CSC 635 Data Mining

## Assignment 4 Report

### Submitted to:

### Dr. Jamil Saquer

### Author(s):

### Rafail Islam

**Perceptron Learning**

**Introduction**

In this assignment, we implemented perceptron learning algorithm to classify linearly separable data set. The test and train data set contains coordinates of points. Our task is to find out the decision boundary line to separate two classes of the data point. To achieve this goal, we implemented perceptron learning algorithm. After building perceptron learning algorithm, we train the model and update weights and bias according to the process described in lecture slide. After training and test the model, we plotted the data set and learned hyperplane aka the decision boundary line to visualize the classifier.

For the extra credit part, I was looking for linearly separable data set. I choose one of the most common real word data set- the iris flower data set [1]. This data set has four attributes: sepal length, sepal width, petal length, and petal width. This data set is linearly separable but it takes two decision boundary as it has 3 classes: Iris Setosa, Iris Versicolour, and Iris Virginica. To make this data set linearly separable, we did a simple trick: we converted the classes into 2 classes so that it can be classified by a perceptron.

**Background**

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt. In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. It is a type of linear classifier, i.e. a classification algorithm that makes all of its predictions based on a linear predictor function combining a set of weights with the feature vector.

Activation function is an important part of perceptron learning algorithm. We used threshold activation function and it works fine for both of the data sets.

**Implementation**

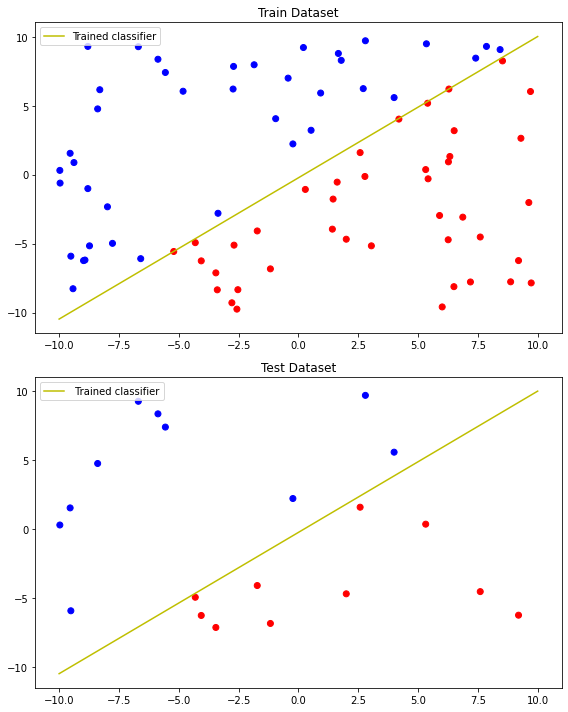
I implemented this algorithm according to the lecture slide. This algorithm is easy to understand and implemented. Firstly, we initialized weights and bias randomly. Then we calculated summation by doing dot product of input vectors/features with the weights. The bias is then added with the dot product. The summation is then passed into the activation function which returns the predicted class label. If the predicted class label is not equal to the actual class label we do weight and bias update by the equation described in lecture slide. We did the update for each data points in the data set. And we repeated the same task until all of the inputs are correctly classified. That said, we set out stopping criteria as until all inputs got classified.

However, in some data set it might not be the best stopping criteria. Because the data set might have few data that are not linearly separable. But in our case, both of our data set is learning separable. That’s why it works fine.

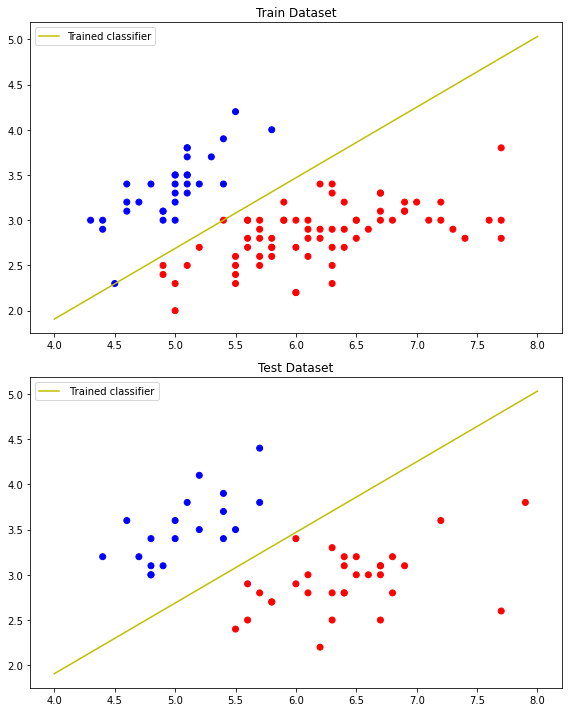
**Experimental Setup and Results**

To implement this algorithm, we used jupyter notebook and ipython. We did not do data prepossessing for part I but we did a little bit data prepossessing in extra credit part. We made the data set linearly separable manually. The prepossessing is described in introduction section.

After training the perceptron model, we tested it with the test data set and we got 100% accuracy for both data set. We plotted the data set and the decision boundary. Figure 1 shows the trained hyperplane with both training and testing data set.

Figure 1 : Plot hyperplane (decision boundary) and data sets

In the extra credit part, we train and test the model with 4 attributes of the data set*, and we got 100% accuracy. However, it is not simply possible to draw four attributes (4D) in 2D plane. To plot the data set and decision boundary, we again train and test the model with 2 attributes and got 100% accuracy. Figure 2 shows the data set and hyperplane for the Iris data set.*

*Figure 2. Iris data set and trained decision boundary.*

**Conclusion**

The perceptron algorithm is the most simplest among all types of machine learning techniques. Though it is simple it can be used for binary classifier. However, this algorithm can not classify multiple classes. Multi-layer perceptron can be used to classify multi-class problems. Apart from the algorithm itself, stopping criteria for the training process is crucial part. I used ‘until all classified’ criteria for this assignment. However, it would not be best for many of the problems. We can apply multiple stopping criteria for that such as max number of epochs and ‘desired minimum accuracy’ whichever come first.

**References**

[1]. Uci.Edu, 2020, archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data. Accessed 29 Oct. 2020.

**Code**

*"""*

Program: hw4.ipynb

Author: Rafail Islam

CSC 635

"""

#----------------------------------------import-----------------------------------------

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**import** pandas **as** pd

**from** sklearn**.**model\_selection **import** train\_test\_split

# Read data from txt file

train **=** np**.**loadtxt**(**"train.txt"**)**

test **=** np**.**loadtxt**(**"test.txt"**)**

Xtrain **,** ytrain **=** train**[:,:-**1**],**train**[:,-**1**]**

Xtest**,** ytest **=** test**[:,:-**1**],**test**[:,-**1**]**

# class for percetron model

**class** **Perceptron(object):**

**def** \_\_init\_\_**(**self**,** no\_of\_inputs**,** learning\_rate**=**0.01**):**

""" this constructor initialize variables

"""

self**.**learning\_rate **=** learning\_rate

self**.**weights **=** 1.0 **\*** np**.**random**.**randint**(-**10**,**10**,** size**=**no\_of\_inputs**)**

self**.**bias **=** 1.0 **\*** np**.**random**.**randint**(-**10**,**10**,** size**=**1**)**

**def** activation**(**self**,**summation**):**

""" thresold activation function.

"""

**return** 1. **if** summation **>** 0 **else** **-**1.

**def** predict**(**self**,** inputs**):**

""" This function calculate summataion for a given input

and return its label prediction by using activation function.

"""

# Calculate summation [x\*w+b]

summation **=** np**.**dot**(**inputs**,** self**.**weights**)** **+** self**.**bias**[**0**]**

# Make prediction

prediction **=** self**.**activation**(**summation**)**

**return** prediction

**def** train**(**self**,** Xtrain**,** ytrain**):**

""" this function takes train dataset and train the perceptron.

"""

i **=** 0

# Repeat Train loop untill all inputs are correctly classified

**while** **True:**

classified **=** 0

# Train for all inputs in train data set

**for** x**,** y\_actual **in** **zip(**Xtrain**,** ytrain**):**

# Determine predict ouput

y\_pred **=** self**.**predict**(**x**)**

# If miss classified : update weights and bias

**if** y\_pred **!=** y\_actual **:**

# Update weights

self**.**weights **+=** self**.**learning\_rate **\*** **(**y\_actual **-** y\_pred**)** **\*** x

# Update bias

self**.**bias**[**0**]** **+=** self**.**learning\_rate **\*** **(**y\_actual **-** y\_pred**)**

**else:**

classified **+=**1

i **+=**1

#print("Epoch: ",i,"classified: ",classified)

# Stop traing if all inputs are correctly classified

**if** classified **==** **len(**Xtrain**):**

**break**

**def** test**(**self**,**Xtest**,**ytest**):**

""" this function takes test dataset and calculate accuracy of the test dataset.

"""

c **=** 0.

**for** x**,** y\_actual **in** **zip(**Xtest**,**ytest**):**

# Predict lable

y\_pred **=** self**.**predict**(**x**)**

# Compare actual and predicted label

**if** y\_pred **==** y\_actual**:**

c **+=**1

**print(str(**x**).**ljust**(**15**,**' '**),** " Actual label: "**,str(**y\_actual**).**ljust**(**4**,**' '**),**" Predicted label: "**,**y\_pred**)**

# calculate accuracy in percente

accuracy **=** **(** c**/** **len(**ytest**))\***100

**print(**"Accuracy rate: %.2f%%"**%**accuracy**)**

**print()**

**print(**"Learned weights are: "**,**self**.**weights**)**

**print(**"Learned bias: "**,**self**.**bias**[**0**])**

# Build, train, test model

model **=** Perceptron**(**2**,**0.001**)**

model**.**train**(**Xtrain**,**ytrain**)**

model**.**test**(**Xtest**,**ytest**)**

#----------------------------------------part 2-----------------------------------------

# Plot dataset and hyperplane

color1 **=** **[**'red' **if** c **==** 1 **else** 'blue' **for** c **in** ytrain**]**

color2 **=** **[**'red' **if** c **==** 1 **else** 'blue' **for** c **in** ytest**]**

# Create Hyperplane

slope **=** **-**model**.**weights**[**0**]** **/** model**.**weights**[**1**]**

x **=** np**.**linspace**(-**10**,**10**)**

y **=** slope **\*** x **-** **(**model**.**bias**[**0**])** **/** model**.**weights**[**1**]**

fig **=**plt**.**figure**(**figsize**=(**8**,**10**))**

axes1 **=** fig**.**add\_subplot**(**211**)**

axes2 **=** fig**.**add\_subplot**(**212**)**

# Plot train dataset

axes1**.**scatter**(**Xtrain**[:,**0**],** Xtrain**[:,**1**],** c **=** color1**)**

# Plot hyperplane

axes1**.**plot**(**x**,**y**,**'y'**,**label**=**"Trained classifier"**)**

axes1**.**legend**()**

axes1**.**set\_title**(**"Train Dataset"**)**

# Plot test dataset

axes2**.**scatter**(**Xtest**[:,**0**],** Xtest**[:,**1**],** c **=** color2**)**

# Plot hyperplane

axes2**.**plot**(**x**,**y**,**'y'**,**label**=**" Trained classifier"**)**

axes2**.**legend**()**

axes2**.**set\_title**(**"Test Dataset"**)**

fig**.**tight\_layout**()**

plt**.**show**()**

#----------------------------------------part 3-----------------------------------------

#Extra Credits

# load data3 requires Internet connection to load data from online database

**def** load\_data2**():**

url **=**'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'

data **=** pd**.**read\_csv**(**url**,**header **=** **None)**

**print(**data**)**

# make the dataset linearly separable

data**[**4**]** **=** np**.**where**(**data**.**iloc**[:,** **-**1**]==**'Iris-setosa'**,** **-**1**,** 1**)**

**print(**data**)**

#convert to numpy ndarray

data **=** data**.**to\_numpy**().**astype**(**'float32'**)**

**return** data

data **=** load\_data2**()**

# Split train test data

X\_train**,** X\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**data**[:,:-**1**],**data**[:,-**1**],**test\_size**=**0.33**,** random\_state**=**42**)**

#Train the model with all 4 features

model1 **=** Perceptron**(**X\_train**.**shape**[**1**],** 0.001**)**

model1**.**train**(**X\_train**,**y\_train**)**

model1**.**test**(**X\_test**,**y\_test**)**

# # we can not plot 4 feature in 2d plane. Let's use only 2 feaute and plot it.

# used only first 2 features/attributes of the data set

X\_train**,** X\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**data**[:,:-**3**],**data**[:,-**1**],**test\_size**=**0.33**,** random\_state**=**42**)**

# build, tain, test perceptron

model2 **=** Perceptron**(**X\_train**.**shape**[**1**],** 0.001**)**

model2**.**train**(**X\_train**,**y\_train**)**

model2**.**test**(**X\_test**,**y\_test**)**

# # Plot the datasets and hyperplane

color1 **=** **[**'red' **if** c **==** 1 **else** 'blue' **for** c **in** y\_train**]**

color2 **=** **[**'red' **if** c **==** 1 **else** 'blue' **for** c **in** y\_test**]**

# Create Hyperplane

slope **=** **-**model2**.**weights**[**0**]** **/** model2**.**weights**[**1**]**

x **=** np**.**linspace**(**4**,**8**)**

y **=** slope **\*** x **-** **(**model2**.**bias**[**0**])** **/** model2**.**weights**[**1**]**

fig **=**plt**.**figure**(**figsize**=(**8**,**10**))**

axes1 **=** fig**.**add\_subplot**(**211**)**

axes2 **=** fig**.**add\_subplot**(**212**)**

# Plot train dataset

axes1**.**scatter**(**X\_train**[:,**0**],** X\_train**[:,**1**],** c **=** color1**)**

# Plot hyperplane

axes1**.**plot**(**x**,**y**,**'y'**,**label**=**"Trained classifier"**)**

axes1**.**legend**()**

axes1**.**set\_title**(**"Train Dataset"**)**

# Plot test dataset

axes2**.**scatter**(**X\_test**[:,**0**],** X\_test**[:,**1**],** c **=** color2**)**

# Plot hyperplane

axes2**.**plot**(**x**,**y**,**'y'**,**label**=**" Trained classifier"**)**

axes2**.**legend**()**

axes2**.**set\_title**(**"Test Dataset"**)**

fig**.**tight\_layout**()**

plt**.**show**()**

#---------------------------------End of Program-------------------------------