CS57800: Statistical Machine Learning

Homework 2

Name: Mohammad Haseeb
Purdue ID: mhaseeb@purdue.edu

Due: October 09, 2018 on Tuesday

1 Vanilla Perceptron

1.1 Algorithm

I closely followed the algorithm included in the TA review slides. Here is the pseudo code for my final algorithm:

Algorithm 1 PerceptronTrain(TrainData,NumEpochs,LearningRate,Digit)

```
$weights_d = 0$ for all d=1...numFeatures+1 ;                    // +1 for the bias term
for iter = 1 ... numEpochs do
    for (img,imgLabel) in TrainData do
        $y = 0$
        if imgLabel == Digit then
            $y = 1$
        else
            $y = -1$
        end
        $activation = weights \cdot img$ ;  // compute activation for this image example
         by taking dot product
        if $y \times activation \leq 0$ then
            $weights_d = weights_d + learningRate \times y \times img_d$ for all d=1...numFeatures+1 ;
             // update weights and bias
        end
    end
end
return weights
```

1.2 Effect of Training Set Size

Note: All F1 scores in the report are shown as percentages.

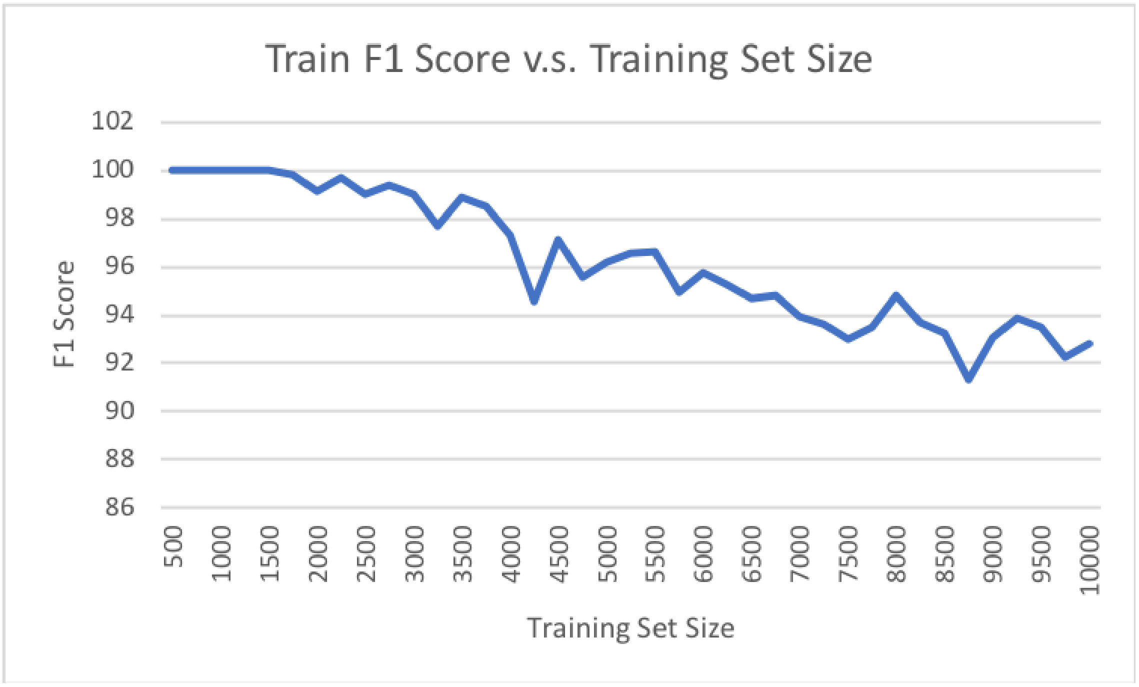

Figure 1: Effect of Training Set Size on F1 Score for Training Data

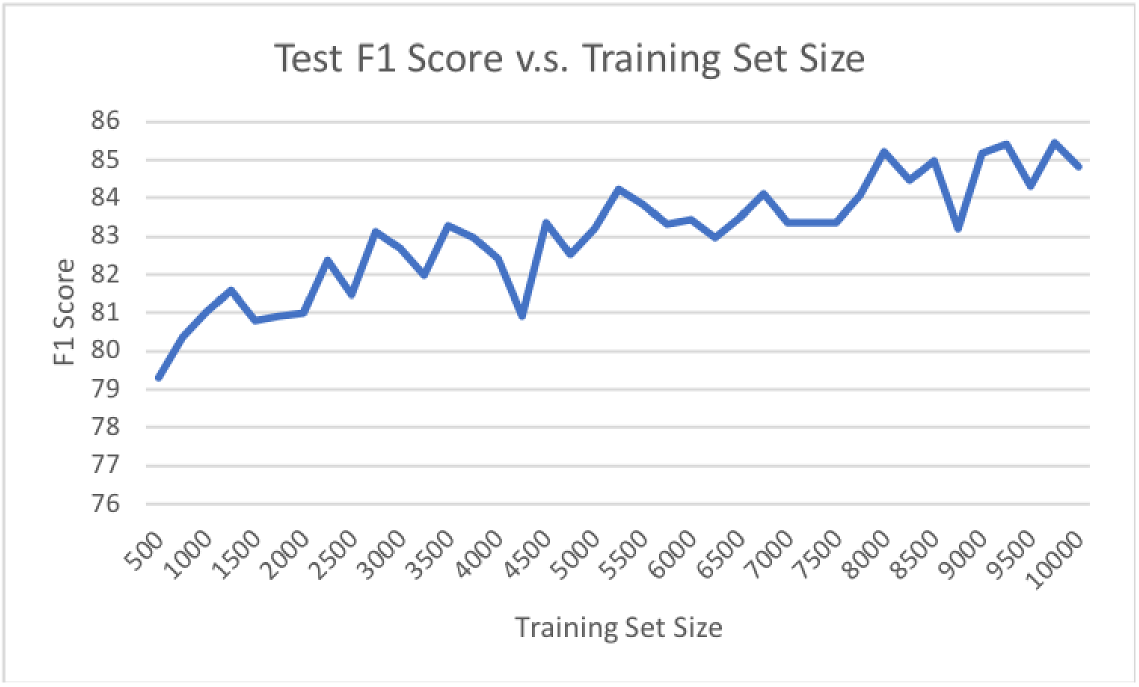

Figure 2: Effect of Training Set Size on F1 Score for Test Data

As it can be seen from the graphs, as the size of the training set increases, the F1 score decreases for the training data and increases for the test data. This is expected because as there are more and more training examples to learn from, the perceptron generalizes better, therefore, the training error increases. The best value for the size of the training set is 9750-10000.

1.3 Effect of Number of Epochs

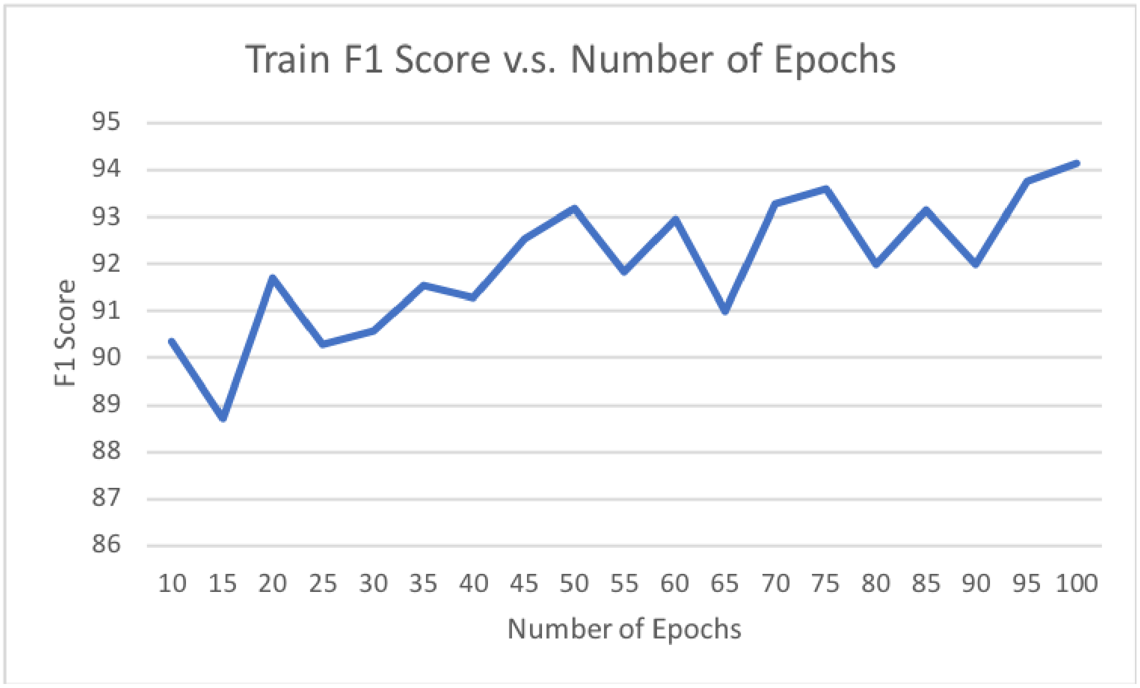

Figure 3: Effect of Number of Epochs on F1 Score for Train Data

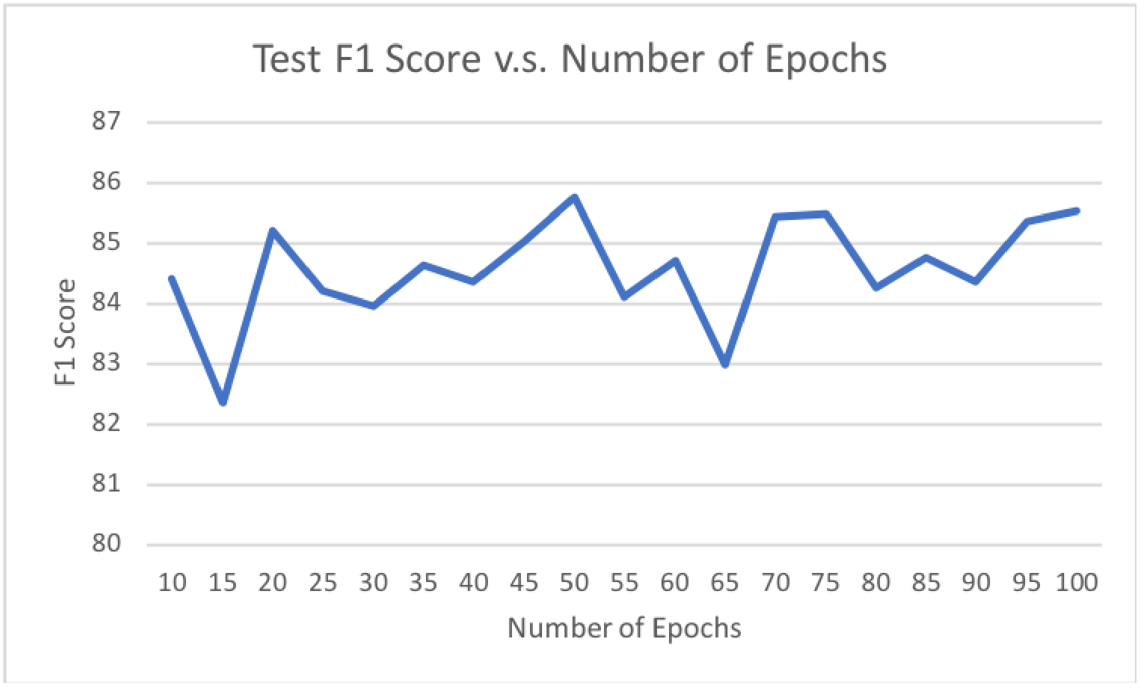

Figure 4: Effect of Number of Epochs on F1 Score for Test Data

As the number of epochs (or time spent on training) increases, the perceptron starts to memorize the training set and therefore, we can see a general increasing trend in the F1 score for training data, with the best training F1 score at 100 epochs. However, the best F1 score for testing data is at 50 epochs and decreases after that, which is what we would expect because as the number of epochs increase, the model tends to over fit on the training data and we expect lower F1 score on test data.

1.4 Effect of Learning Rate

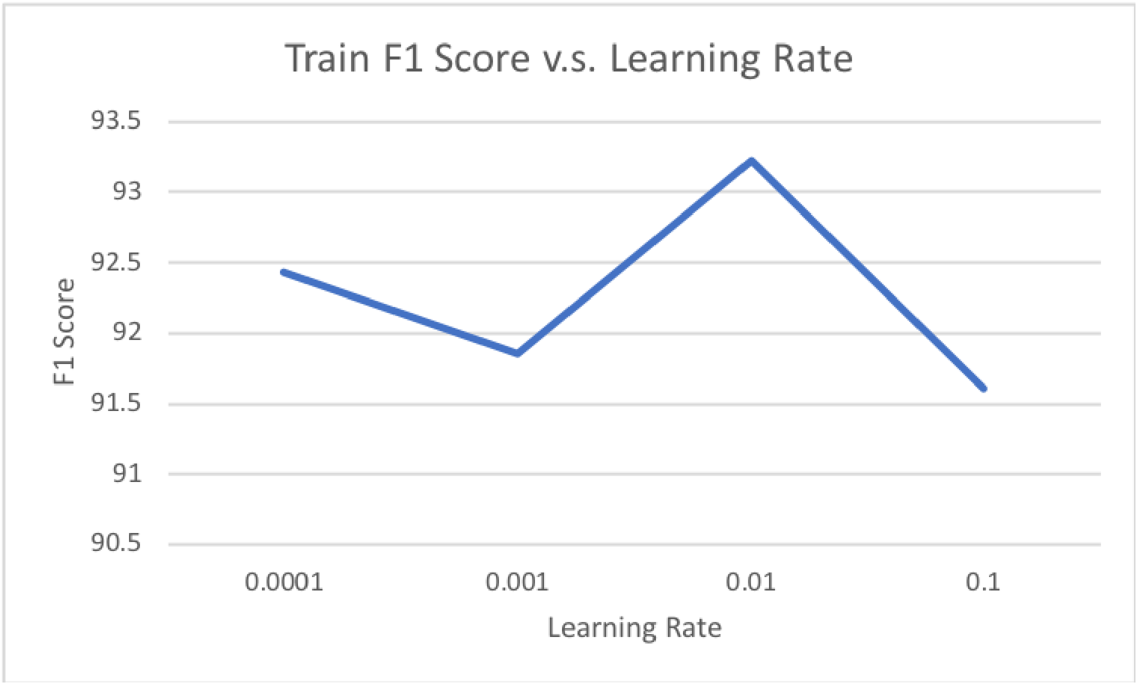

Figure 5: Effect of Learning Rate on F1 Score for Train Data

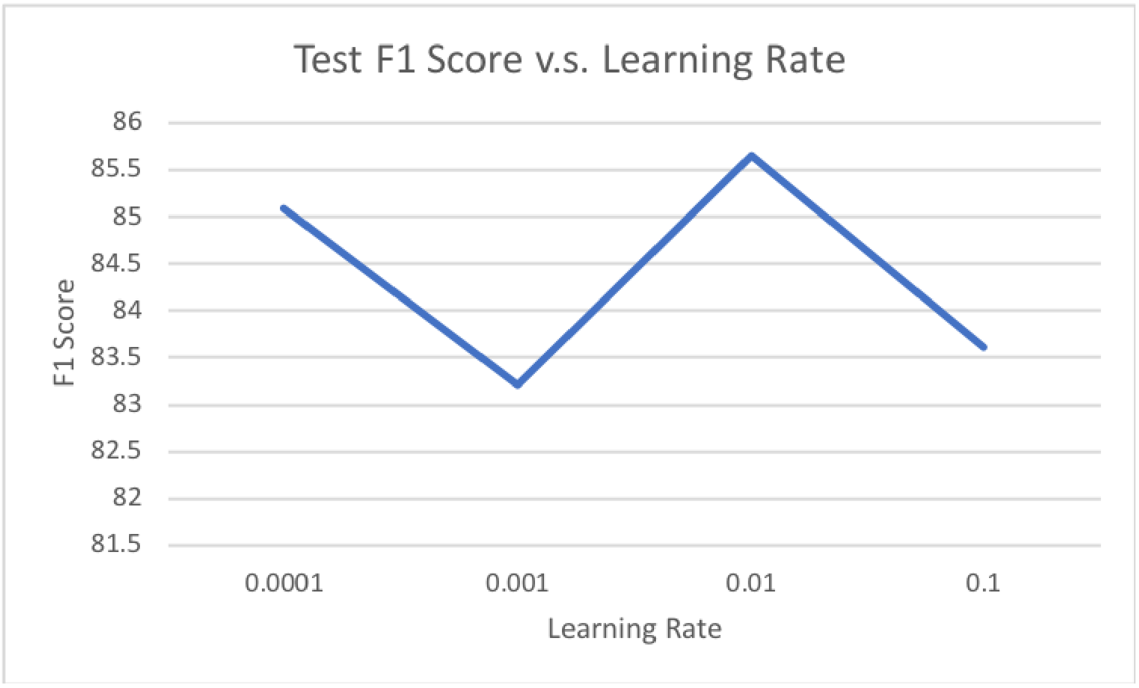

Figure 6: Effect of Learning Rate on F1 Score for Test Data

The learning rate in the perceptron algorithm determines how *fast* our algorithm learns the hypothesis (weights) and converges. In this case, after training the model for 50 epochs, we see similar trends in the learning curves for both the training and the testing data. The best learning rate is 0.01.

2 Average Perceptron

2.1 Algorithm

The algorithm for the average perceptron is very similar to the vanilla perceptron, the only difference is that we add the weight at every iteration to our running average of weights. This ensures that the hypothesis that lasts longer gets more weight. Finally, we predict based on the average weights vector.

Algorithm 2 AvgPerceptronTrain(TrainData,NumEpochs,LearningRate,Digit)

```
$weights_d = 0$ for all d=1...numFeatures+1 ;                // +1 for the bias term
$avgWeights_d = 0$ for all d=1...numFeatures+1 ;             // +1 for the bias term
for iter = 1 ... numEpochs do
    for (img,imgLabel) in TrainData do
        $y = 0$
        if imgLabel == Digit then
            $y = 1$
        else
            $y = -1$
        end
        $activation = weights \cdot img$ ;  // compute activation for this image example
         by taking dot product
        if $y \times activation \leq 0$ then
            $weights_d = weights_d + learningRate \times y \times img_d$ for all d=1...numFeatures+1 ;
             // update weights and bias
        end
        $avgWeights_d = avgWeights_d + weights_d$ for all d=1...numFeatures+1 ;      // update
         average weights and bias
    end
end
return $avgWeights$
```

2.2 Effect of Training Set Size

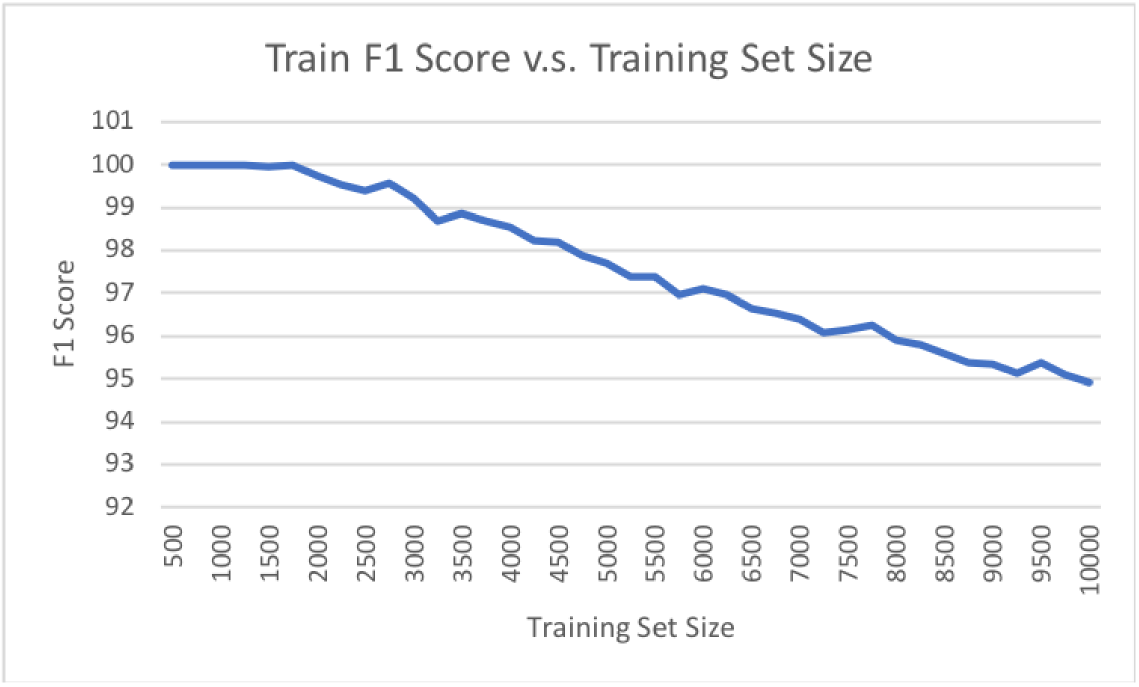

Figure 7: Effect of Training Set Size on F1 Score for Training Data

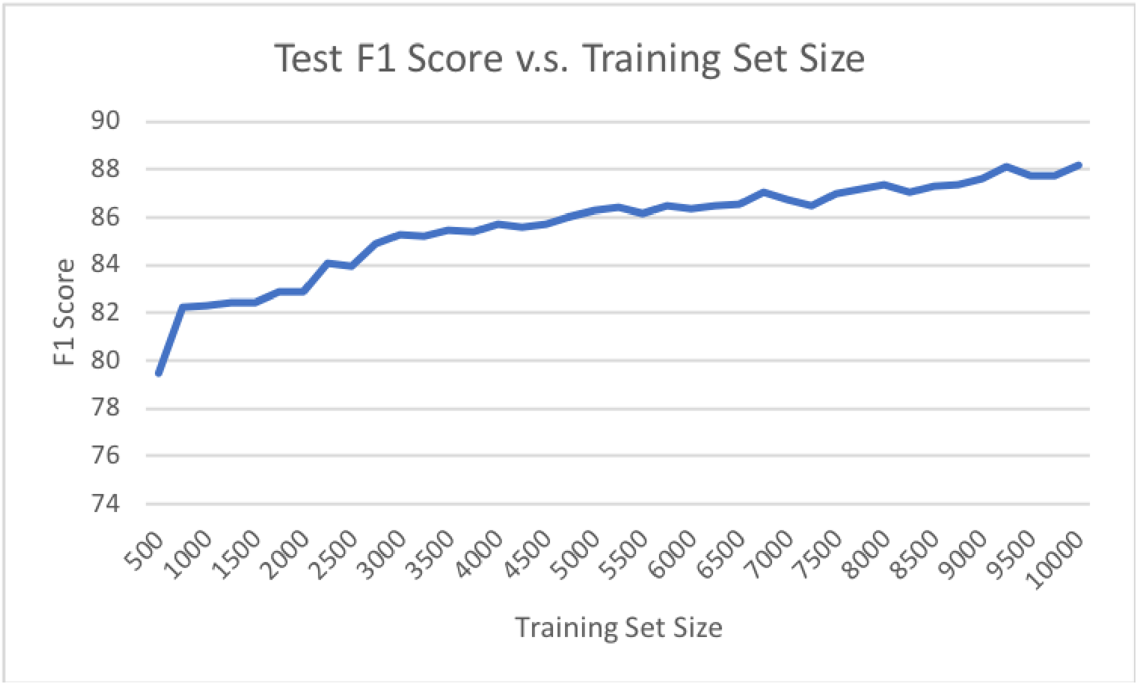

Figure 8: Effect of Training Set Size on F1 Score for Test Data

As the size of the training set increases, the F1 score decreases for the training data and increases for the test data. This is expected because as there are more and more training examples to learn from, the perceptron generalizes better, therefore, the training error increases. The best value for the size of the training set is 10000.

2.3 Effect of Number of Epochs

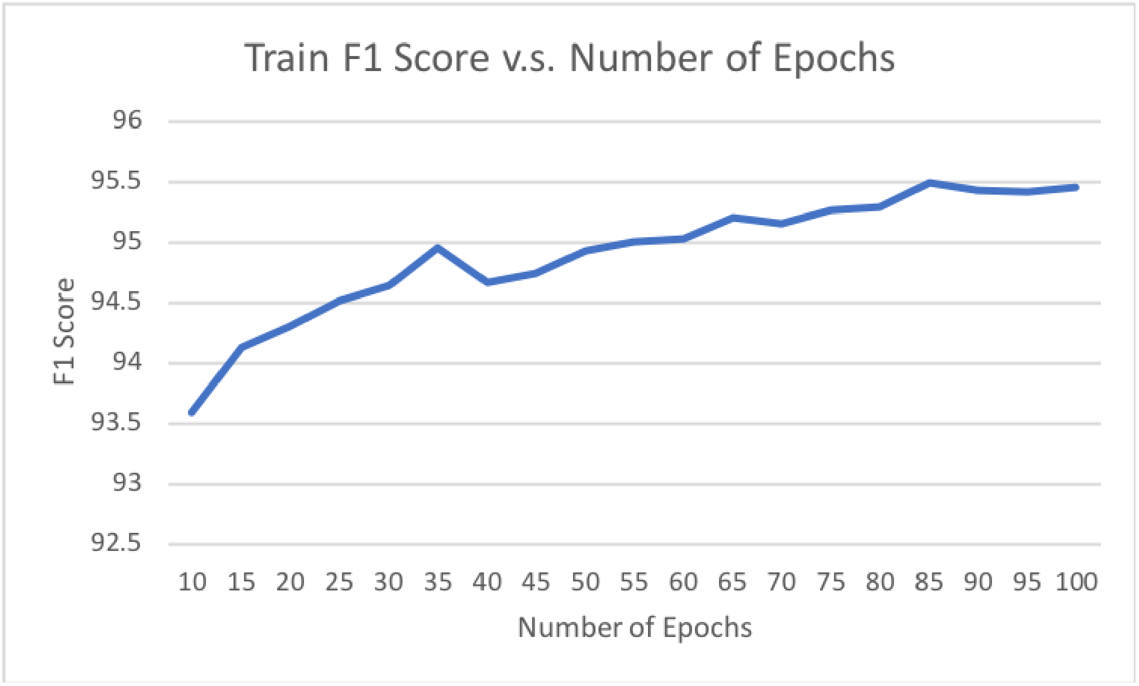

Figure 9: Effect of Number of Epochs on F1 Score for Train Data

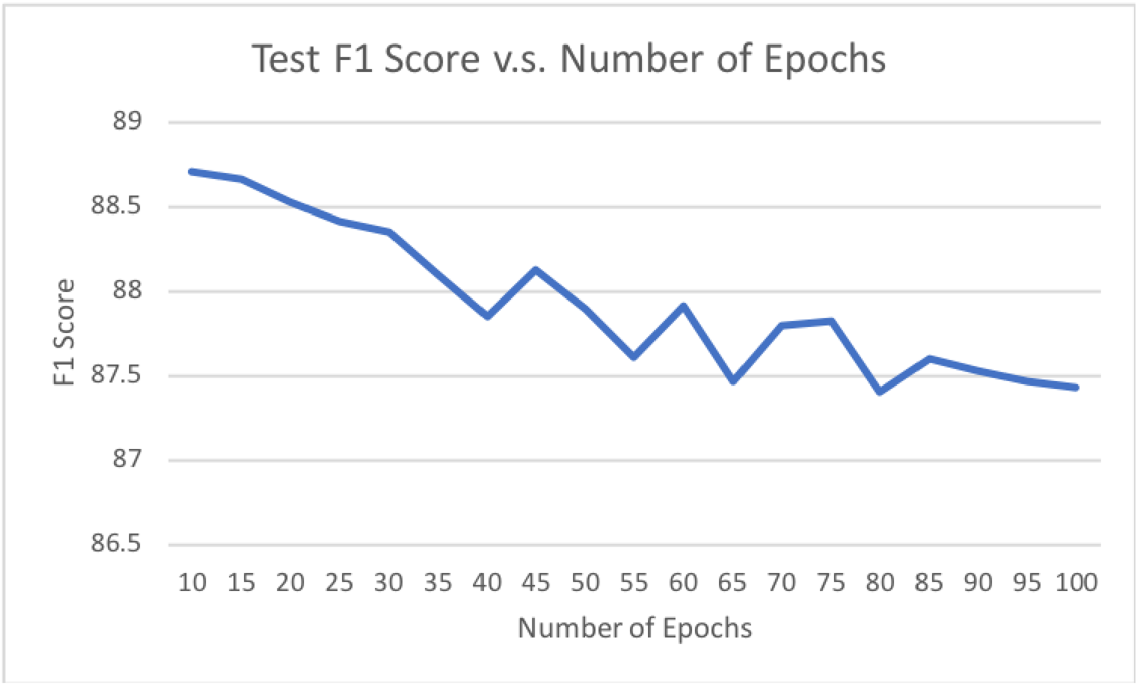

Figure 10: Effect of Number of Epochs on F1 Score for Test Data

It can be seen more clearly here that increasing the number of epochs causes over fitting. As the number of epochs (or time spent on training) increases, the perceptron starts to memorize the training set and therefore, we can see a general increasing trend in the F1 score for training data, with the best training F1 score at 100 epochs. However, the best F1 score for testing data is at 10 epochs and decreases after that, which is what we would expect because as the number of epochs increase, the model tends to over fit on the training data and we expect lower F1 score on test data.

2.4 Effect of Learning Rate

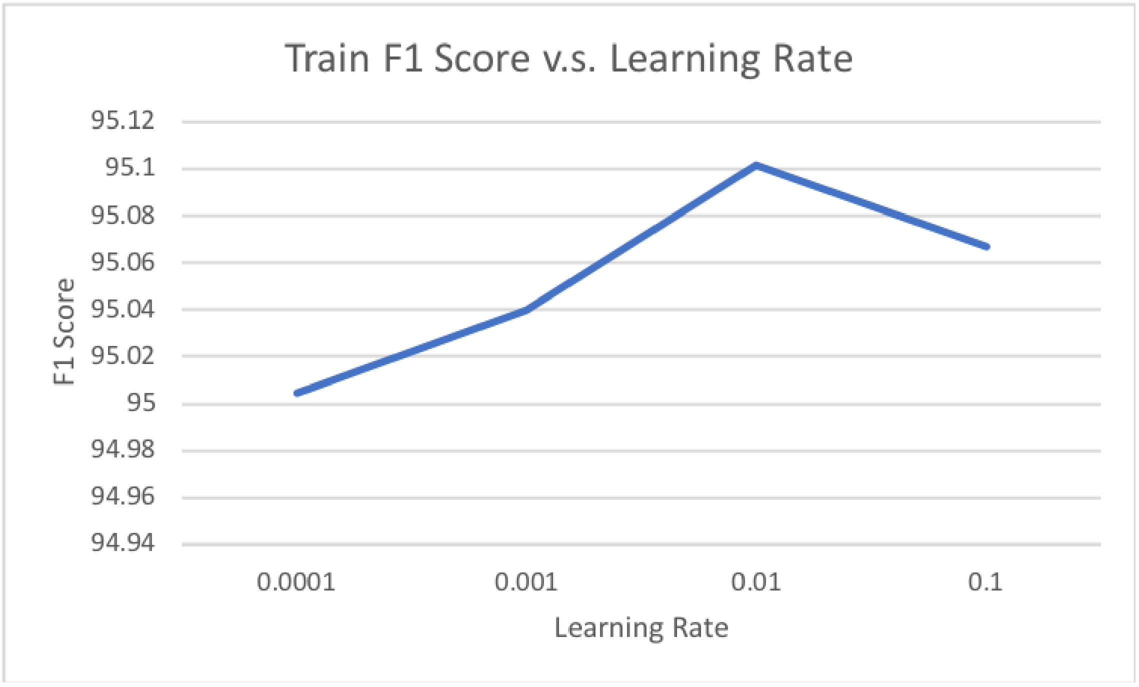

Figure 11: Effect of Learning Rate on F1 Score for Train Data

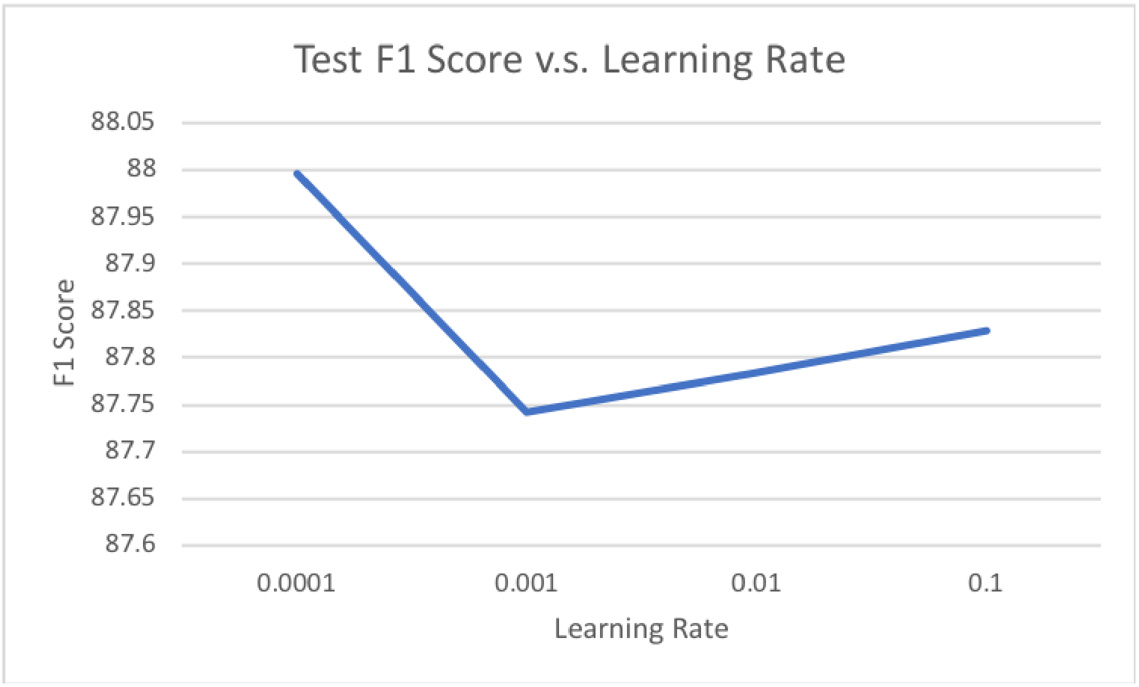

Figure 12: Effect of Learning Rate on F1 Score for Test Data

Varying the learning rate between 0.0001 and 0.1 does not provide significant changes in the F1 score (an absolute difference of 1). Because the learning rate determines how fast the weights are varied and learned, and as we saw before that the best performance on test data was with number of epochs set to 10, it is clear the the average perceptron converges fairly faster. This is why we do not see much difference in the F1 scores between different learning rates since the algorithm a similar weights' model is learned no matter the size of the learning rate. The best learning rate according to the test data results is 0.0001.

3 Winnow

3.1 Algorithm

The vanilla winnow algorithm was giving me average F1 test score of approx. 60%. I then tried the average version of the winnow algorithm and that increased my avg. F1 test score to approx. 77%. I have provided the pseudo code for the average winnow algorithm.

Algorithm 3 AvgWinnowTrain(TrainData,NumEpochs,Thetha,MultplicationFactor,Digit)

```
$weights_d = 1$ for all d=1...numFeatures+1 ;                // +1 for the bias term
$avgWeights_d = 1$ for all d=1...numFeatures+1 ;             // +1 for the bias term
for $iter = 1 \dots numEpochs$ do
    for (img,imgLabel) in TrainData do
        $y = 0$
        if $imgLabel == Digit$ then
            $y = 1$
        else
            $y = -1$
        end
        $activation = weights \cdot img$ ;  // compute activation for this image example
         by taking dot product
        if $(y == 1$ and $activation \geq Theta)$ or $(y == -1$ and $activation < Theta)$ then
            $avgWeights_d = avgWeights_d + weights_d$ for all d=1...numFeatures+1 ;   // update
             average weights and bias
            continue
        end
        $onesIdxs = img.where(img_i == 1)$ ;       // indexes of img where value is 1
        for idx in onesIdxs do
            if $y == 1$ and $activation < Theta$ then
                $weights_{idx} = weights_{idx} \times MultiplicationFactor$
            else if $y == -1$ and $activation \geq Theta$ then
                $weights_{idx} = weights_{idx} \times 1/MultiplicationFactor$
        end
        $avgWeights_d = avgWeights_d + weights_d$ for all d=1...numFeatures+1 ;       // update
         average weights and bias
    end
end
return $avgWeights$
```

3.2 Hyper Parameter Tuning

I decided to tune three hyper-parameters: (1) Training Set Size, (2) Number of Epochs and (3) Multiplication Factor. The learning curves for each of these are in the following sections. I tuned each hyper-parameter by training the algorithm with different values of the parameter while keeping all other hyper-parameters to default values. I then chose the value of the hyper-parameter as the one that gave the best results on the test data.

3.2.1 Effect of Training Set Size

Default values:

- Number of Epochs: 50

- Multiplication Factor: 2
- Theta: 785

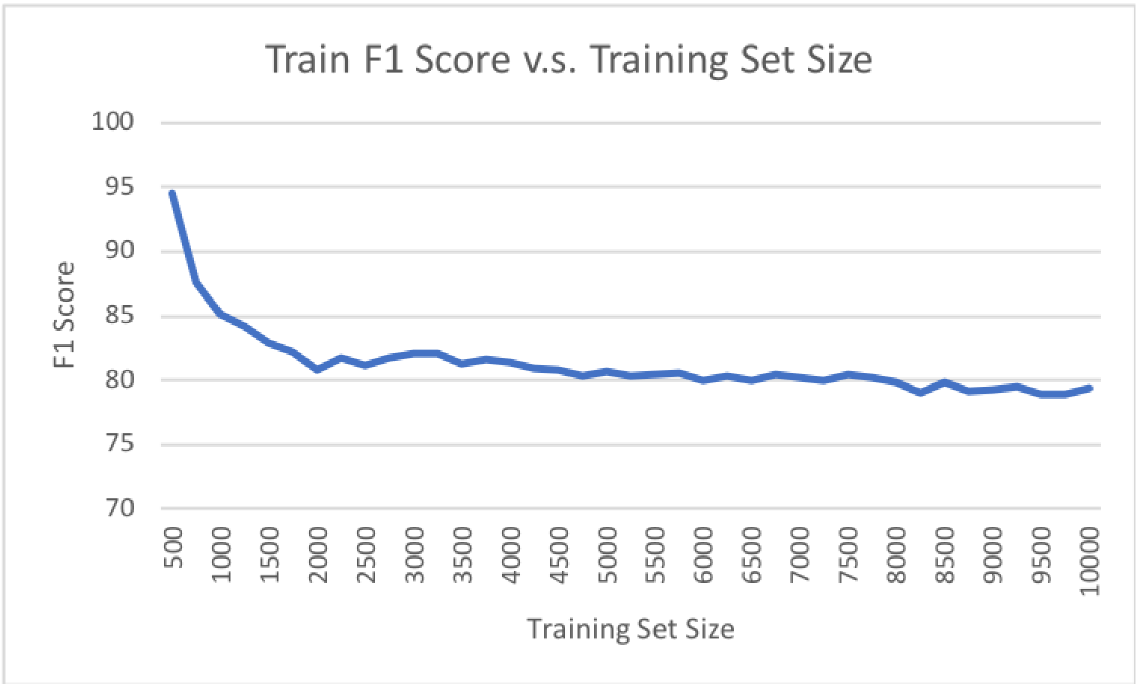

Figure 13: Effect of Training Set Size on F1 Score for Training Data

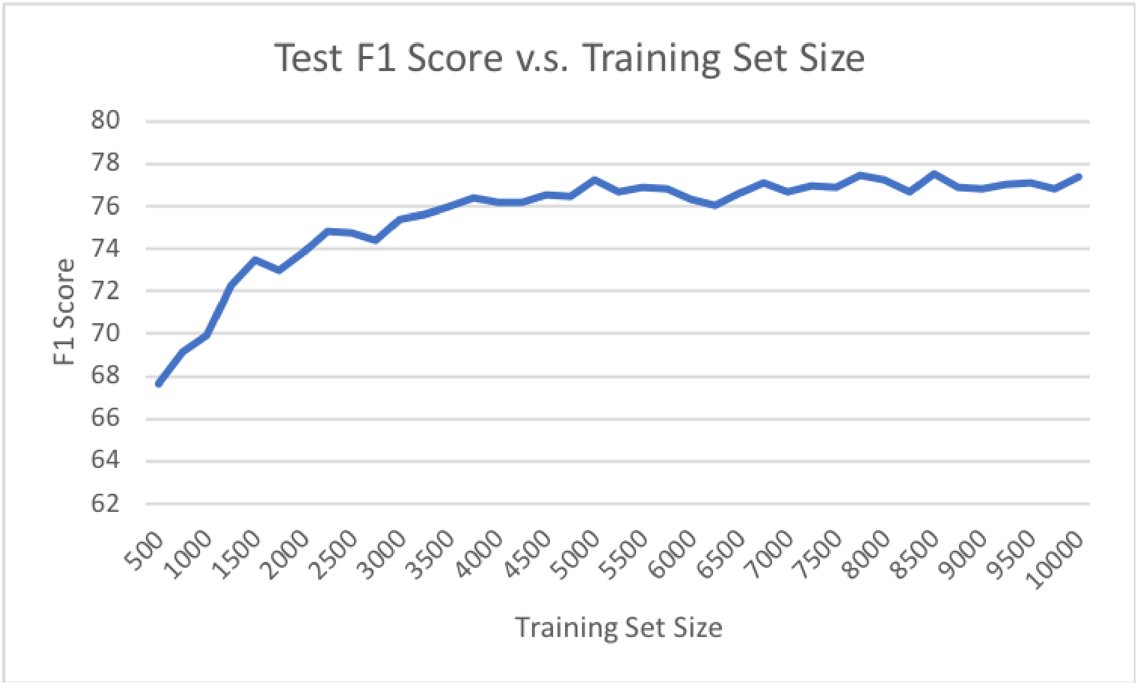

Figure 14: Effect of Training Set Size on F1 Score for Test Data

As it can be seen from the graphs, as the size of the training set increases, the F1 score decreases for the training data and increases for the test data. This is expected because as there are more and more training examples to learn from, the perceptron generalizes better, therefore, the training error increases and there is no over fitting. The best value for the size of the training set is 10000.

3.2.2 Effect of Number of Epochs

Default values:

- Training Set Size: 10000
- Multiplication Factor: 2
- Theta: 785

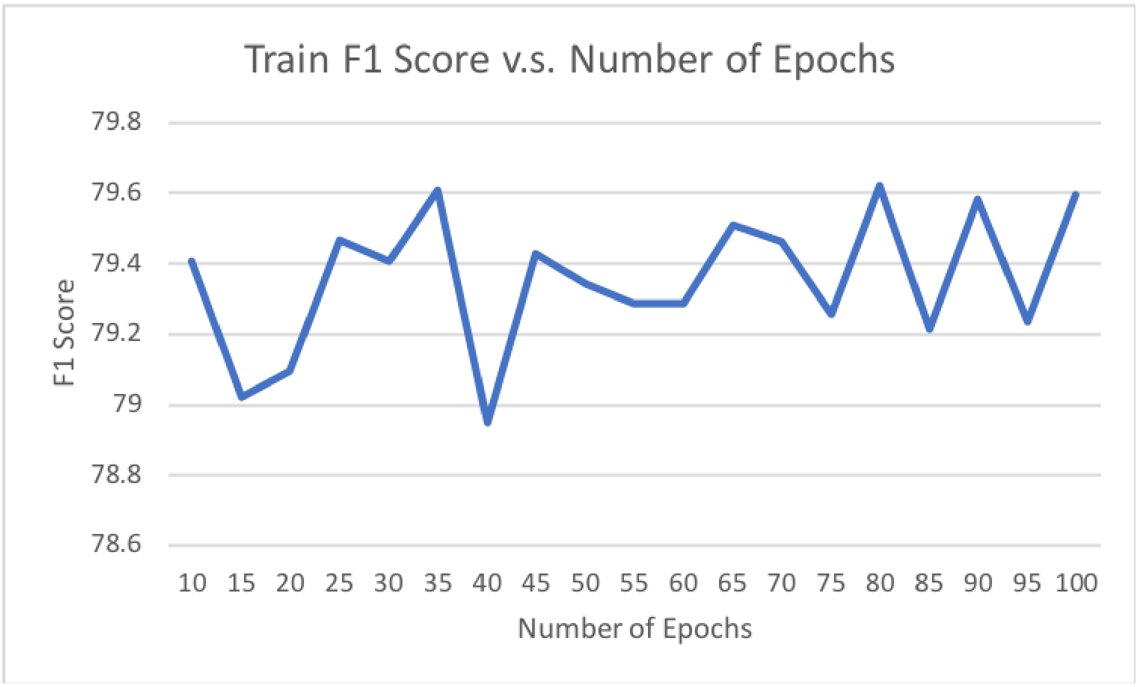

Figure 15: Effect of Number of Epochs on F1 Score for Train Data

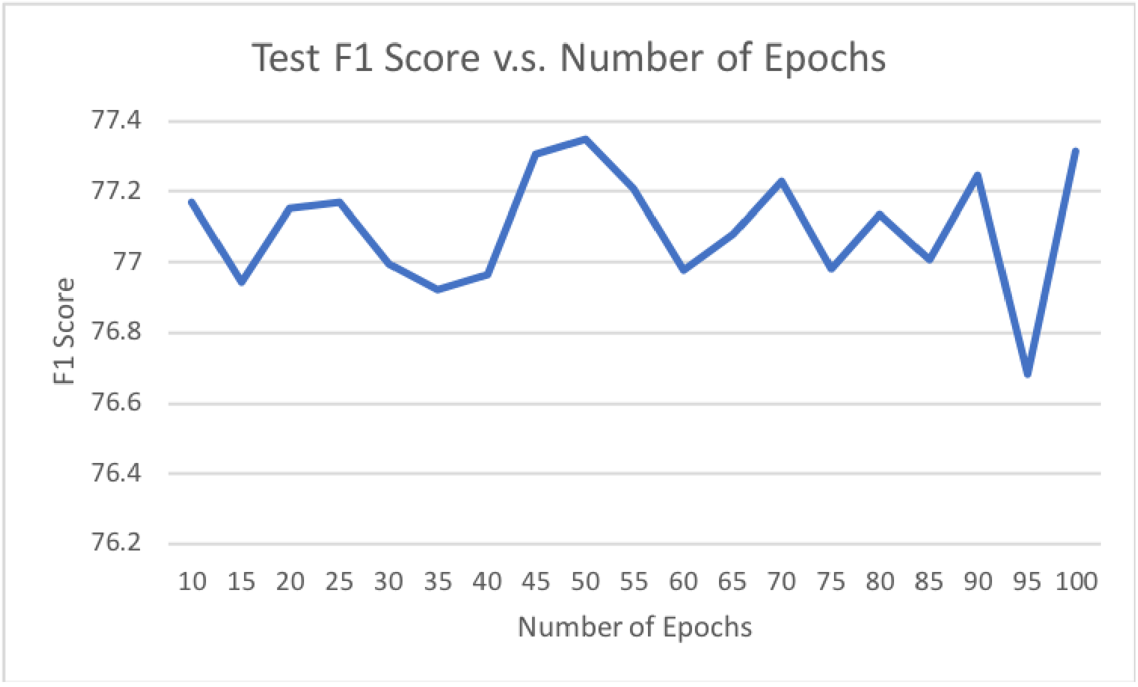

Figure 16: Effect of Number of Epochs on F1 Score for Test Data

As the number of epochs (or time spent on training) increases, the perceptron starts to memorize the training set and therefore, we can see a general increasing trend in the F1 score for training data, with the best training F1 score at 80-100 epochs. However, the best F1 score for testing data is at 50 epochs and decreases after that, which is what we would expect because as the number of epochs increase, the model tends to over fit on the training data and we expect lower F1 score on test data. However, it looks from the learning curves that there is not a lot of over fitting going on.

3.2.3 Effect of Multiplication Factor

Before deciding on tuning this parameter, I tested with incorporating the *exponential* learning rate in the update rule, as was mentioned int the TA review slides' appendix. However, the results of incorporating that were worse than not having it altogether. I then thought about tweaking the multiplication factor in the winnow between 1.5 to 4.5 (I actually tried until 12.0 but the trends are the same). The results are presented below.

Default values:

- Training Set Size: 10000
- Number of Epochs: 50
- Theta: 785

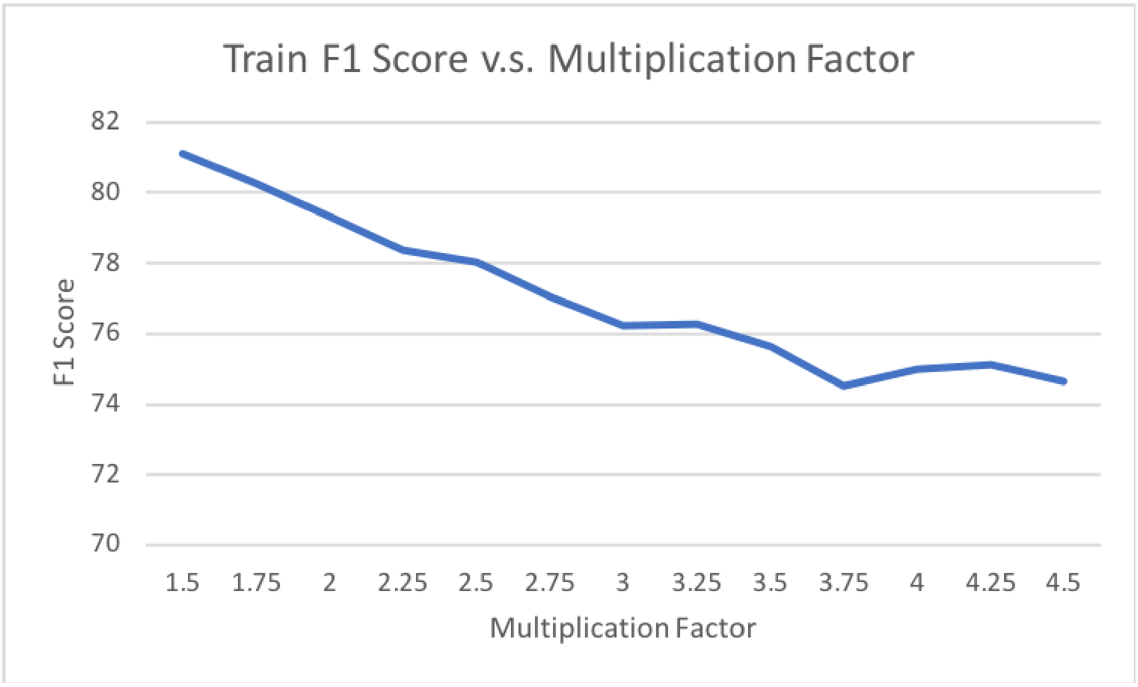

Figure 17: Effect of Multiplication Factor on F1 Score for Train Data

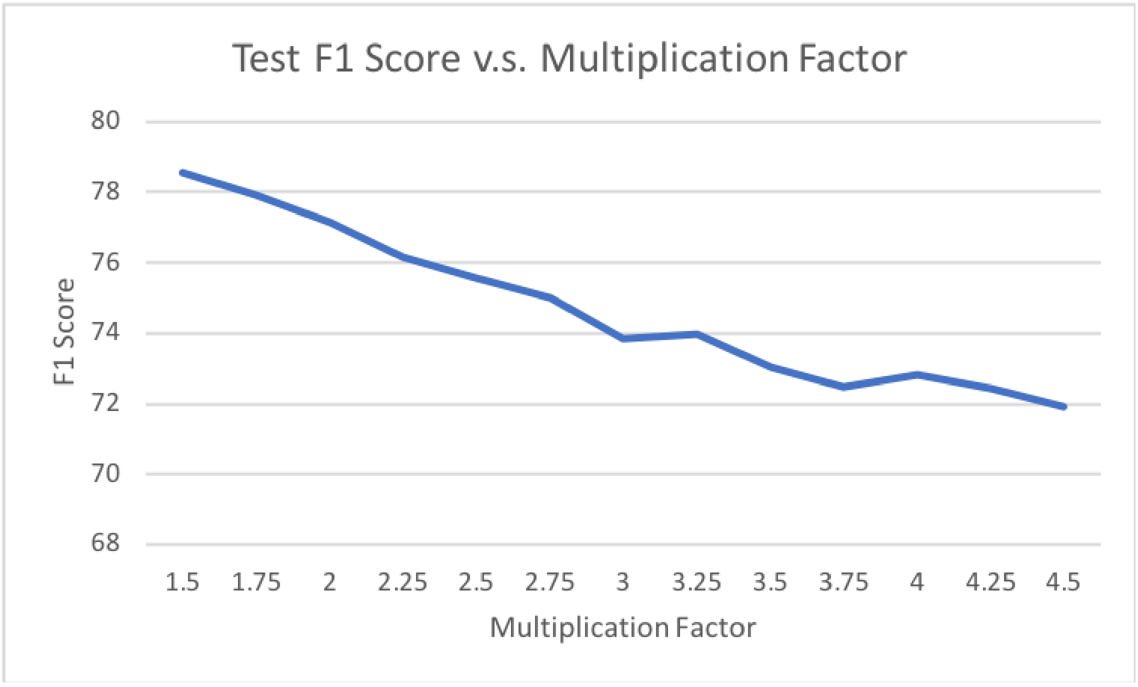

Figure 18: Effect of Multiplication Factor on F1 Score for Test Data

Just like it was the case with the perceptron, the multiplication factor determines by how much the values of the weights is varied. Having too large a factor can result in some features being overly expressive when they should not have been while others being extremely non-expressive.

The converse also applies. The learning curves show this trend that as the multiplication factor is increased, the F1 score for both the training and testing data decreases. The best multiplication factor, interestingly, turns out to be 1.5 for this problem and data set.

4 Open Ended Questions

1. We observe similar trends in the learning curves for the all the three algorithms for hyper-parameters like training set size and number of epochs. This can be reasoned due to the similarity in their underlying nature (increasing or decreasing weights). The best test F1 score is with the average perceptron algorithm as it provides additive upgrades which allow for the weights to be tweaked more accurately. Moreover, having an average perceptron allows us to take into account the hypothesis that performed the best for the most time.

2. An algorithm A is said to be a mistake bound algorithm for a concept class C if, for any concept c in C, the maximum number of mistakes made by A while learning c is a polynomial in n and $size(c)$, where n is the dimensionality of an example in c.

3. We can use an alteration of the elimination algorithm for monotone conjunctions. Say there are n features then we initialize our hypothesis to be $h(x) = x_1 \wedge \neg x_1 \wedge ... \wedge x_n \wedge \neg x_n$. Now, if the predicted label for some example x is False and the true label is True, remove all literals from h that are False in x. This also includes the negated literals in h that are True in x. We do this because all the literals in h that are not active in x do not need to be present in h.

   To show that this is a mistake bound algorithm, we show two things: (1) the maximum number of mistakes that this algorithm makes before converging is bounded and (2) the algorithm is polynomial in the number of dimensions n of the examples and the input data size $size(c)$.

   Note that when this algorithm makes its first mistake on an example, the number of literals in h go from $2n$ to $\leq n$. This is because we remove all literals from h that are not True in x. Thereafter, on every mistake, we remove at least 1 literal from h. Therefore, the total number of mistakes that this algorithm can make is $n + 1$.

   To show that this algorithm is polynomial in the number of dimensions n and the size of a concept c, $size(c)$, notice that we iterate through $size(c)$ many times to see all the examples. In each example, we iterate through the n literals to see which of these are False and if that literal is present in h, we remove it. Therefore, the worst case complexity for this would be $O(n \cdot size(c))$ which is polynomial in n and $size(c)$.

4. (a) Yes. Since we are using the perceptron algorithm, which is a linear classification algorithm, both the classifiers will converge because the dataset is linearly separable.

   (b) Assuming that we have very large number of epochs to train both the classifiers, both the classifiers will have a training error of zero. The training error will be zero because the dataset is linearly separable, therefore, once the separating hyperplane has been learned, the training examples would always be classified correctly.

Because we assume that the number of epochs is very large, The first classifier (trained on sorted examples) would also eventually learn a hypothesis that has a decent classifier margin. Although initially the weights learned by the classifier might be heaviily skewed towards positive examples, having a large number of epochs would allow the perceptron to take into account the negative examples and learn the best weights that separate the dataset.