Perceptron

Question 1

- Implemente o Perceptron para classificar o dataset Iris, separando as classes “setosa” e “versicolor”.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
#Load the dataset
from sklearn.datasets import load_iris
iris = load_iris()
X = iris['data']
Y = iris['target']

#Separando setosa e versicolor
x = X[(Y==0)|(Y==1)]
y = Y[(Y==0)|(Y==1)]

#Mudando os valores de y para implementar o perceptron
y = np.where(y == 0, 1, -1)
```

In [3]:
```
#Função para predição da saída
def predict(x,w,b):
    return np.where((np.dot(x, w) + b)>= 0, 1, -1)

#Setup dos valores inicias
w=np.zeros(x.shape[1])
b=0
learning_rate=0.1
iter=100 #qtd de interações escolhida arbitrariamente

#Primeira predição
p1=predict(x,w,b)
print('\nOutput without weight:\n',p1)

#Contagem de erros
erro=0
for i in range(len(p1)):
    if p1[i]!=y[i]:
        erro=erro+1
print('\nQuantity of worng predictions:',erro)
errado=0

#Atualização do peso - 10 interações
for h in range(iter):
    for i in range(len(y)):
        delta= (y[i]-predict(x,w,b)[i])*learning_rate
        w = delta*x[i] + w
        b= b+ delta


#Segunda predição
p2=predict(x,w,b)
print('\n\nOutput after weight implementation:\n',p2)

#Contagem de erros
erro=0
for i in range(len(p2)):
    if p2[i]!=y[i]:
        erro=erro+1
print('\nQuantity of wrong predictions:',erro)
```

```
Output without weight:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

Quantity of worng predictions: 50


Output after weight implementation:
 [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1]

Quantity of wrong predictions: 0
```

In [4]:
```
p2
```

Out[4]:
```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1])
```

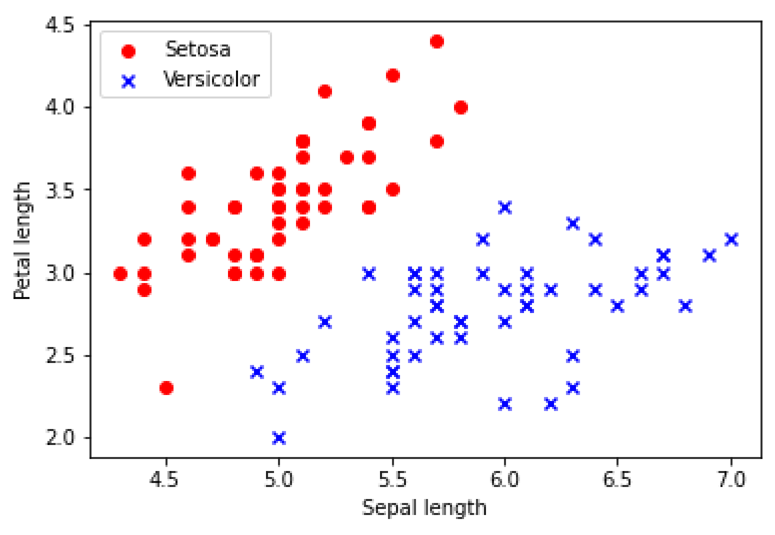
In [5]:
```
#Gráfico ilustrativo de 2 dimensões para representar a separação das classes

x = x[0:100]
y = y[0:100]

# plot Iris Setosa samples
plt.scatter(x[:50, 0], x[:50, 1], color='red', marker='o', label='Setosa')
# plot Iris Versicolour samples
plt.scatter(x[50:100, 0], x[50:100, 1], color='blue', marker='x', label='Versicolor')

# show the legend
plt.xlabel("Sepal length")
plt.ylabel("Petal length")
plt.legend(loc='upper left')

# show the plot
plt.show()
```



Question 2

- Implemente o Perceptron Multiclasses euse no datasert Iris, separando as 3 classes (é possível?)

Não é possível utilizar o Perceptron para separar as 3 classes linearmente, pois este algoritmo atua somente separando **duas** classes. Todavia, para esta implementação serão separadas as classes de 2 em 2.

- Setosa e versicolor já foram separadas no exercício anterior.
- Versicolor e virginica serão separadas abaixo. Depois disto, os resultados destas duas classificações serão unidos para gerar a separação das três classes.

In [6]:
```
#Load the dataset
from sklearn.datasets import load_iris
iris = load_iris()
X = iris['data']
Y = iris['target']

#Separando versicolor e virginica
x = X[(Y==1)|(Y==2)]
y = Y[(Y==1)|(Y==2)]

#Mudando os valores de y para implementar o perceptron
y = np.where(y == 1, 1, -1)

#Função para predição da saída
def predict(x,w,b):
    return np.where((np.dot(x, w) + b)>= 0, 1, -1)

#Setup dos valores inicias
w=np.zeros(x.shape[1])
b=0
learning_rate=0.1
iter=100 #qtd de interações escolhida arbitrariamente

#Primeira predição
p1=predict(x,w,b)
print('\nOutput without weight:\n',p1)

#Contagem de erros
erro=0
for i in range(len(p1)):
    if p1[i]!=y[i]:
        erro=erro+1
print('\nQuantity of worng predictions:',erro)
errado=0

#Atualização do peso - 10 interações
for h in range(iter):
    for i in range(len(y)):
        delta= (y[i]-predict(x,w,b)[i])*learning_rate
        w = delta*x[i] + w
        b= b+ delta


#Segunda predição
p22=predict(x,w,b)
print('\n\nOutput after weight implementation:\n',p2)

#Contagem de erros
erro=0
for i in range(len(p2)):
    if p2[i]!=y[i]:
        erro=erro+1
print('\nQuantity of wrong predictions:',erro)
```

```
Output without weight:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

Quantity of worng predictions: 50


Output after weight implementation:
 [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1]

Quantity of wrong predictions: 0
```

In [7]:
```
#Junção dos resultados para separar as três classes
y = iris['target']
p2= np.where(p2 == 1, 0, 1)
p22 =np.where(p22 == 1, 1, 2)

ans=[]
for i in p2:
    ans.append(i) # setosa
for i in p22[50:100]:
    ans.append(i) #virginica
#ans

e=0
a=0
for i in range(len(ans)):
    if ans[i]!=y[i]:
        e=e+1
    else:
        a=a+1
print('Answer:\n',ans)
print('Quantity of erros:',e)
print('Quantity of right predictions:',a)
```

```
Answer:
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2]
Quantity of erros: 0
Quantity of right predictions: 150
```

O algoritmo Perceptron não é muito utilizado sozinho, uma vez que só pode separar 2 classes por vez. Todavia, ele foi a base para o

s, as Multilayers Perceptrons (MLP).