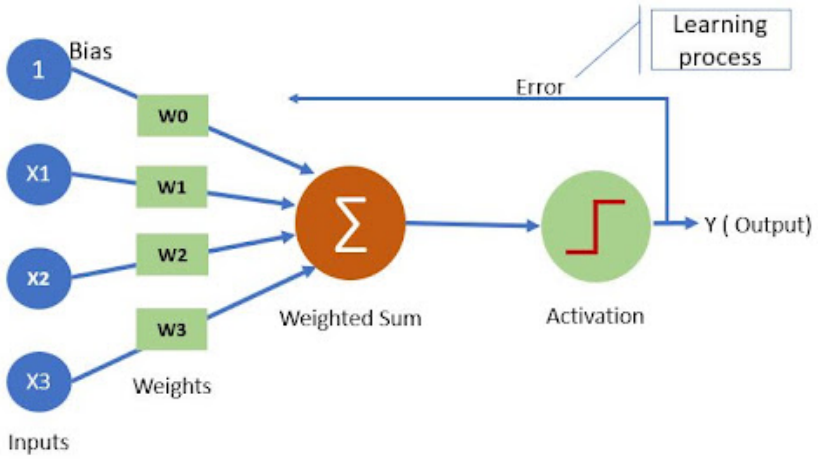
CS435 assignment

Name : Muahmmad faleh almutairi
ID : 381117591

-binary classifier Perceptron

It's a classification algorithm, a supervised one that takes the shape or the idea of a single neural cell, where inputs or features produce a single output that output either helps correct the next patch of weights, if the activation (the predicted value) doesn't match the actual correct classification y value if it's in the training process or it gives a single prediction either correct or wrong then that prediction gets compared to the actual y value updating only on error,



Single Layer Perceptron

As seen in the image a set of inputs or features and a single value called bias which is essentially acts as a safe measure, in that it prevents a non-zero threshold, that value along with the weights and features get tested by applying the activation equation,

$$a = \sum_{i=0}^{D} w_d x_d = \boldsymbol{w}^\top \boldsymbol{x}$$

As seen in the equation a is the result of a summation or a loop which covers all elements, where each element in the W matrix gets multiplied by the corresponding value of the x matrix that value is then added to the bias value to make sure no cases of a zero threshold, then finally the activation value gets tested against the y values (the actual truth values)

Pseudocode perceptron(NumOfIt, train_list, yTrain, list_test, yTest)

- W_d <- 0 for all d =1 ,,,, d d>=5
- B <-0
- For It = 0 …… NumOfIt do
- For all (x,y) in (train_list , yTrain) do
- A <-call Predict(x_d,W_d,B)
- If (ya) <=0 do
- Error <- +1.0
- W_d <- W_d +yx_d
- B <-y
- Else do
- Correct <-+ 1.0
- End if
- End for
- Predictions <- call looptest(list_test, yTest, W_d, B)
- cur_test_acc, cur_test_err <- call accuracy(predictions, yTest)
- tp, fp, tn, fn <- call getTpnfpn(yTest, predictions)
- correct_test<- add to list (cur_test_acc)
- correct_test_con<- add to list (Accuracy(tp,tn,fp,fn))
- Fscore<- add to list (call Precision(tp, fp, fn))
- correct_train<- add to list (((correct / len(yTrain)) * 100.0))
- error_test<- add to list (cur_test_err)
- error_train<- add to list (error)
- End for
- return correct_test, error_test, correct_train, error_train, Fscore, correct_test_con

Pseudocode Predict(x, W, B)

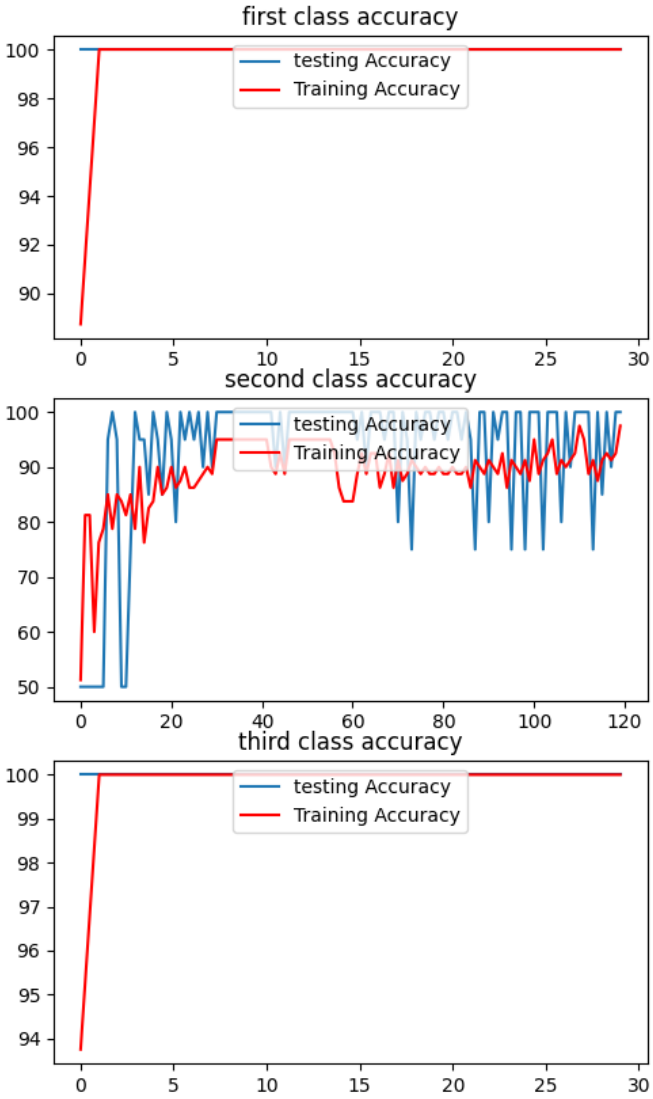
- a <- np.dot(x, W) + B
- return np.sign(a)

Pseudocode looptest(xT, yT, W, B)

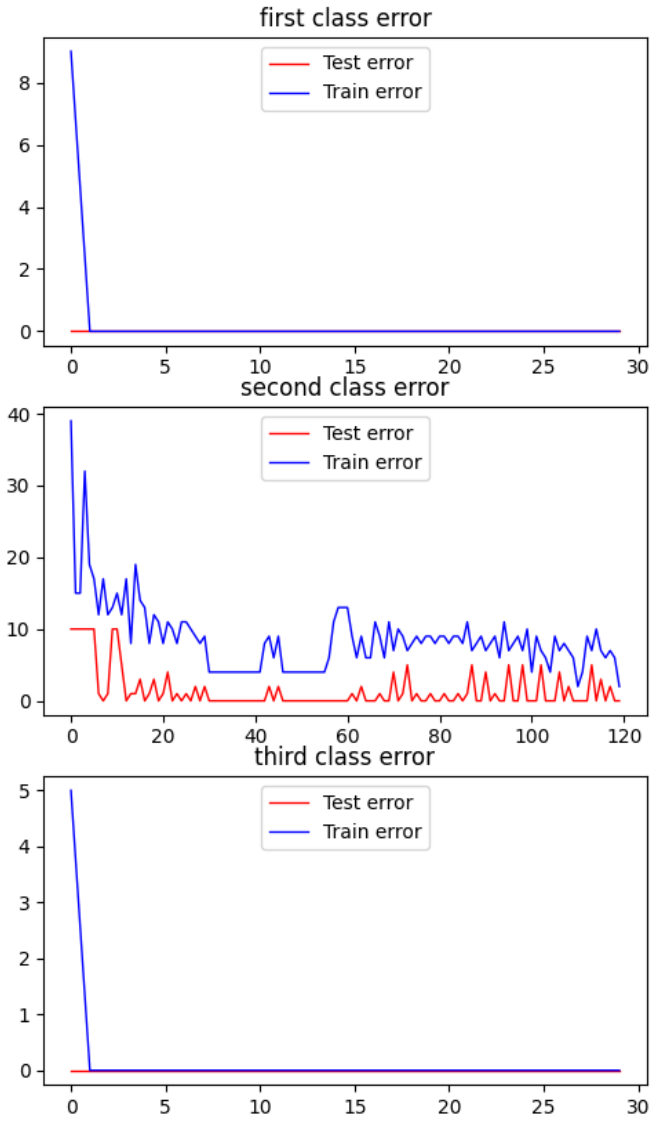
- predictions is list
- For all (x, y) in (xT, yT):
- predictions<- to list call(Predict(x, W, bais))
- End for
- return predictions

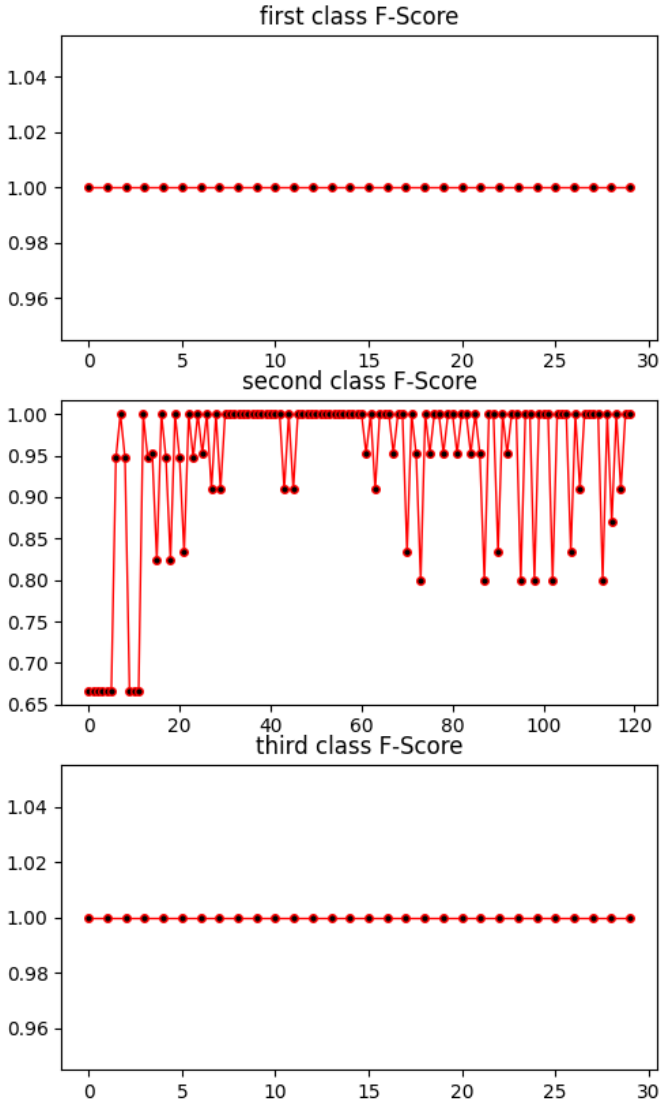
It goes like this in the algorithm if y*a is <=0 then our prediction is wrong and the weights needs to be adjusted, we ably the following equation in which we add the value of multiplying the value of y with the value of x (both are matrixes of the same size since y is a classifier) then said value gets added to the weight updating them in the process, and since we have y values as (1,-1) for correct or incorrect, if a negative value is added meaning we incorrectly classified a negative instance as a positive so we need to lower the activation and we do that by decreasing the W values, but if we incorrectly classified a positive value as negative value we need to do the opposite, increasing the W value by adding a positive x value to it.

$$w = w + yx$$



We can see that the first class is easily separated from any other class so the algorithm was able to discriminate between it and the other classes meaning **one** iterations is enough, but for class 2 and 3 not so much i found that iterations from **35 to 40 and** yielded the best result,





After 30 iterations depending on the shuffle(**if no shuffle is done the algorithm starts to forget what it had learned giving a 60% accuracy**) it could be between 85 and 96(class 2-3) if you see a value **-1 in f-score this means N/A where tp and tn were both 0**.