teste do componente X
#Reduzindo a dimensão para três componente, ou seja, pegando as variáveis com maior pontuação, total de três.
>pca = PCA(n_components = 3)
>X_treinamento = pca.fit_transform(X_treinamento)
>X_teste = pca.transform(X_teste)

#Avaliando o resultado do PCA com as três variáveis com maior pontuação
>print(pca.explained_variance_ratio_)
>print(pca.singular_values_)
[0.236725960.166307130.11519457]
[1104.11806114925.4394275770.20897228]

#Criação do modelo, treinamento,
>svm = SVC()
>svm.fit(X_treinamento, y_treinamento)

#Avalia o modelo com os dados de teste
>svm.score(X_teste,y_teste)
0.819502591706539

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade de gene-

ralização de um modelo a partir de um conjunto de dados.
>cv_scores = cross_val_score(svm, previsores, classe, cv=5,scoring='accuracy')
>print(f'Scores = cv_scores')
>print(f'Média dos Scores = cv_scores.mean()')
Scores = [0.56525120.570462020.568548150.567333790.56704969]
Média dos Scores = 0.5677289688748217

#Obtenção das previsões
>previsoes_svm = svm.predict(X_teste)
>previsoes_svm

#Visualização da matriz de confusão
>print(confusion_matrix(y_teste,previsoes_svm))
>print(pd.crosstab(y_teste,previsoes_svm,rownames=['Real'],colnames=['Predito'],margins= True))
A Figura 31 exibe a matriz de confusão.

Predito	0	1	All
Real			
0	76395	23933	100328
1	7589	92723	100312
All	83984	116656	200640

Figura 31. Matriz de Confusão SVM.

#Gerando o vetor para ser usado no gráfico de barras e radar plot
>svm_vetor = converteVetor(confusion_matrix(y_teste,previsoes_svm))
>print(svm_vetor)

#Calculando e exibindo as métricas: Acurácia, Sensibilidade, Especificidade, Precisão, Recall, F1-Score, Kappa e curva ROC
>acuracia_svm = accuracy_score(y_teste_svm,previsoes_svm)
>especificidade_svm = specificity_score(y_teste_svm,previsoes_svm)
>precisao_svm = precision_score(y_teste_svm,previsoes_svm)
>recall_svm = recall_score(y_teste_svm,previsoes_svm)
>f1Score_svm = f1_score(y_teste_svm,previsoes_svm)
>curva_roc_escore_svm = roc_auc_score(y_teste_svm,previsoes_svm)
>kappa_svm = cohen_kappa_score(y_teste_svm,previsoes_svm)
>print(f'Acurácia:round(acuracia_svm,2)')
>print(f'Especificidade:round(especificidade_svm,2)')
>print(f'Precisão:round(precisao_svm,2)')
>print(f'Recall ou Sensibilidade:round(recall_svm,2)')
>print(f'F1-Score:round(f1Score_svm,2)')
>print(f'Kappa:round(kappa_svm,2)')
>print(f'Curva ROC:round(curva_roc_escore_svm,2)')
Acurácia:0.84
Especificidade:0.76
Precisão:0.79
Recall ou Sensibilidade:0.92

F1-Score:0.85
Kappa:0.68
Curva ROC:0.84

```
#Plotando a Curva ROC da SVM.
>rfp_svm, rvp_svm,lim5 = roc_curve(y_teste,previsoes_svm)
>pyplot.plot(rfp_svm, rvp_svm, marker='.', label='SVM='+str(round(curva_roc_escore_svm,4)),
color="red")
>pyplot.plot([0, 1], [0, 1], color='black', linestyle='–')

#alterando o nome dos eixos
>pyplot.xlabel('1- Especificidade')
>pyplot.ylabel('Sensibilidade')

#Legenda
>pyplot.legend()

#Mostrando o gráfico
>pyplot.show()
```

O resultado desse sequência de comandos pode ser visualizada na Figura 32.

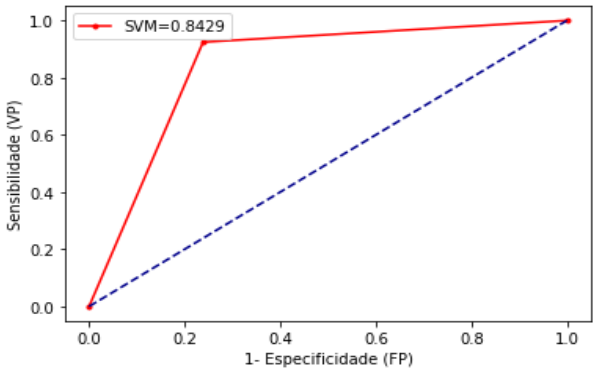

Figura 32. Curva ROC da SVM.

XGBoost. Executando o XGBoost.

```
#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e 70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for executado
>X_treinamento_xgb, X_teste_xgb, y_treinamento_xgb, y_teste_xgb = train_test_split(previsores,
classe,test_size = 0.3,random_state = 0)
>print(len(X_treinamento_xgb))
>print(len(X_teste_xgb))
468158
200640

#Realizando a parte de pré-processamento no conjunto de treinamento, teste e ajustar a escala
padrão.
```

```
>sc = StandardScaler()
>X_treinamento_xgb= sc.fit_transform(X_treinamento_xgb)
>X_teste_xgb = sc.transform(X_teste_xgb)

#Criação do modelo, treinamento,
xgboost_m = XGBClassifier()
xgboost_m.fit(X_treinamento_xgb, y_treinamento_xgb)

#Avalia o modelo com os dados de teste
>xgboost_m.score(X_teste_xgb,y_teste_xgb)
0.863302432216906

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados.
>cv_scores = cross_val_score(xgboost_m, previsores, classe, cv=5,scoring='accuracy')
>print(f'Scores = cv_scores')
>print(f'Média dos Scores = cv_scores.mean()')
Scores = [0.730240730.879448270.879971590.882138770.88026974]
Média dos Scores = 0.850413819390716

#Obtenção das previsões
>previsoes_xgb = gradient.predict(X_teste_xgb)
>previsoes_xgb

#Visualização da matriz de confusão
>print(confusion_matrix(y_teste_xgb,previsoes_xgb))
>print(pd.crosstab(y_teste_xgb,previsoes_xgb,rownames=['Real'],colnames=['Predito'],margins=
True))
```

A Figura 33 exibe a matriz de confusão.

```
[[80074 20254]
 [ 7173 93139]]
Predito      0       1     All
Real
0        80074   20254  100328
1         7173   93139  100312
All      87247  113393  200640
```

Figura 33. Matriz de Confusão XGBoost.

```
#Gerando o vetor para ser usado no gráfico de barras e radar plot
>xgb_vetor = converteVetor(confusion_matrix(y_teste_xgb,previsoes_xgb))
>xgb_vetor

#Calculando e exibindo as métricas: Acurácia, Sensibilidade, Especificidade, Precisão, Recall, F1-Score, Kappa e curva ROC
>acuracia_xgb = accuracy_score(y_teste_xgb,previsoes_xgb)
>especificidade_xgb = specificity_score(y_teste_xgb,previsoes_xgb)
>precisao_xgb = precision_score(y_teste_xgb,previsoes_xgb)
>recall_xgb = recall_score(y_teste_xgb,previsoes_xgb)
```

```
>f1Score_xgb = f1_score(y_teste_xgb,previsoes_xgb)
>curva_roc_escore_xgb = roc_auc_score(y_teste_xgb,previsoes_xgb)
>kappa_xgb = cohen_kappa_score(y_teste_xgb,previsoes_xgb)
>print(f'Acurácia:round(acuracia_xgb,2)')
>print(f'Especificidade:round(especificidade_xgb,2)')
>print(f'Precisão:round(precisao_xgb,2)')
>print(f'Recall ou Sensibilidade:round(recall_xgb,2)')
>print(f'F1-Score:round(f1Score_xgb,2)')
>print(f'Kappa:round(kappa_xgb,2)')
>print(f'Curva ROC:round(curva_roc_escore_xgb,2)')
Acurácia:0.87
Especificidade:0.81
Precisão:0.83
Recall ou Sensibilidade:0.92
F1-Score:0.87
Kappa:0.73
Curva ROC:0.87

#Plotando a Curva ROC do KNN, RF, Árvore de Decisão,Gradient Boosting e XGBoost
>rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
>rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
>rfp_ar, rvp_ar,lim3 = roc_curve(y_teste_ar,previsoes_ar)
>rfp_gra, rvp_gra,lim4 = roc_curve(y_teste_gra,previsoes_gra)
>rfp_xgb, rvp_xgb,lim5 = roc_curve(y_teste_xgb,previsoes_xgb)
>pyplot.plot(rfp_knn, rvp_knn, marker='.', label='KNN='+str(round(curva_roc_escore_knn,2)),
color="orange")
>pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random Forest='+str(round(curva_roc_escore_rf,2)),
color="blue")
pyplot.plot(rfp_ar, rvp_ar, marker='.', label='Árvore Decisão='+str(round(curva_roc_escore_ar,2)),
color="yellow")
>pyplot.plot(rfp_gra, rvp_gra, marker='.', label='Gradient Boosting='+str(round(curva_roc_escore_gra,2)),
color="purple")
>pyplot.plot(rfp_xgb, rvp_xgb, marker='.', label='XGBoost='+str(round(curva_roc_escore_xgb,2)),
color="pink")
>pyplot.plot([0, 1], [0, 1], color='black', linestyle='–')

#alterando o nome dos eixos
>pyplot.xlabel('1- Especificidade')
>pyplot.ylabel('Sensibilidade')

#Legenda
>pyplot.legend()

#Mostrando o gráfico
>pyplot.show()
```

O resultado desse sequência de comandos pode ser visualizada na Figura 34.

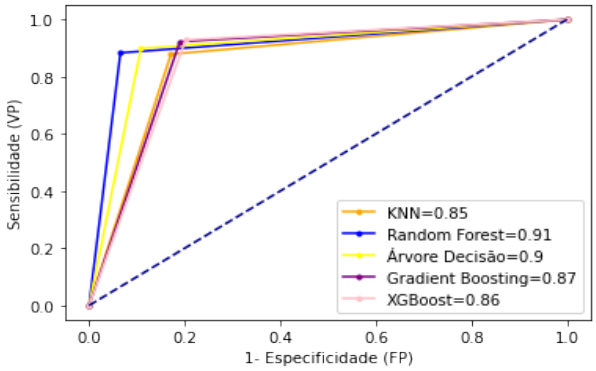

Figura 34. Curva ROC do KNN, RF, Árvore de Decisão, Gradient Boosting e XGBoost.

RESULTADOS FINAIS

Essa seção apresenta os resultados finais da execução dos modelos de Machine Learning, com o intuito de compará-los e definir qual foi o melhor para a resolução do problema apresentado no início do tutorial. A seguir para efeito de comparação serão apresentados gráficos como o radar plot e barras. Também será exibido uma tabela com as métricas calculadas.

#Começa aqui a preparação para gerar o gráfico de radar plot
>categories = $['VN', 'FP', 'FN', 'VP']$
>categories = $[*categories, categories[0]]$
#KNN
>knn_vetor = $[*knn\_vetor, knn\_vetor[0]]$
#Ramdom Forest
>floresta_vetor = $[*floresta\_vetor, floresta\_vetor[0]]$
#Árvores de decisão
>arvore_vetor = $[*arvore\_vetor, arvore\_vetor[0]]$
Gradient
>gradient_vetor = $[*gradient\_vetor, gradient\_vetor[0]]$
#XGBoost
>xgb_vetor = $[*xgb\_vetor, xgb\_vetor[0]]$

#Imprimindo o gráfico
>label_loc = np.linspace(start=0, stop=2 * np.pi, num=len(knn_vetor))
>plt.figure(figsize=(8, 8))
>plt.subplot(polar=True)
>plt.plot(label_loc,knn_vetor,label='KNN')
>plt.plot(label_loc, floresta_vetor, label='Random Forest')
>plt.plot(label_loc, arvore_vetor, label='Árvore de Decisão')
>plt.plot(label_loc, gradient_vetor, label='Gradient Boosting')
>plt.plot(label_loc, xgb_vetor, label='XGBoost')
>plt.title('Comparação entre Matrizes', size=20, y=1.05)
>lines, labels = plt.thetagrids(np.degrees(label_loc), labels=categories)
>plt.legend()
>plt.show()
A Figura 35 apresenta o resultado dos comandos para gerar o gráfico de Radar Plot. 35.
Na (Figura 35) não está apresentado a SVM. Conforme havia explanado, houve a necessidade de

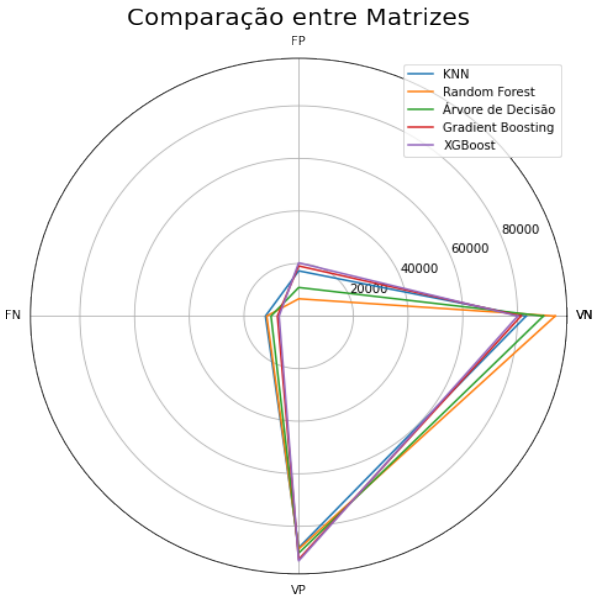

Figura 35. Radar Plot do KNN, RF, Árvore de Decisão, Gradient Boosting e XGBoost.

executar localmente. Portanto, os comandos para gerar o Radar Plot da SVM são apresentados a seguir.

#Radar Plot
#SVM
>categories = $['VN','FP','FN','VP']$
>categories = $[*categories, categories[0]]$
>svm_vetor = $[*svm\_vetor, svm\_vetor[0]]$

#Imprimindo o Gráfico
>label_loc = np.linspace(start=0, stop=2 * np.pi, num=len(svm_vetor))
>plt.figure(figsize=(8, 8))
>plt.subplot(polar=True)
>plt.plot(label_loc, svm_vetor, label='SVM')
>plt.title('Comparação entre Matrizes', size=20, y=1.05)
>lines, labels = plt.thetagrids(np.degrees(label_loc), labels=categories)
>plt.legend()
>plt.show()
A Figura 36 apresenta o resultado dos comandos para gerar o gráfico de Radar Plot da SVM.

O próximo gráfico apresentado é o de Barras. Os comandos para gerá-lo são exibidos a seguir.
#Gráfico de Barras
>data = $[['KNN', knn\_vetor[0], knn\_vetor[1], knn\_vetor[2], knn\_vetor[3]]$,
$['RandomForest', floresta\_vetor[0], floresta\_vetor[1], floresta\_vetor[2], floresta\_vetor[3]]$,
$['rvoreDeciso', arvore\_vetor[0], arvore\_vetor[1], arvore\_vetor[2], arvore\_vetor[3]]$,
$['Gradient', gradient\_vetor[0], gradient\_vetor[1], gradient\_vetor[2], gradient\_vetor[3]]$,
$['XGBoost', xgb\_vetor[0], xgb\_vetor[1], xgb\_vetor[2], xgb\_vetor[3]]]$
>df1 = pd.DataFrame(data,columns=['Matrizes','VN', 'FP', 'FN', 'VP'])
>df1.plot(x="Matrizes", y=['VN', 'FP', 'FN', 'VP'], kind="bar",figsize=(15,8))
>plt.show()

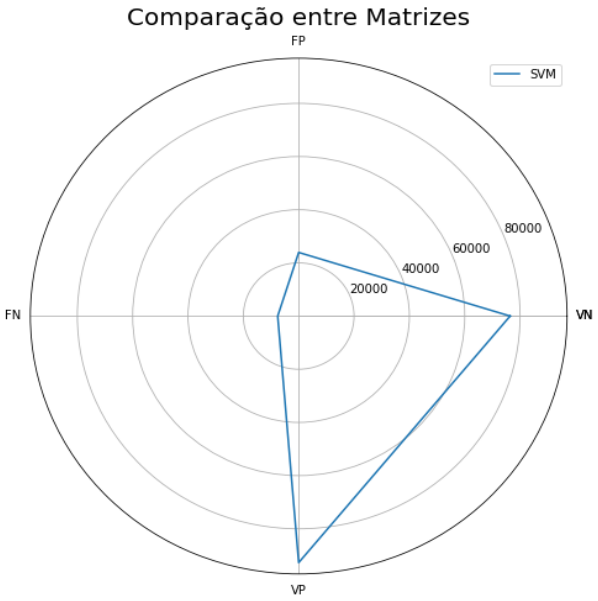

Figura 36. Radar Plot da SVM.

A Figura 37 exibi o resultado dos comandos para gerar o gráfico de barra comparativo entre os modelos de ML KNN, RF, Árvore de Decisão, Gradient Boosting e XGBoost.

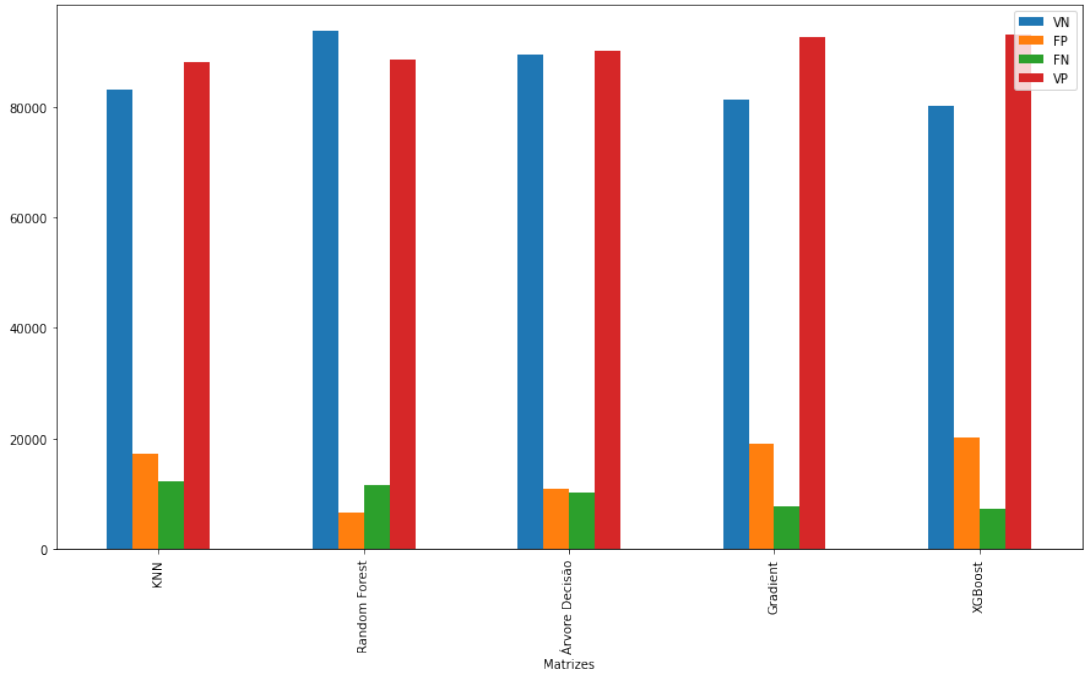

Figura 37. Gráfico de Barras dos modelos do KNN, RF, Árvore de Decisão, Gradient Boosting e XGBoost.

A seguir são apresentados os comandos para gerar o gráfico de barras da SVM.

```
#Gráfico de Barras
>data = [[’SVM’,svm_vetor[0],svm_vetor[1],svm_vetor[2],svm_vetor[3]]]
>df1 = pd.DataFrame(data,columns=[’Matrizes’,’VN’, ’FP’, ’FN’, ’VP’])
>df1.plot(x="Matrizes", y=[’VN’, ’FP’, ’FN’, ’VP’], kind="bar",figsize=(15,8))
>plt.show()
```

Tabela 1. Comparação entre os algoritmos de Machine Learning utilizados e os resultados das métricas calculadas.

Algoritmos	Acurácia	Especificidade	Precisão	Sensibilidade	F1-Score	Kappa
KNN	0.85	0.83	0.84	0.88	0.86	0.71
RF	**0.91**	**0.93**	**0.93**	**0.88**	**0.91**	**0.82**
Arv. Decisão	0.90	0.89	0.89	0.90	0.90	0.79
Gra. Boost.	0.87	0.81	0.83	0.92	0.87	0.73
SVM	0.84	0.76	0.79	0.92	0.85	0.68
XGBoost	0.86	0.80	0.82	0.93	0.87	0.73

A Figura 38 exibi o resultado dos comandos para gerar o gráfico de barras da SVM.

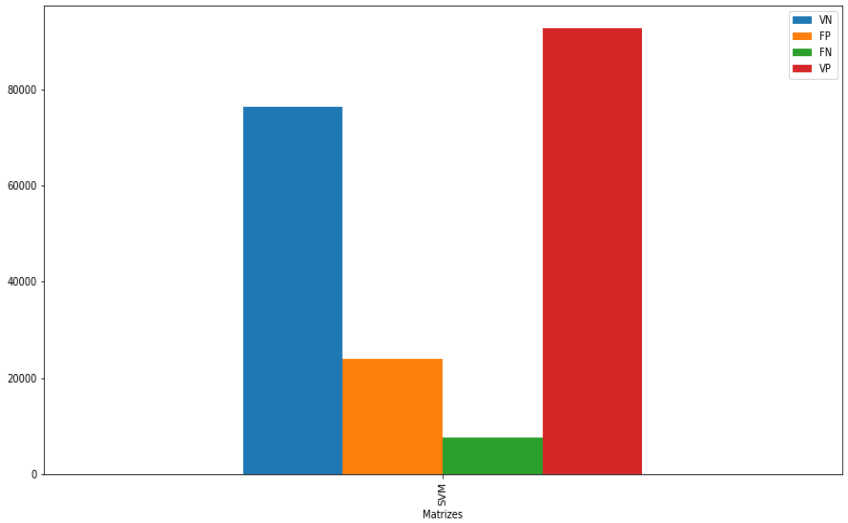

Figura 38. Gráfico de Barras da SVM.

Já Tabela 1 para fins comparativos entre os modelos KNN, RF, Arv. Decisão, Gra. Boost, SVM e XGBoost e as métricas Acurácia, Especificidade, Precisão, Sensibilidade, F1-Score, Kappa.
Ao analisar os Gráficos (Figuras 35,36,37,38), Tabela 1 e Curva ROC(Figura 26), contata-se que o Random Forest, em negrito, obteve a melhor performance, em todas as métricas, especialmente na Kappa. Segundo a própria documentação do scikit-learn[5], valores acima de 0.8 ou 80% são consideradas de boa concordância o que, dado o resultado de 0.82 do Random Forest, praticamente anula a aleatoriedade. Vale destacar que todos os algoritmos utilizaram parâmetros padrão.

CONCLUSÕES

Os trabalho foi desenvolvido de acordo com regras e normas da disciplina de Tópicos Avançados em Bioinformática. Para alcançar os resultados apresentados, métricas e gráficos dos algoritmos de Machine Learning foram estudados para posterior aplicação e validação dos modelos. Também necessitou-se, antes disso, uma análise exploratória dos dados para conhecê-los e identificar possíveis problemas que possam de alguma maneira enviesar o resultado. Esses foram corrigidos adequadamente. Todos os algoritmos atingiram bons resultados, contudo, o

Random Forest teve os melhores, conforme elucidado na seção de "Resultados Finais".

Referências

[1] Ciência dos Dados. VALIDAÇÃO DE MÁQUINAS PREDITIVAS - MÉTRICAS DE AVALIAÇÃO. Disponível em:<https://youtu.be/qwW8AjRqPz4>. Acesso em 16/07/2021.
[2] Corrêa E. Pandas Python: Data Wrangling para Ciência de Dados.Casa do Código. 24 de janeiro de 2020.
[3] FERRARI, D. G.; DE CASTRO SILVA, L. N. Introdução a mineração de dados.
[4] Mariano D. Disponível em:<https://bioinfo.com.br/metricas-de-avaliacao-em-machine-learning-acuracia-sensibilidade-precisao-especificidade-e-f-score/>. Acesso em 16/07/2021.
[5] Cohen's kappa. Disponível em:<https://scikit-learn.org/stable/modules/model_evaluation.html#cohen-kappa>. Acesso em 16/07/2021.
[6] Kaggle. Disponível em:<https://www.kaggle.com/anmolkumar/health-insurance-cross-sell-prediction>. Acesso em 16/07/2021.

ANEXO A - CÓDIGO-FONTE DA ANÁLISE EXPLORATÓRIA DOS DADOS.

```
# -*- coding: utf-8 -*-
"""Trabalho de Machine Learning - AED.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1XDfb5I3Zk3_1EkZckdhU61ETknGp_jhx
"""

import pandas as pd
import seaborn as srn
import statistics as sts
import matplotlib.pyplot as plt
from google.colab import files
from sklearn import preprocessing

from google.colab import drive
drive.mount('/content/drive')

uploaded = files.upload()

df_train = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/train.csv",sep=",")
df_train.head()

df_test = pd.read_csv("/content/test.csv",sep=",")
df_test.head()

print(df_train.shape)
print(df_test.shape)

"""**Um gŕafico de Scatter Matriz será implementado nos dados brutos. Assim como o Scatter plots que consiste em um tipo de gráfico comumente utilizado para observar o comportamento entre duas variáveis, o Scatter Matriz utiliza todo o Data Frame, pois que irá gerar os scatter plots para todos os pares possíveis de atributos.**"""

#Scatter Matriz
#criando um dicionario para mapear cada classe para uma cor
classe_cor = {0 : 'red',1 : 'blue'}
#criando uma lista com as cores de cada exemplo
cores = [classe_cor[nome] for nome in df_train.Response]
# gerando matriz de scatter plots
#pd.scatter_matrix(df_train, color=cores)
pd.plotting.scatter_matrix(df_train, color=cores,figsize=(16,12))

"""## **Explorando os dados categóricos (qualitativos). Para esses utilizarei o gráfico de barras, pizza e linha para obter uma visão mais clara da distribuição dos dados. Não utilizei o Histograma e Boxplot, pois esses são para dados quantitativos.**

**1 - Atributo Gender**

**Agrupando os dados de gênero dos clientes**
"""

group = df_train.groupby(['Gender']).size()
group

"""**Visualizar o dados de gênero dos clientes em um gráfico de barras.**"""

srn.countplot(df_train['Gender'])

"""**2 - Atributo Vehicle_Age**
```

```
**Agrupando  a idade dos veículos**
"""

group = df_train.groupby(['Vehicle_Age']).size()
group

""""**Visualizar a idade dos clientes em um gráfico de pizza**"""

group.plot.pie(autopct="%.1f%%");

""""**3 - Atributo Vehicle_Damage**

**Agrupando os dados de dano do veículo**
"""

group = df_train.groupby(['Vehicle_Damage']).size()
group

""""**Visualizar os dados de dano do veículo em um gráfico de barras**"""

srn.countplot(df_train['Vehicle_Damage'])

""""**4 - Atributo Driving_License**

**Analisando o dado Driving_License. Valores possíveis:**

*   0 não tem licença;
*   1 tem licença
"""

#Agrupando os dados
group = df_train.groupby(['Driving_License']).size()
group

""""**Visualizar os dados de licença em um gráfico de pizza**"""

group.plot.pie(autopct="%.1f%%");

""""**5 - Atributo Region_Code**

**Explorando o Region_Code**
"""

group = df_train.groupby(['Region_Code']).size()

""""**Visualizar os dados de código da região em um gráfico de linhas. Essa opção
foi utilizada devido quantidade enorme de agrupamento, uma vez que dificultaria
a visualização em gráfico de barras padrão**"""

group.plot.line(color='blue')
#srn.distplot(df_train['Age']).set_title('Age')

""""**6 - Atributo  Previously_Insured**

**Analisando o campo Previously_Insured. Valores possíveis:**

*   0  cliente não tem seguro de veículos;
*   1  cliente tem seguro de veículos;
"""

#Agrupando os dados
group = df_train.groupby(['Previously_Insured']).size()
group
```

```
"""**Analisando o campo Previously_Insured em um gráfico de pizza**"""

group.plot.pie(autopct="%.1f%%");

"""**7 - Policy_Sales_Channel**

**Analisando o campo Policy_Sales_Channel. Esse campo é um código anônimo para o canal de divulgação ao cliente, ou seja. Agentes diferentes, por correio, por telefone, pessoalmente, etc.**
"""

group = df_train.groupby(['Policy_Sales_Channel']).size()
group

"""**Visualizar os dados de código de política do canal de vendas em um gráfico de linhas. Essa opção foi utilizada devido quantidade enorme de agrupamento, uma vez que dificultaria a visualização em gráfico de barras padrão.**"""

group.plot.line(color="red")

"""**Ao analisar os dados qualitativos não foi não foi detectado nenhuma anornalia como: dados fora do padrão (escrita) ou células nulas. Contudo, alguns esses serão convertidos para númericos e, desse modo, implementar os algorítmos de machine learning**

# **Explorando os dados quantitativos. Para esses utilizarei o Histograma e Boxplot e, assim obter uma visão mais clara da distribuição dos dados.**

**8 - Explorando a variável Age**
"""

df_train['Age'].describe()

"""**Ao observar esses dados de Age, constata-se que a mediana e média estão próximas, portanto, isso indica que não exite grandes outliers, ou seja, existe uma distribuição regular dos dados. Também no valor mínimo é possível identificar que não existe valor igual a 0.Para ter uma visão completa da distribuição, será utilizado o histograma e o boxplot. Alguns valores importantes: mean(média), std(desvio padrão), 50% (mediana) serão explorados**"""

#Visualizar os dados de Age por um boxplot
srn.boxplot(df_train['Age']).set_title('Age')

"""**O Boxplot indica que a idade está entre 20 e 85 anos e não há outliers**"""

#Visualizar os dados de Age em um histograma
srn.distplot(df_train['Age']).set_title('Age')

"""**Para a idade foi gerado um histograma assimétrico. A freqüência decresce bruscamente em um dos lados de forma gradual no outro, produzindo uma calda mais longa em um dos lados. Assimetria à direita, mediana é inferior a média. Assimetria à esquerda, mediana é superior à média. O gráfico confirma o que foi calculado na função describe**

**9 - Atributo Annual_Premium**

**Analisando o atributo Annual_Premium. Esse é o valor que o cliente precisa pagar como prêmio no ano.**
"""

df_train['Annual_Premium'].describe()
```

```
#Visualizar os dados de Annual_Premium por um boxplot
srn.boxplot(df_train['Annual_Premium']).set_title('Annual_Premium')

"""**Analisando o boxplot pode-se obervar que há outliers. Nesse gráfico existe
um desvio padrão à direita**"""

#Visualizar os dados de Annual_Premium em um histograma
srn.distplot(df_train['Annual_Premium']).set_title('Annual_Premium')

"""**Histograma com dois picos. Isso indica que houve misturas de dados com
médias, obtidos em duas condições distintas. Esse gráfico responderia o desvio
padrão do boxplot, contudo, pasra que não posso causar um enviesamento so
algortmo irei normalizar com a função min-max. Isso será realizado na correção
de dados.**

**10 - Atributo Vintage**

**Analisando o campo Vintage. Número de dias queo cliente esteve associado à
empresa**
"""

df_train['Vintage'].describe()

#Visualizar os dados de Vintage por um boxplot
srn.boxplot(df_train['Vintage']).set_title('Vintage')

#Visualizar os dados de Vintage por um histograma
srn.distplot(df_train['Vintage']).set_title('Vintage')

"""**Histograma do tipo platô ou achatado.  Indice que os dados são combinados
de  várias distribuições com médias diferentes.**

**10 - Explorando a Classe - Atributo Response**
"""

group = df_train.groupby(['Response']).size()
group

"""**Exibindo os dados da classe em um gráfico de barras.**"""

srn.countplot(df_train['Response'])

"""**Como pode-se observar os dados estão desbalanceados, portanto, um
balanceamento antes de executar o modelo será necessário. Caso contrário o
modelo ficará enviesado.**

# **Finalizando a análise exploratória. Verificando se existe valores
nulos(NAN).**
"""

#Verificar os valores nulos
#Eliminar a valores nulos
df_train.isnull().sum()

"""**Não há nenhum campo nulo.**"""
```

ANEXO B - CÓDIGO-FONTE DA CORREÇÃO DOS DADOS E EXECUÇÃO DOS ALGORITMOS K-NEAREST NEIGHBOR(KNN), RANDOM FOREST, DECISION TREE, GRADIENT BOOSTING, SUPPORT VECTOR MACHINE(SVM) E XGBOOST

```
# -*- coding: utf-8 -*-
"""Trabalho de Machine Learning - Correcao e Execucao.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1ovfiQbFSr0FgmPCafiXAFyjRaiSsB8_3
"""

!pip install scikit-plot

#Instalando o xgboost
!pip install xgboost

import numpy as np
import pandas as pd
import seaborn as srn
import statistics as sts
import matplotlib.pyplot as plt
from google.colab import files
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score,confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score,roc_curve,auc,r2_score
from sklearn.neighbors import KNeighborsClassifier
from scipy import stats
from sklearn.model_selection import cross_val_score
from imblearn.metrics import specificity_score
import matplotlib.pyplot  as pyplot
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE

#uploaded = files.upload()

df_train = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/train.csv',sep=',');
df_train.head()

#Separar os atributos previsores da classe
#Criando um dataFrame realizar o One-Hot encoder
previsores_df = df_train.iloc[:,1:11]
#df_train[df_train.columns[1:11]]
#Aqui quer dizer que irei percorrer todas as linas inserir em classe a coluna
12
classe = df_train.iloc[:,11].values
#Exibir os valores
print(classe)

previsores_df.head()

"""**Atributo Gender**

**Converter para número com o One-Hot Encoder. O One-Hot Encoder foi selecionado
pois ele atribui valores numéricos sem dar um peso ou hierarquia.  Nessa
estratégia, cada valor de categoria é convertido em uma nova coluna e atribuído
um valor 1 ou 0 (notação para verdadeiro / falso) à coluna.**
"""

#oneHotEncoder com a função  get_dummies do pandas
```

```
#Agora tentando remodelar com (-1, 1). Fornecemos coluna como 1, mas linhas como
desconhecidas
previsores_df = pd.concat([previsores_df,pd.get_dummies(previsores_df['Gender'],
prefix='Gender')],axis=1)
previsores_df.drop(['Gender'],axis=1, inplace=True)

previsores_df.head()

"""**Atributo Vehicle_Age**

**Converter para número o Vehicle_Age. Usarei o encoder:**
"""

# Transformação dos atributos categóricos em atributos numéricos.
labelencoder = LabelEncoder()
previsores_df['Vehicle_Age'] =
labelencoder.fit_transform(previsores_df['Vehicle_Age'])
group = previsores_df.groupby(['Vehicle_Age']).size()
group

previsores_df.head()

"""**Atributo Vehicle_Damage. Farei a converão para número manualmente**  """

previsores_df.loc[previsores_df['Vehicle_Damage']=='No','Vehicle_Damage'] = '0'
previsores_df.loc[previsores_df['Vehicle_Damage']=='Yes','Vehicle_Damage'] = '1'
group = previsores_df.groupby(previsores_df['Vehicle_Damage']).size()
group

previsores_df.head()

"""**Normalizando. Essa está desativada, pois irei fazer de uma só vez em todas
as variáveis de uma vez**"""

#Normalizando o atributo Annual_Premium
'''min_p = min(previsores_df['Annual_Premium'])
max_p = max(previsores_df['Annual_Premium'])
print(min_p)
print(max_p)
previsores_df['Annual_Premium'] = (previsores_df['Annual_Premium'] - min_p) /
(max_p - min_p)

#Normalizando o atributo Region_Code
min_p = min(previsores_df['Region_Code'])
max_p = max(previsores_df['Region_Code'])
print(min_p)
print(max_p)
previsores_df['Region_Code'] = (previsores_df['Region_Code'] - min_p) / (max_p -
min_p)

#Normalizando o atributo Vintage
min_p = min(previsores_df['Vintage'])
max_p = max(previsores_df['Vintage'])
print(min_p)
print(max_p)
previsores_df['Vintage'] = (previsores_df['Vintage'] - min_p) / (max_p - min_p)

#Normalizando o Policy_Sales_Channel
min_p = min(previsores_df['Policy_Sales_Channel'])
max_p = max(previsores_df['Policy_Sales_Channel'])
print(min_p)
print(max_p)
previsores_df['Policy_Sales_Channel'] = (previsores_df['Policy_Sales_Channel'] -
min_p) / (max_p - min_p)
```

```
'''

previsores_df.head()

#Aqui quer dizer que irei percorrer todas as linhas inserir em previsores as
colunas de 0 a 10
previsores = previsores_df.iloc[:,0:11].values
print(previsores)
print(classe)

"""**Balanceamento das Classes  com o método Smote.**"""

smt = SMOTE()
previsores,classe = smt.fit_sample(previsores,classe)

#Fazendo a contagem para ver como ficou
yi = classe.astype(int)
np.bincount(yi)

#visualização em um gráfico
ax = srn.countplot(x=yi)

"""**Criando uma função para usar nos gráficos de Radar Plot e Barras com os
resultados da matriz de confusão**"""

def converteVetor(m):
    vetor = []
    for i in range(len(m)):
        for j in range(len(m[i])):
            vetor.append(m[i][j])
    return vetor

"""**Executando o algoritomo KNN**"""

#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e
70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for
executado
X_treinamento_knn, X_teste_knn, y_treinamento_knn, y_teste_knn =
train_test_split(previsores, classe,
                                                                test_size =
0.3,random_state = 0)
print(len(X_treinamento_knn))
print(len(X_teste_knn))

#realizando a parte de pré-processamento no conjunto de treinamento, teste e
ajustar a escala padrão.
sc = StandardScaler()
X_treinamento_knn= sc.fit_transform(X_treinamento_knn)
X_teste_knn = sc.transform(X_teste_knn)

#Aplicação da função PCA no treinamento  e conjunto de teste do componente X
#pca_knn = PCA(n_components = 2)
#X_treinamento_knn = pca_knn.fit_transform(X_treinamento_knn)
#X_teste_knn = pca_knn.transform(X_teste_knn)
#explained_variance = pca_knn.explained_variance_ratio_

# Criação do modelo, treinamento,
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_treinamento_knn, y_treinamento_knn)

#avalia o modelo com os dados de teste
knn.score(X_teste_knn,y_teste_knn)
```

```
#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade
de generalização de um modelo,
#a partir de um conjunto de dados.
cv_scores = cross_val_score(knn, previsores, classe, cv=5, scoring='accuracy')
print(f'Scores = {cv_scores}')
print(f'Média dos Scores = {cv_scores.mean()}')

#obtenção das previsões
previsoes_knn = knn.predict(X_teste_knn)
previsoes_knn

# Visualização da matriz de confusão
print(confusion_matrix(y_teste_knn,previsoes_knn))
print(pd.crosstab(y_teste_knn,previsoes_knn,rownames=['Real'],colnames=['Predito
'],margins= True))

#gerando o vetor
knn_vetor =  converteVetor(confusion_matrix(y_teste_knn,previsoes_knn))
knn_vetor

#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,
F1-Score
acuracia_knn = accuracy_score(y_teste_knn,previsoes_knn)
especificidade_knn = specificity_score(y_teste_knn,previsoes_knn)
precisao_knn = precision_score(y_teste_knn,previsoes_knn)
recall_knn = recall_score(y_teste_knn,previsoes_knn)
f1Score_knn = f1_score(y_teste_knn,previsoes_knn)
curva_roc_escore_knn = roc_auc_score(y_teste_knn,previsoes_knn)
kappa_knn = cohen_kappa_score(y_teste_knn,previsoes_knn)
print(f'Acurácia:{round(acuracia_knn,2)}')
print(f'Especificidade:{round(especificidade_knn,2)}')
print(f'Precisão:{round(precisao_knn,2)}')
print(f'Recall ou Sensibilidade:{round(recall_knn,2)}')
print(f'F1-Score:{round(f1Score_knn,2)}')
print(f'Kappa:{round(kappa_knn,2)}')
print(f'Curva ROC:{round(curva_roc_escore_knn,2)}')

#skpltf.metrics.plot_roc(y_teste,previsoes, title="Roc Curve for XBG
Classifier", figsize=(10, 8))
#plt.show()

#Curva ROC
rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
pyplot.plot(rfp_knn, rvp_knn, marker='.',
label='KNN='+str(round(curva_roc_escore_knn,2)),color="orange")
pyplot.plot([0, 1], [0, 1], color='black', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade')
pyplot.ylabel('Sensibilidade')
# Legenda
pyplot.legend()
# Mostrando o gráfico
pyplot.show()

"""**Random forest**"""

#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e
70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for
executado
X_treinamento_rf, X_teste_rf, y_treinamento_rf, y_teste_rf =
train_test_split(previsores, classe,
                                                                    test_size =
0.3,random_state = 0)
```

```
print(len(X_treinamento_rf))
print(len(X_teste_rf))

#realizando a parte de pré-processamento no conjunto de treinamento, teste e
ajustar a escala padrão.
sc = StandardScaler()
X_treinamento_rf = sc.fit_transform(X_treinamento_rf)
X_teste_rf = sc.transform(X_teste_rf)

#Aplicação da função PCA no treinamento  e conjunto de teste do componente X
#pca = PCA(n_components = 2)
#X_treinamento_rf = pca.fit_transform(X_treinamento_rf)
#X_teste_rf = pca.transform(X_teste_rf)
#explained_variance = pca.explained_variance_ratio_

# Criação do modelo, treinamento, obtenção das previsões e da taxa de acerto
floresta = RandomForestClassifier(n_estimators = 10)
floresta.fit(X_treinamento_rf, y_treinamento_rf)

#avalia o modelo com os dados de teste
floresta.score(X_teste_rf,y_teste_rf)

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade
de generalização de um modelo,
#a partir de um conjunto de dados.
cv_scores = cross_val_score(floresta, previsores, classe, cv=5,
scoring='accuracy')
print(f'Scores = {cv_scores}')
print(f'Média dos Scores = {cv_scores.mean()}')

#obtenção das previsões
previsoes_rf = floresta.predict(X_teste_rf)
previsoes_rf

# Visualização da matriz de confusão
print(confusion_matrix(y_teste_rf,previsoes_rf))
print(pd.crosstab(y_teste_rf,previsoes_rf,rownames=['Real'],colnames=['Predito']
,margins= True))

#gerando o vetor
floresta_vetor =  converteVetor(confusion_matrix(y_teste_rf,previsoes_rf))
floresta_vetor

#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,
F1-Score
acuracia_rf = accuracy_score(y_teste_rf,previsoes_rf)
especificidade_rf = specificity_score(y_teste_rf,previsoes_rf)
precisao_rf = precision_score(y_teste_rf, previsoes_rf)
recall_rf = recall_score(y_teste_rf, previsoes_rf)
f1Score_rf = f1_score(y_teste_rf,previsoes_rf)
curva_roc_escore_rf = roc_auc_score(y_teste_rf, previsoes_rf)
kappa_rf = cohen_kappa_score(y_teste_rf, previsoes_rf)
print(f'Acurácia:{round(acuracia_rf,2)}')
print(f'Especificidade:{round(especificidade_rf,2)}')
print(f'Precisão:{round(precisao_rf,2)}')
print(f'Recall ou Sensibilidade:{round(recall_rf,2)}')
print(f'F1-Score:{round(f1Score_rf,2)}')
print(f'Kappa:{round(kappa_rf,2)}')
print(f'Curva ROC:{round(curva_roc_escore_rf,2)}')

#Curva ROC
rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
pyplot.plot(rfp_knn, rvp_knn, marker='.',
```

```
label='KNN='+str(round(curva_roc_escore_knn,2)),color="orange")
pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random
Forest='+str(round(curva_roc_escore_rf,2)),color="blue")
pyplot.plot([0, 1], [0, 1], color='black', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
pyplot.legend()
# Mostrando o gráfico
pyplot.show()

"""**Árvores de Decisão**"""

#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e
70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for
executado
X_treinamento_ar, X_teste_ar, y_treinamento_ar, y_teste_ar =
train_test_split(previsores, classe,
                                                                    test_size =
0.3,random_state = 0)
print(len(X_treinamento_ar))
print(len(X_teste_ar))

#realizando a parte de pré-processamento no conjunto de treinamento, teste e
ajustar a escala padrão.
sc = StandardScaler()
X_treinamento_ar= sc.fit_transform(X_treinamento_ar)
X_teste_ar = sc.transform(X_teste_ar)

#Aplicação da função PCA no treinamento  e conjunto de teste do componente X
#pca = PCA(n_components = 2)
#X_treinamento_ar = pca.fit_transform(X_treinamento_ar)
#X_teste_ar = pca.transform(X_teste_ar)
#explained_variance = pca.explained_variance_ratio_

# Criação do modelo, treinamento, obtenção das previsões e da taxa de acerto
arvore = DecisionTreeClassifier()
arvore.fit(X_treinamento_ar, y_treinamento_ar)

#avalia o modelo com os dados de teste
arvore.score(X_teste_ar,y_teste_ar)

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade
de generalização de um modelo,
#a partir de um conjunto de dados.
cv_scores = cross_val_score(arvore, previsores, classe, cv=5,
scoring='accuracy')
print(f'Scores = {cv_scores}')
print(f'Média dos Scores = {cv_scores.mean()}')

#obtenção das previsões
previsoes_ar = arvore.predict(X_teste_ar)
previsoes_ar

# Visualização da matriz de confusão
print(confusion_matrix(y_teste_ar,previsoes_ar))
print(pd.crosstab(y_teste_ar,previsoes_ar,rownames=['Real'],colnames=['Predito']
,margins= True))

#gerando o vetor
arvore_vetor =  converteVetor(confusion_matrix(y_teste_ar,previsoes_ar))
arvore_vetor
```

```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,
F1-Score
acuracia_ar = accuracy_score(y_teste_ar,previsoes_ar)
especificidade_ar = specificity_score(y_teste_ar,previsoes_ar)
precisao_ar = precision_score(y_teste_ar, previsoes_ar)
recall_ar = recall_score(y_teste_ar, previsoes_ar)
f1Score_ar = f1_score(y_teste_ar,previsoes_ar)
curva_roc_escore_ar = roc_auc_score(y_teste_ar, previsoes_ar)
kappa_ar = cohen_kappa_score(y_teste_ar, previsoes_ar)
print(f'Acurácia:{round(acuracia_ar,2)}')
print(f'Especificidade:{round(especificidade_ar,2)}')
print(f'Precisão:{round(precisao_ar,2)}')
print(f'Recall ou Sensibilidade:{round(recall_ar,2)}')
print(f'F1-Score:{round(f1Score_ar,2)}')
print(f'Kappa:{round(kappa_ar,2)}')
print(f'Curva ROC:{round(curva_roc_escore_ar,2)}')

#Curva ROC
rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
rfp_ar, rvp_ar,lim3 = roc_curve(y_teste_ar,previsoes_ar)
pyplot.plot(rfp_knn, rvp_knn, marker='.',
label='KNN='+str(round(curva_roc_escore_knn,2)),color="orange")
pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random
Forest='+str(round(curva_roc_escore_rf,2)),color="blue")
pyplot.plot(rfp_ar, rvp_ar, marker='.', label='Árvore
Decisão='+str(round(curva_roc_escore_ar,2)),color="yellow")
pyplot.plot([0, 1], [0, 1], color='black', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
# Legenda
pyplot.legend()
# Mostrando o gráfico
pyplot.show()

"""**Gradient Boosting**"""

#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e
70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for
executado
X_treinamento_gra, X_teste_gra, y_treinamento_gra, y_teste_gra =
train_test_split(previsores, classe,
                                                                    test_size =
0.3,random_state = 0)
print(len(X_treinamento_gra))
print(len(X_teste_gra))

#realizando a parte de pré-processamento no conjunto de treinamento, teste e
ajustar a escala padrão.
sc = StandardScaler()
X_treinamento_gra = sc.fit_transform(X_treinamento_gra)
X_teste_gra = sc.transform(X_teste_gra)

#Aplicação da função PCA no treinamento  e conjunto de teste do componente X
#pca = PCA(n_components = 2)
#X_treinamento_gra = pca.fit_transform(X_treinamento_gra)
#X_teste_gra = pca.transform(X_teste_gra)
#explained_variance = pca.explained_variance_ratio_

# Criação do modelo, treinamento, obtenção das previsões
gradient = GradientBoostingClassifier()
gradient.fit(X_treinamento_gra, y_treinamento_gra)
```

```
#avalia o modelo com os dados de teste
gradient.score(X_teste_gra,y_teste_gra)

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade
de generalização de um modelo,
#a partir de um conjunto de dados.
cv_scores = cross_val_score(gradient, previsores, classe, cv=5,
scoring='accuracy')
print(f'Scores = {cv_scores}')
print(f'Média dos Scores = {cv_scores.mean()}')

#obtenção das previsões
previsoes_gra = gradient.predict(X_teste_gra)
previsoes_gra

# Visualização da matriz de confusão
print(confusion_matrix(y_teste_gra,previsoes_gra))
print(pd.crosstab(y_teste_gra,previsoes_gra,rownames=['Real'],colnames=['Predito
'],margins= True))

#gerando o vetor
gradient_vetor =  converteVetor(confusion_matrix(y_teste_gra,previsoes_gra))
gradient_vetor

#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,
F1-Score
acuracia_gra = accuracy_score(y_teste_gra,previsoes_gra)
especificidade_gra = specificity_score(y_teste_gra,previsoes_gra)
precisao_gra = precision_score(y_teste_gra, previsoes_gra)
recall_gra = recall_score(y_teste_gra, previsoes_gra)
f1Score_gra = f1_score(y_teste_gra,previsoes_gra)
curva_roc_escore_gra = roc_auc_score(y_teste_gra, previsoes_gra)
kappa_gra = cohen_kappa_score(y_teste_gra, previsoes_gra)
print(f'Acurácia:{round(acuracia_gra,2)}')
print(f'Especificidade:{round(especificidade_gra,2)}')
print(f'Precisão:{round(precisao_gra,2)}')
print(f'Recall ou Sensibilidade:{round(recall_gra,2)}')
print(f'F1-Score:{round(f1Score_gra,2)}')
print(f'Kappa:{round(kappa_gra,2)}')
print(f'Curva ROC:{round(curva_roc_escore_gra,2)}')

#Curva ROC
rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
rfp_ar, rvp_ar,lim3 = roc_curve(y_teste_ar,previsoes_ar)
rfp_gra, rvp_gra,lim4 = roc_curve(y_teste_gra,previsoes_gra)
pyplot.plot(rfp_knn, rvp_knn, marker='.',
label='KNN='+str(round(curva_roc_escore_knn,2)),color="orange")
pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random
Forest='+str(round(curva_roc_escore_rf,2)),color="blue")
pyplot.plot(rfp_ar, rvp_ar, marker='.', label='Árvore
Decisão='+str(round(curva_roc_escore_ar,2)),color="yellow")
pyplot.plot(rfp_gra, rvp_gra, marker='.', label='Gradient
Boosting='+str(round(curva_roc_escore_gra,2)),color="purple")
pyplot.plot([0, 1], [0, 1], color='black', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
# Legenda
pyplot.legend()
# Mostrando o gráfico
pyplot.show()
```

```
"""**SVM**

**A Execução da SVM com validação cruzada foi realizada localmente  em outro fonte e em um máquina local, devido ao tempo excedido de 8 horas de execução e, não suportado pelo Colab.**
"""

#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e 70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for executado
X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(previsores, classe,
                                                                  test_size = 0.3,random_state = 0)
print(len(X_treinamento))
print(len(X_teste))

#realizando a parte de pré-processamento no conjunto de treinamento, teste e ajustar a escala padrão.
sc = StandardScaler()
X_treinamento= sc.fit_transform(X_treinamento)
X_teste = sc.transform(X_teste)

#Aplicação da função PCA no treinamento  e conjunto de teste do componente X
#Fiz isso, pois o primeiro exemplo sem, ficou mais de três dia e não finalizou
pca = PCA(n_components = 3)
X_treinamento = pca.fit_transform(X_treinamento)
X_teste = pca.transform(X_teste)

#Avaliando o resultado da máquina
print(pca.explained_variance_ratio_)
print(pca.singular_values_)

# Criação do modelo, treinamento, obtenção das previsões
svm = SVC()
svm.fit(X_treinamento, y_treinamento)

#avalia o modelo com os dados de teste
svm.score(X_teste,y_teste)

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo,
#a partir de um conjunto de dados.
cv_scores = cross_val_score(svm, previsores, classe, cv=5, scoring='accuracy')
print(f'Scores = {cv_scores}')
print(f'Média dos Scores = {cv_scores.mean()}')

#obtenção das previsões
previsoes_svm = svm.predict(X_teste)
previsoes_svm

# Visualização da matriz de confusão
print(confusion_matrix(y_teste,previsoes_svm))
print(pd.crosstab(y_teste,previsoes_svm,rownames=['Real'],colnames=['Predito'],margins= True))

#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall, F1-Score
acuracia_svm = accuracy_score(y_teste,previsoes_svm)
especificidade_svm = specificity_score(y_teste,previsoes_svm)
precisao_svm = precision_score(y_teste, previsoes_svm)
recall_svm = recall_score(y_teste, previsoes_svm)
f1Score_svm = f1_score(y_teste,previsoes_svm)
```

```
curva_roc_escore_svm = roc_auc_score(y_teste, previsoes_svm)
kappa_svm = cohen_kappa_score(y_teste, previsoes_svm)
print(f'Acurácia:{round(acuracia_svm,4)}')
print(f'Especificidade:{round(especificidade_svm,4)}')
print(f'Precisão:{round(precisao_svm,4)}')
print(f'Recall ou Sensibilidade:{round(recall_svm,4)}')
print(f'F1-Score:{round(f1Score_svm,4)}')
print(f'Kappa:{round(kappa_svm,4)}')
print(f'Curva ROC:{round(curva_roc_escore_svm,4)}')

#Curva ROC
rfp_knn, rvp_knn,lim1 = roc_curve(y_teste,previsoes_knn)
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste,previsoes_rf)
rfp_ar, rvp_ar,lim3 = roc_curve(y_teste,previsoes_ar)
rfp_gra, rvp_gra,lim4 = roc_curve(y_teste,previsoes_gra)
rfp_svm, rvp_svm,lim5 = roc_curve(y_teste,previsoes_svm)
pyplot.plot(rfp_knn, rvp_knn, marker='.',
label='KNN='+str(round(curva_roc_escore_knn,2)),color="orange")
pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random
Forest='+str(round(curva_roc_escore_rf,2)),color="blue")
pyplot.plot(rfp_ar, rvp_ar, marker='.', label='Árvore
Decisão='+str(round(curva_roc_escore_ar,2)),color="yellow")
pyplot.plot(rfp_gra, rvp_gra, marker='.', label='Gradient
Boosting='+str(round(curva_roc_escore_gra,2)),color="purple")
pyplot.plot(rfp_svm, rvp_svm, marker='.',
label='SVM='+str(round(curva_roc_escore_svm,4)),color="red")
pyplot.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
# Legenda
pyplot.legend()
# Mostrando o gráfico
pyplot.show()

"""**XYZ - XGBoost**"""

#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e
70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for
executado
X_treinamento_xgb, X_teste_xgb, y_treinamento_xgb, y_teste_xgb =
train_test_split(previsores, classe,
                                                                test_size =
0.3,random_state = 0)
print(len(X_treinamento_xgb))
print(len(X_teste_xgb))

#realizando a parte de pré-processamento no conjunto de treinamento, teste e
ajustar a escala padrão.
sc = StandardScaler()
X_treinamento_xgb = sc.fit_transform(X_treinamento_xgb)
X_teste_xgb = sc.transform(X_teste_xgb)
X_teste_xgb

#Aplicação da função PCA no treinamento  e conjunto de teste do componente X
#pca = PCA(n_components = 2)
#X_treinamento_xgb = pca.fit_transform(X_treinamento_xgb)
#X_teste_xgb = pca.transform(X_teste_xgb)
#explained_variance = pca.explained_variance_ratio_

# Criação do modelo, treinamento, obtenção das previsões
xgboost_m = XGBClassifier()
xgboost_m.fit(X_treinamento_xgb, y_treinamento_xgb)
```

```
#avalia o modelo com os dados de teste
xgboost_m.score(X_teste_xgb,y_teste_xgb)

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade
de generalização de um modelo,
#a partir de um conjunto de dados.
cv_scores = cross_val_score(xgboost_m, previsores, classe, cv=5,
scoring='accuracy')
print(f'Scores = {cv_scores}')
print(f'Média dos Scores = {cv_scores.mean()}')

#obtenção das previsões
previsoes_xgb = xgboost_m.predict(X_teste_xgb)
previsoes_xgb

# Visualização da matriz de confusão
print(confusion_matrix(y_teste_xgb,previsoes_xgb))
print(pd.crosstab(y_teste_xgb,previsoes_xgb,rownames=['Real'],colnames=['Predito
'],margins= True))

#gerando o vetor
xgb_vetor =   converteVetor(confusion_matrix(y_teste_xgb,previsoes_xgb))

#Curca ROC isoladamente
curva_roc_escore_xgb = roc_auc_score(y_teste_xgb, previsoes_xgb)
curva_roc_escore_xgb

#Avaliação da máquina preditiva
r2_score(y_teste_xgb,previsoes_xgb)

#Vericando os atributos com maior peso
#feature_import = xgboost_m.get_booster().get_score(importance_type = 'weight')
#keys = list(feature_import.keys())
#values = list(feature_import.values())

#data = pd.DataFrame(data= values, index =
keys,columns=['score']).sort_values(by = 'score',ascending = True)
#data.plot(kind = 'barh')

#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,
F1-Score
acuracia_xgb = accuracy_score(y_teste_xgb,previsoes_xgb)
especificidade_xgb = specificity_score(y_teste_xgb,previsoes_xgb)
precisao_xgb = precision_score(y_teste_xgb, previsoes_xgb)
recall_xgb = recall_score(y_teste_xgb, previsoes_xgb)
f1Score_xgb = f1_score(y_teste_xgb,previsoes_xgb)
curva_roc_escore_xgb = roc_auc_score(y_teste_xgb, previsoes_xgb)
kappa_xgb = cohen_kappa_score(y_teste_xgb, previsoes_xgb)
print(f'Acurácia:{round(acuracia_xgb,2)}')
print(f'Especificidade:{round(especificidade_xgb,2)}')
print(f'Precisão:{round(precisao_xgb,2)}')
print(f'Recal ou Sensibilidade:{round(recall_xgb,2)}')
print(f'F1-Score:{round(f1Score_xgb,2)}')
print(f'Kappa:{round(kappa_xgb,2)}')
print(f'Curva ROC:{round(curva_roc_escore_xgb,2)}')

#Curva ROC
rfp_knn, rvp_knn,lim1 = roc_curve(y_teste_knn,previsoes_knn)
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste_rf,previsoes_rf)
rfp_ar, rvp_ar,lim3 = roc_curve(y_teste_ar,previsoes_ar)
rfp_gra, rvp_gra,lim4 = roc_curve(y_teste_gra,previsoes_gra)
#rfp_svm, rvp_svm,lim5 = roc_curve(y_teste,previsoes_svm)
rfp_xgb, rvp_xgb,lim5 = roc_curve(y_teste_xgb,previsoes_xgb)
```

```
pyplot.plot(rfp_knn, rvp_knn, marker='.',
label='KNN='+str(round(curva_roc_escore_knn,2)),color="orange")
pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random
Forest='+str(round(curva_roc_escore_rf,2)),color="blue")
pyplot.plot(rfp_ar, rvp_ar, marker='.', label='Árvore
Decisão='+str(round(curva_roc_escore_ar,2)),color="yellow")
pyplot.plot(rfp_gra, rvp_gra, marker='.', label='Gradient
Boosting='+str(round(curva_roc_escore_gra,2)),color="purple")
#pyplot.plot(rfp_svm, rvp_svm, marker='.',
label='SVM='+str(round(curva_roc_escore_svm,4)),color="red")
pyplot.plot(rfp_xgb, rvp_xgb, marker='.',
label='XGBoost='+str(round(curva_roc_escore_xgb,2)),color="pink")
pyplot.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
# Legenda
pyplot.legend()
# Mostrando o gráfico
pyplot.show()

"""**Comparando e discutindo a diferença da visualização dos dados em um radar
plot e em um gráfico de barras**

**Radar Plot**
"""

#Predito   N       P
#Real
#N        VN      FP
#P        FN      VP
categories = ['VN', 'FP', 'FN', 'VP']
categories = [*categories, categories[0]]
#KNN
knn_vetor = [*knn_vetor,knn_vetor[0]]
#Ramdom Forest
floresta_vetor = [*floresta_vetor,floresta_vetor[0]]
#Árvores de decisão
arvore_vetor = [*arvore_vetor,arvore_vetor[0]]
#Gradient
gradient_vetor = [*gradient_vetor,gradient_vetor[0]]
#SVM
#svm_matriz = [76395,23933,7589,92723]
#svm_matriz = [*svm_matriz,svm_matriz[0]]
#XGBoost
#xgb_vetor = [80506,19822,7273,93039]
xgb_vetor = [*xgb_vetor,xgb_vetor[0]]

label_loc = np.linspace(start=0, stop=2 * np.pi, num=len(knn_vetor))
plt.figure(figsize=(8, 8))
plt.subplot(polar=True)
plt.plot(label_loc,knn_vetor,label='KNN')
plt.plot(label_loc, floresta_vetor, label='Random Forest')
plt.plot(label_loc, arvore_vetor, label='Árvore de Decisão')
plt.plot(label_loc, gradient_vetor, label='Gradient Boosting')
#plt.plot(label_loc, svm_matriz, label='SVM')
plt.plot(label_loc, xgb_vetor, label='XGBoost')
plt.title('Comparação entre Matrizes', size=20, y=1.05)
lines, labels = plt.thetagrids(np.degrees(label_loc), labels=categories)
plt.legend()
plt.show()

"""**Gráfico de Barras**"""
```

```
data = [['KNN',knn_vetor[0],knn_vetor[1],knn_vetor[2],knn_vetor[3]],
      ['Random
Forest',floresta_vetor[0],floresta_vetor[1],floresta_vetor[2],floresta_vetor[3]]
,
      ['Árvore
Decisão',arvore_vetor[0],arvore_vetor[1],arvore_vetor[2],arvore_vetor[3]],

['Gradient',gradient_vetor[0],gradient_vetor[1],gradient_vetor[2],gradient_vetor
[3]],
      #['SVM',svm_matriz[0],svm_matriz[1],svm_matriz[2],svm_matriz[3]],
      ['XGBoost',xgb_vetor[0],xgb_vetor[1],xgb_vetor[2],xgb_vetor[3]]
     ]
df1 = pd.DataFrame(data,columns=['Matrizes','VN', 'FP', 'FN', 'VP'])
df1.plot(x="Matrizes", y=['VN', 'FP', 'FN', 'VP'], kind="bar",figsize=(15,8))
plt.show()
```

ANEXO C - CÓDIGO-FONTE DO ALGORITMO MÁQUINA VETOR DE SUPORTE(SVM)

```
import numpy as np
import pandas as pd
import seaborn as srn
import statistics as sts
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score,confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score,roc_curve,auc,r2_score
from sklearn.neighbors import KNeighborsClassifier
from scipy import stats
from sklearn.model_selection import cross_val_score
from imblearn.metrics import specificity_score
import matplotlib.pyplot  as pyplot
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE


#Lendo o arquivo
df_train = pd.read_csv('train.csv',sep=',')
df_train.head()

#Separar os atributos previsores da classe
#Criando um dataFrame realizar o One-Hot encoder
previsores_df = df_train.iloc[:,1:11]
#df_train[df_train.columns[1:11]]
#Aqui quer dizer que irei percorrer todas as linas inserir em classe a coluna
12
classe = df_train.iloc[:,11].values
#Exibir os valores
print(classe)

#oneHotEncoder com a função  get_dummies do pandas
#Agora tentando remodelar com (-1, 1). Fornecemos coluna como 1, mas linhas como
desconhecidas
previsores_df = pd.concat([previsores_df,pd.get_dummies(previsores_df['Gender'],
prefix='Gender')],axis=1)
previsores_df.drop(['Gender'],axis=1, inplace=True)
previsores_df.head()

# Transformação dos atributos categóricos em atributos numéricos.
labelencoder = LabelEncoder()
previsores_df['Vehicle_Age'] =
labelencoder.fit_transform(previsores_df['Vehicle_Age'])
group = previsores_df.groupby(['Vehicle_Age']).size()
previsores_df.head()

previsores_df.loc[previsores_df['Vehicle_Damage']=='No','Vehicle_Damage'] = '0'
previsores_df.loc[previsores_df['Vehicle_Damage']=='Yes','Vehicle_Damage'] = '1'
group = previsores_df.groupby(previsores_df['Vehicle_Damage']).size()
previsores_df.head()

#Aqui quer dizer que irei percorrer todas as linhas inserir em previsores as
colunas de 0 a 10
previsores = previsores_df.iloc[:,0:11].values
print(previsores)
print(classe)

#Balanceamento das Classes com o método Smote.
```

```
smt = SMOTE()
previsores,classe = smt.fit_resample(previsores,classe)

#Fazendo a contagem para ver como ficou
yi = classe.astype(int)
np.bincount(yi)
#visualização em um gráfico
ax = srn.countplot(x=yi)

#Divisão da base de dados entre treinamento e teste. Usamos 30% para testar e
70% para treinar.
#Random_state = 0 para sempre obter a mesma divisão da base quando o código for
executado
X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(previsores,
classe,
                                                                  test_size =
0.3,random_state = 0)
print(len(X_treinamento))
print(len(X_teste))

#realizando a parte de pré-processamento no conjunto de treinamento, teste e
ajustar a escala padrão.
sc = StandardScaler()
X_treinamento= sc.fit_transform(X_treinamento)
X_teste = sc.transform(X_teste)

# Criação do modelo, treinamento, obtenção das previsões
svm = SVC()
svm.fit(X_treinamento, y_treinamento)


#avalia o modelo com os dados de teste
svm.score(X_teste,y_teste)

#Cross Validation. A validação cruzada é uma técnica para avaliar a capacidade
de generalização de um modelo,
#a partir de um conjunto de dados.
print('Fazendo a validação cruzada')
cv_scores = cross_val_score(svm, previsores, classe, cv=5, scoring='accuracy')
print(f'Scores = {cv_scores}')
print(f'Média dos Scores = {cv_scores.mean()}')

#obtenção das previsões
previsoes_svm = svm.predict(X_teste)
previsoes_svm

# Visualização da matriz de confusão
print(confusion_matrix(y_teste,previsoes_svm))
print(pd.crosstab(y_teste,previsoes_svm,rownames=['Real'],colnames=['Predito'],m
argins= True))
#Criando um função para tranformar a matriz em vetor
#para usar no Radar Plot e Barras

def converteVetor(m):
    vetor = []
    for i in range(len(m)):
        for j in range(len(m[i])):
            vetor.append(m[i][j])
    return vetor


svm_vetor =   converteVetor(confusion_matrix(y_teste,previsoes_svm))
print(svm_vetor)
```

```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,
F1-Score
acuracia_svm = accuracy_score(y_teste,previsoes_svm)
especificidade_svm = specificity_score(y_teste,previsoes_svm)
precisao_svm = precision_score(y_teste, previsoes_svm)
recall_svm = recall_score(y_teste, previsoes_svm)
f1Score_svm = f1_score(y_teste,previsoes_svm)
curva_roc_escore_svm = roc_auc_score(y_teste, previsoes_svm)
kappa_svm = cohen_kappa_score(y_teste, previsoes_svm)
print(f'Acurácia:{round(acuracia_svm,4)}')
print(f'Especificidade:{round(especificidade_svm,4)}')
print(f'Precisão:{round(precisao_svm,4)}')
print(f'Recall ou Sensibilidade:{round(recall_svm,4)}')
print(f'F1-Score:{round(f1Score_svm,4)}')
print(f'Kappa:{round(kappa_svm,4)}')
print(f'Curva ROC:{round(curva_roc_escore_svm,4)}')

#Curva ROC
rfp_svm, rvp_svm,lim5 = roc_curve(y_teste,previsoes_svm)
pyplot.plot(rfp_svm, rvp_svm, marker='.',
label='SVM='+str(round(curva_roc_escore_svm,4)),color="red")
pyplot.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
# Legenda
pyplot.legend()
# Mostrando o gráfico
pyplot.show()

#Radar Plot
#SVM
categories = ['VN', 'FP', 'FN', 'VP']
categories = [*categories, categories[0]]
svm_vetor = [*svm_vetor,svm_vetor[0]]

#Imprimindo o Gŕafico
label_loc = np.linspace(start=0, stop=2 * np.pi, num=len(svm_vetor))
plt.figure(figsize=(8, 8))
plt.subplot(polar=True)
plt.plot(label_loc, svm_vetor, label='SVM')
plt.title('Comparação entre Matrizes', size=20, y=1.05)
lines, labels = plt.thetagrids(np.degrees(label_loc), labels=categories)
plt.legend()
plt.show()

#Gráfico de Barras
data = [['SVM',svm_vetor[0],svm_vetor[1],svm_vetor[2],svm_vetor[3]]]
df1 = pd.DataFrame(data,columns=['Matrizes','VN', 'FP', 'FN', 'VP'])
df1.plot(x="Matrizes", y=['VN', 'FP', 'FN', 'VP'], kind="bar",figsize=(15,8))
plt.show()
```