Multiclass classification (OvR)

APPLIED MACHINE LEARNING

Assignment 1_GR34

Ahmed Hallaba Omar Sorour
Kamel El-Sehly

Table of Contents

1. Objectives

In this assignment we will implement multiclass classification algorithm and apply it on the iris dataset using. **One versus Rest** methodology will be used to achieve this goal. Iris dataset contains three classes (Iris-Setosa,Iris-Versicolour, Iris-Virginica), and four features (sepal length, sepal width, petal length, petal width) so we aim to create a model that can predict the proper class from given two features. OvR involves training a binary class classifier for each class. During testing process, each classifier will predict the confidence of each class and OvR will select the one with highest confidence.

2. Implementation Functions:

2.1. Load iris dataset:

first of all, we imported the required libraries and the iris dataset from the scikit-learn library.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import svm, datasets
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import classification_report, confusion_matrix
```

```
In [2]: def loadDataset():
            dataset = datasets.load_iris()
            features = dataset.data
            labels = dataset.target
            return dataset, features, labels
```

2.2. Preparing the dataset:

we created this function to strip the first two features out of the dataset.

```
def prepareDataset (X, y):
    features = X
    labels = y
    return np.array(features)[:, 2:4], np.array(labels)
```

2.3 Preparing functions for plotting the data: using the code provided in the tutorials, plotData(): for plotting the points, plotRegions(): for class region plotting

```
def plotRegions(model):
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    xx = np.linspace(xlim[ 0], xlim[1])
    yy = np.linspace(ylim[ 0], ylim[1])
    XX, YY = np.meshgrid(xx, yy)
    z = np.vstack([XX.ravel(), YY.ravel()]).T
    ZZ = model.decision_function(z) .reshape(XX.shape)
    ax.contourf(XX, YY, ZZ, colors=[ 'c','y'], levels=0, alpha=0.2)
```

```
def plotData(features, labels, cls, class_names,class_act):
    colors = []
    markers = []
    for i in range(3):
        if(i==class_act):
            colors.append('b')
            markers.append('o')
        else:
            colors.append('k')
            markers.append('*')

    for class_index in range(len(cls)):
        plt.scatter(features[labels == cls[class_index], 0], features[labels ==cls[class_index], 1], c=colors[class_index],
                    marker=markers[class_index],label=class_names[cls[class_index]])
    Title = "Iris Dataset with {n} classes" .format(n=int(len(cls)))
    plt.title(Title)
    plt.xlabel('Petal Length (cm)' )
    plt.ylabel('Petal width (cm)' )
    plt.legend()
```

2.4 Plotting the miss-classed points for each class: using the function plotComData()

```
def plotCompData(features, labels, cls, class_names):
    colors = ['r','g']
    markers = ['*','+']
    for class_index in range(len(cls)):
        plt.scatter(features[labels == cls[class_index], 0], features[labels ==cls[class_index], 1], c=colors[class_index],
                    marker=markers[class_index],label=class_names[cls[class_index]])
    Title = "Compare plot prediction and actual data"
    plt.title(Title)
    plt.xlabel('Petal Length (cm)' )
    plt.ylabel('Petal width (cm)' )
    plt.legend()
```

2.5 retrieving the classes' names : the (getClassNames) retrieves a dictionary in which the keys represents the numeric representation of the classes and the value represents the names of the classes.

```
def getClassNames (target_names):
    cls_all = {}
    for i, label in enumerate(target_names):
        cls_all[i] = label
    return cls_all
```

2.6 Selecting only one class to be identified: We created a function called (binarize) that maps the value of the class to be identified in each model to 1 and the values of the other classes to -1.

```
def binarize(labels,pv_class):
    result=[]
    for i in range(len(labels)):
        if(labels[i]==pv_class):
            result.append(1)
        else:
            result.append(-1)
    return result
```

2.7 Training the models: we used scikit-learn library to train model using logistic regression and SVM algorithms by the provided functions from the library (LogisticRegression(),svm.SVC())

```
def logisticreg(X,y,class_act):
    y=binarize(y,class_act)
    clf = LogisticRegression(random_state=0,).fit(X, y)
    y_pred =clf.predict(X)
    y_proba = clf.predict_proba(X)
    return clf,clf.score(X, y)*100, confusion_matrix(y, y_pred),y_pred,y_proba
```

```
def suppvecmach(X,y,class_act):
    y=binarize(y,class_act)
    clf = svm.SVC(decision_function_shape='ovo',probability=True).fit(X, y)
    y_pred = clf.predict(X)
    y_proba = clf.predict_proba(X)
    return clf,clf.score(X, y)*100, confusion_matrix(y, y_pred),y_pred,y_proba
```

2.8 We created a function that we named (performance) for plotting the confusion matrix and the score along with the data that were mis-classified in another figure:

```
def performance(y_predict):
    print('Confusion matrix\n')
    hm=sn.heatmap(confusion_matrix(y,y_predict), annot=True)
    plt.show()
    counter=0
    comp=[]
    for i in range(len(y)):
        if y[i]!=y_predict[i]:
            counter+=1
            comp.append(0)
        else:
            comp.append(1)

    comp=np.array(comp)

    plotCompData(X,comp,[0,1],['Wrong','Correct'])
    print('Score',(1-counter/len(y))*100,'%')
```

3. Requirements and results

3.1 Load the Iris dataset: we loaded the dataset using the function described in (2.1)

```
dataset, features, labels = loadDataset()
class_names = getClassNames(dataset.target_names)
X, y = prepareDataset(features, labels)
cls=[0,1,2]
```

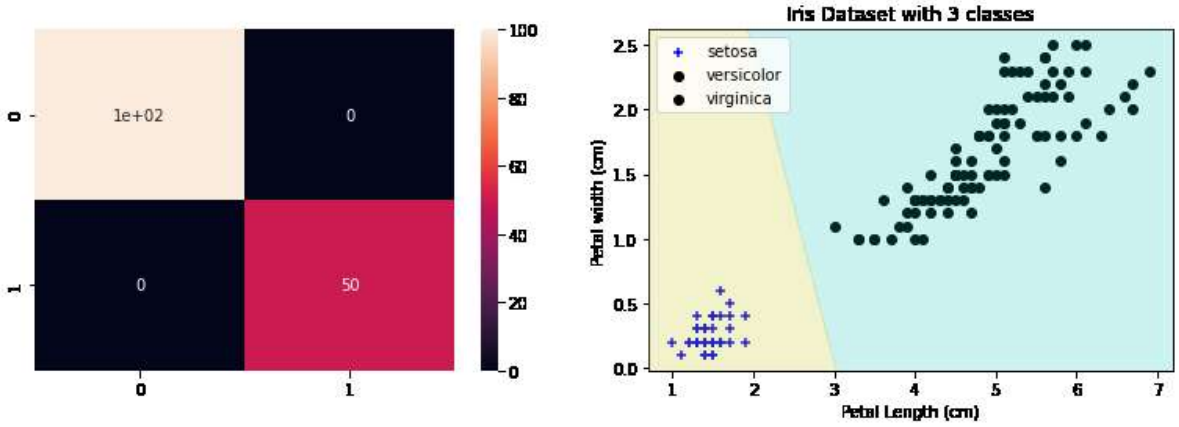
3.2 Build OvR-LR and OvR-SVM and test on 2D Iris dataset (which contains

3 classes). For each class: using the function described in (2.2)

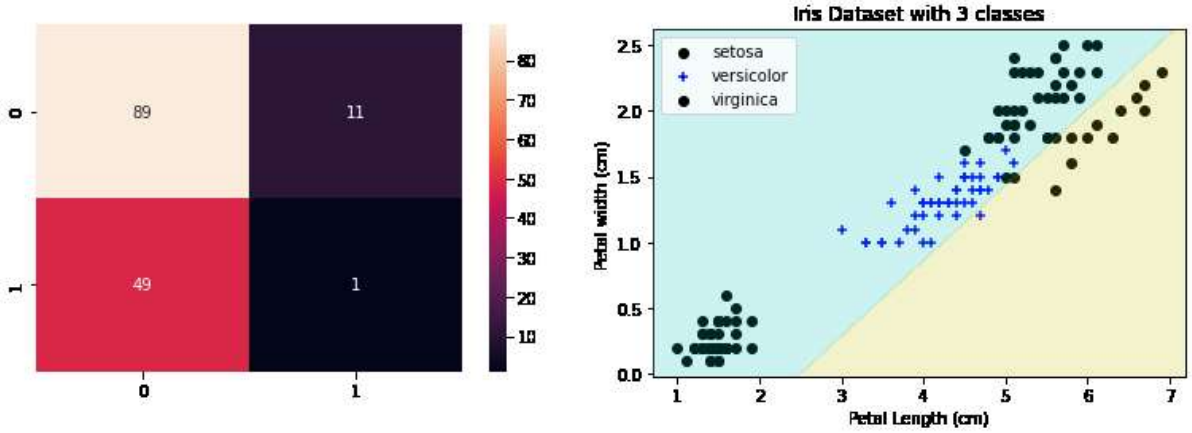
```
dataset, features, labels = loadDataset()
class_names = getClassNames(dataset.target_names)
X, y = prepareDataset(features, labels)
cls=[0,1,2]
```

3.3. Obtain the binarized label (1 for positive class, -1 for negative class): we used the function declared in (2.6) this function has been embedded under another function that creates the classifiers (models) (2.7). So, we first recall the classification function it first binarizes the label then passes the labeled classes to the model.

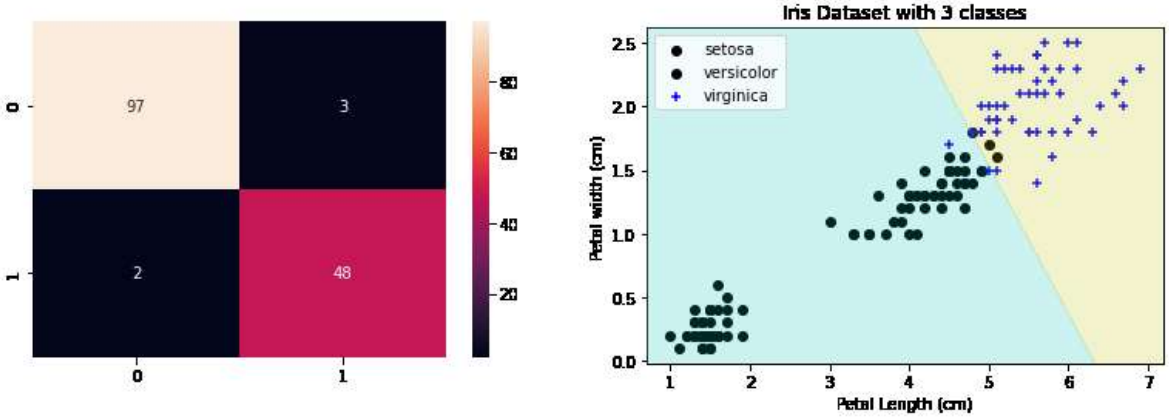
3.3.1 Obtain the LR's confusion matrix and accuracy:



Logistic regression with active class 0 Score 100.0 %
(Setosa class V.S Rest class)

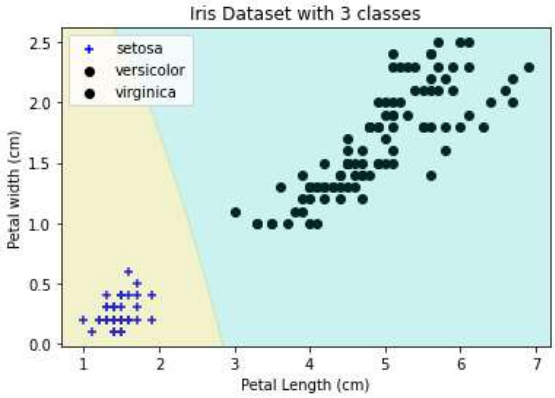


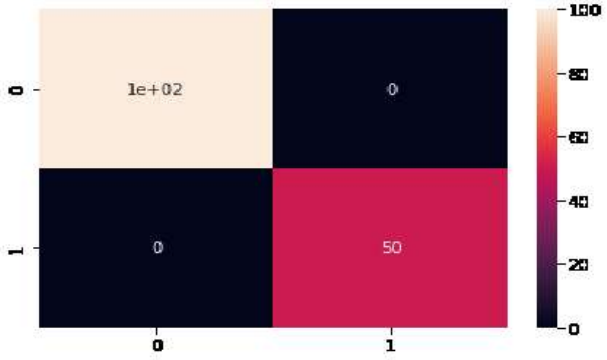
Logistic regression with active class 1
Score 60.0 % (Versicolor V.S other Classes)



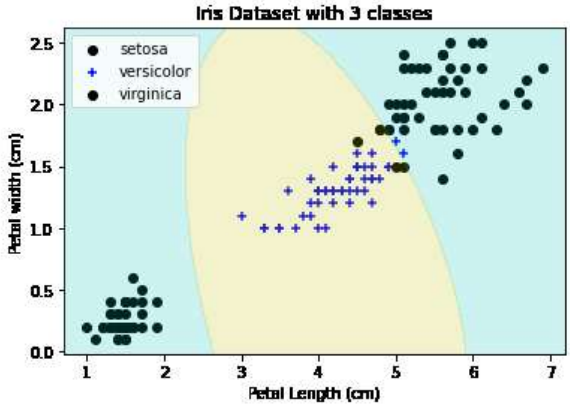
Logistic regression with active class 2
Score 96.66666666666667 % (Verginica class V.S other Classes)

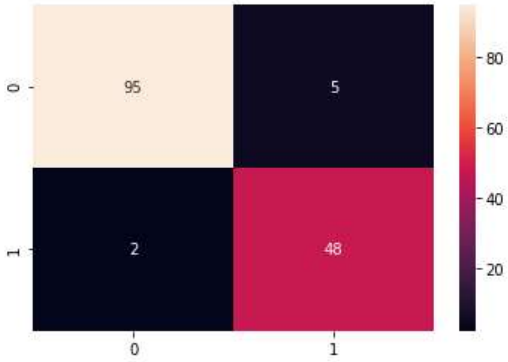
3.3.2 Obtain the SVM's confusion matrix and accuracy:



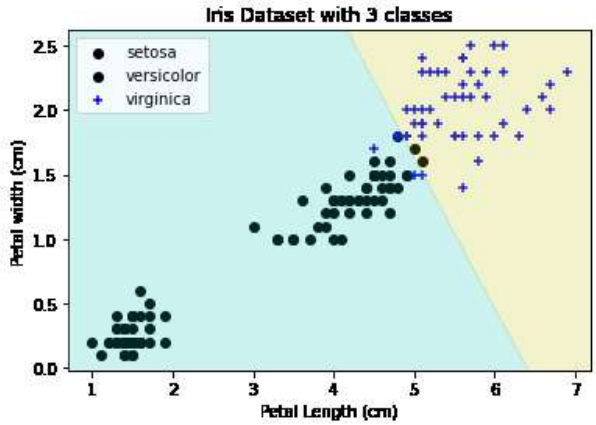


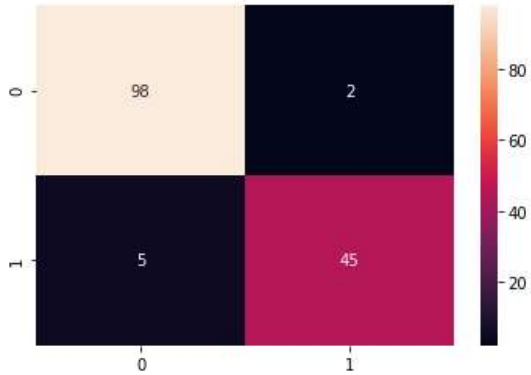
Support Vector Machine with active class 0
Score 100.0 %





Support Vector Machine with active class 1
Score 95.33333333333334 %



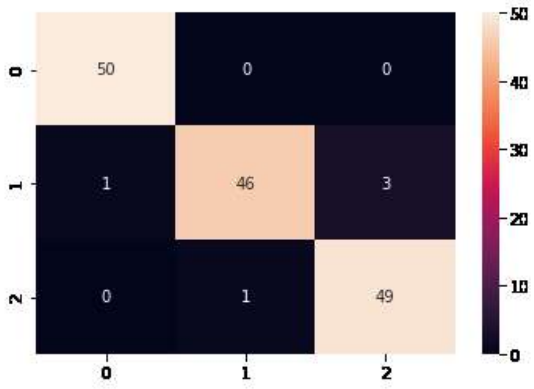


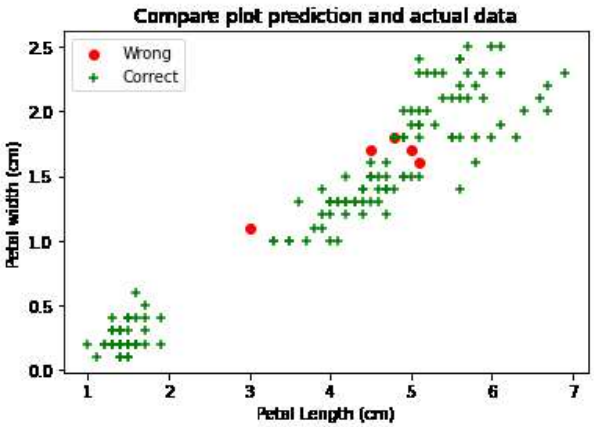
Support Vector Machine with active class 2
Score 95.33333333333334 %

3.4 For the aggregation of the results obtained from the three models we used two different way s to achieve this purpose:

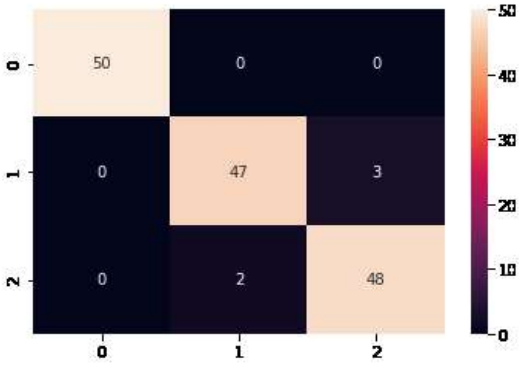
3.4.1 Aggregation using argmax function:

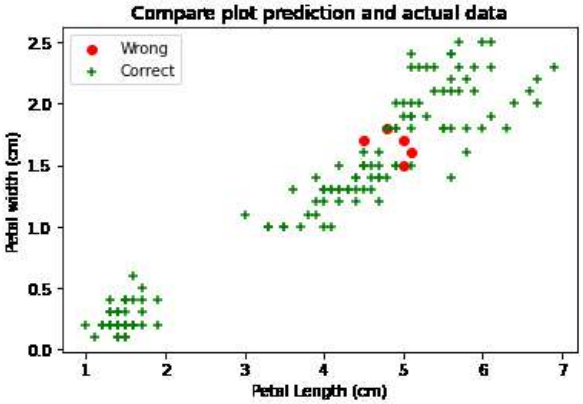
(a) For logistic Regression:





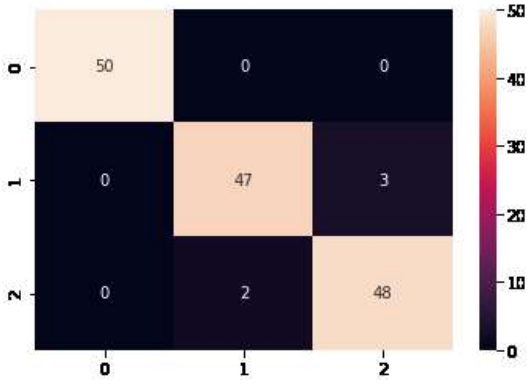
(b) For SVM:

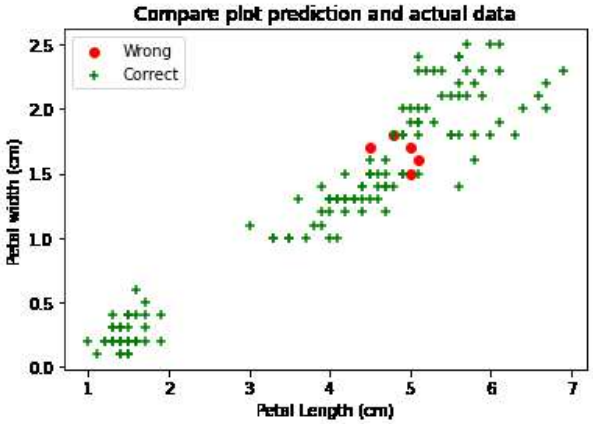




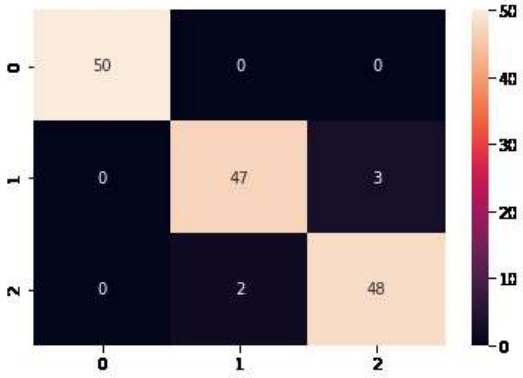
3.4.2 Aggregation using voting:

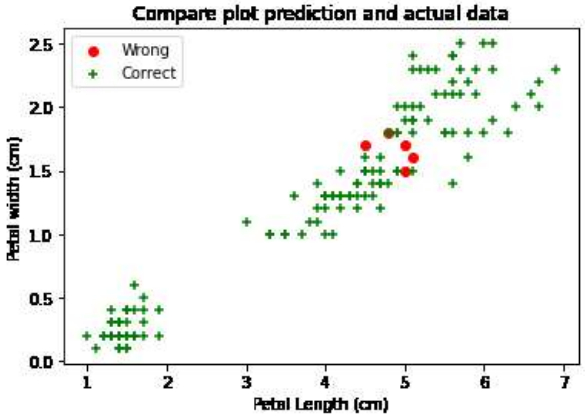
(a) For logistic Regression:





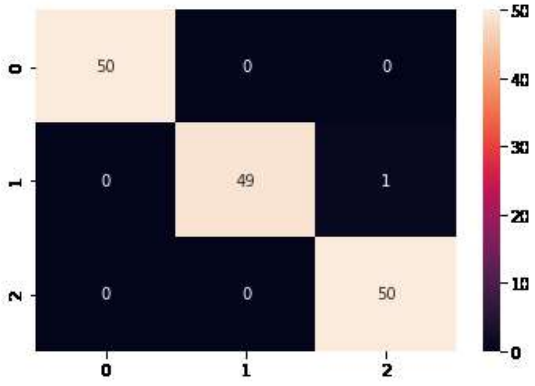
(b) For SVM:

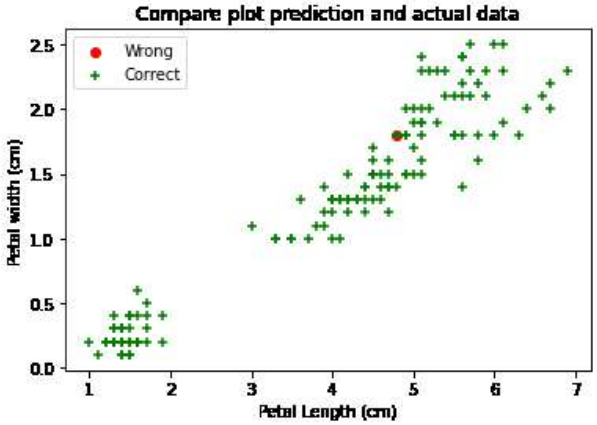




3.5 Using decision tree classification algorithm:

As an alternative scenario we created another model using the decision tree algorithm below are the results obtained





4. Conclusion

In this assignment we use the multiclass classification using OvR methodology in which we split the problem into multiclass binary classification problem and create model for predicting each class separately and after this we use an aggregation function the returns the best confidence for each model and the corresponding label. We also use an alternative scenario in which we train another model using the decision tree algorithm that achieved accuracy better than the multi class binary classification using argmax because the decision tree deals with probabilities better than returning the maximum confidence only using argmax.