**Disciplina: Aprendizado de Máquina - Aprendizagem de Máquina e PCA (Principal Components Analysis) (24EXPG9555C)**  
  
**Atividade - N1:**

* **Importante!**  
  Não foi fornecido o arquivo doação.csv, também não foi possível localizar esse dataset na internet, por conta disso, escrevi um arquivo csv que contém as 10 linhas demonstradas na imagem do enunciado do estudo de caso e repeti essas linhas para ganhar um pouco mais de linhas para trabalhar.  
  Dessa forma, não foi possível chegar no modelo ideal esperado, já que não haviam dados o suficiente para o treino do algoritmo.  
    
  Segue abaixo como ficou o arquivo:  
    
  doação.csv:  
  Recencia,Frequencia,Quantidade,Tempo,Doou

2,50,12500,98,1

0,13,3250,28,1

1,16,4000,35,1

2,20,5000,45,1

1,24,6000,77,0

23,2,500,38,0

21,2,500,52,0

23,3,750,62,0

39,1,250,39,0

72,1,250,72,0

2,50,12500,98,1

0,13,3250,28,1

1,16,4000,35,1

2,20,5000,45,1

1,24,6000,77,0

23,2,500,38,0

21,2,500,52,0

23,3,750,62,0

39,1,250,39,0

72,1,250,72,0

* Fiz algumas alterações no algoritmo mencionado no exercício com o intuito de trabalhar melhor as escalas com a classe StandardScaler da biblioteca sklearn.preprocessing e também utilizei o GridSearchCV para identificar os melhores atributos do modelo para o treinamento.
* Segue abaixo o algoritmo atualizado:  
    
  import pandas as pd

from sklearn.utils import resample

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import ConfusionMatrixDisplay

#Carregar os dados

dados\_crus = pd.read\_csv('doacao.csv', header = 0,

                          na\_values = "?", comment='\t',

                          sep=",", skipinitialspace=True)

dataset = dados\_crus.copy()

#dataset

#Separar em Sim doou e Não doou

df\_SIM = dataset[dataset['Doou']==1]

df\_NAO = dataset[dataset['Doou']==0]

#Olhar se tem os mesmos valores de SIM e NÃO

#print(len(df\_SIM))

#print(len(df\_NAO))

if len(df\_NAO) > len(df\_SIM):

    #Reduzir o numero da NAO para ficar igual ao SIM

    df\_NAO\_RED = resample(df\_NAO,

                          replace = False,

                          n\_samples = len(dataset) - len(df\_SIM),

                          random\_state = 42

                          )

    #len(df\_NAO\_RED)

else:

    #Reduzir o numero da SIM para ficar igual ao NAO

    df\_SIM\_RED = resample(df\_SIM,

                          replace = False,

                          n\_samples = len(df\_NAO),

                          random\_state = 42

                          )

    #len(df\_SIM\_RED)

#Juntar novamente os dados apos a redução

if len(df\_NAO) > len(df\_SIM):

    df\_novo = pd.concat([df\_SIM, df\_NAO\_RED])

else:

    df\_novo = pd.concat([df\_SIM\_RED, df\_NAO])

#len(df\_novo)

#Separa os dados de entrada (X) e dos dados de saída (y)

X = df\_novo.drop('Doou', axis = 1).copy()

y = df\_novo['Doou'].copy()

# Centralizar e padronizar os dados

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Centralizar e padronizar os dados

# Cada coluna deve ter média = 0 e desvio padrão = 1

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, random\_state=42, test\_size=0.10)

#X\_train\_pad = scale(X\_train)

#X\_test\_pad = scale(X\_test)

#print(len(X\_train), len(X\_test))

# Iniciar a construção da SVM com ajuste de hiperparâmetros usando GridSearchCV

param\_grid = {

    'C': [0.1, 1, 10, 100, 1000],

    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],

    'kernel': ['rbf', 'linear', 'poly', 'sigmoid']

}

#Determina os melhores atributos para a tomada de decisão

grid = GridSearchCV(SVC(random\_state=42), param\_grid, refit=True, verbose=2)

grid.fit(X\_train, y\_train)

# Melhor estimador encontrado pelo GridSearchCV

best\_estimator = grid.best\_estimator\_

# Previsões e avaliação do modelo

y\_pred = best\_estimator.predict(X\_test)

#Incia a construção da SVM

#clf\_svm = SVC(random\_state=42)

#clf\_svm.fit(X\_train, y\_train)

#Matriz de confusão para análisar os dados de teste

ConfusionMatrixDisplay.from\_estimator(best\_estimator,

                      X\_test,

                      y\_test,

                      values\_format = 'd',

                      display\_labels=['Sim', 'Nao'])

* O resultado foi uma matriz de confusão com 1 positivo verdadeiro, 0 positivo falso, 0 negativo falso e 1 negativo verdadeiro.
* Para aplicar o treinamento aos dados da proposta do problema, organizei esses dados em um arquivo csv (doação\_atividade.csv) e esse ficou no seguinte formato:  
    
  Recencia,Frequencia,Quantidade,Tempo,Doou

2,2,500,4

18,2,500,23

21,3,750,35

21,3,750,64

* Então complementei o algoritmo com o seguinte trecho:  
  # Dados fornecidos para previsão

novos\_dados = pd.DataFrame({

    'Recencia': [2, 18, 21, 21],

    'Frequencia': [2, 2, 3, 3],

    'Quantidade': [500, 500, 750, 750],

    'Tempo': [4, 23, 35, 64]

})

# Padronizar os novos dados com o mesmo scaler usado anteriormente

novos\_dados\_scaled = scaler.transform(novos\_dados)

# Fazer previsões com o modelo treinado

novas\_previsoes = best\_estimator.predict(novos\_dados\_scaled)

# Exibir as previsões

print("Previsões para os novos dados (1 para Doou e 0 para Não Doou):")

print(novas\_previsoes)

* O resultado foi o seguinte:  
  Previsões para os novos dados (1 para Doou e 0 para Não Doou): [0 0 0 0]