Respostas Segunda prova de Aprendizado de Máquina

- Aluno: Edwin Jahir Rueda Rojas

- code e os dados: https://github.com/ejrueda/MasterUFPA/tree/master/Materias/Aprendizado%20de%20maquina/Tarefas/prova_2 (https://github.com/ejrueda/MasterUFPA/tree/master/Materias/Aprendizado%20de%20maquina/Tarefas/prova_2)

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
from PIL import Image
%matplotlib inline
```

Ponto 1

1) [2.0 pts] Use os dados Breast Cancer Wisconsin (Diagnostic) Data Set do UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)) (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))). Use validação cruzada para avaliar qual dos algoritmos tem maior acurácia nos dados:

- SVM Linear
- SVM RBF
  Decida que tipo de padronização (normalização) dos dados você usará para cada algoritmo (ou nenhuma). justifique.

In [2]:
```
data = pd.read_csv("./data/wdbc.csv", header=None)
print(data.shape)
data = data.drop(0, axis=1)
data.head()
```

```
(569, 32)
```

Out[2]:

	1	2	3	4	5	6	7	8	9	10	...	22	23	24	25	26	27	28
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000

5 rows × 31 columns

In [3]:
```
X = data.iloc[:, 1:]
y = np.array(data.iloc[:,0].values)
```

In [4]:
```
for idx in range(len(y)):
    if y[idx]=="M":
        y[idx]=1
    elif y[idx]=="B":
        y[idx]=0
```

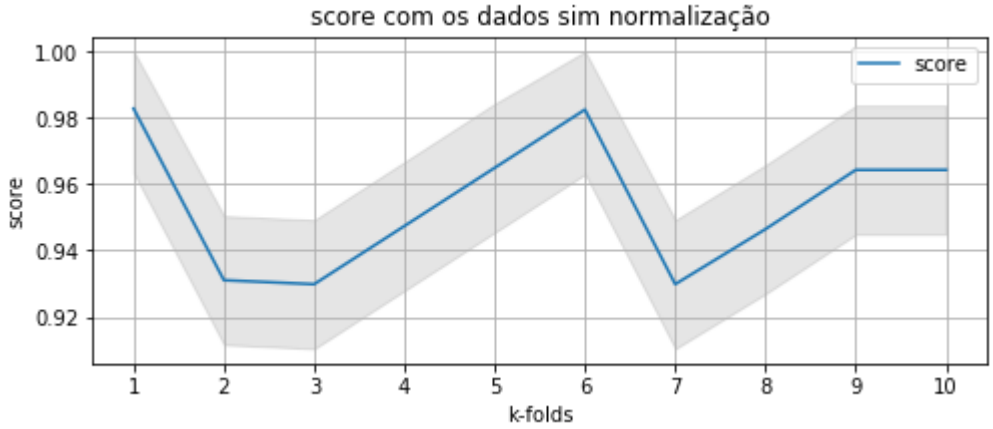
In [5]:
```
y = y.astype(int)
```

In [6]:
```
from sklearn import svm
from sklearn.model_selection import cross_val_score

clf1 = svm.SVC(kernel='linear') #kernel lineal
score1 = cross_val_score(clf1, X, y, n_jobs=2, cv=10)
```

In [7]:
```
plt.figure(figsize=(8,3))
plt.xticks(range(1,11), range(1,11))
plt.grid()
plt.title("score com os dados sim normalização")
plt.xlabel("k-folds")
plt.ylabel("score")
x = range(1,11)
plt.plot(x, score1, label="score", linewidth=1.5)
print("score promedio: ", np.mean(score1))
upper = score1 + np.std(score1)
upper[upper>1]=1
plt.fill_between(x, upper, score1 - np.std(score1), alpha=0.2, color='grey')
plt.legend();
```

```
score promedio:  0.9543179068360554
```



Scalando os dados

In [8]:
```
def data_scale(df, l_cols=-1):
    """
    df: DataFrame de entrada
    l_cols: lista de columnas a escalar, por defecto -1 para escalar todas
    return: retorna un df con las columnas estandarizadas así:
    (X - mean(X))/(max(X)-min(X))
    """
    df_scale = df.copy()
    if l_cols == -1:
        l_cols = df.columns
    for col in l_cols:
        #se cambia cada columna
        #df_scale[col] = (df_scale[col] - min(df_scale[col]))/(max(df_scale[col]) - min(df_scale[
        df_scale[col] = (df_scale[col] - min(df_scale[col]))/((max(df_scale[col]) - min(df_scale[
    return df_scale
```

In [9]:
```
data.columns
```

Out[9]:
```
Int64Index([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
            18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31],
           dtype='int64')
```

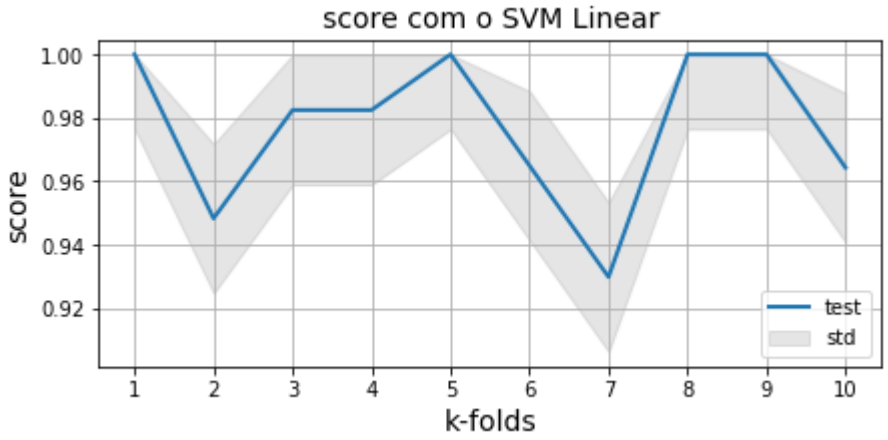
In [10]:
```
#Escalando os dados
X_scale = data_scale(X)
```

In [11]:
```
from sklearn import svm
from sklearn.model_selection import cross_val_score

clf2 = svm.SVC(kernel='linear') #kernel lineal
score2 = cross_val_score(clf2, X_scale, y, n_jobs=2, cv=10)

plt.figure(figsize=(7,3))
plt.xticks(range(1,11), range(1,11))
plt.grid()
plt.title("score com o SVM Linear", size=14)
plt.xlabel("k-folds", size=14)
plt.ylabel("score", size=14)
plt.plot(range(1,11), score2, label="test", linewidth=2)
print("score promedio com o SVM Linear", np.mean(score2))
upper2 = score2 + np.std(score2)
upper2[upper2>1]=1
plt.fill_between(x, upper2, score2 - np.std(score2), alpha=0.2, color='grey', label="std")
plt.legend(loc=4);
```

```
score promedio com o SVM Linear 0.9772210699161695
```



In [12]:
```
from sklearn import svm
from sklearn.model_selection import cross_val_score

clf3 = svm.SVC(kernel='rbf') #kernel lineal
score3 = cross_val_score(clf3, X_scale, y, n_jobs=2, cv=10)

plt.figure(figsize=(7,3))
plt.xticks(range(1,11), range(1,11))
plt.grid()
plt.title("score com o SVM RBF", size=14)
plt.xlabel("k-folds", size=14)
plt.ylabel("score", size=14)
plt.plot(x, score3, label="test", linewidth=2, color="green")
print("score promedio do SVM RBF: ", np.mean(score3))
upper2 = score3 + np.std(score3)
upper2[upper2>1]=1
plt.fill_between(x, upper2, score3 - np.std(score3), alpha=0.2, color='grey', label="std")
plt.legend(loc=4);
```

```
score promedio do SVM RBF:  0.9527180019013048
```

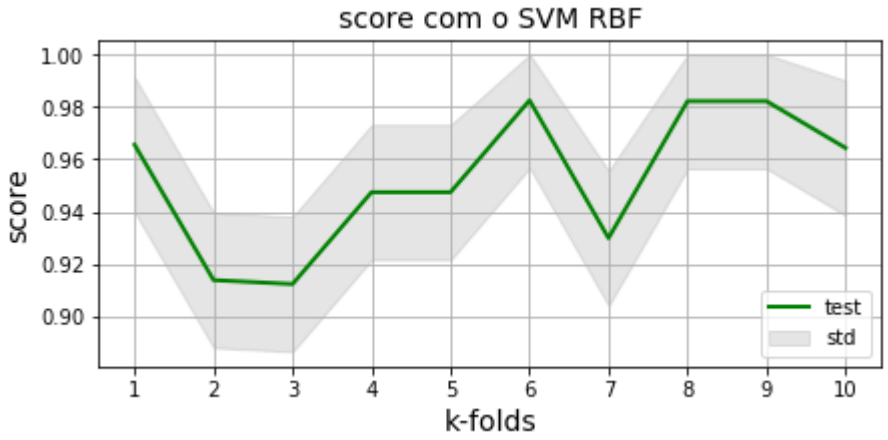
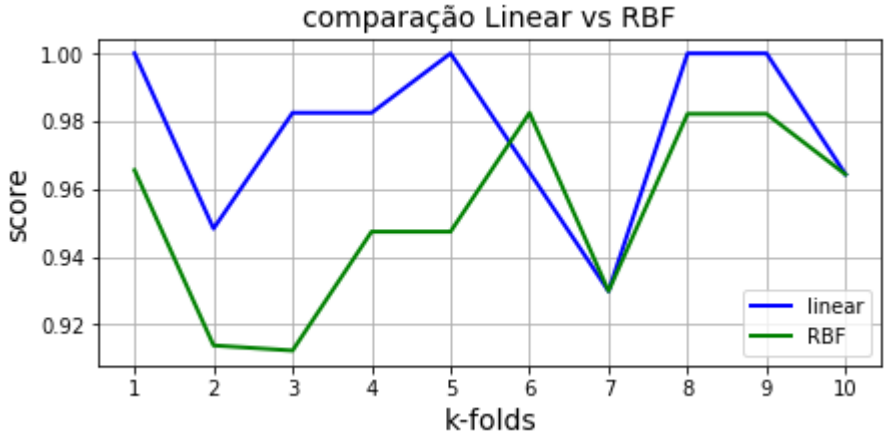


gráfico da relação dos scores

In [13]:
```
plt.figure(figsize=(7,3))
plt.xticks(range(1,11), range(1,11))
plt.grid()
plt.title("comparação Linear vs RBF", size=14)
plt.xlabel("k-folds", size=14)
plt.ylabel("score", size=14)
plt.plot(x, score2,  linewidth=2, color="blue", label="linear")
plt.plot(x, score3,  linewidth=2, color="green", label="RBF")
plt.legend(loc=4);
```



Cloclução

- o modelo SVM com o kernel linear é melhor que o SVM com kernel RBF já que ele é melhor o igual em todos os k-folds, como o gráfico anterior mostra.

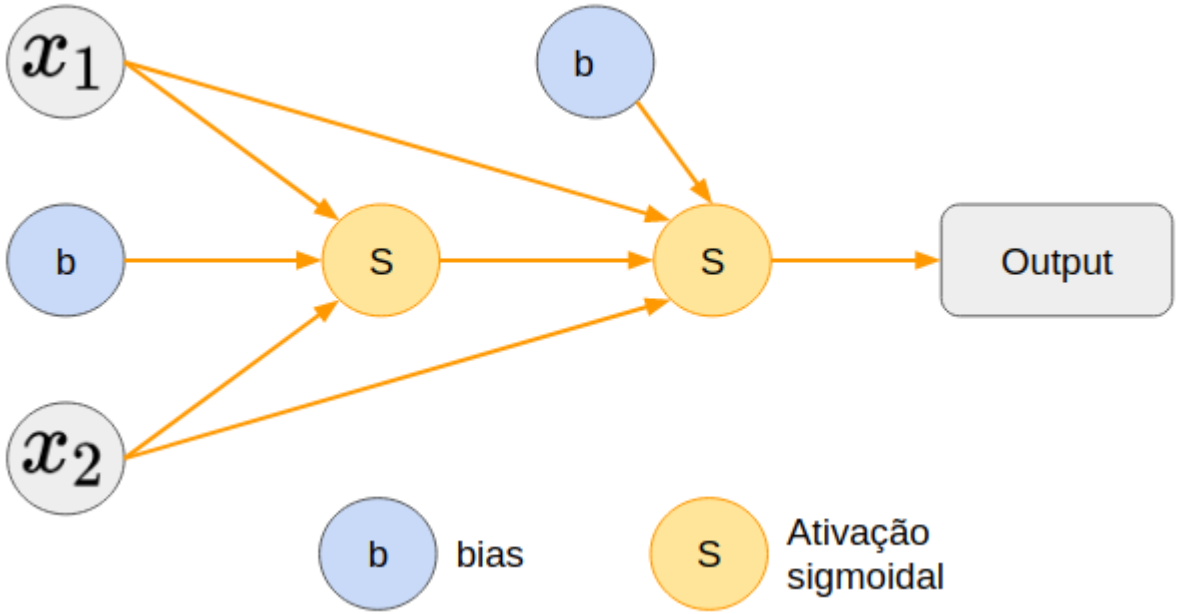
Ponto 2

2) [2.0 pts] Implemente em uma linguagem de programação de sua escolha uma Rede Neural Artificial Multilayer Perceptron treinada com o algoritmo backprogation que resolva o problema do OU-EXCLUSIVO. Para validar sua implementação utilize a arquitetura apresentada na aula sobre Redes Neurais. Mostre que os resultados foram os mesmos. Avalie como ficaria a solução se considerarmos uma arquitetura com 2 neurônios na camada escondida e um neurônio na camada de saída. Lembrando que a função de ativação do neurônio deve ser sigmoide.

In [2]:
```
print("-------------------- Arquitectura da rede 1 --------------------")
Image.open("rede1.png")
```

```
-------------------- Arquitectura da rede 1 --------------------
```

Out[2]:



In [2]:

```
import numpy as np
class RNN_MLP:

    def __init__(self, X, y):
        self.X = X
        self.y = y
        self.lr = 0.4 #learning rate
        #inicialización de los pesos en cero
        self.theta1 = np.zeros((X.shape[1] + 1, 1))
        self.theta2 = np.zeros((X.shape[1] + 2, 1))
        self.count = 0

    def sigmoid(self, z):
        """
        função para calcular a sigmoide
        """
        return (1/(1 + np.exp(-z)))

    def get_weigths(self):
        """
        função para retornar os pesos da rede entrenada
        """
        return (self.theta1, self.theta2)

    def get_count(self):
        """
        retorna o número de iterações que a rede
        precisó para converger
        """
        return self.count

    def predict(self, X):
        """
        função para predecir a saída esperada
        """
        y_p = []
        for idx in range(X.shape[0]):
            x1 = np.concatenate(([1],X[idx]))
            h1 = self.sigmoid(self.theta1.T.dot(x1)) #hipótesis da camada oculta
            #para computar a saída da camada oculta
            x2 = np.concatenate(([1],X[idx],h1))
            h2 = self.sigmoid(self.theta2.T.dot(x2)) #hipótesis final
```

```
43              y_p.append(np.float(h2))
44          y_p = np.array(y_p)
45          y_p[y_p>=0.5] = 1
46          y_p[y_p<0.5] = 0
47
48          return y_p
49
50      def error(self, X, y):
51          """
52          função para o calculo do erro da rede
53          """
54          y_p = self.predict(X)
55          return np.mean((y_p - y)**2)
56
57
58      def train(self,X,y, bias=1):
59          """
60          função para o treinamento da rede
61          X: matriz com os dados de treinamento
62          y: vector objetivo
63          """
64          while (self.error(X,y) != 0):
65              for idx in range(X.shape[0]):
66                  x1 = np.concatenate(([1],X[idx]))
67                  h1 = self.sigmoid(self.theta1.T.dot(x1)) #hipótesis da camada oculta
68                  #para computar a saída da camada oculta
69                  x2 = np.concatenate(([1],X[idx],h1))
70                  h2 = self.sigmoid(self.theta2.T.dot(x2)) #hipótesis final
71                  #------------------------------
72                  #Backpropagation
73                  #------------------------------
74                  erro_saida = (y[idx] - h2)*h2*(1 - h2)
75                  self.theta2 = (self.theta2.T + (self.lr*erro_saida*x2)).T
76                  erro_interno = h1*(1 - h1)*erro_saida*self.theta2[-1]
77                  self.theta1 = (self.theta1.T + (self.lr*erro_interno*x1)).T
78
79              self.count += 1
```

In [3]:
```
X = np.array([[1,0],[0,0],[0,1],[1,1]]) #Entrada XOR
y = np.array([1,0,1,0]) #salida objetivo
obj = RNN_MLP(X, y) #clase para entrenar
obj.train(X,y) #treino do modelo
print("predição do modelo:", obj.predict(X))
print("error do modelo:", obj.error(X, y))
```

```
predição do modelo: [1. 0. 1. 0.]
error do modelo: 0.0
```

In [4]:
```
print("learning rate:", 0.4)
print("número de iterações do modelo:", obj.count)
print("pesos da camada de entrada:")
print(obj.get_weigths()[0])
print("pesos da camada oculta:")
print(obj.get_weigths()[1])
```

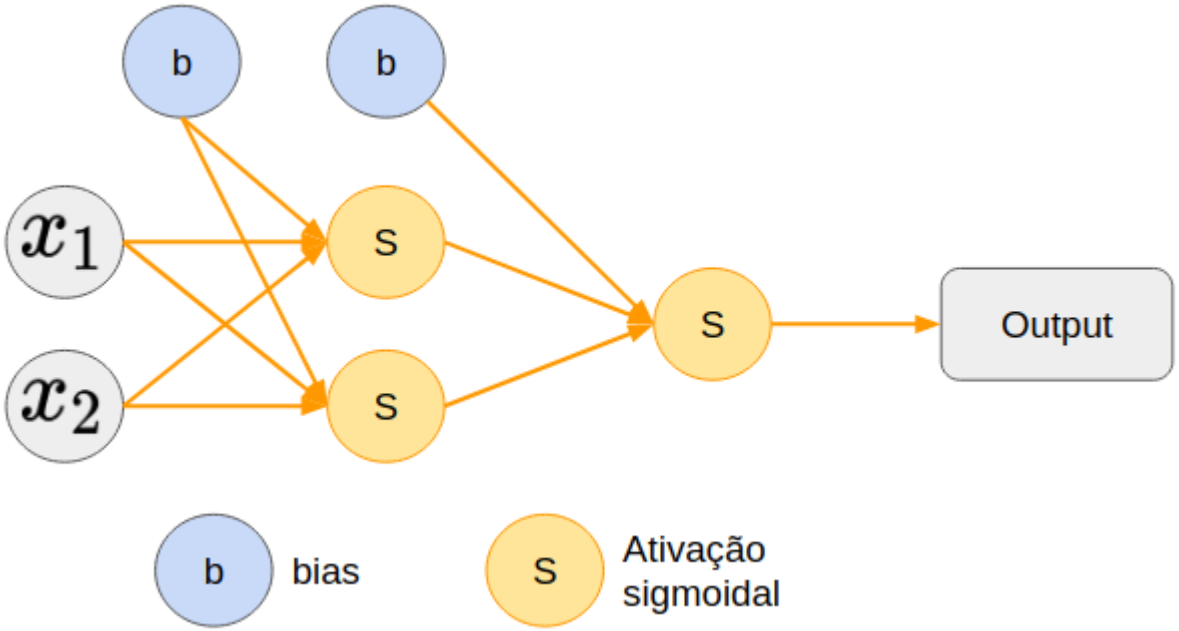
```
learning rate: 0.4
número de iterações do modelo: 1575
pesos da camada de entrada:
[[0.65829301]
 [1.43380589]
 [1.45335442]]
pesos da camada oculta:
[[-0.79544225]
 [-0.25472723]
 [-0.26251389]
 [ 1.18628356]]
```

Rede com dois neurônios na camada escondida

In [5]:
```
print("-------------------- Arquitectura da rede 2 --------------------")
Image.open("rede2.png")
```

```
-------------------- Arquitectura da rede 2 --------------------
```

Out[5]:



In [204]:

```python
import numpy as np
class RNN_MLP_2:

    def __init__(self, X, y, lr=0.1):
        self.X = X
        self.y = y
        self.lr = lr #learning rate
        self.num_nce = 2 #número de neurônios na camada escondida
        self.num_nsa = 1 #número de nerônios na camada de saída
        #inicialización de los pesos en cero
        #número de pesos na camada escondida(como o bias)
        self.theta1 = np.random.random((X.shape[1]+1, self.num_nce))
        #número de pesos na camada de saída(com o bias)
        self.theta2 = np.random.random(((self.num_nce + 1)*self.num_nsa, 1))
        self.count = 0 #para contar as iterações que a rede faz para converger
    def sigmoid(self, z):
        """
        função para calcular a sigmoide
        """
        return (1/(1 + np.exp(-z)))

    def get_weigths(self):
        """
        função para retornar os pesos da rede entrenada
        """
        return (self.theta1, self.theta2)

    def get_count(self):
        """
        retorna o número de iterações que a rede
        precisó para converger
        """
        return self.count

    def predict(self, X):
        """
        função para predecir a saída esperada
        """
        y_p = []
        for idx in range(X.shape[0]):
            #entrada para a camada inicial
            x1 = np.concatenate(([1],X[idx]))
```

```
43              #print(self.theta1.shape, x1.shape)
44              h1 = self.sigmoid(x1.dot(self.theta1)) #hipótesis da camada oculta
45              #para computar a saída da camada oculta
46              x2 = np.concatenate(([1],h1))
47              h2 = self.sigmoid(x2.dot(self.theta2)) #hipótesis final
48              y_p.append(np.float(h2))
49          y_p = np.array(y_p)
50          y_p[y_p>=0.5] = 1
51          y_p[y_p<0.5] = 0
52
53          return y_p
54
55      def error(self, X, y):
56          """
57          função para o calculo do erro da rede
58          """
59          y_p = self.predict(X)
60          return np.mean((y_p - y)**2)
61
62
63      def train(self,X,y, bias=1):
64          """
65          função para o treinamento da rede
66          X: matriz com os dados de treinamento
67          y: vector objetivo
68          """
69          for i in range(40000): #para quando a rede tenha como erro zero
70              for idx in range(X.shape[0]):
71                  #entrada para a camada inicial
72                  x1 = np.concatenate(([1],X[idx]))
73                  #print(self.theta1.shape, x1.shape)
74                  h1 = self.sigmoid(x1.dot(self.theta1)) #hipótesis da camada oculta
75                  #para computar a saída da camada oculta
76                  x2 = np.concatenate(([1],h1))
77                  h2 = self.sigmoid(x2.dot(self.theta2)) #hipótesis final
78                  #---------------------------------
79                  #Backpropagation
80                  #---------------------------------
81                  erro_saida = (y[idx] - h2)*h2*(1 - h2)
82                  self.theta2 = self.theta2 + ((self.lr*erro_saida*x2).T).reshape(3,1)
83                  erro_interno = h1*(1 - h1)*sum(erro_saida*self.theta2)
84                  self.theta1 = self.theta1 + (self.lr*erro_interno.reshape(2,1).dot(x1.reshape(1,3
85              if self.error(X, y) == 0:
```

```
86                 break
87             self.count += 1
```

In [210]:

```
1 X = np.array([[1,0],[0,0],[0,1],[1,1]]) #Entrada XOR
2 y = np.array([1,0,1,0]) #salida objetivo
3 obj = RNN_MLP_2(X, y, lr=0.1) #clase para entrenar
4 obj.train(X,y) #treino do modelo
5 print("predição do modelo:", obj.predict(X))
6 print("error do modelo:", obj.error(X, y))
```

```
predição do modelo: [1. 0. 1. 0.]
error do modelo: 0.0
```

In [211]:

```
1 print("pesos da camada de entrada:")
2 print(obj.get_weigths()[0])
3 print("pesos da camada oculta:")
4 print(obj.get_weigths()[1])
```

```
pesos da camada de entrada:
[[0.38806811 0.25898045]
 [4.9152757  5.03040498]
 [4.91805205 5.0376233 ]]
pesos da camada oculta:
[[-7.57961129]
 [ 2.60026043]
 [ 5.68566434]]
```

Ponto 3

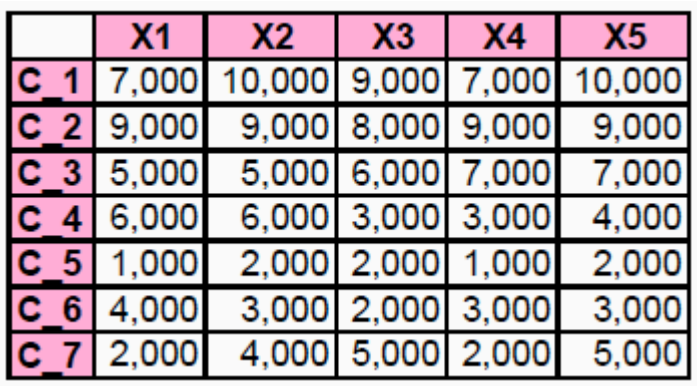
3) [2.0 pts] Dado o conjunto de dados abaixo:

- a) aplique o método de agrupamento aglomerativo utilizando a métrica single-link e o critério de dissimilariade distância Euclidiana.
- b) aplique o algoritmo K-means utilizando distância Euclidiana considerando K = 2. O Algoritmo deve parar caso não apresente convergência após 5 iterações. Considere também que os centros iniciais são: cliente1 e cliente4.
- c) Avalie qual melhor solução de clusterização considerando 3 grupos.

In [7]:
```
print("---------------- Dados ----------------")
Image.open("ponto3.png")
```

```
---------------- Dados ----------------
```

Out[7]:



	X1	X2	X3	X4	X5
C_1	7,000	10,000	9,000	7,000	10,000
C_2	9,000	9,000	8,000	9,000	9,000
C_3	5,000	5,000	6,000	7,000	7,000
C_4	6,000	6,000	3,000	3,000	4,000
C_5	1,000	2,000	2,000	1,000	2,000
C_6	4,000	3,000	2,000	3,000	3,000
C_7	2,000	4,000	5,000	2,000	5,000

a) aplique o método de agrupamento aglomerativo utilizando a métrica single-link e o critério de dissimilariade distância Euclidiana.

In [17]:
```
data = pd.read_csv("./data/data_ponto3.csv", names=["X1","X2","X3","X4","X5"])
data
```

Out[17]:

	X1	X2	X3	X4	X5
0	7	10	9	7	10
1	9	9	8	9	9
2	5	5	6	7	7
3	6	6	3	3	4
4	1	2	2	1	2
5	4	3	2	3	3
6	2	4	5	2	5

In [3]:
```
c_1 = data.iloc[0].values
c_2 = data.iloc[1].values
c_3 = data.iloc[2].values
c_4 = data.iloc[3].values
c_5 = data.iloc[4].values
c_6 = data.iloc[5].values
c_7 = data.iloc[6].values
```

- Para o cálculo da distância Euclidiana

In [31]:
```
np.round(np.sqrt(sum((c_5 - c_6)**2)),4)
```

Out[31]: 3.873

In [5]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
Image.open("Selección_021.png")
```

--------------- Single Linkage (Euclidiana) ---------------

Out[5]:

	C_1	C_2	C_3	C_4	C_5	C_6
C_2	3,3166	-	-	-	-	-
C_3	6,8557	6,6332	-	-	-	-
C_4	10,247	10,198	6	-	-	-
C_5	15,7797	16,1864	10,0995	7,0711	-	-
C_6	13,1149	13	7,2801	3,873	3,873	-
C_7	11,2694	12,1655	6,3246	5,099	4,899	4,3589

$D(C_3, C_1C_2) = \min\{6.8557, 6.6332\} = 6.6332$
$D(C_4, C_1C_2) = \min\{10.247, 10.198\} = 10.198$
$D(C_5, C_1C_2) = \min\{15.7797, 16.1864\} = 16.1864$
$D(C_6, C_1C_2) = \min\{13.1149, 13\} = 13$
$D(C_7, C_1C_2) = \min\{11.2694, 12.1655\} = 11.2694$

In [6]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
Image.open("Selección_022.png")
```

--------------- Single Linkage (Euclidiana) ---------------

Out[6]:

	C_3	C_4	C_5	C_6	C_7
C_4	6	-	-	-	-
C_5	10,0995	7,0711	-	-	-
C_6	7,2801	3,873	3,873	-	-
C_7	6,3246	5,099	4,899	4,3589	-
C_1C_2	6,6332	10,198	16,1864	13	11.2694

$D(C_3, C_4C_6) = \min\{6, 7.2801\} = 6$
$D(C_5, C_4C_6) = \min\{7.0711, 3.873\} = 3.873$
$D(C_7, C_4C_6) = \min\{5.099, 4.3589\} = 4.3589$
$D(C_1C_2, C_4C_6) = \min\{10.198, 13\} = 10.198$

In [7]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
Image.open("Selección_023.png")
```

```
--------------- Single Linkage (Euclidiana) ---------------
```

Out[7]:

	C_3	C_5	C_7	C_1C_2
C_5	10,0995	-	-	-
C_7	6,3246	4,899	-	-
C_1C_2	6,6332	16,1864	11.2694	-
C_4C_6	6	3,873	4,3589	10,198

$D(C_3, C_5\mathbf{C_4C_6}) = \min\{10.0995, 6\} = 6$
$D(C_7, C_5\mathbf{C_4C_6}) = \min\{4.899, 11.2694\} = 4.899$
$D(C_1C_2, C_5\mathbf{C_4C_6}) = \min\{16.1864, 10.198\} = 10.198$

In [8]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
Image.open("Selección_024.png")
```

```
--------------- Single Linkage (Euclidiana) ---------------
```

Out[8]:

	C_3	C_7	C_1C_2
C_7	6,3246	-	-
C_1C_2	6,6332	11.2694	-
$C_5C_4C_6$	6	4,899	10,198

$D(C_3, C_7\mathbf{C_5C_4C_6}) = \min\{6.3246, 6\} = 6$
$D(C_1C_2, C_7\mathbf{C_5C_4C_6}) = \min\{11.2694, 10.198\} = 10.198$

In [9]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
Image.open("Selección_025.png")
```

```
--------------- Single Linkage (Euclidiana) ---------------
```

Out[9]:

	C_3	C_1C_2
C_1C_2	6,6332	-
$C_7C_5C_4C_6$	6	10,198

$D(C_1C_2, C_3\mathbf{C_7C_5C_4C_6}) = \min\{6.6332, 10.198\} = 6.6332$

In [11]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
Image.open("Selección_026.png")
```

```
--------------- Single Linkage (Euclidiana) ---------------
```

Out[11]:

	C_1C_2
$C_3C_7C_5C_4C_6$	6.6332

In [13]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
print("Resulta em:")
Image.open("Selección_027.png")
```

```
--------------- Single Linkage (Euclidiana) ---------------
Resulta em:
```

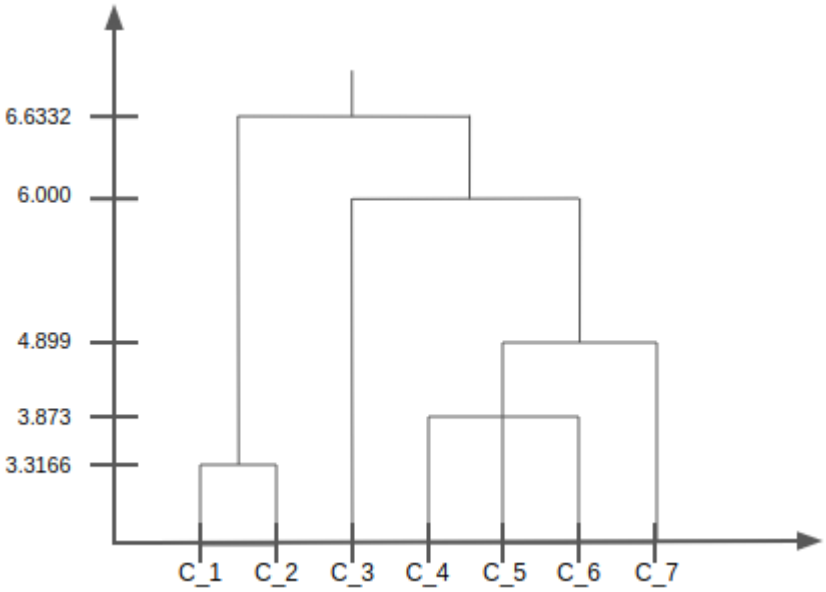
Out[13]:

Nó	Fusão	Nível
1	C_1 e C_2	3,3166
2	C_4 e C_6	3,873
3	C_5 e C_4C_6	3,873
4	C_7 e $C_5C_4C_6$	4,899
5	C_3 e $C_7C_5C_4C_6$	6
6	C_1C_2 e $C_3C_7C_5C_4C_6$	6.6332

In [4]:
```
print("--------------- Single Linkage (Euclidiana) ---------------")
print("Gráfico:")
Image.open("Selección_028.png")
```

```
--------------- Single Linkage (Euclidiana) ---------------
Gráfico:
```

Out[4]:



b) aplique o algoritmo K-means utilizando distância Euclidiana considerando K = 2. O Algoritmo deve parar caso não apresente convergência após 5 iterações. Considere também que os centros iniciais são: cliente1 e cliente4.

In [26]:
```
data
#seleção dos centros inicias
c1 = data.iloc[0].values
c4 = data.iloc[3].values
print("centroide 1: ", c1)
print("centroide 2: ", c4)
```

```
centroide 1:  [ 7 10  9  7 10]
centroide 2:  [6 6 3 3 4]
```

In [56]:
```
from sklearn.cluster import KMeans

num_iter = 5 #o algoritmo só apresenta 5 iterações
num_cen = 2 #numero de centros finais
centroides = np.concatenate((c1,c4)).reshape(2,5) #centros iniciais
kmeans = KMeans(n_clusters=num_cen, init=centroides, max_iter=num_iter, n_init=1).fit(data)
```

In [57]:
```
new_centers = kmeans.cluster_centers_
print("---------- Resultado ----------")
print("centro 1: ", new_centers[0])
print("centro 2: ", new_centers[1])
```

```
---------- Resultado ----------
centro 1:  [8.  9.5 8.5 8.  9.5]
centro 2:  [3.6 4.  3.6 3.2 4.2]
```

c) Avalie qual melhor solução de clusterização considerando 3 grupos.

In [58]:
```
#o algoritmo vai para ao convergir
#os centros inicias são escohlidos da forma random
num_cen2 = 3 #numero de centros finais
kmeans2 = KMeans(n_clusters=num_cen2).fit(data)
```

In [61]:
```
new_centers2 = kmeans2.cluster_centers_
print("---------- Resultado ----------")
print("centro 1: ", new_centers2[0])
print("centro 2: ", new_centers2[1])
print("centro 3: ", new_centers2[2])
```

```
---------- Resultado ----------
centro 1:  [3.25 3.75 3.   2.25 3.5 ]
centro 2:  [8.  9.5 8.5 8.  9.5]
centro 3:  [5. 5. 6. 7. 7.]
```

Ponto 4

4) [2.0 pts] Implemente o método K-means. Os parâmetros de entrada são número K de clusters, o número M máximo de iterações, e um arquivo ARFF com o conjunto de treino (assuma que todos os atributos do ARFF devem ser levados em conta). O critério deparada não precisa ser limitado a usar apenas o valor de M. Faça um tratamento (leve em conta) para o caso de algum cluster ficar com nenhum vetor associado a ele

A classe KNN é feita para resolver o algoritmo k-means, ela tem as seguintes funções

- **Inicialização:** a inicialização recebe os parâmetros k:número de cluster, num_max_iter:número máximo de iterações do algoritmo e um arff: o cual e o arquivo .arff.
- **get_cent_inicias:** retorna os centrois calculados inicialmente pelo algoritmo.
- **get_groups:** retorna um dicionário com os índices dos clusters computados, indices do "toDataFrame()".
- **get_iters:** retorna o número de iterações que o programa fez para convergir.
- **get_centroides:** retorna os centroides computados.
- **toDataFrame:** retorna o DataFrame feito com os dados arff.
- **train:** treina o modelo e retorna os centroides computados.

In [15]:

```
import numpy as np
import pandas as pd
class KNN():
    """
    Classe para gerar k clusters com o algoritmo k-means,
    tendo en conta a distância Euclidian.
    Parâmetros:
    k: número de cluster a gerar
    num_max_iter: número maximo de iterções para a conversão do algoritmo
    arff: arquivo .arff com os dados de treino
    """
    def __init__(self, k, num_max_iter, arff):
        """ inialização das variáveis"""
        self.k = k
        self.num_max_iter = num_max_iter
        self.arff = arff
        self.d_grupos = {}
        self.centroides = {}
        self.centro_inicias = {}
        self.iters = 0

    def toDataFrame(self):
        """
        converte o arquivo arff num DataFrame
        """
        firts = True
        header = []
        flag_data = False
        data = []
        for line in self.arff:
            if flag_data == False:
                if "@attribute" in line.lower():
                        attri = line.lower().split()
                        columnName = attri[attri.index("@attribute")+1]
                        header.append(columnName)
                elif "@data" in line.lower():
                    flag_data = True

            elif flag_data==True:
                l = line.replace("\n","").split(",")
                if "%" in l:
                    continue
```

```
                l = [float(i) for i in l] #converte de string á float
                if firts:
                    data = np.array(l).reshape(1,len(l))
                    #print(data.shape)
                    firts = False
                else:
                    data = np.concatenate((data,np.array(l).reshape(1,len(l))))
                    #break
        df = pd.DataFrame(data=data, columns=header)
        return df

    def dist_euclidean(self, x1, x2):
        """
        para calculo da distância euclidiana
        """
        return np.sqrt(sum((x1 - x2)**2))

    def get_centroides(self):
        """
        para obter os centroides computados pelo k-means,
        retorna um dicionário com os centroides.
        """
        return self.centroides

    def get_groups(self):
        """
        retorna um dicinário com os índices dos dados de cada cluster
        """
        return self.d_grupos

    def get_cent_inicias(self):
        """
        retorna os centroides iniciais computados
        """
        return self.centro_inicias

    def get_iters(self):
        """
        retorna as iterações que fez o algoritmo
        """
        return self.iters

    def train(self):
```

```
        """
        função para treinar o algoritmo k-means
        """
        data_train = self.toDataFrame()
        assert data_train.shape[0]>=self.k, "o valor de k tem que ser menor o igual que o número
        #indices dos clusters
        idx = np.random.choice(data_train.index, size=self.k, replace=False)
        for i in range(self.k):
            self.centroides[i] = data_train.iloc[idx[i]].values
        #inicialização da lista dos centros
        self.centro_inicias = self.centroides.copy()
        for j in range(self.num_max_iter):
            self.iters += 1
            for d in self.centroides:
                self.d_grupos[d] = []
            #calcular a qual dado pertence o dado
            for ix, row in data_train.iterrows():
                flag_f = True
                for g in self.centroides:
                    if flag_f:
                        aux = self.dist_euclidean(self.centroides[g], row.values)
                        flag_f = False
                        aux_g = g
                    elif aux > self.dist_euclidean(self.centroides[g], row.values):
                        aux = self.dist_euclidean(self.centroides[g], row.values)
                        aux_g = g

                ax_d = self.d_grupos[aux_g]
                ax_d.append(ix)
                self.d_grupos.update({aux_g:ax_d})

            aux_centro = self.centroides.copy() #para comparar si os centros mudam
            #Para o calculo dos novos centroides
            for d in self.d_grupos.keys():
                if len(self.d_grupos[d]) != 0:
                    aux_data = data_train.iloc[self.d_grupos[d]]
                    aux_data.append(pd.DataFrame([self.centroides[d]], columns=aux_data.columns)
                                    ignore_index=True)
                    self.centroides.update({d:aux_data.mean().values})

            #Si os centros não mudam o algoritmo vai parar
            count = 0
            for i in self.centroides:
```

```
                if (self.centroides[i] == aux_centro[i]).all():
                    count += 1
            if count == len(self.centroides):
                break
        return self.centroides
```

In [16]:
```
ruta_arff = "./data/iris_dataset.arff" #ruta do arquivo
with open(ruta_arff, "r") as inFile:
    content = inFile.readlines()
```

- Se cria o objeto e se treina, para 3 clusters com um máximo de 100 iterações.

In [20]:
```
obj = KNN(k=3, num_max_iter=100, arff=content)
g = obj.train()
```

In [21]:
```
1 obj.toDataFrame()
```

Out[21]:

	sepallength	sepalwidth	petallength	petalwidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	6.3	2.5	4.9	1.5
15	6.1	2.8	4.7	1.2
16	6.4	2.9	4.3	1.3
17	6.6	3.0	4.4	1.4
18	6.8	2.8	4.8	1.4
19	6.7	3.0	5.0	1.7
20	6.0	2.9	4.5	1.5
21	5.7	2.6	3.5	1.0
22	5.5	2.4	3.8	1.1
23	5.5	2.4	3.7	1.0
24	5.8	2.7	3.9	1.2

	sepallength	sepalwidth	petallength	petalwidth
25	6.0	2.7	5.1	1.6
26	5.4	3.0	4.5	1.5
27	6.0	3.4	4.5	1.6
28	6.7	3.0	5.2	2.3
29	6.3	2.5	5.0	1.9
30	6.5	3.0	5.2	2.0
31	6.2	3.4	5.4	2.3
32	5.9	3.0	5.1	1.8
33	100.0	120.0	123.0	140.0
34	110.0	100.0	102.0	130.0
35	140.0	90.0	120.0	130.0

In [22]:
```
print("--------- Clusters gerados ---------")
for i in g:
    print("Cluster " + str(i)+":", g[i])
print()
print("--------- Clusters iniciais ---------")
c_i = obj.centro_inicias
for i in c_i:
    print("Cluster " + str(i)+":", c_i[i])
print()
print("--------- Grupos gerados (Cluster, índices) ---------")
gg = obj.get_groups()
for i in gg:
    print("Cluster " + str(i)+":", gg[i])
print()
print("--------- Número de iterações para convergir ---------")
print(obj.get_iters())
```

```
--------- Clusters gerados ---------
Cluster 0: [116.66666667 103.33333333 115.         133.33333333]
Cluster 1: [6.12631579 2.84210526 4.60526316 1.54210526]
Cluster 2: [4.85       3.3        1.43571429 0.2       ]

--------- Clusters iniciais ---------
Cluster 0: [110. 100. 102. 130.]
Cluster 1: [6.3 2.5 5.  1.9]
Cluster 2: [4.8 3.  1.4 0.1]

--------- Grupos gerados (Cluster, índices) ---------
Cluster 0: [33, 34, 35]
Cluster 1: [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]
Cluster 2: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

--------- Número de iterações para convergir ---------
2
```

Ponto 5

5) [2.0 pts] Fazer questão (slide 23 e 24) da aula sobre teste de significância do prof. Ronnie.

$$d_t = d \pm t_{1-\alpha,k-1}\hat{\sigma}_t$$

$$d_t = 0.004 \pm 2.06 * 0.003$$

$$d_t = 0.004 \pm 0.00618$$

$$d_1 = 0.004 + 0.00618 = 0.01018$$

$$d_2 = 0.004 - 0.00618 = -0.00218$$

- Como o intervalo é bem próximo de zero, não se pode dezir que a diferencia entre os modelos seja estadisticamente significativa.