CS57800: Statistical Machine Learning

Homework 2

Vineeth Ravi

October 3, 2018

PU ID : 0030019456

EMAIL ID : ravi24@purdue.edu

1 Introduction

The objective of this assignment is to experiment on the Perceptron algorithm. We will be classifying the images of handwritten digits using two variations of Perceptron. We will train 10 perceptrons that will combinedly learn to classify the handwritten digits. Each Perceptron will have 785 inputs and one output. Each Perceptrons target is one of the 10 digits.

For example, lets index the 10 Perceptrons from 0 to 9. Say we are training for an image of the digit 3. Then only the Perceptron with index 3 will have an expected output 1, all the other Perceptrons will have expected output 0. While predicting the label of an image, the Perceptron which gives the highest value of w.x will be the winner, i.e. the predicted label of the image will be the index of that Perceptron.

We implement this assignment from scratch in Python 2.7 and we do not use any already implemented models like those in the scikit-learn library. We include the plots and describe the results and procedure used below :-

2 Vanilla Perceptron

The Vanilla Perceptron Algorithm has been implemented and the python code file has been submitted along with the report. There are two types of initialization of weights available. We can initialize the weights of the perceptrons to 0 or we can initialize them to a value between [-1,1]. In the python file, by default I have initialized the weights to 0 due to higher F1 Scores, but this can be modified, by just commenting out a line, and using the other line available in the code. The python file, has been well commented, to highlight this aspect. A comparison of these graphs has been included in the Report. The approach used for training the perceptron is the ONE vs

ALL approach. The python code file has been commented to explain the algorithm used with more detail than the pseudo-code included in the report.

Vanilla Perceptron Algorithm :-

Pesudo Code for 10 Vanilla Perceptrons Algorithm with Random Initialization :-

Perceptron Train()

Result: Compute Test F1 Scores and Train F1 Scores and return weights
initialization;
$[w, b] \leftarrow [random(-1, 1)]$:- For all 10 perceptrons' weight vectors
for *k=1,2, ... ,Nepoch* **do**
  Shuffle data every epoch
  for *i=1,2, ... ,Ntrainingsize* **do**
    $(x, y) \in D-$ Training examples
    $y_p \leftarrow argmax(w^T x)$:- w For all 10 perceptrons' weight vectors and instance x
    if $y_p \neq ytrue$ **then**
      Increment weights of perceptron with index = ytrue
      $w[y_p] \leftarrow w[y_p] + (Nlearnrate) * x$;
    else
      continue;
    end
    for *j=1,2, ... ,10* **do**
      Compute $(w^T x)$ and find those perceptrons with values ≥ 0
      if $(w^T x >= 0)$ *and* $(j \neq y_t)$ **then**
        Decrement weights of perceptrons with index $\neq$ ytrue
        $w[y] \leftarrow w[y] + (-Nlearnrate) * x$;
      else
        continue;
      end
    end
  end
end

return w, F1 Score

Perceptron Test()
For Perceptron Test data
Compute
$y_p \leftarrow argmax(w^T x)$
return y_p , F1 Score

Algorithm 1: 10 Vanilla Perceptrons

The example command used to run this python file is :-

"python vanilla_perceptron.py 1000 20 0.001 data"

data - is the name of the folder, where all the 4 input files are stored in compressed .gz format.

The default value of the hyperparameters for generating all of the learning curves are :-
Number of training examples - 10000
Number of Epochs - 50
Learning Rate - 0.001

The below graphs or learning curves are plotted for Random Initialization of weights 'w' :-

Effect of the size of training set in learning :-

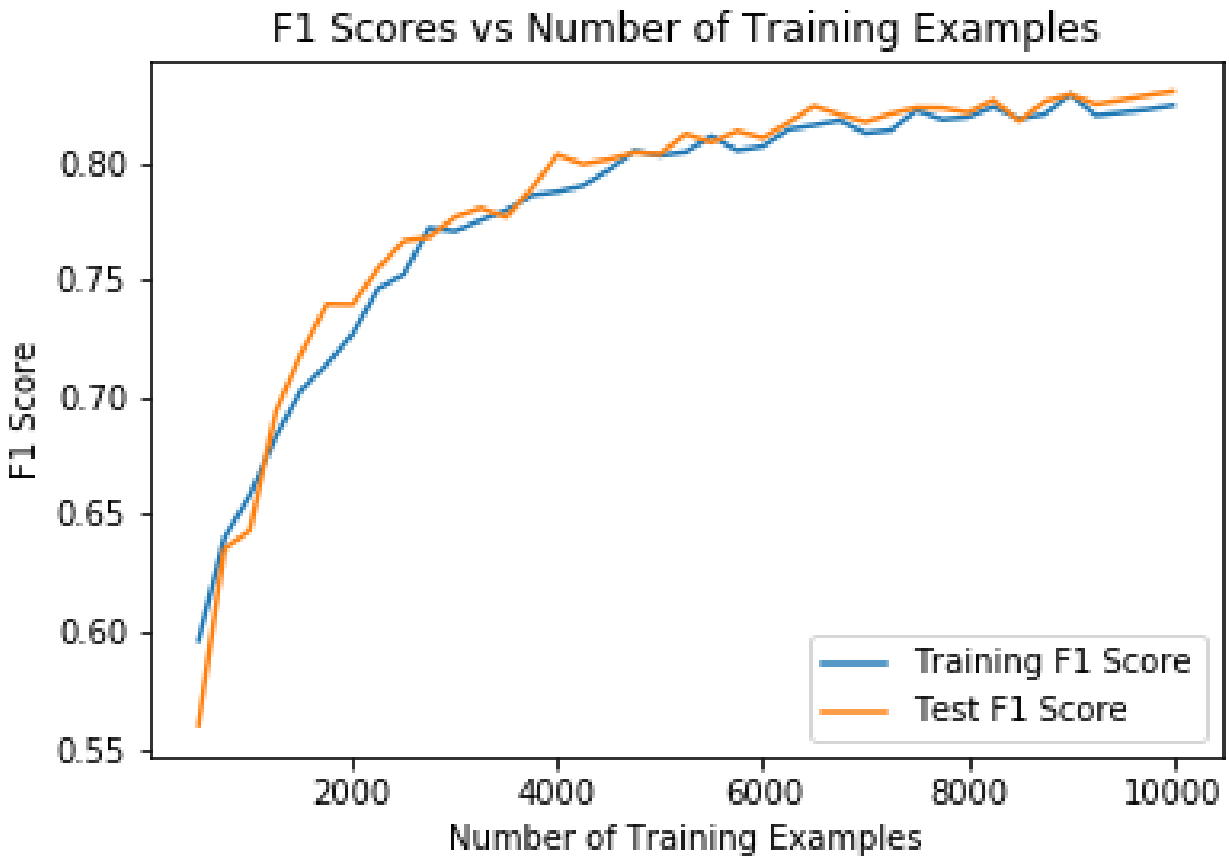

Figure 1: F1 Scores vs Number of Training Examples

Effect of the number of epochs in learning :-

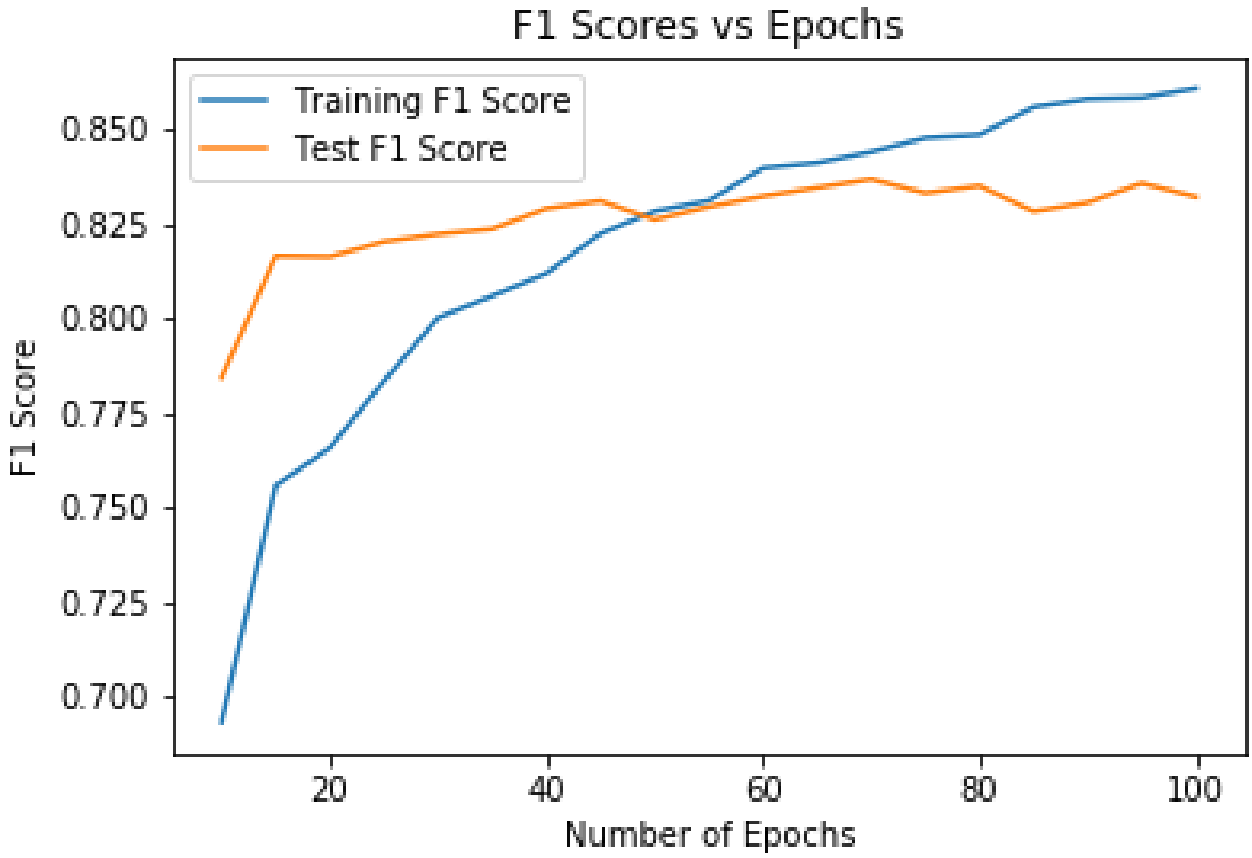

Figure 2: F1 Scores vs Number of Epochs

Effect of learning rate :-

Figure 3: F1 Scores vs Learning Rate

Observations on the learning curves:-

Effect of the size of training set in learning :-
The F1 Score increases as we increase the number of Training Examples Size for both the Training as well as the Test data. We observe the F1 Scores continuously keep increasing as we increase the training data size. This is expected, because as we encounter more training examples, the perceptron is able to learn better from the training data as well as generalize better for the test data. So, the optimal hyper-parameter of training size is 10000.

Effect of the number of epochs in learning :-
The F1 Score initially increases for both the training as well as the test data as we increase the number of epochs. As we keep increasing the number of epochs further from 50, we observe that the training F1 Score increases, but the test F1 score, mostly remains the same (maybe marginally decreases later). This is because of overfitting, to the existing training data, we are not able to gen-

eralize better for the test data as we increase the number of epochs further. Initally the test F1 score is higher than the train F1 Score, till the point of 50 epochs, from where the train F1 Score is higher than the test F1 Score. So, I would suggest the optimal hyper-parameter of Number of Epochs is 50.

Effect of learning rate :-
The F1 Score increases as we increase the learning rate for both the Train and Test F1 Score. The F1 score also finally saturates for learning rates of 0.01 and 0.1 for both the Test and Train data. The training F1 Score overtakes the Test F1 score at the learning rate of 0.001, Beyond which there is an increase in F1 Score till 0.01, after which the F1 Score saturates. This means, that we are no longer learning anything new from the training data, and there is no better generalization which we can use to make better predictions for the test data. So, the optimal hyper-parameter for the learning rate can be 0.001 or 0.01.

3 Average Perceptron

The Average Perceptron Algorithm has been implemented and the python code file has been submitted along with the report. There are two types of initialization of weights available. We can initialize the weights of the perceptrons to 0 or we can initialize them to a value between [-1,1]. In the python file, by default I have initialized the weights to 0 due to higher F1 Scores, but this can be modified, by just commenting out a line, and using the other line available in the code. The python file, has been well commented, to highlight this aspect. A comparison of these graphs has been included in the Report.

In the Average Perceptron Algorithm, we update the average weights of 'w' in each iteration of the Vanilla Perceptron Algorithm. We then use this averaged weight vector for prediction of the test data. The approach used for training the perceptron is the ONE vs ALL approach. We observe that the TEST F1 Scores for the average Perceptron are significantly higher when compared to the Test F1 Scores in the vanilla perceptron, when we initialize the weights to zeros, instead of random values between (-1,1), Though the learning curves , for when we initialize the weights to zero, are not included in this report, I had generated them to make these observations, that we get higher TEST F1 Scores for 0 initialization of weights. To verify this, you can run both the python code files for the vanilla and average perceptron submitted along with this report, You can observe the difference in F1 Score for Test data (Average Perceptron perfroms better than Vanilla Perceptron).

The learning curves for when we initialize the weights to zero, can be generated from the python code, by appropriately commenting some lines as described in the instructions and comments in the python code. This applies to both the Vanilla Perceptron as well as the Average Perceptron. The python code file has been commented to explain this algorithm used with more detail than the pseudo-code included in the report.

Average Perceptron Algorithm :-

Pesudo Code for 10 Averaged Perceptrons Algorithm with Random Initialization :-

Perceptron Train()

Result: Compute Test F1 Scores and Train F1 Scores and return weights

initialization;

$[w, b] \leftarrow [random(-1, 1)]$:- For all 10 perceptrons' weight vectors

$a \leftarrow 0$:- Average weight vector For all 10 perceptrons'

for *k=1,2, ... ,Nepoch* **do**

  Shuffle data every epoch

  for *i=1,2, ... ,Ntrainingsize* **do**

    $(x, y) \in D-$ Training examples

      $y_p \leftarrow argmax(w^T x)$:- w For all 10 perceptrons' weight vectors and instance x

      if $y_p \neq ytrue$ **then**

        Increment weights of perceptron with index = ytrue

        $w[y_p] \leftarrow w[y_p] + (Nlearnrate) * x$;

      else

        continue;

      end

      for *j=1,2, ... ,10* **do**

        Compute $(w^T x)$ and find those perceptrons with values ≥ 0

        if $(w^T x >= 0)$ *and* $(j \neq y_t)$ **then**

          Decrement weights of perceptrons with index $\neq$ ytrue

          $w[y] \leftarrow w[y] + (-Nlearnrate) * x$;

        else

          continue;

        end

      end

      $a \leftarrow a + w$

  end

end

return a, F1 Score

Perceptron Test()

For Perceptron Test data

Compute

$y_p \leftarrow argmax(w^T x)$

return y_p , F1 Score

Algorithm 2: 10 Average Perceptrons

The example command used to run this python file is :-

"python average_perceptron.py 1000 20 0.001 data"

data - is the name of the folder, where all the 4 input files are stored in compressed .gz format.

The default value of the hyperparameters for generating all of the learning curves are :-
Number of training examples - 10000
Number of Epochs - 50
Learning Rate - 0.001

The below graphs or learning curves are plotted for Random Initialization of weights 'w' :-

Effect of the size of training set in learning :-

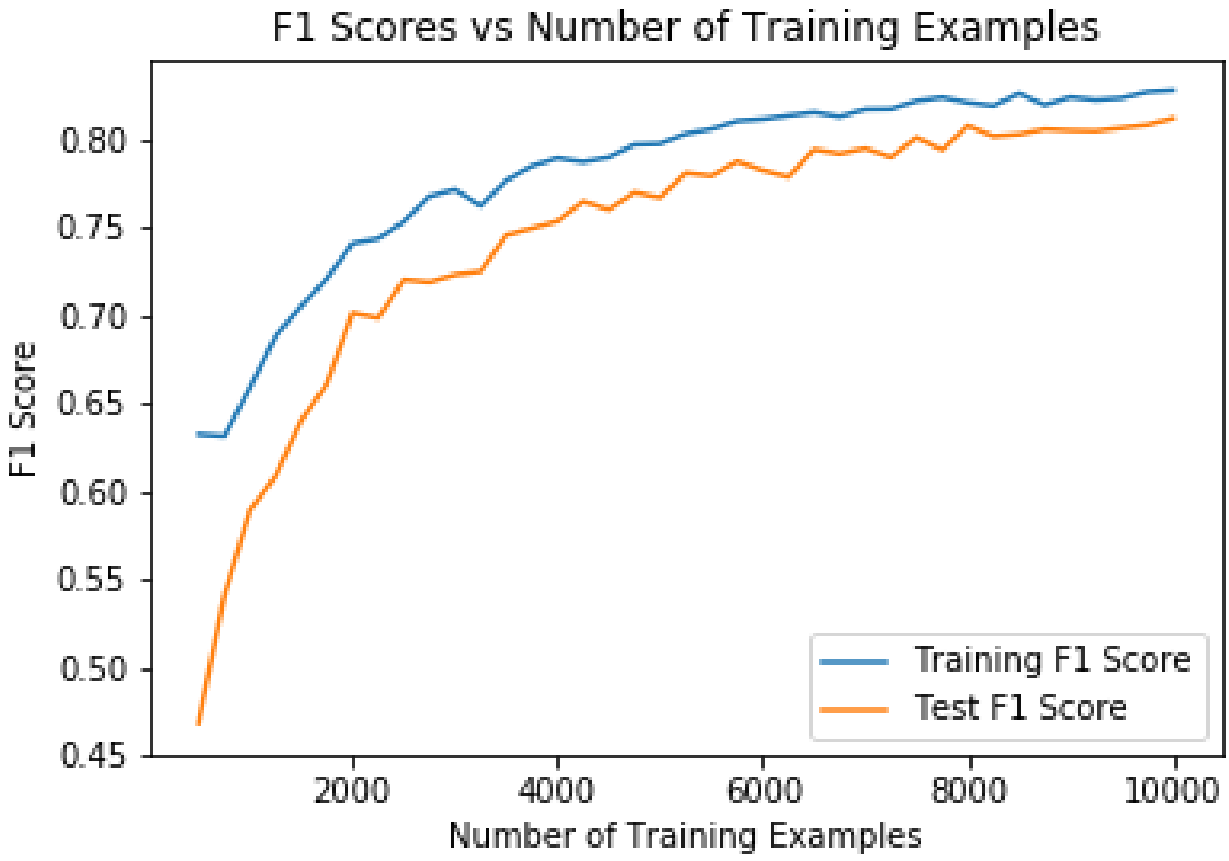

Figure 4: F1 Scores vs Number of Training Examples

Effect of the number of epochs in learning :-

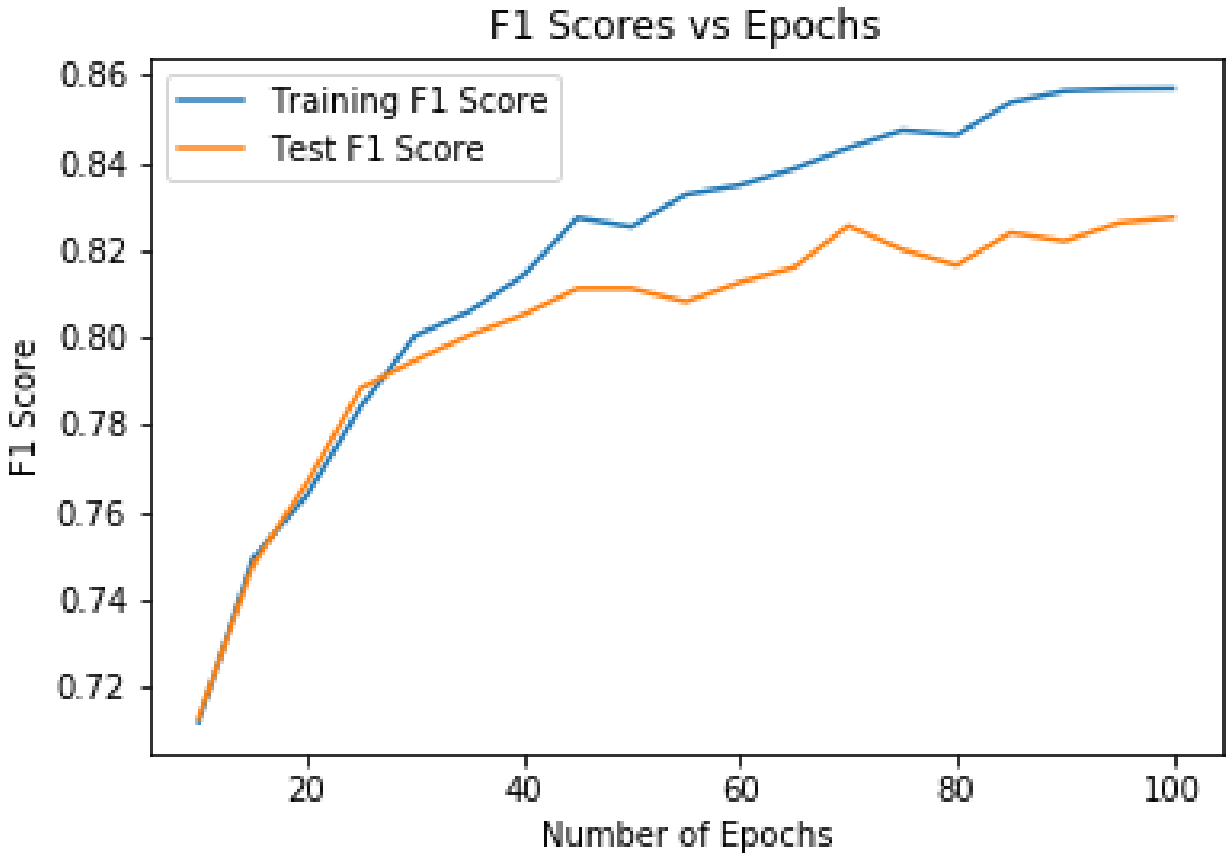

Figure 5: F1 Scores vs Number of Epochs

Effect of learning rate :-

Figure 6: F1 Scores vs Learning Rate

Observations on the learning curves:-

Effect of the size of training set in learning :-
When, we implement the Average Perceptron, the first thing we can observe is that there are higher Test F1 Scores, when compared to the vanilla Perceptron, (Run both the python codes, with the same set of hyper-parameters to observe the difference). Similar to the Vanilla Perceptron, we observe that, as we increase the number of Training Examples, the Training F1 Scores and Test F1 Score increases continuously. This is expected, because as we encounter more training examples, the perceptron is able to learn better from the training data as well as generalize better for the test data. Hence, the optimal hyper-parameter for the number of training examples is 10000.

Effect of the number of epochs in learning :-
As we increase the number of epochs, both the Training F1 Score and Test F1 Scores increase. Till 50 Epochs, both the Training and Test F1 Score increase very fast, but after that the Test F1 Score does not increase by much, when compared to the Training F1 Score, as observed from the graph. For tuning the hyper-parameter for the Number of Epochs, it is better we choose 70, Since there is not significant increase in F1 Score from 70 to 100 and if we increase the number of epochs, we are prone to over-fitting, as suggested by the trend or results of F1 Scores obtained between 70 to 100 epochs in the Learning curves. Hence, the optimal hyper-parameter for the number of epochs is 70.

Effect of learning rate :-
Similar to the trends observed for the vanilla perceptron, we observe that as we increase the learning rate, the Test and Training F1 Scores for the average perceptron also increase, and finally do not have a significant increase in F1 Scores as we change the learning rate from 0.01 to 0.1. The train F1 Scores also have a very minimal (small but marginal decrease) beyond a learning rate of 0.01. Since the Test and Train F1 Scores are excellent as well as similar to each other when the learning rate is equal to 0.1, The optimal hyper-parameter for the learning rate can be 0.1.

4 Winnow

The Winnow Algorithm has been implemented and the python code file has been submitted along with the report. There are two types of initialization of weights available. We can initialize the weights of the perceptrons to 1 or we can initialize them to a value between [0,1]. In the python file, by default I have initialized the weights to 1 due to higher F1 Scores, but this can be modified, by just commenting out a line, and using the other line available in the code. The python file, has been well commented, to highlight this aspect. The Pesudo-Code include in the report is for the Average Winnow.

Here I have implemented the average Winnow, since it had higher F1 Scores, when compared to the normal winnow algorithm. An important hyper-parameter to note down in the Winnow Algorithm is the "threshold" ,We have two options for tuning the threshold hyper-parameter, We can dynamically tune it to :-

$$threshold = \frac{np.max(array)}{LearnRate}$$

or we can statically fix it to (N/4 ,or N/2 or 3N/4, or N) any of these values.
FINALLY I have chosen $threshold$ = N/2, the learning curves for this hyper-parameter is not included in this report, Since they have similar F1 Scores results. This value of threshold can me modified in the Python code file, to generate the learning curves for that particular value of threshold:-

threshold = 784/2 = 392 (PERFROMS WAY BETTER , Higher F1 Scores)

This threshold is used for the estimating the hyperplane or decision boundaries margin (similar to regularization) during the training for the winnow algorithm where I have implemented the ONE

vs ALL approach. The python code file has been commented to explain the algorithm used with more detail than the pseudo-code included in the report.

Winnnow Algorithm :-

Pesudo Code for 10 Average Winnow Algorithm with Random Initialization :-

Winnow Train()

Result: Compute Test F1 Scores and Train F1 Scores and return weights

initialization;

$[w, b] \leftarrow [1]$:- For all 10 perceptrons' weight vectors

$a \leftarrow 0$:- Average weight vector For all 10 perceptrons'

for *k=1,2, ... ,Nepoch* **do**

  Shuffle data every epoch

  for *i=1,2, ... ,Ntrainingsize* **do**

    $(x, y) \in D-$ Training examples

      $y_p \leftarrow argmax(w^T x)$:- w For all 10 perceptrons' weight vectors and instance x

      if $y_p \neq ytrue$ **then**

        Increment weights of perceptron with index = ytrue

        $w[y_p]_i \leftarrow w[y_p]_i * (Nlearnrate), (x_i = 1)$;

      else

        continue;

      end

      for *j=1,2, ... ,10* **do**

        Compute $(w^T x)$ and find those perceptrons with values $\geq$ threshold

        if $(w^T x >= (thresholdvalue)\ and\ (j \neq y_t)$ **then**

          Decrement weights of perceptrons with index $\neq$ ytrue

          $w[y_p]_i \leftarrow w[y_p]_i/(Nlearnrate), (x_i = 1)$;

        else

          continue;

        end

      end

      $a \leftarrow a + w$

  end

end

return a, F1 Score

Perceptron Test()

For Perceptron Test data

Compute

$y_p \leftarrow argmax(w^T x)$

return y_p , F1 Score

Algorithm 3: 10 Winnows Algorithm

Command Line arguments required to run this file :-

"python winnow.py 1000 20 2 data"

data - is the name of the folder, where all the 4 input files are stored in compressed .gz format.

Hyper-parameters used for tuning and creating the learning curves :-
1) Number of Training Examples Size , 2) Number of Epochs , 3) Learning Rate , 4) Threshold Value

Default Values of the Hyper-parameters :-

Number of training examples - 10000
Number of Epochs - 50
Learning Rate - 2
$threshold = 784/2 = 392$

The below graphs or learning curves are plotted for Initialization of weights 'w' = '1' and For NORMAL WINNOW ALGORITHM (NOT AVERAGE WINNOW):-

Effect of the size of training set in learning :-

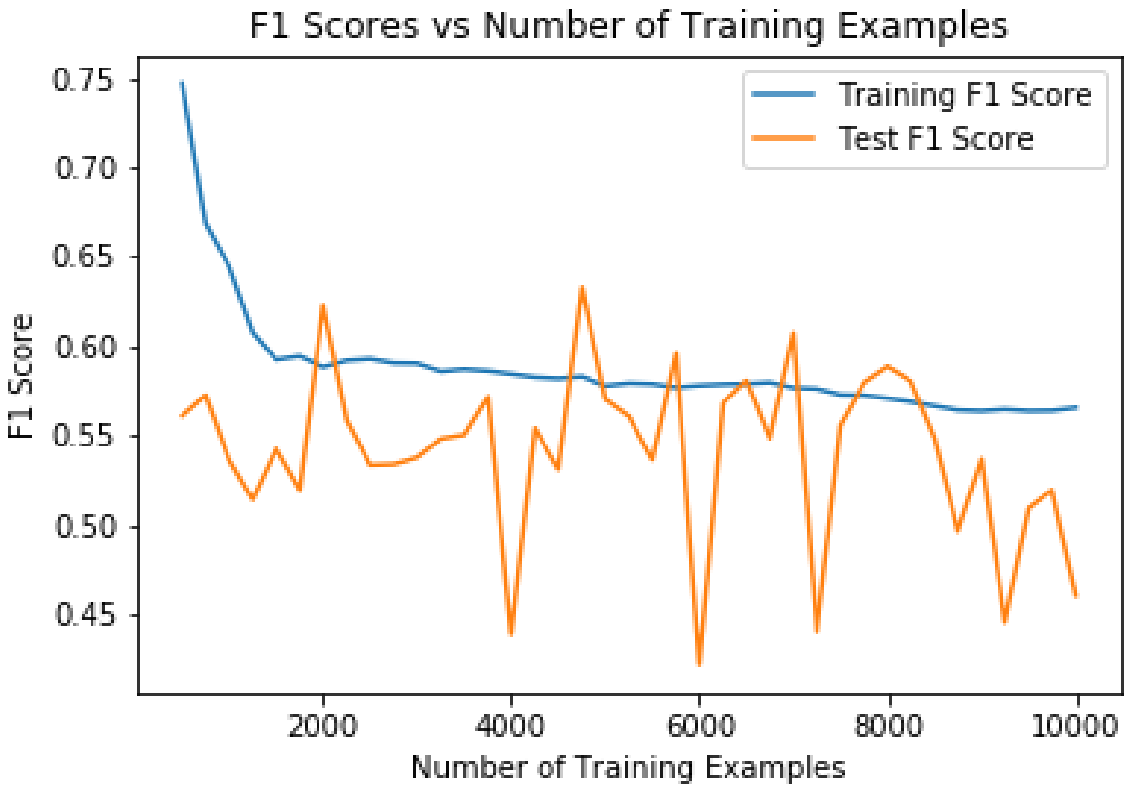

Figure 7: F1 Scores vs Number of Training Examples

Effect of the number of epochs in learning :-

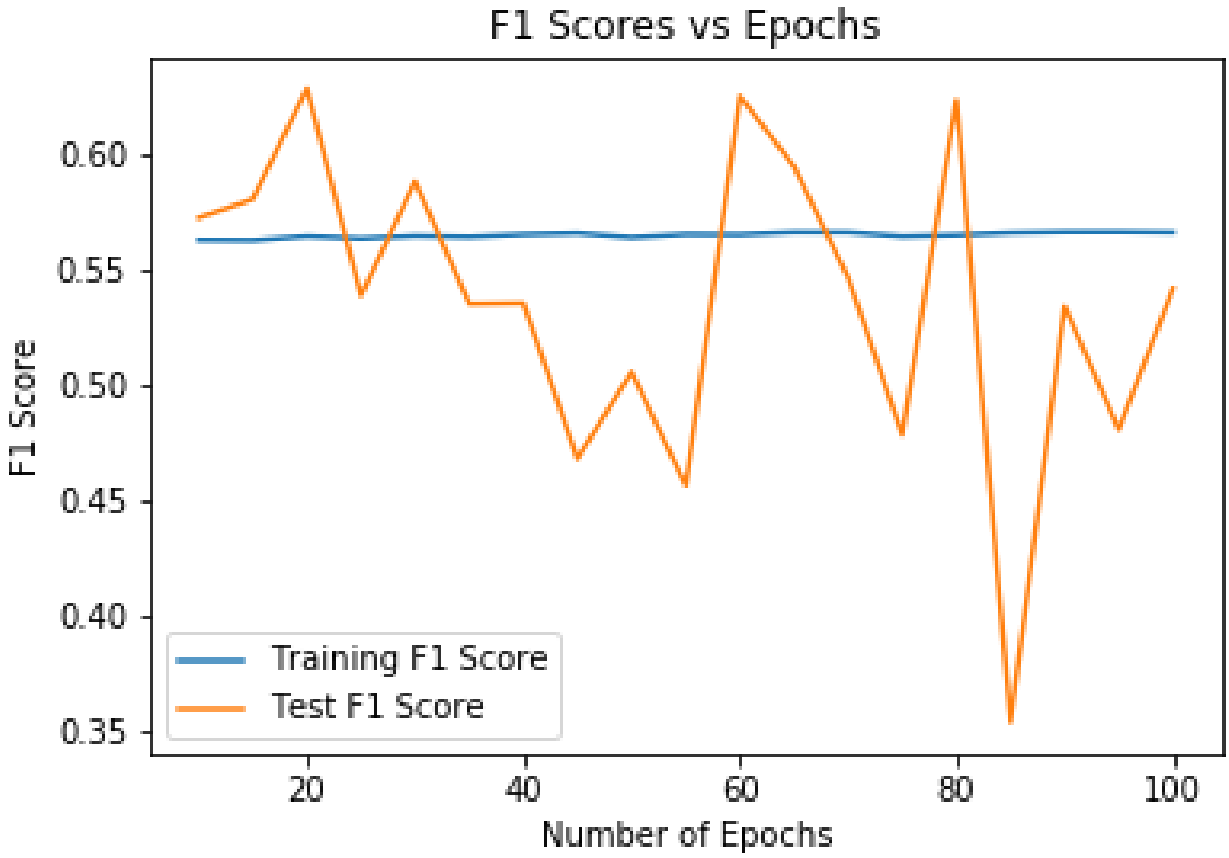

Figure 8: F1 Scores vs Number of Epochs

Effect of learning rate :-

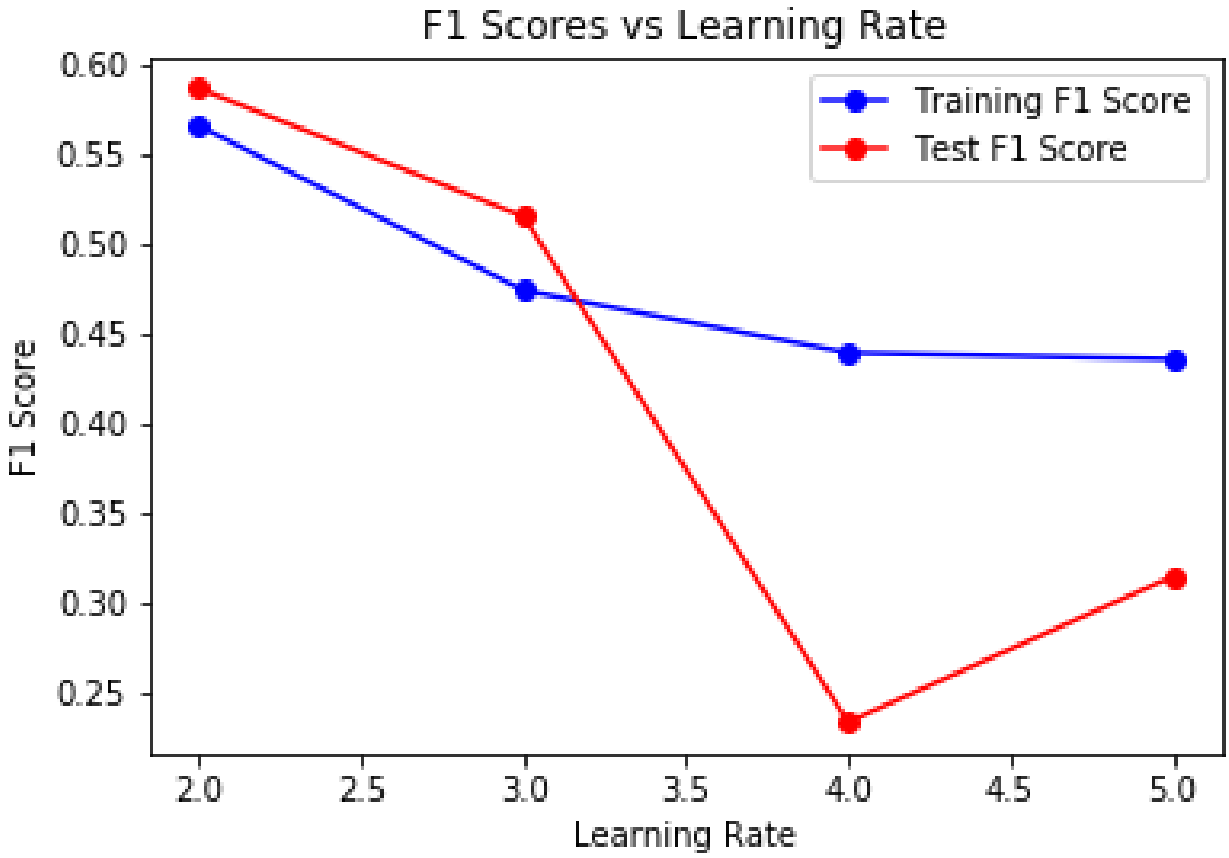

Figure 9: F1 Scores vs Learning Rate

Observations on the learning curves:-

Effect of the size of training set in learning :-

As we increase the number of training examples, we observe that the Training F1 Score decreases and the TEST F1 Scores fluctuates a lot between 0.5 and 0.6. The winnow algorithm does not perform well when compared to perceptron. The Winnow algorithm has multiplicative weight updates, instead of additive updates like the Perceptron algorithm , So it is not able to fit and learn well from the training data properly and generalize well for the test data as we increase the size of Training Examples. Hence the F1 Score is lesser compared to the Perceptron, because of more error (more mistakes). It is very difficult to choose an optimal hyper-parameter, so it can be default 10000, to compare performance with the Perceptrons' algorithm.

Effect of the number of epochs in learning :-
For the Winnow Algorithm, as we increase the number of epochs, the Training F1 Score remains almost the same around 0.56, where as the TEST F1 Score keeps fluctuating between 0.5 and 0.6. The Winnow algorithm has multiplicative weight updates, instead of additive updates like the Perceptron algorithm , So it is not able to fit and learn well from the training data properly and generalize well for the test data as we increase the Number of Epochs. Hence the F1 Score is lesser compared to the Perceptron, because of more error (more mistakes). It is again difficult to choose an optimal hyper-parameter, so by default it can be 50, to compare perfromance with the Perceptrons' algorithm.

Effect of learning rate :-
The learning rate of the Winnow Algorithm is the multiplicative weight update '2' - Nlearn present in the pseudo-code. This is changed from 2,3,4,5 and the F1 Scores at plotted. We observe that as we increase the learning rate to higher values, the F1 Score decreases for both the Test and Training data. This is because, we are learning too fast, and we are under-fitting the data, as we increase the learning rate. Since the Winnow algorithm has multiplicative weight updates, instead of additive updates like the Perceptron algorithm , So it is not able to fit and learn well from the training data properly and generalize well for the test data as we increase the learning rate. Hence the F1 Score is lesser compared to the Perceptron, because of more error (more mistakes). The optimal hyper-parameter value of the learning rate is 2, Since at this learning rate value, the Training and Test F1 Scores are the highest.

5 Answers to Open Ended Questions

1) For the same set of Hyper-Parameters, The Average Perceptron works the best and gives the best Test F1 Scores, when we initialize the weights with zeros. The Vanilla Perceptron algorithm works better than the Winnow algorithm. The F1 Scores for Average and Vanilla Perceptron algorithm, are much higher when compared to the Winnow Algorithm. They also run faster, when compared to the Winnow Algorithm. Average Perceptron better than Vanilla Perceptron better than Winnow. So, I would use the Average Perceptron for best results.

2) The Mistake Bound model, has learning in 3 stages :- The learner gets an unlabeled example, The learner predicts its classification, The learner is told the correct label, and for such a learning Algorithm A that learns class C with a mistake bound M iff mistake(A,C) $\leq$ M. We update only when we make a mistake while training over a stream of examples and bound the number of mistakes, by making the smallest number of mistakes in the long run.

3) We can convert each negation of a variable to a new variable x', and learn over the space of 2n function variables, where we now have monotone conjunctions. Applying the Elimination algorithm, we can at the maximum make 2n mistakes, before identifying the target hypothesis, Since, we can check for the presence of each of the 2*n variables, in the monotone conjuction with

appropriate training examples. The elimination algorithm has a mistake bound of 'n' (derived in class for n variables), So applying to 2n variables, we obtain a mistake bound of 2n. Hence the mistake bound is 2*n, for Boolean Conjunctions.
Alternatively, We can use the "Winnow-Extension" Algorithm (described in the slides of Lecture 6). We now convert the problem to learn monotone functions over 2n variables. Hence, The mistake Bound is O(klog(2n)).

4) i) Yes, both the classifiers will converge, Since perceptron will always converge for a linearly separable data set. Only difference is that the time it takes for the algorithm to converge on the sorted dataset will be longer than the time it takes for convergence on the randomized dataset. (Perceptron Convergence Theorem)

ii) The Training error for both of the classifiers will eventually reach to 0, Since, we know from the Perceptron Convergence Theorem, that for a linearly seperable dataset, the perceptron will eventually converge, giving 0 Training error at the end. So, the Training Error for both of the classifiers will be 0 (zero).

6 Extra: Average WINNOW LEARNING CURVES

The below graphs or learning curves are plotted for Initialization of weights 'w'=1 and for Average WINNOW ALGORITHM :-

Effect of the size of training set in learning :-

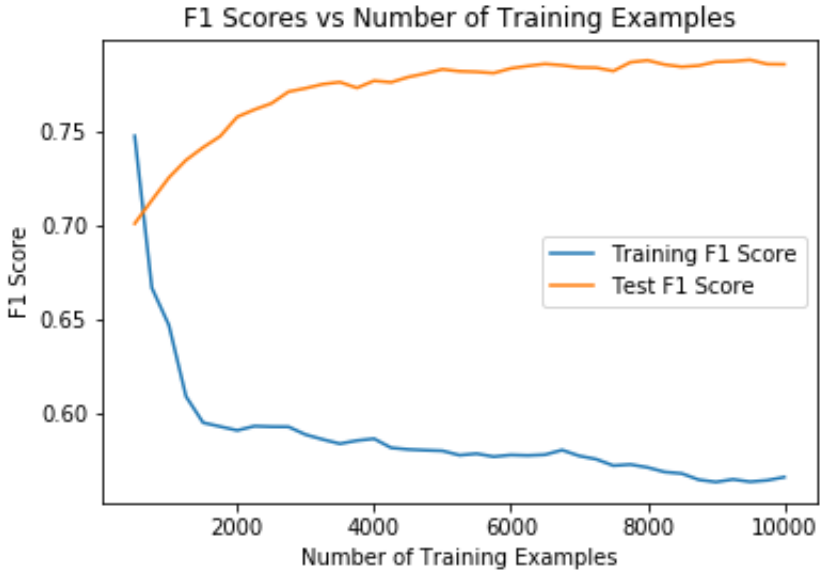

Figure 10: F1 Scores vs Number of Training Examples

Effect of the number of epochs in learning :-

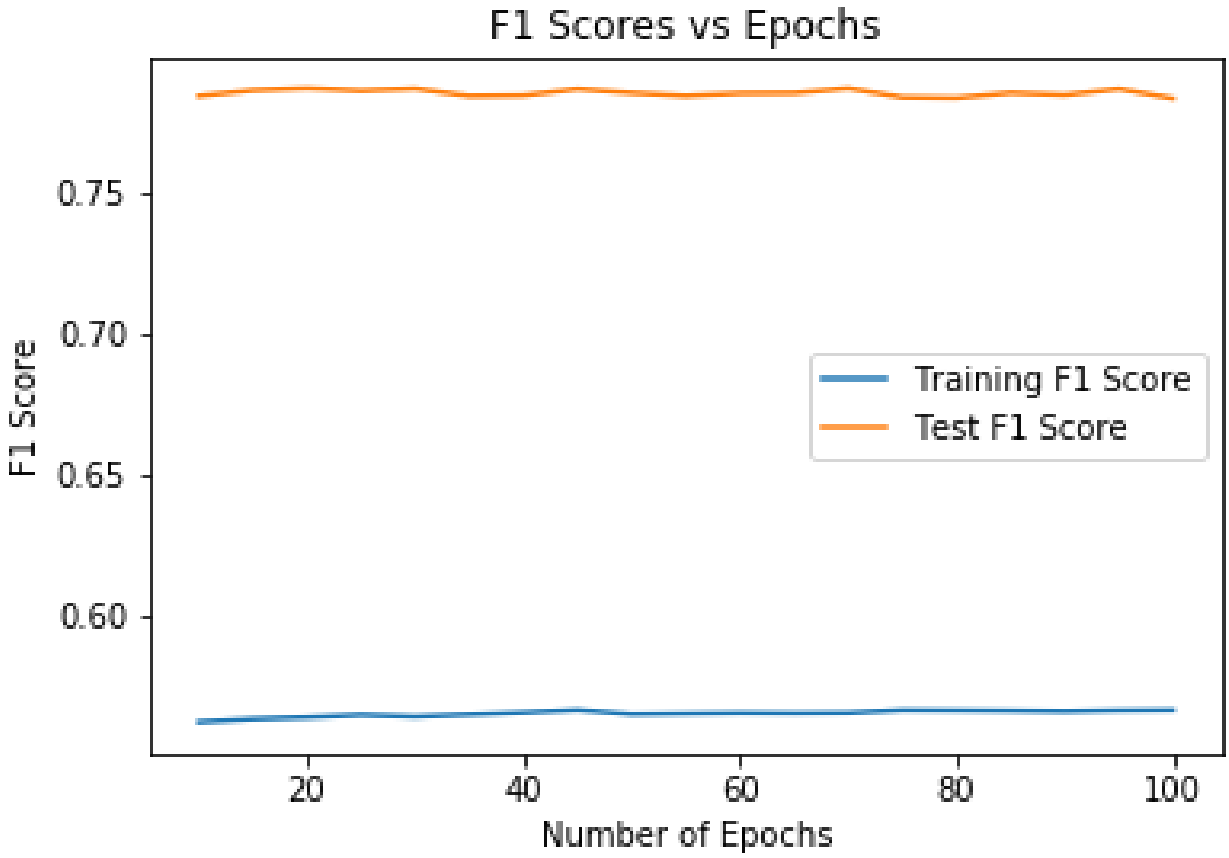

Figure 11: F1 Scores vs Number of Epochs

Effect of learning rate :-

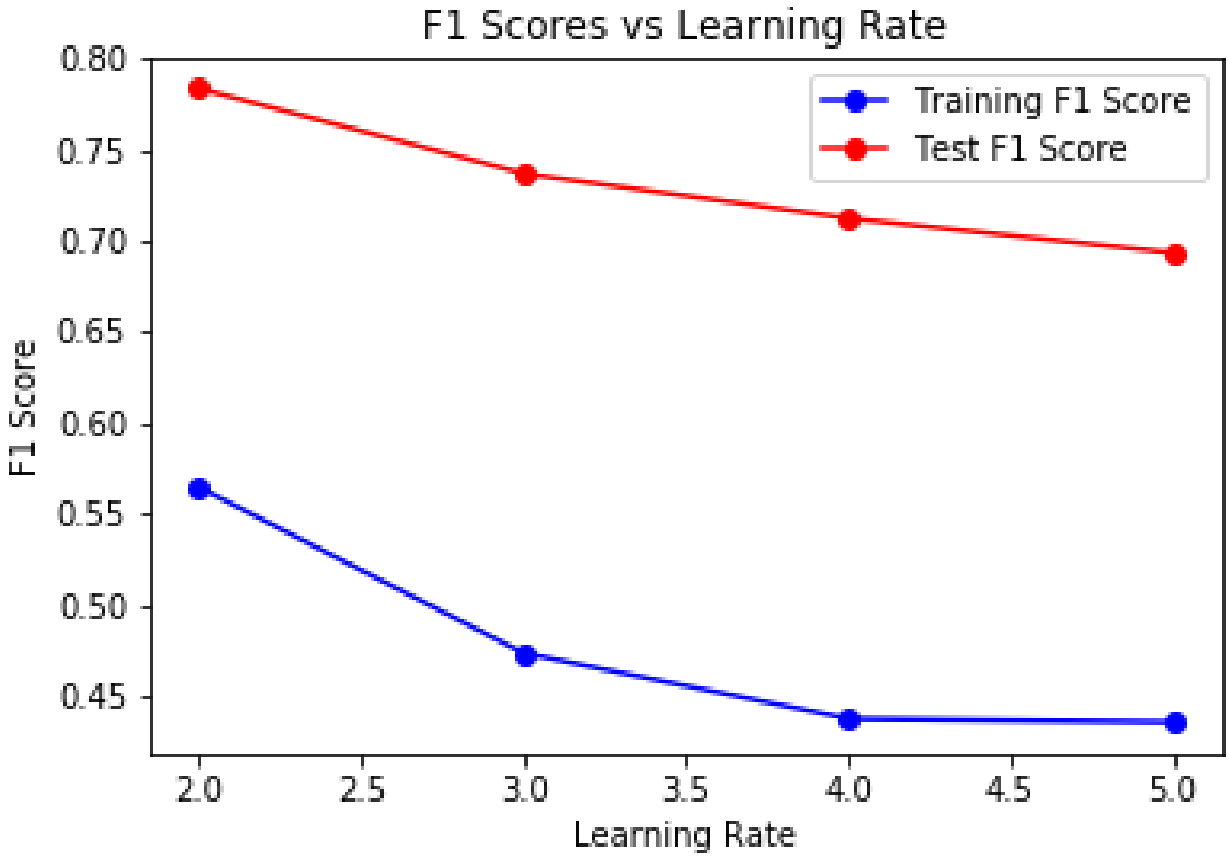

Figure 12: F1 Scores vs Learning Rate

7 Extra: Average Perceptron with Zero Initialization of weights

The below graphs or learning curves are plotted for Initialization of weights 'w'=0 and for Average Perceptron ALGORITHM :-

Effect of the size of training set in learning :-

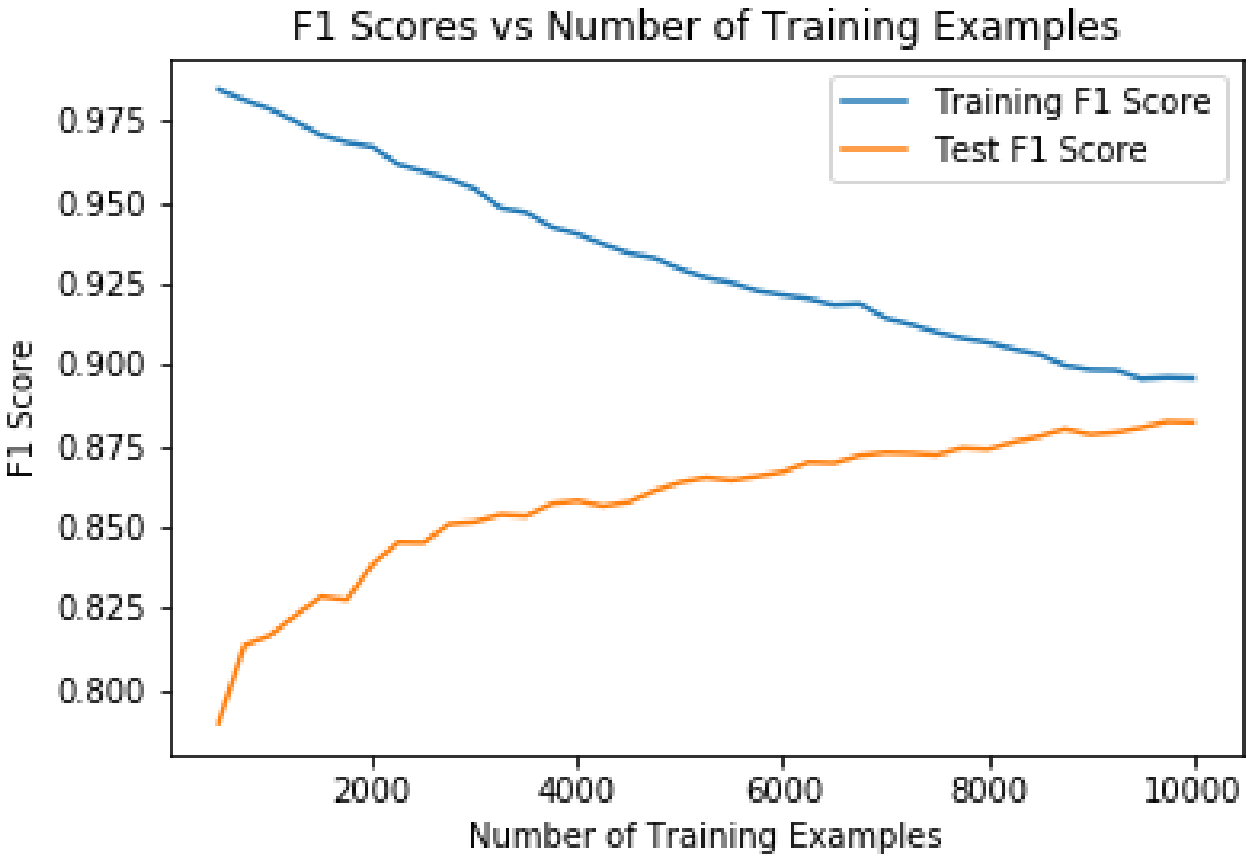

Figure 13: F1 Scores vs Number of Training Examples

Effect of the number of epochs in learning :-

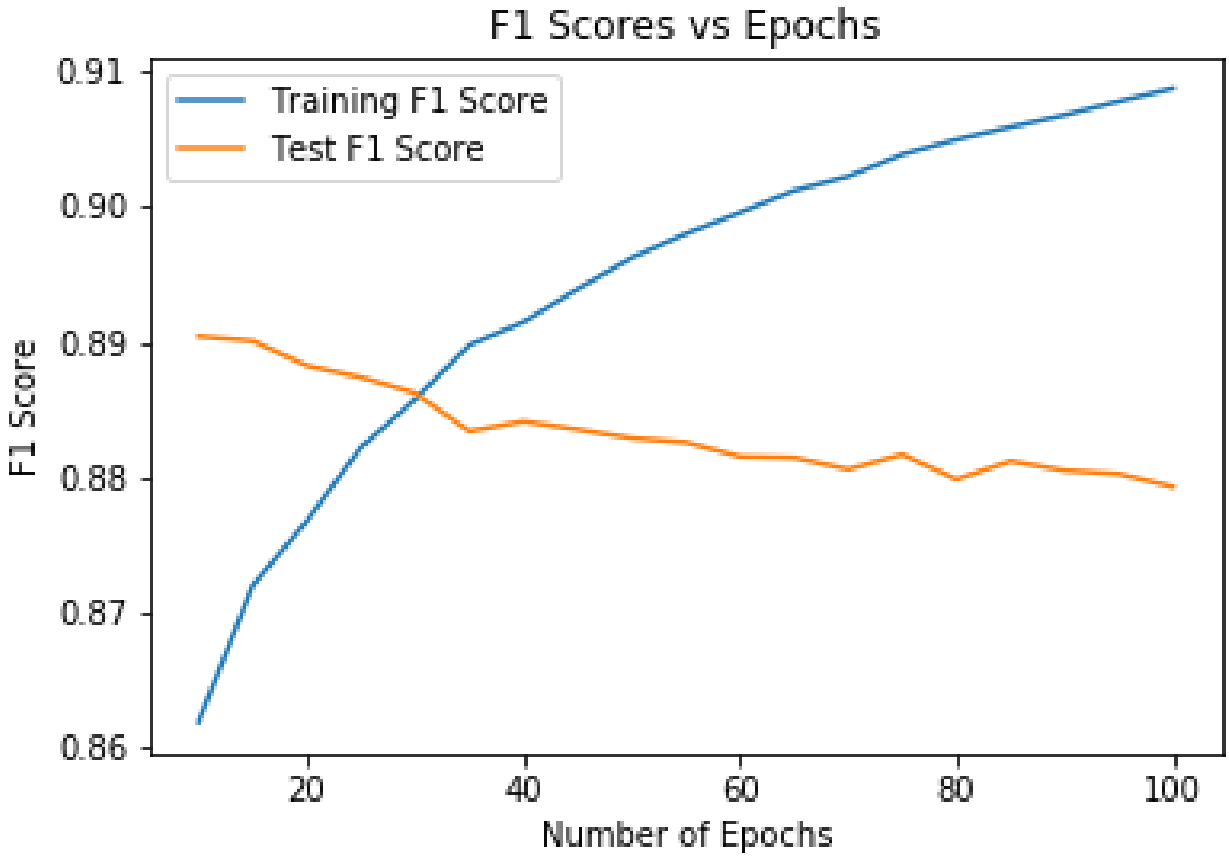

Figure 14: F1 Scores vs Number of Epochs

Effect of learning rate :-

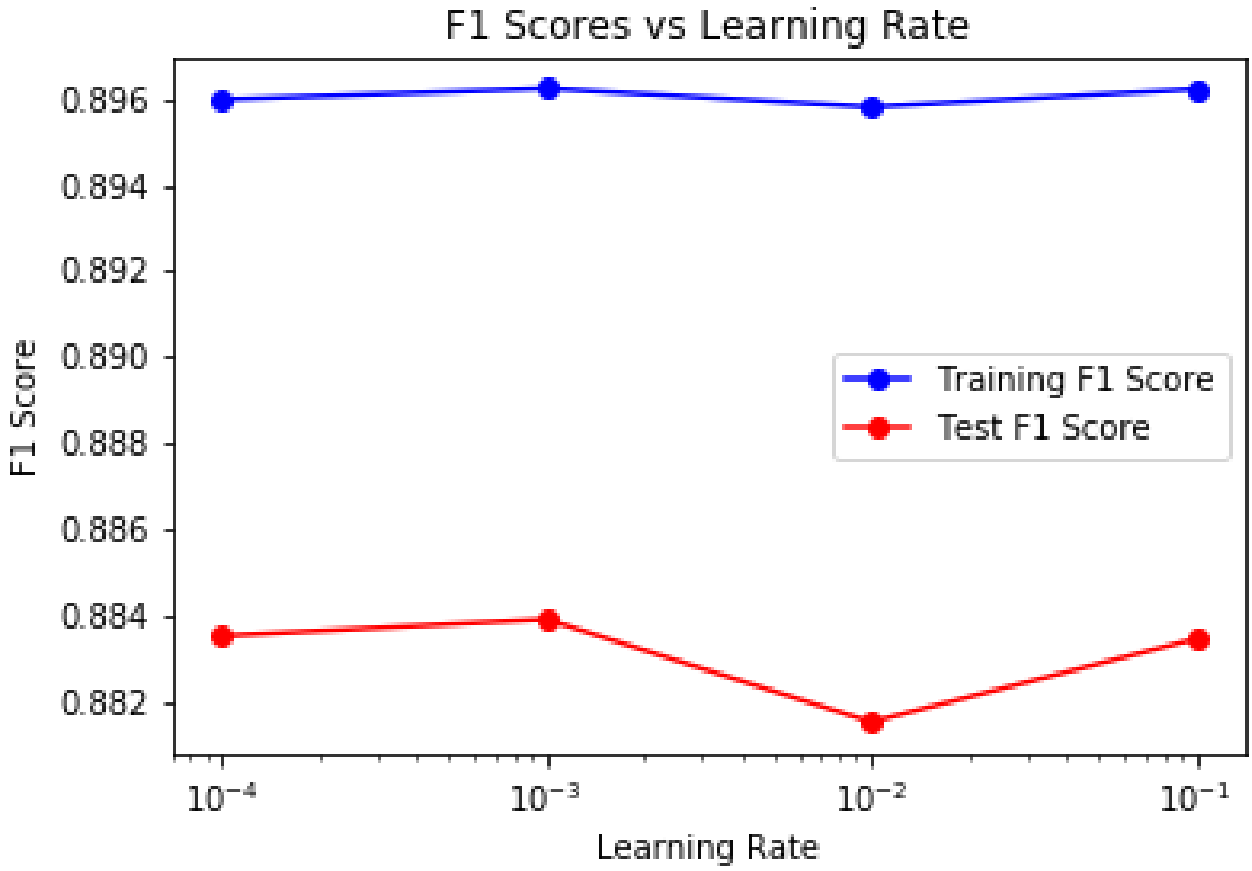

Figure 15: F1 Scores vs Learning Rate

8 Extra: Vanilla Perceptron with Zero Initialization of weights

The below graphs or learning curves are plotted for Initialization of weights 'w'=0 and for Vanilla Perceptron ALGORITHM :-

Effect of the size of training set in learning :-

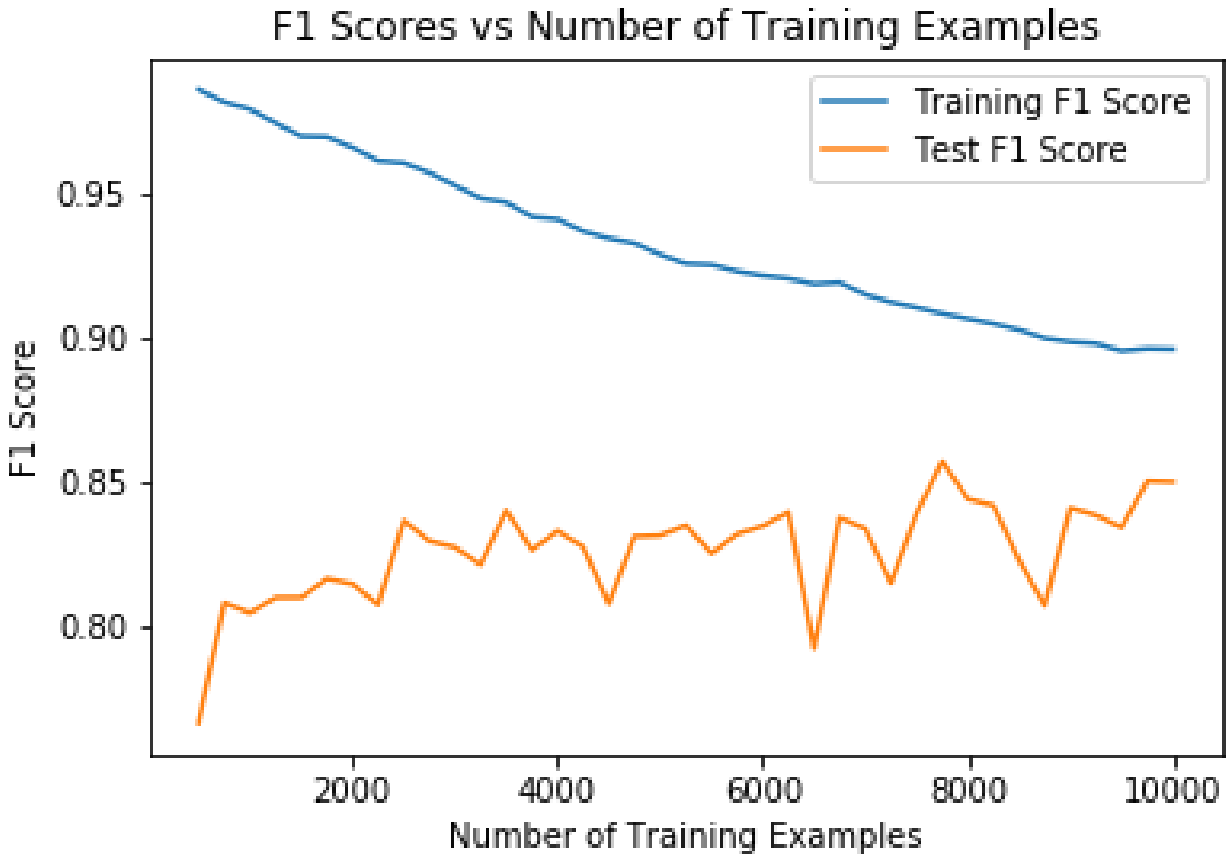

Figure 16: F1 Scores vs Number of Training Examples

Effect of the number of epochs in learning :-

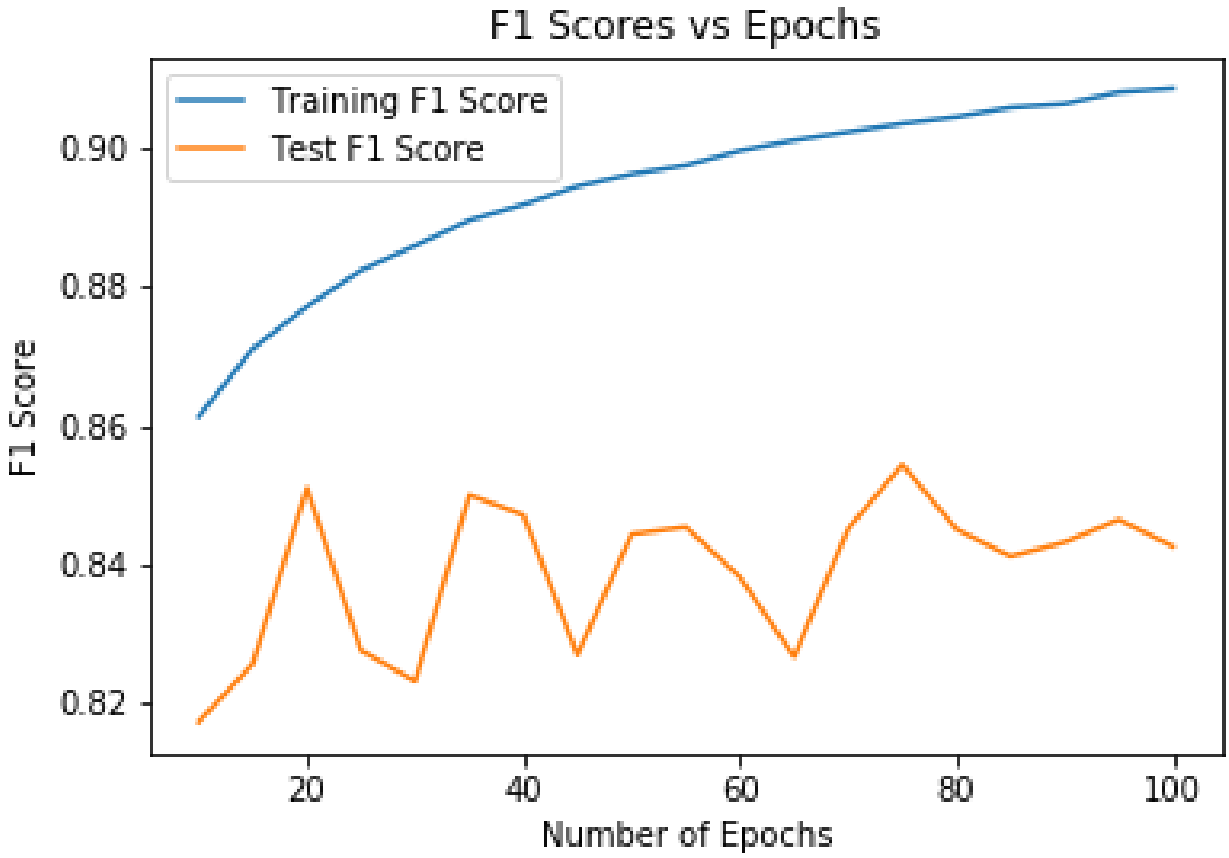

Figure 17: F1 Scores vs Number of Epochs

Effect of learning rate :-

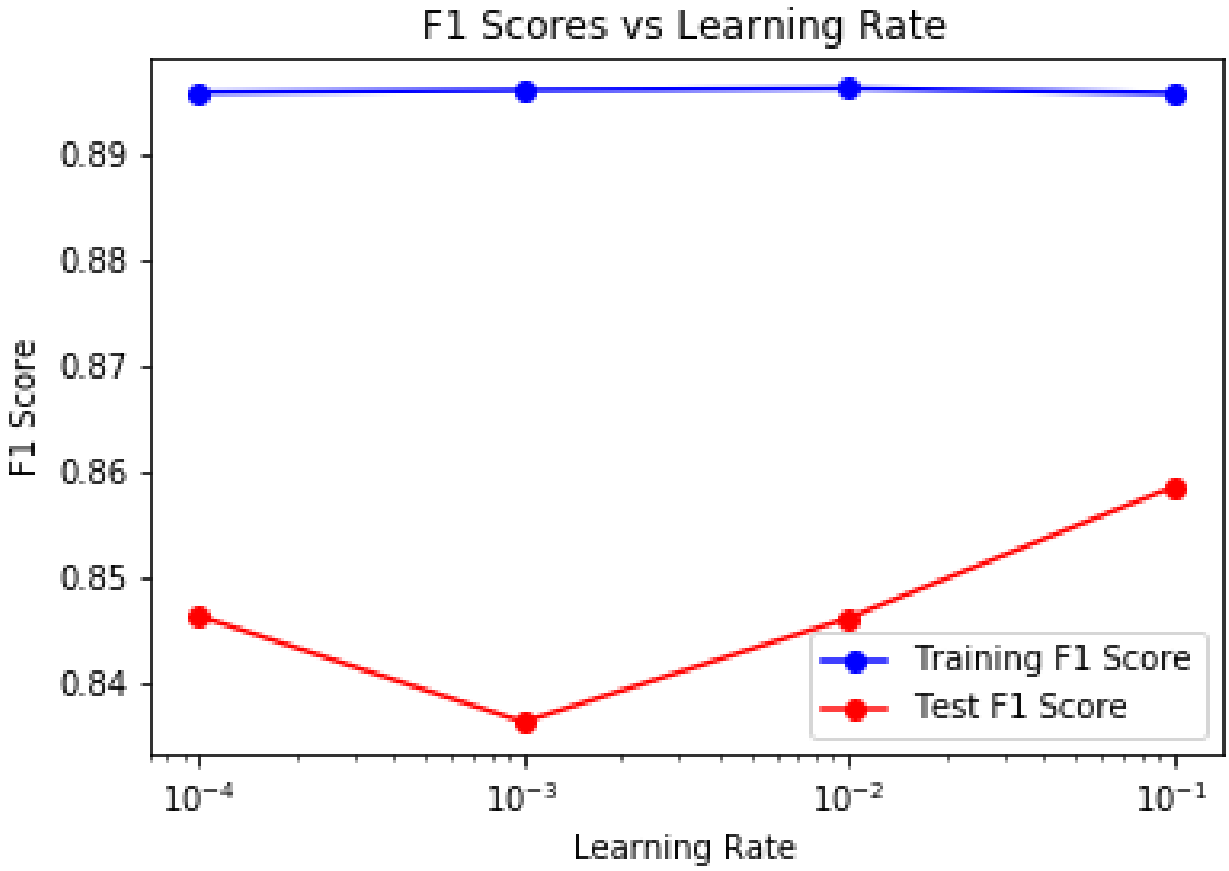

Figure 18: F1 Scores vs Learning Rate

THE END