Multi Layer Perceptron (MLP)

Importar as bibliotecas

In [25]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import normalize
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
import tensorflow as tf
from keras.utils import np_utils
```

Carregar Base de Dados

In [4]:

```
data=pd.read_csv("data/iris.csv")
data.sample(10)
```

Out[4]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
66	5.6	3.0	4.5	1.5	Iris-versicolor
40	5.0	3.5	1.3	0.3	Iris-setosa
103	6.3	2.9	5.6	1.8	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
16	5.4	3.9	1.3	0.4	Iris-setosa
69	5.6	2.5	3.9	1.1	Iris-versicolor
109	7.2	3.6	6.1	2.5	Iris-virginica
36	5.5	3.5	1.3	0.2	Iris-setosa
90	5.5	2.6	4.4	1.2	Iris-versicolor
79	5.7	2.6	3.5	1.0	Iris-versicolor

Codificar as categorias

In [7]:

```
data.loc[data["Species"]=="Iris-setosa","Species"]=0
data.loc[data["Species"]=="Iris-versicolor","Species"]=1
data.loc[data["Species"]=="Iris-virginica","Species"]=2


data=data.iloc[np.random.permutation(len(data))]
print(data.sample(10))
```

```
     SepalLength  SepalWidth  PetalLength  PetalWidth Species
96           5.7         2.9          4.2         1.3       1
11           4.8         3.4          1.6         0.2       0
58           6.6         2.9          4.6         1.3       1
125          7.2         3.2          6.0         1.8       2
92           5.8         2.6          4.0         1.2       1
104          6.5         3.0          5.8         2.2       2
52           6.9         3.1          4.9         1.5       1
139          6.9         3.1          5.4         2.1       2
28           5.2         3.4          1.4         0.2       0
136          6.3         3.4          5.6         2.4       2
```

Separar dados descritivos das classes

In [8]:

```
X=data.iloc[:,:4].values
y=data.iloc[:,4].values
```

In [10]:

```
print("Shape of X",X.shape)
print("Shape of y",y.shape)
```

```
Shape of X (150, 4)
Shape of y (150,)
```

Dividir dados para treinamento, validação e teste

In [20]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
import numpy as np

# Normalizar os dados
X_normalized = normalize(X, axis=0)

'''
70% - conjunto de treinamento
20% - conjunto de validação
10% - conjunto de teste
'''
X_train_val, X_test, y_train_val, y_test = train_test_split(X_normalized, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2

# Transformar os rótulos em categorias
num_classes = 3
y_train = np_utils.to_categorical(y_train, num_classes)
y_val = np_utils.to_categorical(y_val, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

Arquitetura do MLP

In [21]:

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(16,input_dim=4,activation='relu'))
model.add(tf.keras.layers.Dense(64,activation='relu'))
model.add(tf.keras.layers.Dense(64,activation='relu'))

#model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(3,activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 16)                80

 dense_13 (Dense)            (None, 64)                1088

 dense_14 (Dense)            (None, 64)                4160

 dense_15 (Dense)            (None, 3)                 195

=================================================================
Total params: 5,523
Trainable params: 5,523
Non-trainable params: 0
_________________________________________________________________
```

Treinar a rede

In [24]:

```
class_weights = {
    0: 1.0,  # peso da classe 0
    1: 1.0,  # peso da classe 1
    2: 1.0,  # peso da classe 2

}

model.fit(X_train,
          y_train,
          validation_split=0.2,
          batch_size=16,
          epochs=50,
          verbose=1,
          class_weight=class_weights)
```

```
Epoch 70/75
6/6 [==============================] - 0s 8ms/step - loss: 0.0945 - accu
racy: 0.9762 - val_loss: 0.0610 - val_accuracy: 1.0000
Epoch 71/75
6/6 [==============================] - 0s 9ms/step - loss: 0.0951 - accu
racy: 0.9643 - val_loss: 0.0587 - val_accuracy: 1.0000
Epoch 72/75
6/6 [==============================] - 0s 9ms/step - loss: 0.0934 - accu
racy: 0.9643 - val_loss: 0.0614 - val_accuracy: 1.0000
Epoch 73/75
6/6 [==============================] - 0s 8ms/step - loss: 0.0887 - accu
racy: 0.9643 - val_loss: 0.0559 - val_accuracy: 1.0000
Epoch 74/75
6/6 [==============================] - 0s 9ms/step - loss: 0.0906 - accu
racy: 0.9524 - val_loss: 0.0558 - val_accuracy: 1.0000
Epoch 75/75
6/6 [==============================] - 0s 10ms/step - loss: 0.0889 - acc
uracy: 0.9643 - val_loss: 0.0559 - val_accuracy: 1.0000
```

Out[24]:

Testar

In [26]:

```
# Fazer previsões nos dados de teste
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Calcular a matriz de confusão
matriz_confusao = confusion_matrix(np.argmax(y_test, axis=1), y_pred_classes)

# Calcular a acurácia
acuracia = accuracy_score(np.argmax(y_test, axis=1), y_pred_classes)

# Calcular a pontuação F1
f1 = f1_score(np.argmax(y_test, axis=1), y_pred_classes, average='macro')

print("Matriz de Confusão:")
print(matriz_confusao)
print("Acurácia:", acuracia)
print("Pontuação F1:", f1)
```

```
1/1 [==============================] - 0s 82ms/step
Matriz de Confusão:
[[6 0 0]
 [0 5 0]
 [0 0 4]]
Acurácia: 1.0
Pontuação F1: 1.0
```

In [ ]:

```

```