November 13, 2023

1 Lista 1 de Machine Learning - Higor David Oliveira - AM 23/2

1.1 Questão 1

[ ]:
```
# %pip install ucimlrepo
# %pip install matplotlib
# %pip install pandas
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Leitura do banco de dados

[ ]:
```
data_1 = pd.read_csv('data/haberman+s+survival/haberman.data', header = None,␣
 ↪names = ['Age', 'Year_of_operation', 'Number_of_nodes', 'Survival_status'])
data_1.head() # mostra o começo do dataset
```

[ ]:

	Age	Year_of_operation	Number_of_nodes	Survival_status
0	30	64	1	1
1	30	62	3	1
2	30	65	0	1
3	31	59	2	1
4	31	65	4	1

1. Age of patient at time of operation (numerical)
2. Patient's year of operation (year - 1900, numerical)
3. Number of positive axillary nodes detected (numerical)
4. Survival status (class attribute)

a e b) Média e mediana dos atributos para o banco de dados e para cada classe.

[ ]:
```
atribute_index = np.arange(0,3)
print(f'--- Média dos atributos de todas as classes:\n {data_1.iloc[:
 ↪,atribute_index].mean()}')
print(f'--- Mediana dos atributos de todas as classes:\n {data_1.iloc[:
 ↪,atribute_index].median()}')
# print("Média dos atributos de todas as classes:\n", data_1.iloc[:
 ↪,atribute_index].mean())
```

```
# print("Mediana dos atributos de todas as classes:\n", data_1.iloc[:
 ↪,atribute_index].median())

# Separação entre classes
data_1_class1 = data_1[(data_1.Survival_status == 1)]
data_1_class2 = data_1[(data_1.Survival_status == 2)]

print(f'--- Média dos atributos da classe 1:\n {data_1_class1.iloc[:
 ↪,atribute_index].mean()}')
print(f'--- Mediana dos atributos da classe 1:\n {data_1_class1.iloc[:
 ↪,atribute_index].median()}')
print(f'--- Média dos atributos da classe 2:\n {data_1_class2.iloc[:
 ↪,atribute_index].mean()}')
print(f'--- Mediana dos atributos da classe 2:\n {data_1_class2.iloc[:
 ↪,atribute_index].median()}')
# print("Média dos atributos da classe 1:\n", data_1_class1.iloc[:
 ↪,atribute_index].mean())
# print("Mediana dos atributos da classe 1:\n", data_1_class1.iloc[:
 ↪,atribute_index].median())
# print("Média dos atributos da classe 2:\n", data_1_class2.iloc[:
 ↪,atribute_index].mean())
# print("Mediana dos atributos da classe 2:\n", data_1_class2.iloc[:
 ↪,atribute_index].median())
```

```
--- Média dos atributos de todas as classes:
 Age                  52.457516
Year_of_operation    62.852941
Number_of_nodes       4.026144
dtype: float64
--- Mediana dos atributos de todas as classes:
 Age                  52.0
Year_of_operation    63.0
Number_of_nodes       1.0
dtype: float64
--- Média dos atributos da classe 1:
 Age                  52.017778
Year_of_operation    62.862222
Number_of_nodes       2.791111
dtype: float64
--- Mediana dos atributos da classe 1:
 Age                  52.0
Year_of_operation    63.0
Number_of_nodes       0.0
dtype: float64
--- Média dos atributos da classe 2:
 Age                  53.679012
Year_of_operation    62.827160
```

```
Number_of_nodes       7.456790
dtype: float64
--- Mediana dos atributos da classe 2:
 Age                   53.0
Year_of_operation     63.0
Number_of_nodes        4.0
dtype: float64
```

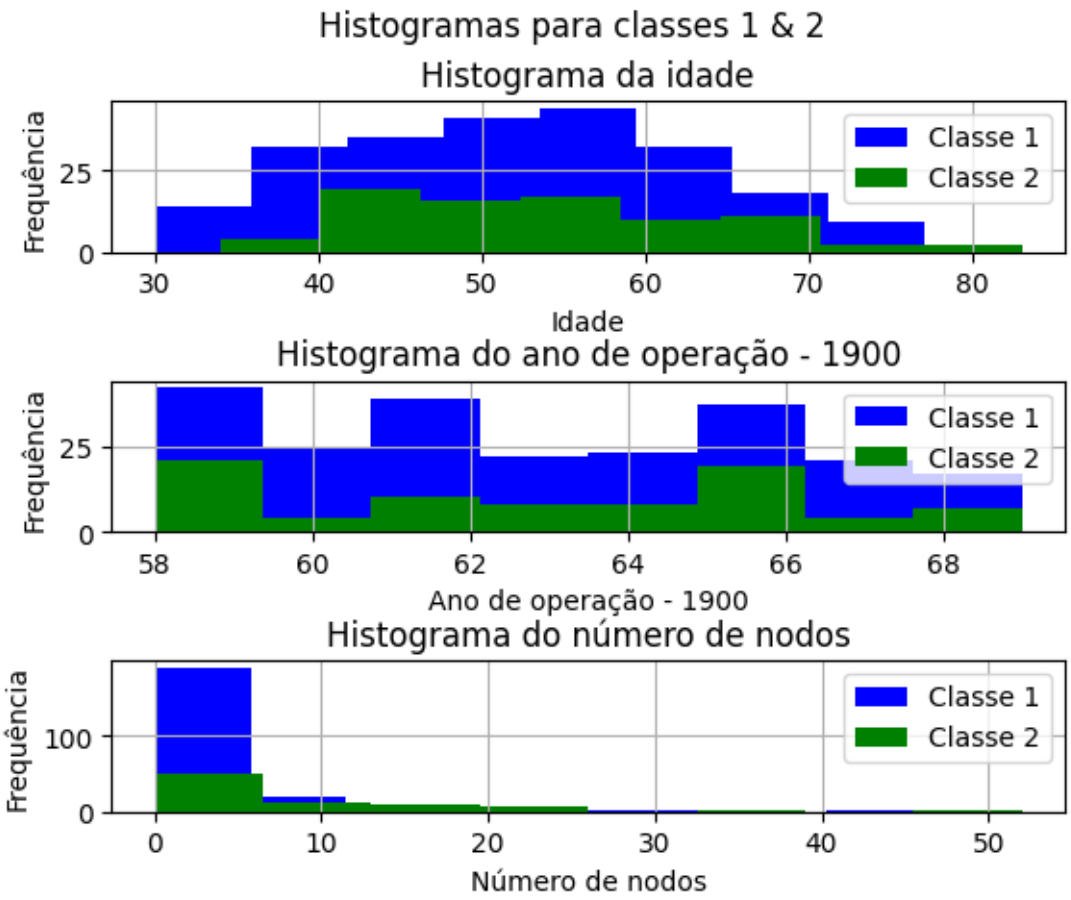
c) Matriz de coeficientes de correlação Para calcular a matriz de correlação, basta usarmos o método do Pandas df.corr()

[ ]:
```
data_1.corr('pearson')
```

[ ]:
```
                          Age  Year_of_operation  Number_of_nodes  \
Age                  1.000000           0.089529        -0.063176
Year_of_operation    0.089529           1.000000        -0.003764
Number_of_nodes     -0.063176          -0.003764         1.000000
Survival_status      0.067950          -0.004768         0.286768

                     Survival_status
Age                         0.067950
Year_of_operation          -0.004768
Number_of_nodes             0.286768
Survival_status             1.000000
```

d) Histograma de cada atributo

[ ]:
```
fig1, axs1 = plt.subplots(3,1)
plt.suptitle("Histogramas para classes 1 & 2")
plt.subplots_adjust(hspace=0.85)
titles = ["Histograma da idade", "Histograma do ano de operação - 1900",␣
 ↪"Histograma do número de nodos"]
xlabels = ["Idade","Ano de operação - 1900","Número de nodos"]
for i in range(0,3):
    axs1[i].hist(data_1_class1.iloc[:,i], bins=8, color="blue")
    axs1[i].hist(data_1_class2.iloc[:,i], bins=8, color="green")
    axs1[i].set_title(titles[i])
    axs1[i].set_ylabel("Frequência")
    axs1[i].set_xlabel(xlabels[i])
    axs1[i].legend(["Classe 1", "Classe 2"])
    axs1[i].grid()
```



Observando a distribuição das duas classes para cada atributo, podemos concluir que a tarefa de classificação não será fácil, pois a distribuição dos atributos está muito entre as duas classes.

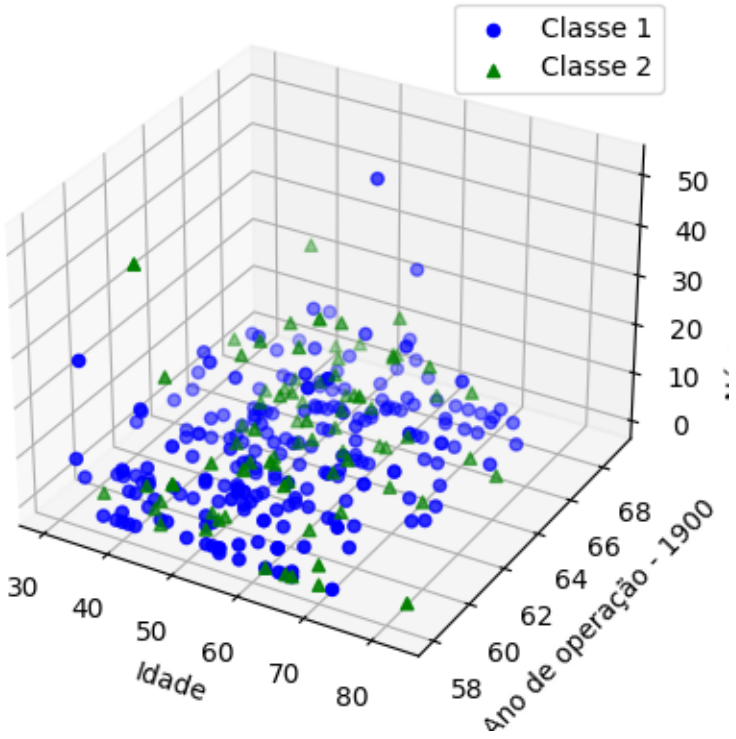
Pelo histograma, ainda é possível perceber que o número de amostras não está balanceado entre as duas classes.

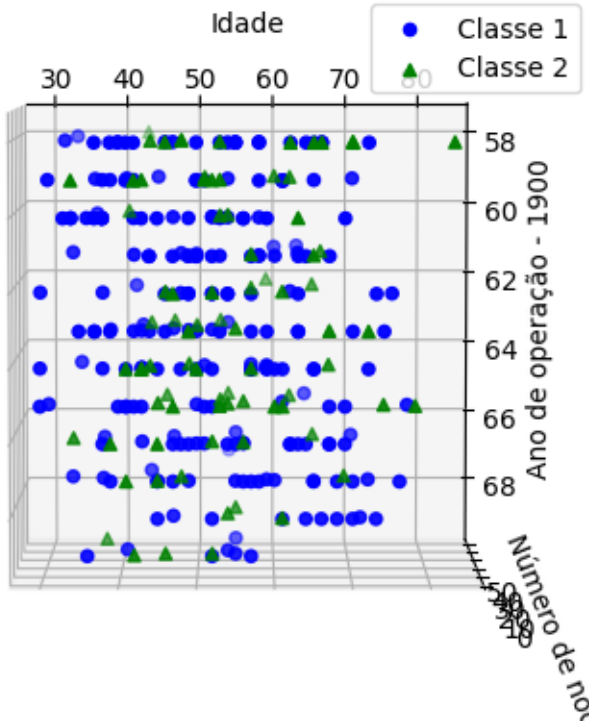
e) Gráfico 3D das amostras

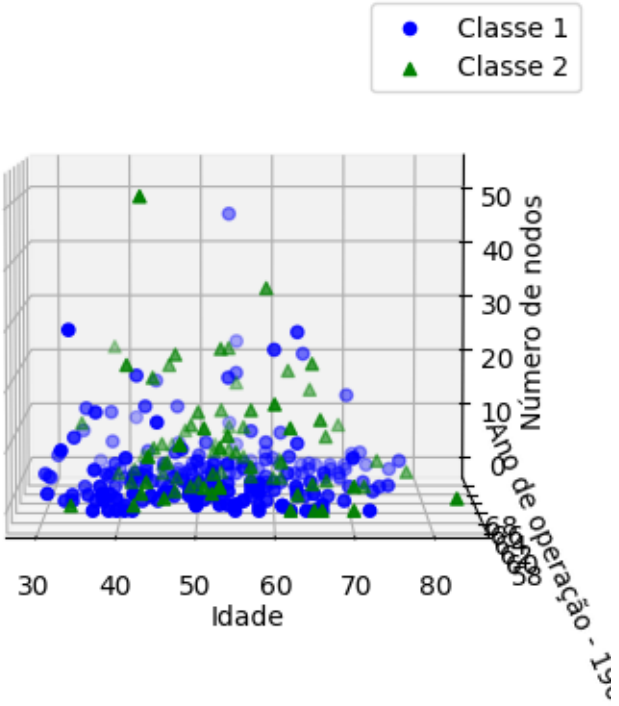
[ ]:
```
fig2 = plt.figure()
ax2 = fig2.add_subplot(projection='3d')
ax2.scatter3D(list(data_1_class1.iloc[:,0]),list(data_1_class1.iloc[:
 ↪,1]),list(data_1_class1.iloc[:,2]), marker='o', color='blue')
ax2.scatter3D(list(data_1_class2.iloc[:,0]),list(data_1_class2.iloc[:
 ↪,1]),list(data_1_class2.iloc[:,2]), marker='^', color='green')
ax2.set_xlabel("Idade")
ax2.set_ylabel("Ano de operação - 1900")
ax2.set_zlabel("Número de nodos")
ax2.legend(["Classe 1", "Classe 2"])

fig3 = plt.figure()
```

```
ax3 = fig3.add_subplot(projection='3d')
ax3.scatter3D(list(data_1_class1.iloc[:,0]),list(data_1_class1.iloc[:
 ↪,1]),list(data_1_class1.iloc[:,2]), marker='o', color='blue')
ax3.scatter3D(list(data_1_class2.iloc[:,0]),list(data_1_class2.iloc[:
 ↪,1]),list(data_1_class2.iloc[:,2]), marker='^', color='green')
ax3.set_xlabel("Idade")
ax3.set_ylabel("Ano de operação - 1900")
ax3.set_zlabel("Número de nodos")
ax3.view_init(elev=-85,azim=-90)
ax3.legend(["Classe 1", "Classe 2"])

fig4 = plt.figure()
ax4 = fig4.add_subplot(projection='3d')
ax4.scatter3D(list(data_1_class1.iloc[:,0]),list(data_1_class1.iloc[:
 ↪,1]),list(data_1_class1.iloc[:,2]), marker='o', color='blue')
ax4.scatter3D(list(data_1_class2.iloc[:,0]),list(data_1_class2.iloc[:
 ↪,1]),list(data_1_class2.iloc[:,2]), marker='^', color='green')
ax4.set_xlabel("Idade")
ax4.set_ylabel("Ano de operação - 1900")
ax4.set_zlabel("Número de nodos")
ax4.view_init(elev=5,azim=-90)
ax4.legend(["Classe 1", "Classe 2"])
```

[ ]: <matplotlib.legend.Legend at 0x2c63e7a4c40>




Idade
Classe 1
Classe 2
30
40
50
60
70
58
60
62
64
66
68
Ano de operação - 1900


Classe 1
Classe 2
50
40
30
20
10
Número de nodos
30
40
50
60
70
80
Idade

Analisando os gráficos plotados acima, que foram usados as 3 características do banco de dados como eixos, é possível concluir que a tarefa de classificação para diferenciar as duas classes muito dificilmente terá uma acurácia alta.

Assim também, vendo a distruibuição da frequência dos valores nos histogramas, vemos que o "formato" da distruibuição dos dados se assemelha tanto para a classe 1 quanto pra classe 2, ficando, assim, difícil fazer um limiar de separação claro entre elas.

1.2 Questão 2

Calcule a informação mútua de cada atributo para a classe do banco de dados car evaluation.

[ ]:
```
# from ucimlrepo import fetch_ucirepo # código extraido do próprio site de␣
 ↪download do banco de dados

# fetch dataset
# car_evaluation = fetch_ucirepo(id=19)

# data (as pandas dataframes)
# data_2 = car_evaluation
# data_2_feat = car_evaluation.data.features
# data_2_targ = car_evaluation.data.targets

# # metadata
# print(car_evaluation.metadata)

# # variable information
# print(car_evaluation.variables)

data_2 = pd.read_csv('data/car+evaluation/car.data', names=['buying', 'maint',␣
 ↪'doors', 'persons', 'lug_boot', 'safety', 'target'])
data_2_feat = data_2.iloc[:,:-1]
data_2_targ = data_2.iloc[:,-1:]

print(data_2.head())
# print("Dados type: ", type(data_2))
# print(data_2_feat.head())
# print(data_2_targ.head())
```

```
  buying  maint doors persons lug_boot safety target
0  vhigh  vhigh     2       2    small    low  unacc
1  vhigh  vhigh     2       2    small    med  unacc
2  vhigh  vhigh     2       2    small   high  unacc
3  vhigh  vhigh     2       2      med    low  unacc
4  vhigh  vhigh     2       2      med    med  unacc
```

Informação Mútua (MI) pode ser calculada 2 a 2 usando a expressão abaixo:

$$M(x_1, x_2) = H(x_1) + H(x_2) - H(x_1, x_2)$$

sendo, H função entropia, dada por:

$$H(x) = -\sum_{x_1} p(x_1) log_2[p(x_1)]$$

E por:

$$H(x_1, x_2) = -\sum_{x_1, x_2} p(x_1, x_2) log_2[p(x_1, x_2)]$$

sendo, p função probabilidade,
A probabilidade será calculada como média simples da frequência

$$p(x_1 = a) = \frac{frequncia(a)}{n^\circ \; amostras \; x_1}$$

Calculando a frequência e entropia de cada dado, temos:

[ ]:
```
data_2_pd = data_2_feat # Base completa
data_2_pd['target'] = data_2_targ

# Calculo da probabilidade de cada elemento da classe alvo
target_count = data_2_targ.groupby(data_2_targ.columns[-1])[data_2_targ.
 ↪columns[-1]].count()
target_freq = target_count.transform(lambda x : x/target_count.sum())
for i, column in enumerate(data_2_feat.columns):
    if column == data_2_pd.columns[-1]:
        continue

    # Calculo da probabilidade de cada elemento da coluna de atributo
    column_count = data_2_feat.groupby(column)[column].count()
    column_freq = column_count.transform(lambda x : x/column_count.sum())

    # Calculo da probabilidade de cada elemento da coluna com a coluna alvo␣
 ↪("target" ou "class")
    column_count_target = data_2_pd.groupby(data_2_pd.columns[-1])[column].
 ↪value_counts()
    column_freq_target = column_count_target.transform(lambda x : x/
 ↪column_count_target.sum())

    # Calculo da informação mútua
    #Entropia do atributo
    column_entropy = -sum(column_freq*np.log2(column_freq))

    #Entropia do alvo
    target_entropy = -sum(target_freq*np.log2(target_freq))
```

```
    #Entropia 2 a 2
    column_entropy_target = -sum(column_freq_target*np.log2(column_freq_target))

    #Informação mútua
    info_mutua = column_entropy + target_entropy - column_entropy_target

    print(f'A informação mútua entre os atributos de {column} e target é de␣
↪aproximadamente {info_mutua:.4f}')
    # print(f'A informação mútua entre os atributos de {column} e target é de␣
↪aproximadamente {info_mutua:.4f} = {column_entropy:.4f} + {target_entropy:.
↪4f} - {column_entropy_target:.4f}')
```

```
A informação mútua entre os atributos de buying e target é de aproximadamente
0.0964
A informação mútua entre os atributos de maint e target é de aproximadamente
0.0737
A informação mútua entre os atributos de doors e target é de aproximadamente
0.0045
A informação mútua entre os atributos de persons e target é de aproximadamente
0.2197
A informação mútua entre os atributos de lug_boot e target é de aproximadamente
0.0300
A informação mútua entre os atributos de safety e target é de aproximadamente
0.2622
```

Observando os valores obtidos, e, sendo a medida de informação mútua um indicativo de dependência entre os dados, podemos concluir que, os atributos de maior relevância para a classe são "safety" e "persons", pois possuem os maiores valores e, logo, não devem ser eliminados.

Já os atributos "lug_boot" e "doors" possuem valor de informação mútua muito pequena, logo, a variação no valor não contribue consideravelmente para a classe e assim, poderiam ser eliminados para simplificar os dados e, possivelmente, melhorar o desempenho de um classificador.

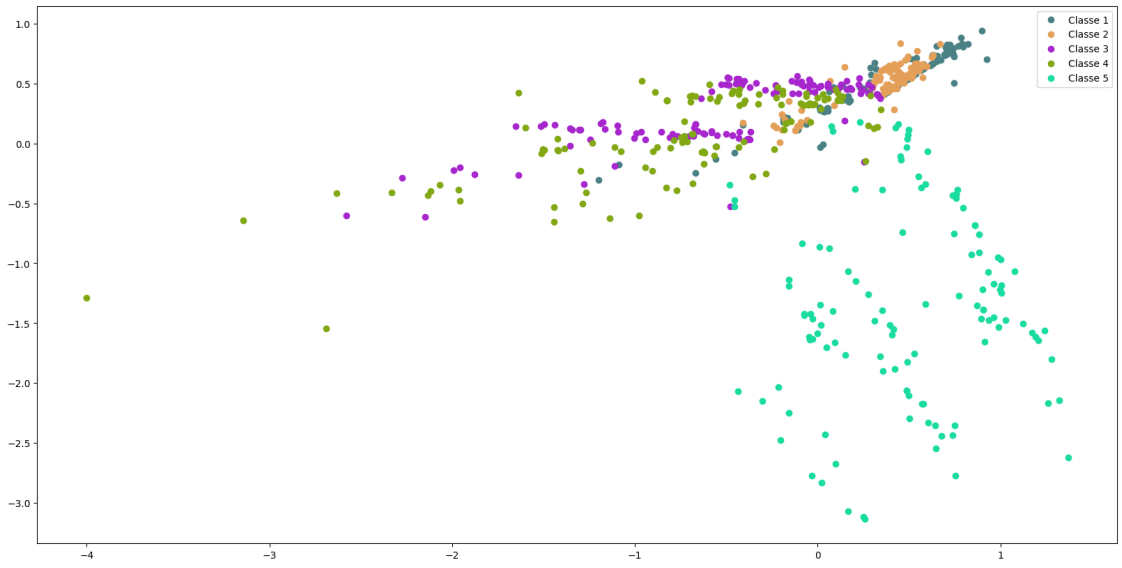
1.3 Questão 3: Base de dados CNAE_9_reduzido

Letra A) Importação dos dados

[ ]:
```
data_3 = pd.read_fwf('data/CNAE_9_reduzido/CNAE_9_reduzido.txt', header = None)
data_3.describe()

data_3_target = pd.DataFrame()
data_3_target['target'] = data_3.iloc[:,0]
data_3_atribute = data_3.iloc[:,1:-1]

#Usando matriz de covariância
eigValues, eigVectors = np.linalg.eig(data_3_atribute.cov())

#Subtrair dos dados a média
data_3_normalized = data_3_atribute - data_3_atribute.mean()
```

```
#Seleciona os dois autovetores associados aos maiores autovalores
#M será a matriz de projeção dos dados
M = eigVectors[:,0:2]

#Projeta os dados nos autovetores principais
proj_data = data_3_normalized @ M

classes = data_3_target[data_3_target.columns[0]].unique()

#Plota o gráfico
plt.figure(figsize=(20,10))
legend = []
data_3_color_map_list = []
for classe in classes:
    color = np.random.uniform(0,1,3)
    data_select = proj_data[data_3_target[data_3_target.columns[0]] == classe]
    plt.plot(data_select[1],data_select[0],marker='o',linestyle='',color =␣
 ↪color, label=classe)
    data_3_color_map_list.append(color)
    legend.append(f'Classe {classe}')
plt.legend(legend)
data_3_color_map_dict = dict(zip(classes, data_3_color_map_list))
```

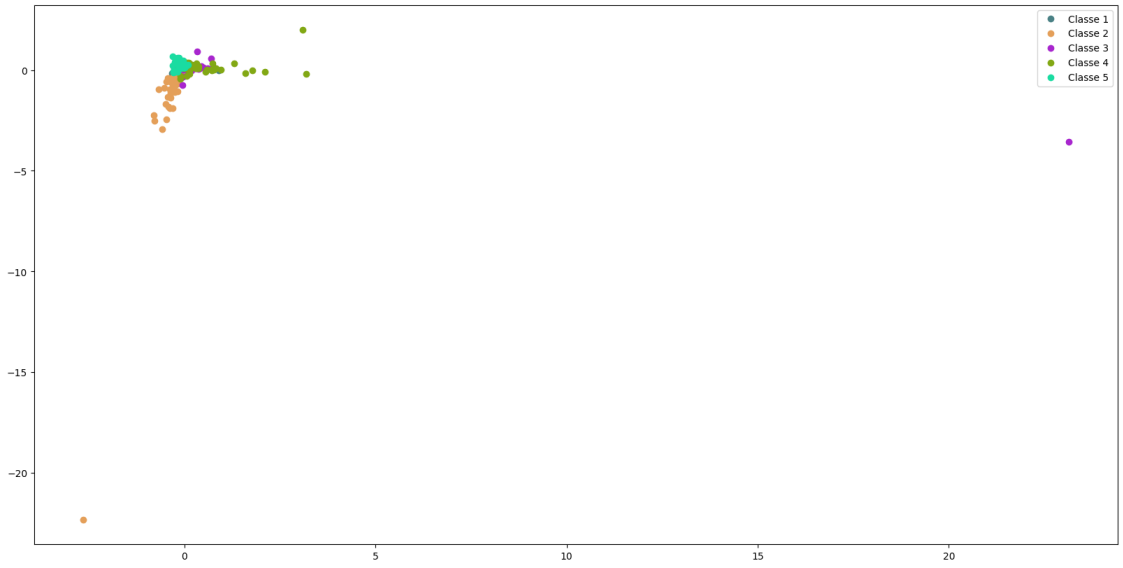
```
C:\Users\higor\AppData\Roaming\Python\Python310\site-
packages\matplotlib\cbook\__init__.py:1340: ComplexWarning: Casting complex
values to real discards the imaginary part
  return np.asarray(x, float)
```



letra B) PCA com branqueamento dos dados Primeiro, vamos extrair a matriz de correlação entre os dados.

[ ]:
```
#Usando matriz de coef. de correlação
eigValues_corr, eigVectors_corr = np.linalg.eig(data_3_atribute.corr())
# eigValues_cov, eigVectors_cov = np.linalg.eig(data_3_atribute.cov())

#Seleciona os dois autovetores associados aos maiores autovalores
#M será a matriz de projeção dos dados
M_full = eigVectors_corr[:,:]

# Calculando a matriz diagonal V. V é a raiz quadrada do inverso da matriz␣
 ↪formada pelos autovalores vindos da correlação
eigVectors_corr_diag = np.diag(eigValues_corr)

data_3_normalized_centralized = (data_3_atribute - data_3_atribute.mean())/
 ↪data_3_atribute.std()
# data_3_normalized_centralized = (data_3_atribute - data_3_atribute.mean())

# Para matrizes diagonais, a raiz quadrada é calculada como raiz quadrada dos␣
 ↪elementos centrais
V = np.linalg.inv(eigVectors_corr_diag)
V = np.apply_along_axis(np.sqrt, 0, V)

# Calcula a matriz de projeção MV
MV = M_full @ V.T
# MV = MV[:,0:2] # Não é necessário, mas simplifica calculos

# #Projeta os dados na matriz de branqueamento
proj_data_branc = data_3_normalized_centralized @ MV

# #Plota o gráfico
plt.figure(figsize=(20,10))
for classe in classes:
    color = data_3_color_map_dict[classe]
    data_select = proj_data_branc[data_3_target[data_3_target.columns[-1]] ==␣
 ↪classe]
    plt.plot(data_select[data_select.columns[0]],data_select[data_select.
 ↪columns[1]],marker='o',linestyle='',color = color, label=classe)
plt.legend(legend)

# Existem 2 atributos que fogem do padrão do restante e são plotados distante,␣
 ↪reduzindo o zoom do plot. Reacertando o zoom
# plt.xlim([-2, 4])
# plt.ylim([-4, 3])
```

[ ]:
```
<matplotlib.legend.Legend at 0x2c6871eb1f0>
```



A técnica de branqueamento dos dados faz com que os dados se tornem descorrelacionado. A descorrelação faz com que os atributos de cada classe se projetem sobre eixos (componentes principais) com maior distinção.

No nosso banco de dados, é possível ver que a descorrelação possibilitou separar melhor os atributos de algumas classes, como para as classes 3 e 2, porém, isso não aconteceu para todas as classes. Para demais classes, os atributos se projetam muito próximos, dificultando o trabalho de classificação por distância, por exemplo.

Letra C) TSNE

[ ]:
```
import tsne_python.tsne as tsne # tsne library under tsne_python\tsne.py

data_3_numpy = data_3_atribute.to_numpy()
data_3_tsne_output = tsne.tsne(data_3_numpy)

data_3_tsne_output_df = pd.DataFrame(data_3_tsne_output, columns=['x','y'])
data_3_tsne_output_df['target'] = data_3_target[data_3_target.columns[0]]

#Plota o gráfico
plt.figure(figsize=(20,10))
for classe in classes:
    color = data_3_color_map_dict[classe]
    data_select = data_3_tsne_output_df.
 ↪loc[data_3_tsne_output_df[data_3_tsne_output_df.columns[-1]] == classe]
    plt.plot(data_select[data_select.columns[0]],data_select[data_select.
 ↪columns[1]],marker='o',linestyle='',color = color, label=classe)
plt.legend(legend)
```

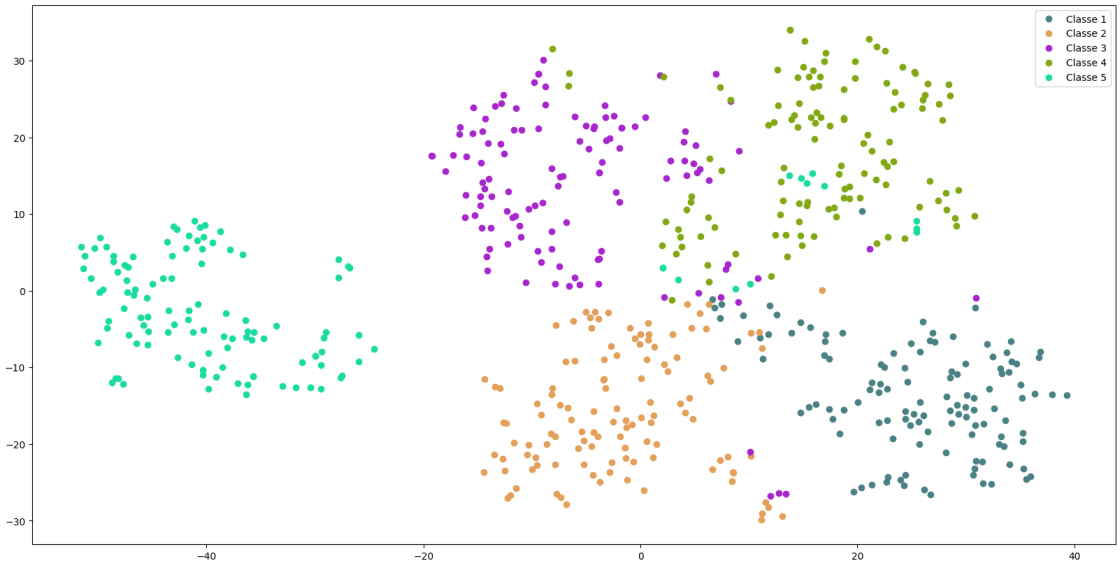
```
Preprocessing the data using PCA…
```

```
Computing pairwise distances…
Computing P-values for point 0 of 600…
Computing P-values for point 500 of 600…
Mean value of sigma: 0.899904
Iteration 10: error is 15.718864
Iteration 20: error is 14.449982
Iteration 30: error is 13.646034
Iteration 40: error is 13.764672
Iteration 50: error is 13.653682
Iteration 60: error is 13.650789
Iteration 70: error is 13.608732
Iteration 80: error is 13.664124
Iteration 90: error is 13.636309
Iteration 100: error is 13.657969
Iteration 110: error is 1.372649
Iteration 120: error is 0.955931
Iteration 130: error is 0.818586
Iteration 140: error is 0.754642
Iteration 150: error is 0.722214
Iteration 160: error is 0.703105
Iteration 170: error is 0.688476
Iteration 180: error is 0.676906
Iteration 190: error is 0.667999
Iteration 200: error is 0.662417
Iteration 210: error is 0.658561
Iteration 220: error is 0.655475
Iteration 230: error is 0.652901
Iteration 240: error is 0.650777
Iteration 250: error is 0.648886
Iteration 260: error is 0.647319
Iteration 270: error is 0.646137
Iteration 280: error is 0.645264
Iteration 290: error is 0.644585
Iteration 300: error is 0.644003
Iteration 310: error is 0.643502
Iteration 320: error is 0.643071
Iteration 330: error is 0.642703
Iteration 340: error is 0.642390
Iteration 350: error is 0.642120
Iteration 360: error is 0.641886
Iteration 370: error is 0.641681
Iteration 380: error is 0.641501
Iteration 390: error is 0.641339
Iteration 400: error is 0.641198
Iteration 410: error is 0.641074
Iteration 420: error is 0.640963
Iteration 430: error is 0.640863
Iteration 440: error is 0.640775
```

```
Iteration 450: error is 0.640697
Iteration 460: error is 0.640626
Iteration 470: error is 0.640561
Iteration 480: error is 0.640502
Iteration 490: error is 0.640449
Iteration 500: error is 0.640401
Iteration 510: error is 0.640356
Iteration 520: error is 0.640311
Iteration 530: error is 0.640245
Iteration 540: error is 0.640107
Iteration 550: error is 0.640000
Iteration 560: error is 0.639940
Iteration 570: error is 0.639894
Iteration 580: error is 0.639853
Iteration 590: error is 0.639813
Iteration 600: error is 0.639777
Iteration 610: error is 0.639752
Iteration 620: error is 0.639732
Iteration 630: error is 0.639714
Iteration 640: error is 0.639697
Iteration 650: error is 0.639682
Iteration 660: error is 0.639668
Iteration 670: error is 0.639654
Iteration 680: error is 0.639641
Iteration 690: error is 0.639627
Iteration 700: error is 0.639608
Iteration 710: error is 0.639565
Iteration 720: error is 0.639504
Iteration 730: error is 0.639477
Iteration 740: error is 0.639464
Iteration 750: error is 0.639455
Iteration 760: error is 0.639447
Iteration 770: error is 0.639440
Iteration 780: error is 0.639433
Iteration 790: error is 0.639425
Iteration 800: error is 0.639387
Iteration 810: error is 0.639329
Iteration 820: error is 0.639275
Iteration 830: error is 0.639211
Iteration 840: error is 0.639187
Iteration 850: error is 0.639177
Iteration 860: error is 0.639171
Iteration 870: error is 0.639166
Iteration 880: error is 0.639162
Iteration 890: error is 0.639158
Iteration 900: error is 0.639155
Iteration 910: error is 0.639152
Iteration 920: error is 0.639149
```

```
Iteration 930: error is 0.639146
Iteration 940: error is 0.639143
Iteration 950: error is 0.639139
Iteration 960: error is 0.639133
Iteration 970: error is 0.639119
Iteration 980: error is 0.639090
Iteration 990: error is 0.639057
Iteration 1000: error is 0.639032
```

[ ]: <matplotlib.legend.Legend at 0x2c63e8fd0f0>



Questão D) Primeiro, definindo as funções de auxílio para os algoritmos:

[ ]:
```python
def distancia_pw2(pt1, pt2):
    # Parameter assertion
    if pt1 is None or pt2 is None:
        print("Empty input parameter for distance", pt1, pt2)
        return
    # if not isinstance(pt1, np.ndarray) or not isinstance(pt2, np.ndarray):
    #     print(f"Wrong input data type. Expected {np.ndarray} got␣
 ↪{type(pt1)},{type(pt2)}")
    #     return

    dist = np.sum((pt1 - pt2)**2,axis=1)  # distancia^2
    return dist

def knn_predict(x_train: pd.DataFrame, y_train: pd.DataFrame, x_test: pd.
 ↪DataFrame, classes, k=1):
```

```
    y_predict = []

    # Convertendo para numpy para otimizar
    x_train = x_train.to_numpy()
    x_test = x_test.to_numpy()

    for i, pt in enumerate(x_test):
        # Calcula distância com todos os elementos
        dist_vet = distancia_pw2(pt, x_train)
        if dist_vet is None:
            print("None distance dataframe")
            return

        # Ordena (pega somente os indices)
        order_up_indexes = np.argsort(dist_vet)
        # Pega os indices dos menores
        nearest_classes = y_train.iloc[order_up_indexes[0:k],0]

        # Levanta a frequencia de cada classe
        class_count = np.zeros(classes.shape[0], dtype=np.uint64)
        for classe in nearest_classes:
            class_elements = np.where(classes == classe)
            class_elements = np.squeeze(np.array(class_elements))
            class_count[class_elements] = class_count[class_elements] + 1
        y_predict.append(classes[np.argmax(class_count)])

    return np.array(y_predict)

def knn_accuracy(y_predict: np.ndarray, y_gnd_truth: np.ndarray):
    NC = np.count_nonzero(y_predict == y_gnd_truth)
    return NC/y_gnd_truth.shape[0]

def knn_recall(y_predict: np.ndarray, y_gnd_truth: np.ndarray):
    VP = np.count_nonzero(np.logical_and(y_predict == 1, y_predict ==␣
 ↪y_gnd_truth))
    FN = np.count_nonzero(np.logical_and(y_predict == 0, y_predict !=␣
 ↪y_gnd_truth))
    if VP + FN == 0:
        return 0
    return VP / (VP + FN)

def knn_precision(y_predict: np.ndarray, y_gnd_truth: np.ndarray):
    VP = np.count_nonzero(np.logical_and(y_predict == 1, y_predict ==␣
 ↪y_gnd_truth))
    FP = np.count_nonzero(np.logical_and(y_predict == 1, y_predict !=␣
 ↪y_gnd_truth))
    if VP + FP == 0:
```

```
        return 0
    return VP / (VP + FP)
```

Usando o modelo de NN e calculando sua acurácia, temos:

[ ]:
```
# Separando os sets - hold out
data_3_train = data_3_atribute.iloc[0:480, :]
data_3_train_class = data_3_target.iloc[0:480, :]
# print(data_3_train)
data_3_test = data_3_atribute.iloc[480:-1, :].reset_index(drop=True)
data_3_test_class = data_3_target.iloc[480:-1, :].reset_index(drop=True)
# print(data_3_test)

y_predict = knn_predict(data_3_train, data_3_train_class, data_3_test, classes)
print(f'Previsões: \n{y_predict}')
y_knn_acc = knn_accuracy(y_predict, data_3_test_class[data_3_test_class.
 ↪columns[0]].to_numpy())
print(f'A acurácia do NN foi de {y_knn_acc*100:.2f} %')
```

```
Previsões:
[1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2
 2 4 5 1 2 3 3 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 5 4 5 1 2 3 4 5 1 2 3 4
 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1
 2 3 4 5 1 2 3 4]
A acurácia do NN foi de 97.48 %
```

Analisando o valor alto de acurácia acima (97,48%), temos um indicativo que as distâncias entre as amostras é bem definida, possívelmente com as amostras afastadas entre amostras de cada classe. Considerando isso e os gráficos dos blocos anteriores, o gráfico que representa melhor as distâncias seriam os plotados na C), ou seja, o do TSNE.

Questão 4: Base de dados Breast Cancer Wisconsin (Diagnostic) Primeiro importando os dados

[ ]:
```
# fetch dataset
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)

# data (as pandas dataframes)
data_4 = breast_cancer_wisconsin_diagnostic
data_4_atribute = breast_cancer_wisconsin_diagnostic.data.features
data_4_target = breast_cancer_wisconsin_diagnostic.data.targets

# metadata
print(breast_cancer_wisconsin_diagnostic.metadata)

# variable information
# print(breast_cancer_wisconsin_diagnostic.variables)
```

```
# print(f'A lista de colunas é:\n{list(data_4_atribute.columns)}')
print(data_4_atribute.head())
```

```
{'uci_id': 17, 'name': 'Breast Cancer Wisconsin (Diagnostic)', 'repository_url':
'https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic',
'data_url': 'https://archive.ics.uci.edu/static/public/17/data.csv', 'abstract':
'Diagnostic Wisconsin Breast Cancer Database.', 'area': 'Health and Medicine',
'tasks': ['Classification'], 'characteristics': ['Multivariate'],
'num_instances': 569, 'num_features': 30, 'feature_types': ['Real'],
'demographics': [], 'target_col': ['Diagnosis'], 'index_col': ['ID'],
'has_missing_values': 'no', 'missing_values_symbol': None,
'year_of_dataset_creation': 1993, 'last_updated': 'Fri Nov 03 2023',
'dataset_doi': '10.24432/C5DW2B', 'creators': ['William Wolberg', 'Olvi
Mangasarian', 'Nick Street', 'W. Street'], 'intro_paper': {'title': 'Nuclear
feature extraction for breast tumor diagnosis', 'authors': 'W. Street, W.
Wolberg, O. Mangasarian', 'published_in': 'Electronic imaging', 'year': 1993,
'url': 'https://www.semanticscholar.org/paper/53f0fbb425bc14468eb3bf96b2e1d41ba8
087f36', 'doi': '10.1117/12.148698'}, 'additional_info': {'summary': 'Features
are computed from a digitized image of a fine needle aspirate (FNA) of a breast
mass.  They describe characteristics of the cell nuclei present in the image. A
few of the images can be found at
http://www.cs.wisc.edu/~street/images/\r\n\r\nSeparating plane described above
was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision
Tree Construction Via Linear Programming." Proceedings of the 4th Midwest
Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a
classification method which uses linear programming to construct a decision
tree.  Relevant features were selected using an exhaustive search in the space
of 1-4 features and 1-3 separating planes.\r\n\r\nThe actual linear program used
to obtain the separating plane in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination
of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992,
23-34].\r\n\r\nThis database is also available through the UW CS ftp
server:\r\nftp ftp.cs.wisc.edu\r\ncd math-prog/cpo-dataset/machine-learn/WDBC/',
'purpose': None, 'funded_by': None, 'instances_represent': None,
'recommended_data_splits': None, 'sensitive_data': None,
'preprocessing_description': None, 'variable_info': '1) ID number\r\n2)
Diagnosis (M = malignant, B = benign)\r\n3-32)\r\n\r\nTen real-valued features
are computed for each cell nucleus:\r\n\r\n\ta) radius (mean of distances from
center to points on the perimeter)\r\n\tb) texture (standard deviation of gray-
scale values)\r\n\tc) perimeter\r\n\td) area\r\n\te) smoothness (local variation
in radius lengths)\r\n\tf) compactness (perimeter^2 / area - 1.0)\r\n\tg)
concavity (severity of concave portions of the contour)\r\n\th) concave points
(number of concave portions of the contour)\r\n\ti) symmetry \r\n\tj) fractal
dimension ("coastline approximation" - 1)', 'citation': None}}
A lista de colunas é:
['radius1', 'texture1', 'perimeter1', 'area1', 'smoothness1', 'compactness1',
'concavity1', 'concave_points1', 'symmetry1', 'fractal_dimension1', 'radius2',
'texture2', 'perimeter2', 'area2', 'smoothness2', 'compactness2', 'concavity2',
```

```
'concave_points2', 'symmetry2', 'fractal_dimension2', 'radius3', 'texture3',
'perimeter3', 'area3', 'smoothness3', 'compactness3', 'concavity3',
'concave_points3', 'symmetry3', 'fractal_dimension3']
   radius1  texture1  perimeter1   area1  smoothness1  compactness1  \
0    17.99     10.38      122.80  1001.0      0.11840       0.27760
1    20.57     17.77      132.90  1326.0      0.08474       0.07864
2    19.69     21.25      130.00  1203.0      0.10960       0.15990
3    11.42     20.38       77.58   386.1      0.14250       0.28390
4    20.29     14.34      135.10  1297.0      0.10030       0.13280

   concavity1  concave_points1  symmetry1  fractal_dimension1  …  radius3  \
0      0.3001          0.14710     0.2419             0.07871  …    25.38
1      0.0869          0.07017     0.1812             0.05667  …    24.99
2      0.1974          0.12790     0.2069             0.05999  …    23.57
3      0.2414          0.10520     0.2597             0.09744  …    14.91
4      0.1980          0.10430     0.1809             0.05883  …    22.54

   texture3  perimeter3   area3  smoothness3  compactness3  concavity3  \
0     17.33      184.60  2019.0       0.1622        0.6656      0.7119
1     23.41      158.80  1956.0       0.1238        0.1866      0.2416
2     25.53      152.50  1709.0       0.1444        0.4245      0.4504
3     26.50       98.87   567.7       0.2098        0.8663      0.6869
4     16.67      152.20  1575.0       0.1374        0.2050      0.4000

   concave_points3  symmetry3  fractal_dimension3
0           0.2654     0.4601             0.11890
1           0.1860     0.2750             0.08902
2           0.2430     0.3613             0.08758
3           0.2575     0.6638             0.17300
4           0.1625     0.2364             0.07678

[5 rows x 30 columns]
```

Como preparação dos dados, vemos que o banco de dados não possui valores faltando. Assim também, ja se separou removendo a classe da tabela de atributos, para evitar ser usado como atributo durante o treino dos modelos. Como o import foi feito pelo framework ucirepo, também ja foi removido qualquer identificador de index na lista, mas se esse estivesse presente como coluna, seria necessário remover.

Além disso, observando os valores (todos númericos), vemos que a escala entre eles é diferente: enquanto há atributos variando entre 0 e 1, existem outros que são >900. A normalização e centralização, feita pelo PCA na questão B, provavelmente se mostrará útil em melhorar a classificação.

A) Aplicando no knn

[ ]:
```
# Separando os sets - hold out para as 300 primeiras para treino e o restante␣
↪para teste
data_4_train = data_4_atribute.iloc[0:300, :]
data_4_train_class = data_4_target.iloc[0:300, :]
```

```
# print(data_4_train)
data_4_test = data_4_atribute.iloc[300:-1, :].reset_index(drop=True)
data_4_test_class = data_4_target.iloc[300:-1, :].reset_index(drop=True)
# print(data_4_test)
data_4_classes = data_4_target[data_4_target.columns[0]].unique()

y_predict = knn_predict(data_4_train, data_4_train_class, data_4_test,␣
 ↪data_4_classes)
print(f'Previsões: \n{y_predict}')
y_knn_acc = knn_accuracy(y_predict, data_4_test_class[data_4_test_class.
 ↪columns[0]].to_numpy())
print(f'A acurácia do NN foi de {y_knn_acc*100:.2f} %')
```

```
Previsões:
['M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'M'
 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'M'
 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M'
 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B'
 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M'
 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B'
 'M' 'M' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B'
 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'M']
A acurácia do NN foi de 88.43 %
```

B) Aplicando agora o PCA sobre os dados, considerando 90% da informação

[ ]:
```
#Usando matriz de covariância
data_4_eigValues, data_4_eigVectors = np.linalg.eig(data_4_atribute.corr())

#Subtrair dos dados a média e normalizar
data_4_normalized = (data_4_atribute - data_4_atribute.mean())/data_4_atribute.
 ↪std()

#Seleciona os dois autovetores associados aos maiores autovalores
#M será a matriz de projeção dos dados
data_4_eigVectors = np.squeeze(data_4_eigVectors)

M = data_4_eigVectors[:,np.where(data_4_eigValues > 1)]
M = np.squeeze(M)

#Projeta os dados nos autovetores principais
```

```
data_4_proj_data = data_4_normalized @ M

print(data_4_proj_data)

print(f'Foram selecionados {M.shape[1]} de {data_4_eigVectors.shape[1]}␣
  ↪atributos usando o critério de fisher.')
```

```
             0          1         2         3         4         5
0     9.184755   1.946870 -1.122179  3.630536  1.194059  1.410184
1     2.385703  -3.764859 -0.528827  1.117281 -0.621228  0.028631
2     5.728855  -1.074229 -0.551263  0.911281  0.176930  0.540976
3     7.116691  10.266556 -3.229948  0.152413  2.958275  3.050737
4     3.931842  -1.946359  1.388545  2.938054 -0.546267 -1.225416
..         …         …         …         …         …         …
564   6.433655  -3.573673  2.457324  1.176279  0.074759 -2.373105
565   3.790048  -3.580897  2.086640 -2.503825  0.510274 -0.246493
566   1.255075  -1.900624  0.562236 -2.087390 -1.808400 -0.533977
567  10.365673   1.670540 -1.875379 -2.353960  0.033712  0.567437
568  -5.470430  -0.670047  1.489133 -2.297136  0.184541  1.616415

[569 rows x 6 columns]
Foram selecionados 6 de 30 atributos usando o critério de fisher.
```

Aplicando NN sobre os dados do PCA

[ ]:
```
# Separando os sets - hold out
data_4_train_2 = data_4_proj_data.iloc[0:300, :]
data_4_train_2_class = data_4_target.iloc[0:300, :]
# print(data_4_train_2)
data_4_test_2 = data_4_proj_data.iloc[300:-1, :].reset_index(drop=True)
data_4_test_2_class = data_4_target.iloc[300:-1, :].reset_index(drop=True)
# print(data_4_test_2)
data_4_classes = data_4_target[data_4_target.columns[0]].unique()

y_predict = knn_predict(data_4_train_2, data_4_train_2_class, data_4_test_2,␣
  ↪data_4_classes)
print(f'Previsões: \n{y_predict}')
y_knn_acc = knn_accuracy(y_predict, data_4_test_2_class[data_4_test_2_class.
  ↪columns[0]].to_numpy())
print(f'A acurácia do NN foi de {y_knn_acc*100:.2f}%')
```

```
Previsões:
['M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'M'
 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'B'
 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
```

```
 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B'
 'M' 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B'
 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'M']
A acurácia do NN foi de 94.03%
```

Acima vemos que após aplicar o PCA, o modelo de NN apresentou um resultado bem melhor em comparação com o desempenho do NN sem nenhum processamento dos dados, sendo que ainda foi selecionando somente parte da informação dos atributos.

A ganha no desempenho provavelmente está relacionada as transformações que o PCA tem sobre os dados: de normalizar e centralizar.

C) Aplicando agora o descriminante de fisher para duas classes (M, B)

[ ]:
```
# Primeiro separando entre as classes, para achar S
data_4_atribute_class_1 = data_4_atribute.loc[data_4_target[data_4_target.
 ↪columns[0]] == data_4_target[data_4_target.columns[0]].unique()[0]]
data_4_atribute_class_2 = data_4_atribute.loc[data_4_target[data_4_target.
 ↪columns[0]] == data_4_target[data_4_target.columns[0]].unique()[1]]

# Calculando S1 e S2 e, enfim, a matriz de projeção, Sw
S1 = (data_4_atribute_class_1.shape[0] - 1)*data_4_atribute_class_1.cov()
S2 = (data_4_atribute_class_2.shape[0] - 1)*data_4_atribute_class_2.cov()
# print(S1)

Sw = S1 + S2
Sw_proj = np.linalg.inv(Sw)
# print(Sw_proj)

# Selecionando n-1 atributos projetados
V = Sw_proj @ (data_4_atribute_class_1.mean() - data_4_atribute_class_2.mean())
# print(V)

data_4_atribute_fisher_proj = data_4_atribute @ V
# Convertendo tipo de pandas Series para pandas DataFrame
data_4_atribute_fisher_proj = data_4_atribute_fisher_proj.to_frame()

print(f'Formato de saída dos dados: {data_4_atribute_fisher_proj.shape}')
print(type(data_4_atribute_fisher_proj))
```

```
Formato de saída dos dados: (569, 1)
<class 'pandas.core.frame.DataFrame'>
```

Consideramos que o fisher foi corretamente aplicado uma vez que reduziu a dimensionalidade dos dados para 1 dimensão (569 amostras x 1 dimensão), por meio de uma projeção.

Aplicando o NN sobre os dados projetados

[ ]:
```
# Separando os sets - hold out
data_4_train_3 = data_4_atribute_fisher_proj.iloc[0:300]
data_4_train_3_class = data_4_target.iloc[0:300, :]
# print(data_4_train_3)
data_4_test_3 = data_4_atribute_fisher_proj.iloc[300:-1].reset_index(drop=True)
data_4_test_3_class = data_4_target.iloc[300:-1, :].reset_index(drop=True)
# print(data_4_test_3)
data_4_classes = data_4_target[data_4_target.columns[0]].unique()

y_predict = knn_predict(data_4_train_3, data_4_train_3_class, data_4_test_3,␣
 ↪data_4_classes)
print(f'Previsões: \n{y_predict}')
y_knn_acc = knn_accuracy(y_predict, data_4_test_3_class[data_4_test_3_class.
 ↪columns[0]].to_numpy())
print(f'A acurácia do NN foi de {y_knn_acc*100:.2f}%')
```

```
Previsões:
['M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'M'
 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'B'
 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B'
 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B'
 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B'
 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M']
A acurácia do NN foi de 96.27%
```

A acurácia do classificador NN sobre os dados aplicado o discriminante do fisher foi maior, como visto acima.

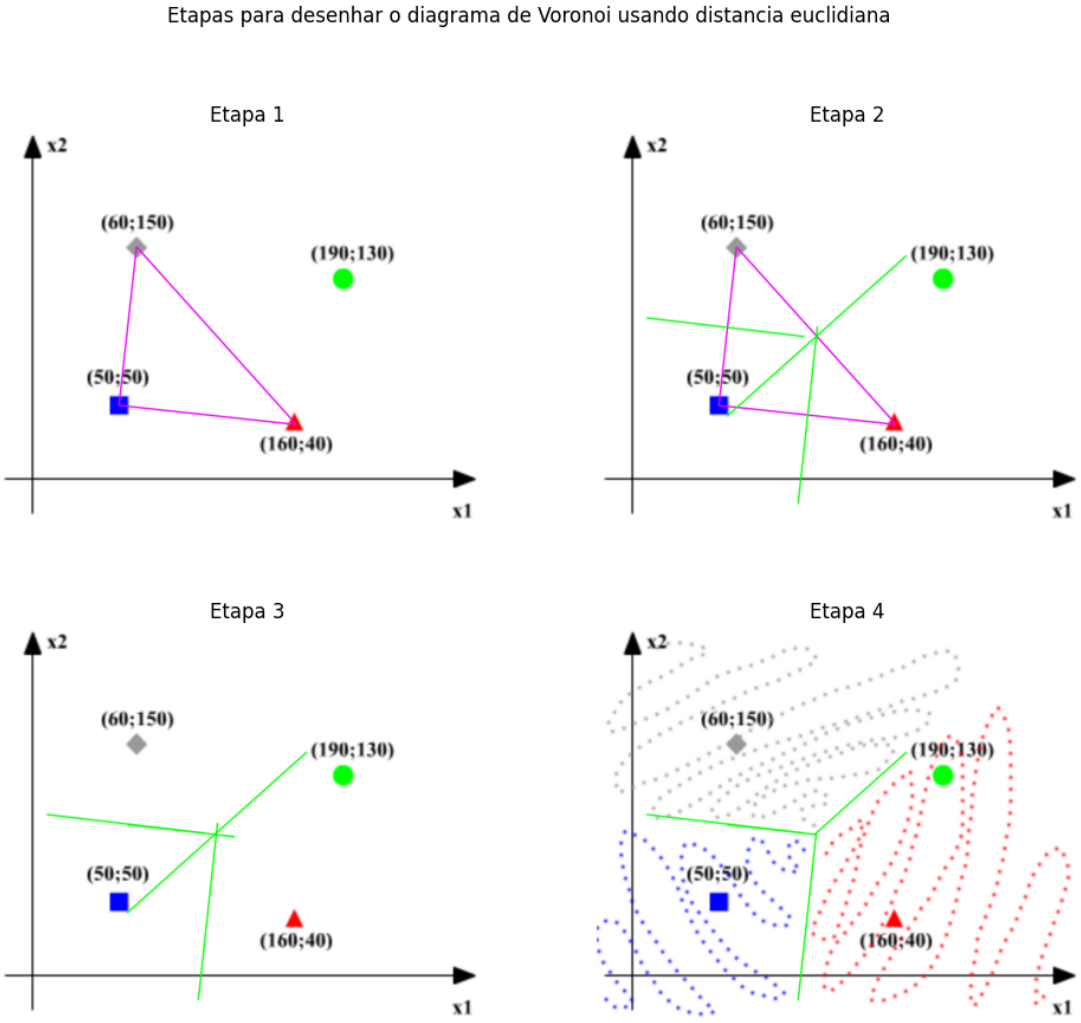
A melhora no desempenho usando o discriminante de fisher é um indicativo de que os atributos nesse banco de dados tem distribuições lineares bem definidas, com variancia uniforme e com coeficientes lineares constantes e distintos entre classes, uma vez que o fisher se baseia em projeção sobre uma reta, tentando maximar a distância entre os centroides projetados de cada classe.

O bom desempenho do PCA também é um indicativo de os dados possuem distribuição linear bem definida, isto é, dados com variancia uniforme, mas não necessariamente com coeficientes lineares constantes uma vez que o PCA calcula a distância dos dados sobre uma componente linear entre os atributos.

Questão 7) Diagrama de Voronoi

A) Distância euclidiana

[ ]:
```
etapa_1 = plt.imread('data/questao_5/step1_euclidiana_2.png')
etapa_2 = plt.imread('data/questao_5/step2_euclidiana_2.png')
etapa_3 = plt.imread('data/questao_5/step3_euclidiana_2.png')
etapa_4 = plt.imread('data/questao_5/step4_euclidiana_2.png')

fig2, axs2 = plt.subplots(2,2, figsize=(12,10))
plt.suptitle("Etapas para desenhar o diagrama de Voronoi usando distancia␣
 ↪euclidiana")
titles = ["Etapa 1", "Etapa 2", "Etapa 3", "Etapa 4"]
figures = [etapa_1, etapa_2, etapa_3, etapa_4]
i = 0
j = 0
for figure, title in zip(figures, titles):
    # print(f'{j},{i%2}')
    axs2[j, i%2].imshow(figure)
    axs2[j, i%2].axis('off')
    axs2[j, i%2].set_title(title)
    if i >= 1:
        j = 1
    i = i + 1
```

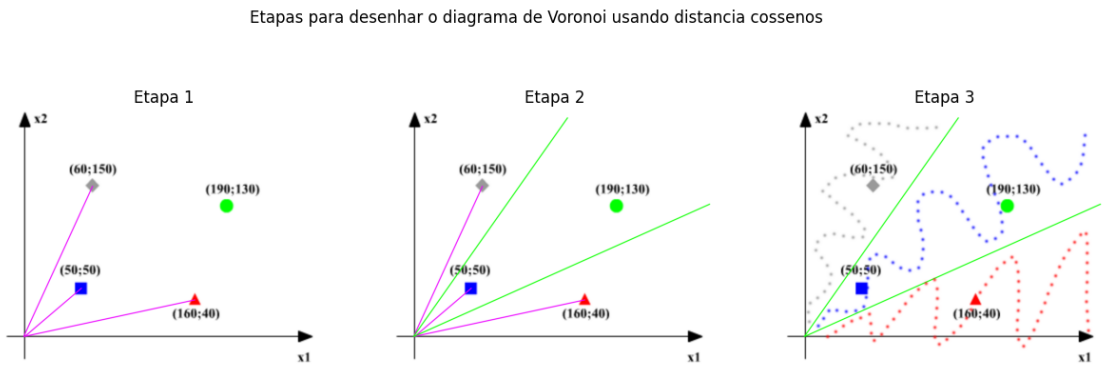


Para o diagrama de Voronoi usando distância euclidiana basta fazer: - Traçar a linha equidistante entre as amostras mais próximas: - Traçar uma linha entre as amostras (etapa 1) - Traçar uma mediatriz nesta linha, a mediatriz é a linha equidistante (etapa 2) - Extender a reta até um pouco depois do cruzamento com outra e apagar as linhas entre amostras (etapa 3) - Analisar e resolvemos os encontros das linhas equidistantes, terminando uma no encontro da outra (etapa 4) - Resolvendo cada encontro, fechamos o poligono (ou as faces) que definem as zonas no diagrama de Voronoi (etapa 4)

B) Usando distância de cossenos

[ ]:
```
etapa_1_cos = plt.imread('data/questao_5/step1_cosseno_2.png')
etapa_2_cos = plt.imread('data/questao_5/step2_cosseno_2.png')
etapa_3_cos = plt.imread('data/questao_5/step3_cosseno_2.png')
# etapa_4_cos = plt.imread('data/questao_5/step4_cosseno.png')

fig3, axs3 = plt.subplots(1,3, figsize=(15,5))
```

```
plt.suptitle("Etapas para desenhar o diagrama de Voronoi usando distancia␣
 ↪cossenos")
titles_cos = ["Etapa 1", "Etapa 2", "Etapa 3"]
figures_cos = [etapa_1_cos, etapa_2_cos, etapa_3_cos]
i = 0
j = 0
for figure, title in zip(figures_cos, titles_cos):
    # print(f'{j},{i%2}')
    axs3[i].imshow(figure)
    axs3[i].axis('off')
    axs3[i].set_title(title)
    if i >= 1:
        j = 1
    i = i + 1
```



Similarmente, usando distância de cossenos basta fazer: - Traçar a bissetriz entre as amostras mais próximas: - Traçar uma linha entre a origem e a amostra (etapa 1) - Traçar a bissetriz dos angulos encontrados entre as amostras (etapa 2) - Analisar as bissetrizes, apagar as linhas entre origem e amostra, definir as zonas (etapa 3)

C) Classificando o circulo usando NN com distância euclidiana e de cossenos

Com o diagrama de Voronoi desenhado acima, podemos facilmente usar as figuras acima para classificar a amostra circulo. Para distância euclidiana, o circulo seria atribuido a classe triângulo Para distância de cossenos, o circulo seria atribuido a classe quadrado

Questão 8) Primeiro, importando os dados

[ ]:
```
data_5 = pd.read_csv('data/htru2/HTRU_2.csv', header = None)
print(data_5.head()) # mostra o começo do dataset

data_5_atribute = data_5[data_5.columns[:]]
data_5_atribute = data_5_atribute.drop(columns=data_5_atribute.columns[-1])
data_5_target = data_5[data_5.columns[-1]].to_frame()
```

```
data_5_classes = data_5_target[data_5_target.columns[0]].unique()
```

```
            0          1         2         3         4          5          6  \
0  140.562500  55.683782 -0.234571 -0.699648  3.199833  19.110426   7.975532
1  102.507812  58.882430  0.465318 -0.515088  1.677258  14.860146  10.576487
2  103.015625  39.341649  0.323328  1.051164  3.121237  21.744669   7.735822
3  136.750000  57.178449 -0.068415 -0.636238  3.642977  20.959280   6.896499
4   88.726562  40.672225  0.600866  1.123492  1.178930  11.468720  14.269573

            7  8
0   74.242225  0
1  127.393580  0
2   63.171909  0
3   53.593661  0
4  252.567306  0
```

Letra A) Usando distância Mahalanobis Primeiro definido a função de knn usando agora a distância de Mahalanobis

[ ]:
```
def distancia_Mahalanobis_pw2(pt1, pt2, cov_matrix):
    # Parameter assertion
    if pt1 is None or pt2 is None or cov_matrix is None:
        print("Empty input parameter for distance", pt1, pt2, type(cov_matrix))
        return None

    diff = pt1 - pt2 # calcula (a-b)
    dist = np.transpose(diff) @ cov_matrix @ diff
    return dist

def rocchio_predict_Mahalanobis(x_train: pd.DataFrame, y_train: pd.DataFrame,␣
 ↪x_test: pd.DataFrame, classes):
    # Calculo dos centroidas
    centroids = []
    for classe in classes:
        attributes_class = x_train.loc[y_train[y_train.columns[0]] == classe]
        attributes_class = attributes_class.reset_index(drop=True)
        centroid = attributes_class.mean(axis=0).to_numpy()
        centroids.append(centroid)

    # Preparação da matriz covariancia inversa transposta
    cov_matrix_t = np.linalg.inv(x_train.cov()).transpose()
    y_predict = []
    for i, pta in x_test.iterrows():
        dist_vet = []
        for centroid in centroids:
```

```
            dist_vet.append(distancia_Mahalanobis_pw2(pt1=pta.to_numpy(),␣
↪pt2=centroid, cov_matrix=cov_matrix_t))

        if dist_vet is None:
            print("None distance dataframe")
            return
        for dist in dist_vet:
            if dist == np.nan:
                print('Nan distance')
            break

        # Ordena o vetor de distâncias e pega os índices da ordem
        order_up_indexes = np.argsort(dist_vet)
        # Pega a classe de menor distância
        y_predict.append(classes[order_up_indexes[0]])
    return np.array(y_predict)
```

Aplicando o modelo definido de rocchio

[ ]:
```
rocchio_acc_score = []
rocchio_recall_score = []
rocchio_precision_score = []
rocchio_y_predict = []
roccio_tempos = []
roccio_tempos.append(time.time())
for i in range(0,5):
    # Dividindo os dados (hold out, 70/30)
    # Train - 70%
    data_5_atribute_train = data_5_atribute.sample(frac=0.7)
    data_5_target_train = data_5_target.iloc[data_5_atribute_train.index]
    # Test - 30%
    data_5_atribute_test = data_5_atribute.drop(data_5_atribute_train.index)
    data_5_target_test = data_5_target.iloc[data_5_atribute_test.index]

    # Resetando indexes para começar em 0
    data_5_atribute_train = data_5_atribute_train.reset_index(drop=True)
    data_5_target_train = data_5_target_train.reset_index(drop=True)
    data_5_atribute_test = data_5_atribute_test.reset_index(drop=True)
    data_5_target_test = data_5_target_test.reset_index(drop=True)

    # Aplicando modelo classificação
    rocchio_y_predict.append(rocchio_predict_Mahalanobis(data_5_atribute_train,␣
↪data_5_target_train, data_5_atribute_test, data_5_classes))

    # Transformando o dataframe de classificação do test para np array
    rocchio_y_gnd_truth = data_5_target_test[data_5_target_test.columns[0]].
↪to_numpy()
```

```
    # Calculando métricas
    rocchio_acc_score.append(knn_accuracy(rocchio_y_predict[-1],␣
 ↪rocchio_y_gnd_truth))
    rocchio_precision_score.append(knn_precision(rocchio_y_predict[-1],␣
 ↪rocchio_y_gnd_truth))
    rocchio_recall_score.append(knn_recall(rocchio_y_predict[-1],␣
 ↪rocchio_y_gnd_truth))
    roccio_tempos.append(time.time())
for acc, precision, recall in zip (rocchio_acc_score, rocchio_precision_score,␣
 ↪rocchio_recall_score):
    print(f'Accuracy:{acc*100:.2f}% Precision:{precision*100:.2f}% Recall:
 ↪{recall*100:.2f}%')

print(f'-------------\nMédias Acurracy: {np.mean(rocchio_acc_score)*100:.2f}%␣
 ↪Precision: {np.mean(rocchio_precision_score)*100:.2f}% Recall: {np.
 ↪mean(rocchio_recall_score)*100:.2f}%')
print(f'Tempo total = {(roccio_tempos[-1] - roccio_tempos[0]):.2f}s. Tempo␣
 ↪médio: {sum([(y - x) for x,y in zip(roccio_tempos,roccio_tempos[1:])])/
 ↪len([(y - x) for x,y in zip(roccio_tempos,roccio_tempos[1:])]):.2f}s')
# tempos_segundos_temp = [(y - x) for x,y in zip(roccio_tempos,roccio_tempos[1:
 ↪])]
# print(f'Tempo médio: {sum(tempos_segundos_temp)/len(tempos_segundos_temp):.
 ↪2f}s')
```

```
Accuracy:97.86% Precision:93.10% Recall:81.97%
Accuracy:97.78% Precision:93.76% Recall:80.79%
Accuracy:97.67% Precision:92.99% Recall:80.73%
Accuracy:97.99% Precision:93.67% Recall:83.81%
Accuracy:97.80% Precision:92.91% Recall:81.70%
-------------
Médias Acurracy:97.82% Precision:93.28% Recall:81.80%
Tempo total = 3.18s. Tempo médio: 0.64s
```

Como podemos ver, os valores de acurácia, precisão e recall foram altos para o modelo de Rocchio usando distancia de mahalanobis. Assim também, o tempo de execução foi bem baixo, o que é esperado do método de Rocchio, que só precisa comparar a distância do atributo com o centroid de cada classe (que são somente 2, no caso).

Letra B) Knn, hold-out (70/30) com validação usando parte do set de treino (35/35 validação)

[ ]:
```
knn_acc_score = []
knn_recall_score = []
knn_precision_score = []
k_best = []
tempos_best = []
y_predict_best = []
```

```
y_gnd_truth_best = []
tempo_start = time.time()
for i in range(0,5):
    # Dividindo os dados (hold out, 70/30)
    # Train - 70%
    data_5_atribute_train = data_5_atribute.sample(frac=0.7)
    data_5_target_train = data_5_target.iloc[data_5_atribute_train.index]
    # Test - 30%
    data_5_atribute_test = data_5_atribute.drop(data_5_atribute_train.index)
    data_5_target_test = data_5_target.iloc[data_5_atribute_test.index]

    # Resetando indexes para começar em 0
    data_5_atribute_train = data_5_atribute_train.reset_index(drop=True)
    data_5_target_train = data_5_target_train.reset_index(drop=True)
    data_5_atribute_test = data_5_atribute_test.reset_index(drop=True)
    data_5_target_test = data_5_target_test.reset_index(drop=True)

    # Dividindo o set de treinamento para validação para achar o melhor k dado␣
 ↪o conjunto
    # Validação 50%
    data_5_atribute_train_validation = data_5_atribute_train.sample(frac=0.5)
    data_5_target_train_validation = data_5_target_train.
 ↪iloc[data_5_atribute_train_validation.index]

    # Teste #2 50%
    data_5_atribute_train_train = data_5_atribute_train.
 ↪drop(data_5_atribute_train_validation.index)
    data_5_target_train_train = data_5_target.iloc[data_5_atribute_train_train.
 ↪index]

    # Resetando indexes para começar em 0
    data_5_atribute_train_validation = data_5_atribute_train_validation.
 ↪reset_index(drop=True)
    data_5_target_train_validation = data_5_target_train_validation.
 ↪reset_index(drop=True)
    data_5_atribute_train_train = data_5_atribute_train_train.
 ↪reset_index(drop=True)
    data_5_target_train_train = data_5_target_train_train.reset_index(drop=True)

    # Achando o melhor k entre 0 e k_max
    k_max = 20
    y_predict = []
    y_knn_acc = []
    y_grund_truth =␣
 ↪data_5_target_train_validation[data_5_target_train_validation.columns[0]].
 ↪to_numpy()
```

```
    tempos = []
    tempos.append(time.time())
    for j in range(1, k_max+1):
        # Knn predict
        y_predict.append(
            knn_predict_numpy(
                data_5_atribute_train_train,
                data_5_target_train_train,
                data_5_atribute_train_validation,
                data_5_classes,
                k=j
            )
        )

        # Get accurancy
        y_knn_acc.append(
            knn_accuracy(y_predict[j-1], y_grund_truth)
        )
        tempos.append(time.time())
        # print(j)

    tempos_segundos_temp = [int(y - x) for x,y in zip(tempos,tempos[1:])]
    print(f'O algoritmo na sua iteração {i+1} demorou {(tempos[-1] -tempos[0]):.
↪02f}s para achar o melhor k. Iterando de k = 1 até k = {k_max}. Média␣
↪{(sum(tempos_segundos_temp)/len(tempos_segundos_temp)):.02f}s')
    # print(f'Iterando de k = 1 até k = {k_max}. Com os tempos[s]:')
    # print(tempos_segundos_temp)

    # Acha melhor melhor k
    k_best.append(
        1 + y_knn_acc.index(max(y_knn_acc))
        )

    # Achado o melhor k, aplica novamente o Knn com todo o conjunto de treino
    tempos_best.append(time.time())
    y_predict_best.append(
        knn_predict_numpy(
            data_5_atribute_train,
            data_5_target_train,
            data_5_atribute_test,
            data_5_classes,
            k=k_best[-1]
        )
    )

    # Convertendo as labels do conjunto de teste para numpy
```

```
    y_gnd_truth_best.append(data_5_target_test[data_5_target_test.columns[0]].
↪to_numpy())

    # Calculando métricas
    knn_acc_score.append(knn_accuracy(y_predict_best[-1], y_gnd_truth_best[-1]))
    knn_recall_score.append(knn_recall(y_predict_best[-1],␣
↪y_gnd_truth_best[-1]))
    knn_precision_score.
↪append(knn_precision(y_predict_best[-1],y_gnd_truth_best[-1]))

    tempos_best.append(time.time())

    print(f'A acurácia do melhor K ({k_best[-1]}) sobre o conjunto de teste foi␣
↪de {knn_acc_score[-1]*100:.2f}%. Demorando {(tempos_best[-1] -␣
↪tempos_best[-2]):.02f} segundos.')
    # print(f'E demorou {int(tempos_best[-1] - tempos_best[-2])} para rodar o␣
↪algoritmo knn com k = {k_best[-1]}')

print("-----------------------")

print(f'Melhores Ks, suas accuracias, recalls e precisions: ')
for k_value, acc, recall, precision in zip(k_best, knn_acc_score,␣
↪knn_recall_score, knn_precision_score):
    print(f'K = {k_value}, acc = {acc*100:.2f}, recall = {recall*100:.2f},␣
↪precision = {precision*100:.2f}')
print(f'-------------\nMédias Acurracy: {np.mean(knn_acc_score)*100:.2f}%␣
↪Precision: {np.mean(knn_precision_score)*100:.2f}% Recall: {np.
↪mean(knn_recall_score)*100:.2f}%')
print(f'Tempo total = {(time.time() - tempo_start):.2f}s. Tempo médio: {sum([(y␣
↪- x) for x,y in zip(tempos_best,tempos_best[1:])])/len([(y - x) for x,y in␣
↪zip(tempos_best,tempos_best[1:])]):.2f}s')

# print(f'Ao total, levou {int(time.time() - tempo_start)} segundos')
```

```
O algoritmo na sua iteração 1 demorou 222s para achar o melhor k. Iterando de k
= 1 até k = 20. Média 10s
A acurácia do melhor K (11) sobre o conjunto de teste foi de 97.82%. Demorando
16 segundos.
O algoritmo na sua iteração 2 demorou 255s para achar o melhor k. Iterando de k
= 1 até k = 20. Média 12s
A acurácia do melhor K (12) sobre o conjunto de teste foi de 97.17%. Demorando
17 segundos.
O algoritmo na sua iteração 3 demorou 259s para achar o melhor k. Iterando de k
= 1 até k = 20. Média 12s
A acurácia do melhor K (12) sobre o conjunto de teste foi de 97.24%. Demorando
17 segundos.
O algoritmo na sua iteração 4 demorou 269s para achar o melhor k. Iterando de k
```

```
= 1 até k = 20. Média 12s
A acurácia do melhor K (14) sobre o conjunto de teste foi de 97.45%. Demorando
30 segundos.
O algoritmo na sua iteração 5 demorou 287s para achar o melhor k. Iterando de k
= 1 até k = 20. Média 13s
A acurácia do melhor K (16) sobre o conjunto de teste foi de 97.28%. Demorando
17 segundos.
-----------------------
Melhores Ks, suas accuracias, recalls e precisions:
K = 11, acc = 97.82, recall = 81.01, precision = 94.58
K = 12, acc = 97.17, recall = 76.27, precision = 91.48
K = 12, acc = 97.24, recall = 76.79, precision = 92.58
K = 14, acc = 97.45, recall = 76.85, precision = 94.83
K = 16, acc = 97.28, recall = 74.79, precision = 93.78
-------------
Médias Acurracy: 97.39% Precision: 93.45% Recall: 77.14%
Tempo total = 1393.77s. Tempo médio: 130.08s
```

[ ]:
```python
# Descobrindo a proporção entre classes
data_8_class_1 = data_5_target[data_5_target[data_5_target.columns[0]] ==␣
 ↪data_5_classes[0]]
data_8_class_2 = data_5_target[data_5_target[data_5_target.columns[0]] ==␣
 ↪data_5_classes[1]]

print(f'Temos {len(data_8_class_1)}({(len(data_8_class_1)/
 ↪len(data_5_target))*100:.02f}%) elementos da classe 0 e␣
 ↪{len(data_8_class_2)}({(len(data_8_class_2)/len(data_5_target))*100:.02f}%)␣
 ↪da classe 1.')
```

```
Temos 16259(90.84%) elementos da classe 0 e 1639(9.16%) da classe 1.
```

Questão C) Comparação Assim como o modelo de classificação de Rocchio, o desempenho do modelo de KNN foi alto (>95%), porém, este demorou em torno de 10x o tempo que o Rocchio demorou, ~0.8s para o Rocchio e ~11s para o KNN no melhor K. Isso desconsiderando o tempo para encontrar o melhor K dentro do set de treino.

Analisando a distribuição das classes, vemos o motivo pelo qual os modelos possuem um valor alto de accurácia: ~91% do dataset é da classe 0 e ~9% é da classe 1, logo, um classificador no-skill ja seria capaz de ter acurácia de ~91% classificando tudo como classe 0. Porém, ai que métricas como recall e precision são importantes. Pois, este mesmo classificador no-skill teria 0% nessas outras duas métricas, uma vez que nunca fariam uma predição positiva (classe 1).

1.4 Questões teóricas

Questão 2) Explique o dilema entre bias e variância e o seu relacionamento com underfitting e overfitting. Bias e variance estão relacionados a capacidade de um modelo de fazer suposições. Um modelo com alto bias é um modelo que faz tem poucas suposições ou suposições muito simples, tornando um modelo que não é capaz de fazer suposições devidas sobre banco de dados complexos ou de aprender com a dados que possuem variação alta. Já um modelo com alta

variance é um modelo que é muito sensível a variações e a complexidade dos dados. Um modelo com alta variance pode considerar incorretamente ruídos ou uma variação como parte da complexidade de um dado. O modelo "presta atenção demais" as variações de cada atributo.

Underfitting e ovefittings estão relacionados ao aprendizado (regulagem de parâmetros) de um modelo sobre um conjunto de dados. Quando um modelo está overfitted sobre um conjunto de dados, este geralmente possui um ótimo desempenho (alto valor de acurácia) sobre o conjunto de treino, mas tem baixo desempenho no dataset de teste ou em dados que se afastem dos padrões do dataset de treino. O overfitting se relaciona na medida que o algoritmo viciou-se a "prestar atenção" à ruídos e variações no dataset de treino, logo, possui alta variance. Geralmente causado por rodar muitas iterações de treino ou por um conjunto de treino muito pequeno. Underfitting trata-se do oposto, o modelo aprendeu pouco sobre o banco de dados de treino. Geralmente possui um baixo desempenho tanto no conjunto de treino quanto no conjunto de teste. O algoritmo "presta pouca atenção" ao conjunto de dados, associando-se a um alto bias.

Tanto para bias x variance quanto para under x overfitting geralmente se tem um trade-off entre eles: quando se tem muito de um, o outro é menor. É trabalho do engenheiro de aprendizado de máquina (ou inteligência artificial) saber modificar o modelo, o banco de dados, levantar e analisar métricas suficientes para encontrar o ideal para cada aplicação.

Questão 3) Em uma empresa é adotado um método de Aprendizado de Máquina para detectar defeito de fabricação de peças mecânicas, sendo que raramente acontece este tipo de problema na fábrica. Um funcionário anuncia empolgado que o sistema alcançou uma acurácia de 99%, porém seu gerente não achou o resultado tão relevante. Responda:

a) Por que o gerente não ficou empolgado com o resultado achado? Pois, de forma similar ao visto na questão 8, mesmo um modelo no-skill conseguiria alcançar uma acurácia alta classificando tudo como não defeituoso, uma vez que o banco de dados é muito desbalanceado entre as classes (negativo e positivo). ##### b) O que o funcionário poderia fazer para confirmar se o método empregado é adequado para o problema? Para melhor firmar sobre o desempenho do modelo, deve ser usado outras métricas como por exmeplo, a precisão, que vai se relacionar a capacidade do modelo de detectar os casos de defeito e ao mesmo tempo de não classificar incorretamente um defeito. Numa pespectiva de diminuir ao máximo o número de produtos defeituosos que saem da fábrica, pode ser usando a matriz de confusão e analisar diretamente os casos de verdadeiros positivos (VP), tentando garantir que este seja mais próximo de 100%.