Lab 2 – Basic Regression and Classification

Author: Gonçalo Aguiar 904475

1. **Calculate the total mean squared error between the predictions of the width of the sepals learned by the regression model and the actual values of this feature - for the training set and separately for the test set. Use the .predict() method to obtain predictions from the trained regression model.**

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

To calculate the total mean squared error the following code was written(the code represents the MSE formula):

- For the training set:

```
predictions = regr.predict(X[train_inds,0:1])
j=0
error =0
for i in predictions:
    aux = i -  X[train_inds,1:][j]
    error = error + pow(aux,2)
    j = j+1


print("\n TrainingError->",error/40)
```

- For the test set:

```
predictions = regr.predict(X[test_inds,0:1])
j=0
error =0
for i in predictions:
    aux = i -  X[test_inds,1:][j]
    error = error + pow(aux,2)
    j = j+1

print("\n TestError->",error/10)
```

The results were the following:

```
TrainingError-> [0.05710711]
```

```
TestError-> [0.08977383]
```

In order to confirm this values I used the sklearn mean_squared_error:

```
from sklearn.metrics import mean_squared_error
print("\nConfirm TRainingError",mean_squared_error(predictions, X[train_inds,1:]))
```

```
from sklearn.metrics import mean_squared_error
print("\nConfirm TEstError->",mean_squared_error(predictions, X[test_inds,1:]))
```

```
Confirm TRainingError 0.05710711096226839
```

```
Confirm TEstError-> 0.08977382563146889
```

2. **Modify the above example to predict the width of sepals based on two other features - the length of sepals (0) and the length of petals (2). To do this, fill in the multiple_regression.py program in places marked with: # !!!. Note the values of the learned parameters in the report and include the obtained graph.**

In order to complete the given program the following lines were written:

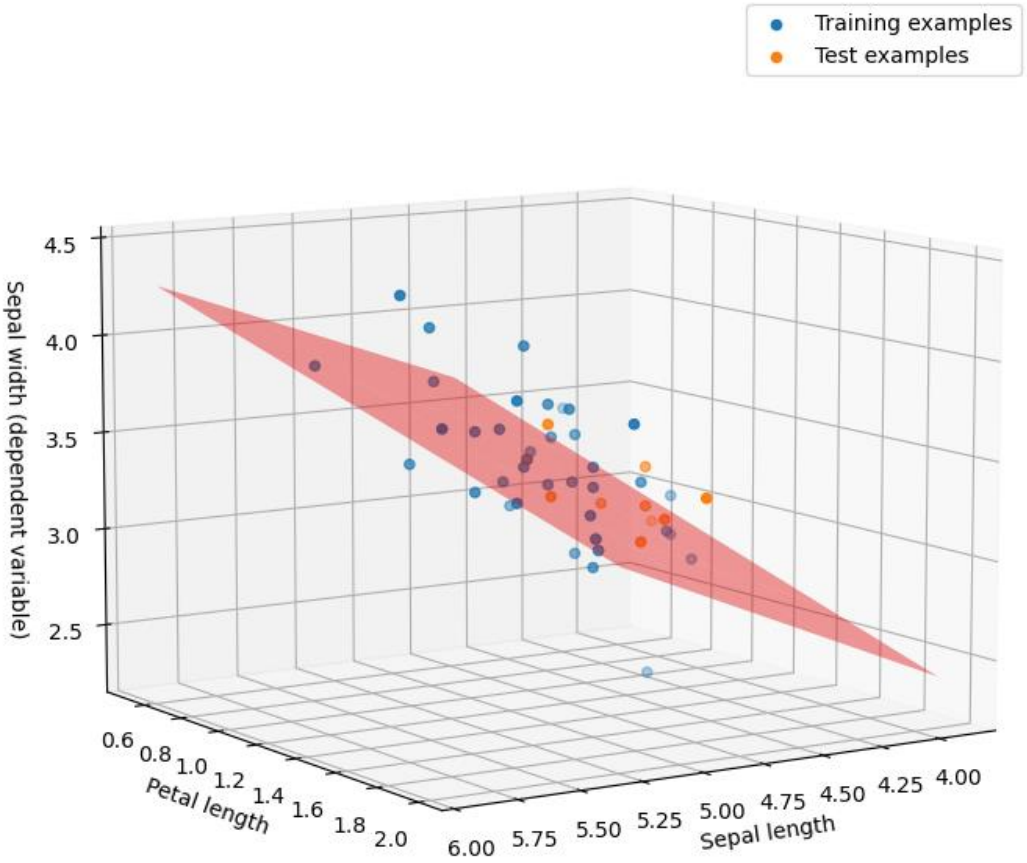
```
# take only examples from class 0 (iris setosa) and first 3 features
# !!!
X = X[Y==0][:,0:3]
Y = Y[Y==0]
```

```
# create the regression model and train it
from sklearn import linear_model
multiregr = linear_model.LinearRegression()
multiregr.fit(X[train_inds,:][:,[0,2]], X[train_inds, 1])
```

The values of the learned parameters(coeficientes and intercept respectively) are :

```
[ 0.86579148 -0.08006363] -0.8081408006674606
```

The resulting graph is the following:



3. **Draw a necessary conclusion from the information that for $wTx < 0$ the classifier returns class 0, and for $wTx \geq 0$ class 1. Then calculate (using variables) the parameters of the decision boundary. Write down the obtained formulas (and possibly their derivation).**
   Since wTx<0 corresponds to class 0 and wTx > 0 corresponds to class 1 , we can set the decision boundary to be the hyperplane where wTx = 0.
   If we take as an example a 2-dimensional space the decision boundary equation becomes wTx = w1x1 + w2x2 = 0.
   From the last equation we can rearrange it and get x2 = -(w1/w2)x1 where x2 is the predicted class(0 or 1) and x1 is the feature input. If we know the values of w1 and w2 we can predict the value of x2 based on a x1 value.

4. **Expand the logistic_regression.py program by adding the calculation of the probability with which the test samples were assigned to classes. Use the .predict_proba() method, which can be called on an instance of the LogisticRegression() object, for this purpose. Paste the probabilities for successive samples, the class assigned by the classifier, and the actual class into the report.**

   In order to complete the task the following code was written:

```
probs = clf.predict_proba(X01_test)

y_pred = clf.predict(X01_test)

for i in range(len(y01_test)):
    print(f"Sample {i}: probabilities {probs[i]}, predicted class {y_pred[i]}, actual class {y01_test[i]}")
```

   The printed result was:

```
Sample 0: probabilities [0.94711971 0.05288029], predicted class 0, actual class 0
Sample 1: probabilities [0.91053441 0.08946559], predicted class 0, actual class 0
Sample 2: probabilities [0.81369998 0.18630002], predicted class 0, actual class 0
Sample 3: probabilities [0.93104458 0.06895542], predicted class 0, actual class 0
Sample 4: probabilities [0.94707105 0.05292895], predicted class 0, actual class 0
Sample 5: probabilities [0.91056407 0.08943593], predicted class 0, actual class 0
Sample 6: probabilities [0.85283656 0.14716344], predicted class 0, actual class 0
Sample 7: probabilities [0.88481547 0.11518453], predicted class 0, actual class 0
Sample 8: probabilities [0.85280608 0.14719392], predicted class 0, actual class 0
Sample 9: probabilities [0.97659995 0.02340005], predicted class 0, actual class 0
Sample 10: probabilities [0.0776379 0.9223621], predicted class 1, actual class 1
Sample 11: probabilities [0.05968883 0.94031117], predicted class 1, actual class 1
Sample 12: probabilities [0.0201279 0.9798721], predicted class 1, actual class 1
Sample 13: probabilities [0.05969565 0.94030435], predicted class 1, actual class 1
Sample 14: probabilities [0.16399255 0.83600745], predicted class 1, actual class 1
Sample 15: probabilities [0.31380572 0.68619428], predicted class 1, actual class 1
Sample 16: probabilities [0.01526377 0.98473623], predicted class 1, actual class 1
Sample 17: probabilities [0.1003752 0.8996248], predicted class 1, actual class 1
Sample 18: probabilities [0.12884126 0.87115874], predicted class 1, actual class 1
Sample 19: probabilities [0.07758575 0.92241425], predicted class 1, actual class 1
```

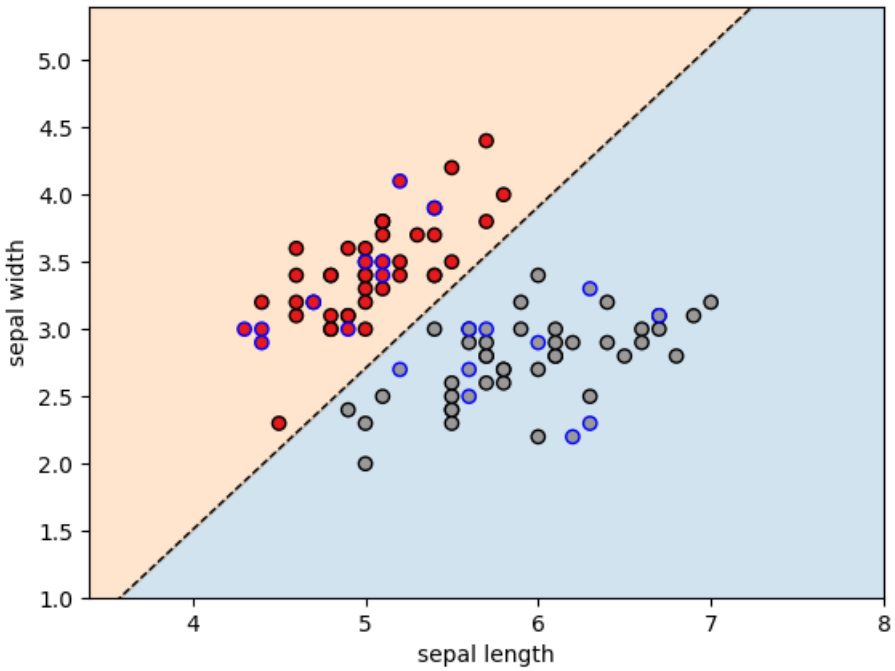
As we can see in the results the classifier got 100% accuracy since it predicted the right class every time for each of the 20 samples.

5. **Add support vector machine classification to the previous logistic regression program (for the same training and test set). The appropriate function is SVC from the sklearn.svm module. When defining the classifier object, provide parameters: kernel='linear', C=1E10. Caution: We set the kernel to 'linear' because we are looking for a linear decision boundary; the parameter C concerns the so-called regularization, which counteracts the phenomenon of overfitting - excessive adjustment of model parameters to training data. Save the graphs and draw your conclusions.**

   In order to fullfil the task the following line was replaced by the logistic regression one.

```
clf = SVC(kernel='linear', C=1E10).fit(X01_train, y01_train)
```

   The resulting plot was:



   After running the changed code a few times and comparing the plots created by the SVC with the plots created by the LogisticRegression I reached the conclusion that the results are very similiar because the decision boundaries are identical. This indicates that both the models are making similar predictions.

   However, the decision boundary of the SVC model looks slightly smoother with the iris dataset.