Imperial College
London

# EE3-23 Machine Learning

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

## Assignment 1

Anand Kasture (CID: 00832896)

Date: February 6, 2017

## Pledge

I, **Anand Kasture**, pledge that this assignment is completely my own work, and that I did not take, borrow or steal work from any other person, and that I did not allow any other person to use, have, borrow or steal portions of my work. I understand that if I violate this honesty pledge, I am subject to disciplinary action pursuant to the appropriate sections of Imperial College London.

# 1 Brexit Confidence Intervals

This question concerns the confidence with which we can deduce the likelihood of an event occurring when provided with poll results on a fixed sample size. The problem can be formally stated as follows: With what confidence level can we *reject* the hypothesis that the probability of the United Kingdom (UK) remaining in the European Union (EU) is at least 50%, when provided with poll results on 2052 people. These results show that 45% of the people preferred to stay within the union versus and 55% supported leaving the EU.

Let the random variable $X$ represent an individual's definitive decision regarding Brexit. We will enforce the independent and identically distributed (i.i.d) assumption on multiple instances of the random variable $X$. The random variable $X$ can be modelled with a Bernoulli Distribution as follows:

$$f_X(x;p) = \begin{cases} p & X = 1 \quad \text{(Leave)} \\ 1-p & X = 0 \quad \text{(Remain)} \end{cases}$$

Note: $f_X(x;p)$ represents the probability mass function of this distribution.

We will invoke Hoeffding's inequality to solve this problem. Hoeffding's inequality places an upper bound on the probability that the sum of random variables deviates from its expected value by a certain amount. Equation 1 presents a formal definition of the one-sided variant. Alternatively, Equation 1 can be also be re-written as shown in Equation 2.

Let $a_k \leq b_k$ be real numbers. Let $X_1, X_2, ..., X_N$ be a sequence of i.i.d real-valued random variables such that $X_k \in [a_k, b_k]$ for every $k = 1, 2, ..., N$. Then, Hoeffding's inequality states that for any $\epsilon \geq 0$,

$$\Pr\left(\sum_{k=1}^{N} X_k - \sum_{k=1}^{N} \mathbb{E}[X_k] \geq \epsilon\right) \leqslant \exp\left(\frac{-2\varepsilon^2}{\sum_{k=1}^{N}(b_k - a_k)^2}\right) \tag{1}$$

For any $0 < \delta \leq 1$, with probability of at least $1 - \delta$,

$$\sum_{k=1}^{N} X_k < \sum_{k=1}^{N} \mathbb{E}[X_k] + \sqrt{\frac{1}{2}\ln(1/\delta)\sum_{k=1}^{N}(b_k - a_k)^2} \tag{2}$$

For our problem, $a_k = 0$, $b_k = 1$, $\hat{p} = 0.55$, and $N = 2052$. Note that $\hat{p}$ represents the empirical probability of leaving the EU, as obtained from the poll results.

We will solve for $\delta$ in Equation 2, and obtain a solution to the overall problem by evaluating the resultant expression with our parameter values.

$$\sum_{k=1}^{N} X_k < \sum_{k=1}^{N} \mathbb{E}[X_k] + \sqrt{\frac{1}{2}\ln(1/\delta) \sum_{k=1}^{N} (b_k - a_k)^2} \tag{3a}$$

$$N\hat{p} < Np + \sqrt{\frac{1}{2}\ln(1/\delta) \cdot N} \tag{3b}$$

$$\hat{p} - p < \frac{1}{N}\sqrt{\frac{1}{2}\ln(1/\delta) \cdot N} \tag{3c}$$

The problem of finding the confidence to reject the hypothesis that the probability of the UK remaining the the EU is at least 50% is equivalent to finding the confidence with which we can accept that the probability of the UK leaving the EU is at most 50%. Therefore, we set $p = 0.5$ in Equation 3c.

$$\begin{aligned}\delta &= \exp(-2N(p-\hat{p})^2) = \exp(-2N\epsilon^2) \\ &= \exp(-2 \times 2052 \times 0.05)^2) \\ &= 3.5 \times 10^{-5}\end{aligned}$$

Therefore, we can reject the hypothesis that the likelihood of the UK remaining in the UK is at least 50%, with $(1 - \delta) =$ **99.996%** confidence.

# 2 Convergence of the Perceptron Algorithm

In this section, we will investigate the convergence properties of the perceptron algorithm, by working through four algebraic problems. We will use the following notation:

- $N$ - Number of $\mathbf{x}_k^j \rightarrow y_k^j$ mappings
- $\mathbf{x}_k^j$ - $j^{th}$ feature vector at iteration $k$
- $y^j$ - Binary output classification for $\mathbf{x}_k^j$
- $\mathbf{w}_k$ - Weight vector at iteration $k$
- $\mathbf{w}_\star$ - Optimal weight vector s.t. $\text{sign}(\mathbf{w}_\star^T \mathbf{x}^j) = y^j$ for $j = 1, 2, ..., N$

Note: $\mathbf{x}_k^j \in \mathbb{R}^{d+1}$, $\mathbf{w}_k \in \mathbb{R}^{d+1}$, $y^j \in \{1, -1\}$ for $j = 1, 2, ..., N$

## Part A

Define $\rho = \min_j \quad y^j \mathbf{w}_\star^T \mathbf{x}^j$. Show that $\rho > 0$.

By definition, the presence of $\mathbf{w}_\star$ implies that $\text{sign}(\mathbf{w}_\star^T \mathbf{x}^j) = y^j$ for $j = 1, 2, ..., N$. Therefore, the terms $\mathbf{w}_\star^T \mathbf{x}^j$ $y^j$ will have the same signs. This in turn necessitates the product of the two terms to always hold a positive value.

## Part B

Show that for any $k$ before the algorithm stops, $\mathbf{w}_\star^T \mathbf{w}_k \geq t\rho$.

| | |
|---|---|
| $\mathbf{w}_k = \mathbf{w}_{k-1} + y_{k-1}\mathbf{x}_{k-1}$ | perceptron weight update rule |
| $\mathbf{w}_\star^T \mathbf{w}_k = \mathbf{w}_\star^T \mathbf{w}_{k-1} + y_{k-1}\mathbf{w}_\star^T \mathbf{x}_{k-1}$ | multiplying both sides by $\mathbf{w}_\star^T$ |
| $\mathbf{w}_\star^T \mathbf{w}_k \geq \mathbf{w}_\star^T \mathbf{w}_{k-1} + \rho$ | using the definition of $\rho$ |
| | |
| $\mathbf{w}_\star^T \mathbf{w}_{k-1} \geq \mathbf{w}_\star^T \mathbf{w}_{k-2} + \rho$ | extending result to $k \leftarrow k-1$ |
| $\mathbf{w}_\star^T \mathbf{w}_k \geq \mathbf{w}_\star^T \mathbf{w}_{k-2} + 2\rho$ | re-writing for iteration $k$ |
| $\vdots$ | |
| $\mathbf{w}_\star^T \mathbf{w}_k \geq \mathbf{w}_\star^T \mathbf{w}_0 + k\rho$ | working backwards to $k = 1$ |
| $\mathbf{w}_\star^T \mathbf{w}_k \geq k\rho$ | since $\mathbf{w}_0 = \mathbf{0}$ |

## Part C

Show that for any $k$ before the algorithm stops, $\|\mathbf{w}_k\|^2 \leq tR^2$ where $R = \max_j \|\mathbf{x}_k\|$

| | |
|---|---|
| $\mathbf{w}_k = \mathbf{w}_{k-1} + y_{k-1}\mathbf{x}_{k-1}$ | perceptron weight update rule |
| $\|\mathbf{w}_k\|^2 = \|\mathbf{w}_{k-1}\|^2 + \|\mathbf{x}_{k-1}\|^2 + 2y_{k-1}\mathbf{w}_{k-1}^T\mathbf{x}_{k-1}$ | $\ell^2$- norm of both sides |
| $\|\mathbf{w}_k\|^2 \leq \|\mathbf{w}_{k-1}\|^2 + \|\mathbf{x}_{k-1}\|^2$ | mistake $\Rightarrow 2y_{k-1}\mathbf{w}_{k-1}^T\mathbf{x}_{k-1} \leq 0$ |
| $\|\mathbf{w}_k\|^2 \leq \|\mathbf{w}_{k-1}\|^2 + R^2$ | using the definition of $R$ |
| | |
| $\|\mathbf{w}_{k-1}\|^2 \leq \|\mathbf{w}_{k-2}\|^2 + R^2$ | extending result to $k \leftarrow k-1$ |
| $\|\mathbf{w}_k\|^2 \leq \|\mathbf{w}_{k-2}\|^2 + 2R^2$ | re-writing for iteration $k$ |
| $\vdots$ | |
| $\|\mathbf{w}_k\|^2 \leq \|\mathbf{w}_0\|^2 + kR^2$ | working backwards to $k = 1$ |
| $\|\mathbf{w}_k\|^2 \leq kR^2$ | since $\mathbf{w}_0 = \mathbf{0}$ |

## Part D

Show that for any $k$ before the algorithm stops, $k \leq R^2\|\mathbf{w}_\star\|^2/\rho^2$, and conclude that the algorithm stops after at most $k \leq R^2\|\mathbf{w}_\star\|^2/\rho^2$ updates/iterations.

| | |
|---|---|
| $k\rho \leq \mathbf{w}_\star^T\mathbf{w}_k$ | using result from Part B |
| $k\rho \leq \left\|\mathbf{w}_\star^T\right\|\|\mathbf{w}_k\|$ | using the Cauchy-Schwarz inequality |
| $k^2\rho^2 \leq \left\|\mathbf{w}_\star^T\right\|^2\|\mathbf{w}_k\|^2$ | squaring both sides |
| $k^2\rho^2 \leq \left\|\mathbf{w}_\star^T\right\|^2 kR^2$ | using result from Part C |
| $k \leq \frac{R^2}{\rho^2}\left\|\mathbf{w}_\star^T\right\|^2$ | |

# 3 Experiments with the Perceptron Learning Algorithm

In this section, we will first present initial results on a simple MATLAB implementation of the perceptron algorithm using synthetic training and test data sets.

Thereafter, we will investigate the classification performance of the algorithm for a varying number of training data points with different margin sizes. The performance metric here is the error probability i.e. the proportion of test data points that are wrongly classified.

Lastly, we will modify the perceptron algorithm in order to extend this approach to real world data. Namely, we will present training error and test error results for the classification of two handwritten digits.

## Part A - Implementing Perceptron

We have successfully implemented a simple perceptron function in MATLAB. The perceptron learning algorithm represents a very clear and concise approach to the construction of a linear classifier. Therefore, our MATLAB function is mostly based on the version that has been presented in the lecture slides.

However, the software level implementation of the perceptron algorithm may vary. For example, because the weights are iteratively updated using the first mis-classified point that has been identified, the order in which we iterate through the set of feature vectors can have an impact on the algorithm's convergence performance. However, as an initial start, we simply iterate through the set from the first index to the last i.e. without pre-processing the data set.

In order to visualise the linear classifier generated by our perceptron algorithm, we ran our MATLAB function against a varying number of feature vectors from the same 2-D training data set. Figure 1 illustrates two plots produced for $N = 2$ and $N = 4$, where $N$ represents the number of feature vectors. The equation used to generate the training data set is $x_2 = x_1 + 0.1$. This equation represents the true classifier, and is also plotted on the graphs. Hence, it follows that the region shaded in yellow represents the error probability of our own implementation.

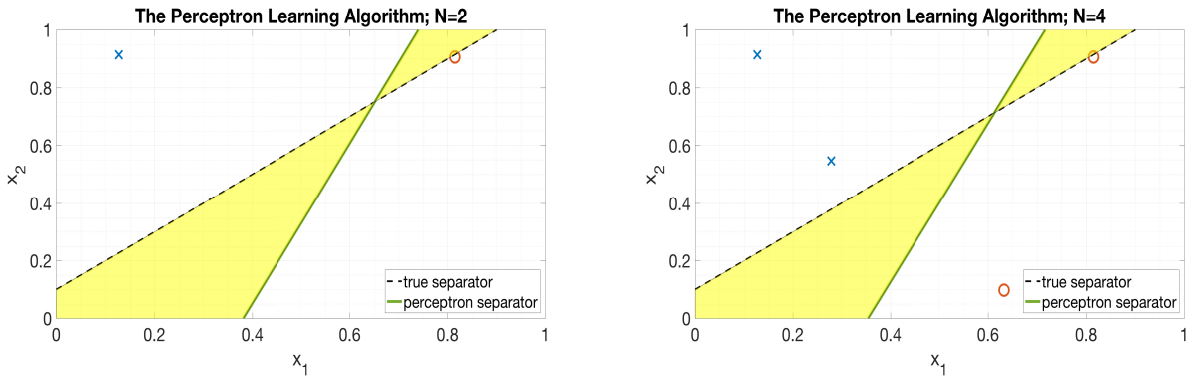


*Figure 1: Perceptron Learning Algorithm for N=2 and N=4*

The synthetic 2-D training data set was generated using the `rand` function within MATLAB. This produces a random number that is uniformly distributed in the range $[0,1]$. We reset the random number generator seed to its default value at the top of our script in order to replicate the time invariant property of a simple real world training data set.

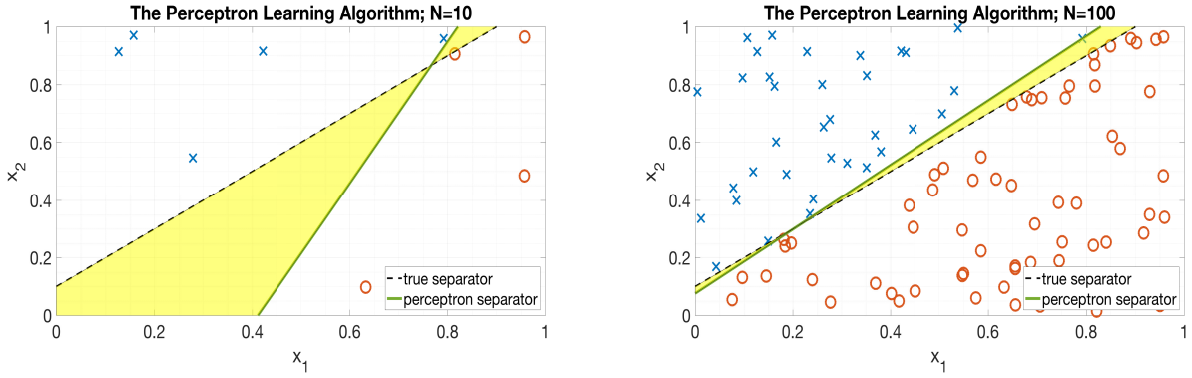


*Figure 2: Perceptron Learning Algorithm for N=10 and N=100*

Figure 2 illustrates two plots produced for $N = 10$ and $N = 100$. We observe that the yellow region for $N = 100$ is significantly smaller than that for $N = 10$. This is an expected result, since a higher number of training data points enforce stricter constraints on the algorithm.

## Part B - Classification Performance

### (1) A Closed Loop Formula for the Test Error

This section requires us to present a closed form formula on the test error of any separator line $ax + b$, when the true classifier is defined by $y = x + 0.1$. The test error, in theory, corresponds to the area represented by conflicting separator inequalities. Obtaining a fully representative closed form solution that encapsulates every unique polygon combinations within the unit square proved to be a notoriously difficult task. One way of computing the theoretical test error was to classify the types of regions that are produced as a function of $a$ and $b$, and using the MATLAB (integral) function. However, this approach does not accommodate y-axis limits, leading to the requirement for corrective steps post integration.

Therefore, all test error calculations in this assignment are performed empirically. The test data set is generated using the same logic as that for the generation of training data, except that it comprises of a significantly larger number of input vectors in order to produce a reasonably accurate estimate of the theoretical error probability.

### (2) Test Error Plots

Figure 3 illustrates how the empirical test error varies with the number of training points. As was the case in **Part A**, the error probability (the region shaded in yellow) roughly decreases with an increasing number of training points.

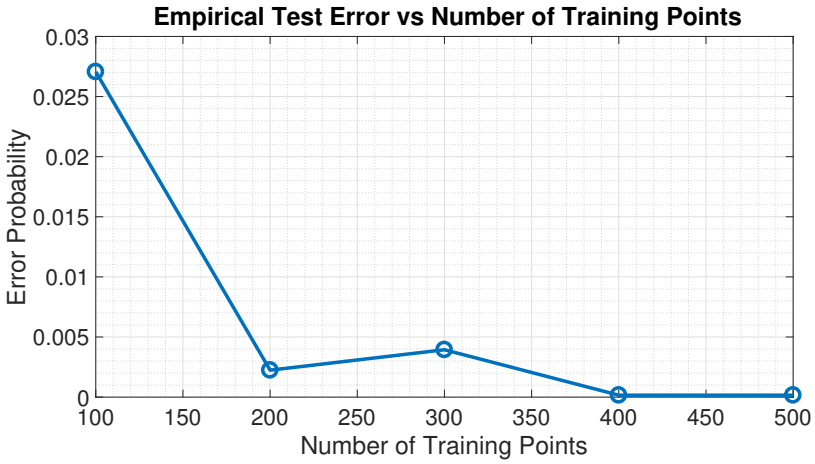


*Figure 3: Empirical Test Error Vs Number of Training Points*

### (3) 90% Confidence Interval Plot for Test Error Values

The test error values shown in Figure 3 are random to a certain extent since they inherently depend on a randomly generated training data set. We can obtain a more reliable plot by repeating the experiment multiple times.

Figure 4 illustrates a 90% confidence interval for the test error plotted against the number of training points. These results were obtained by repeating the test error calculations 100 different times for randomly generated training data sets of size $N = 100, 200, ..., 500$. Figure 4 depicts the same downward trend error probability with an increasing number of training points that we observed in Figure 3 and Figure 2.

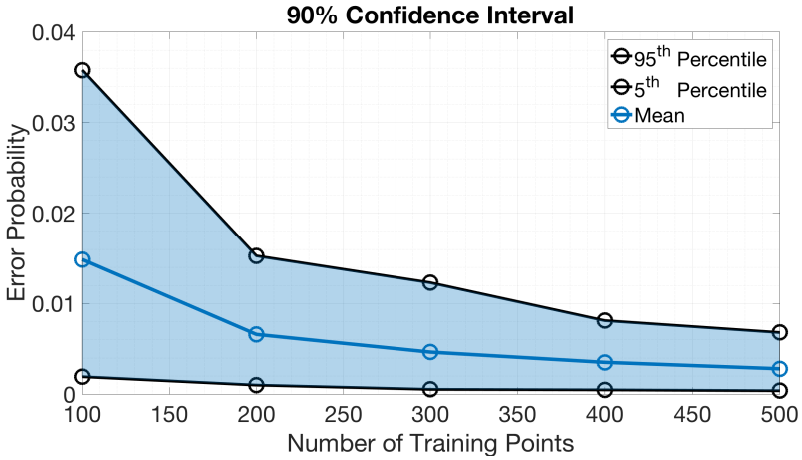


*Figure 4: 90% Confidence Interval for Test Error vs Number of Training Points*

## (4) Impact of Using Margin Separated Test Data

In this section, we will investigate the impact that the size of the margin in a margin separated test data set has on the test error. We will produce graphs in the same manner that we have done thus far, except that the equation for the *true* classifier is now given by $|x_2 - x_1 - 0.1| > \gamma$. The use of the absolute operator leads to two true classifiers that are separated by a margin. The *width* of this margin is governed by $\gamma$.

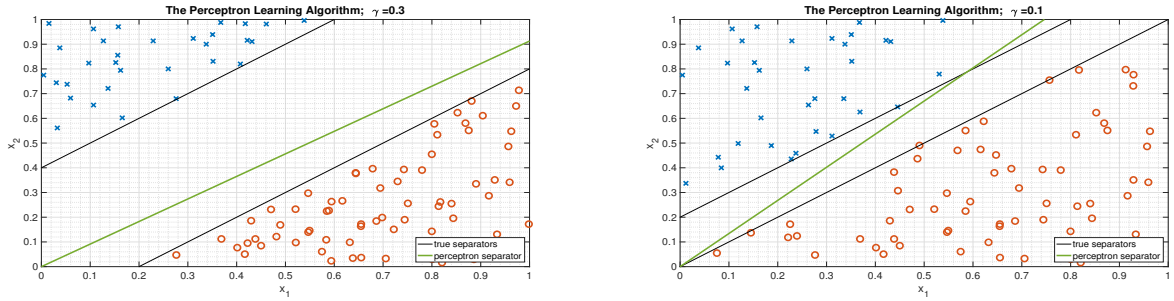


*Figure 5: Perceptron Learning Algorithm with $N = 100$ and $\gamma = 0.3$ and $\gamma = 0.1$*

Figure 5 and Figure 6 illustrate the linear classifier produced by the perceptron algorithm when the training data is subject to different sized margins. Note that the size of the training data set has been fixed to 100 feature vectors. While we have not made an explicit attempt to highlight the erroneous regions on these figures, we may still infer with reasonable confidence that the number of perceptron iterations/updates and the test error probabilities are likely to increase with a decreasing margin width. In order to confirm this hypothesis, We will now present the error probability plots seen in (2) and (3) for $\gamma = 0.3, 0.1, 0.01$ and $0.001$.

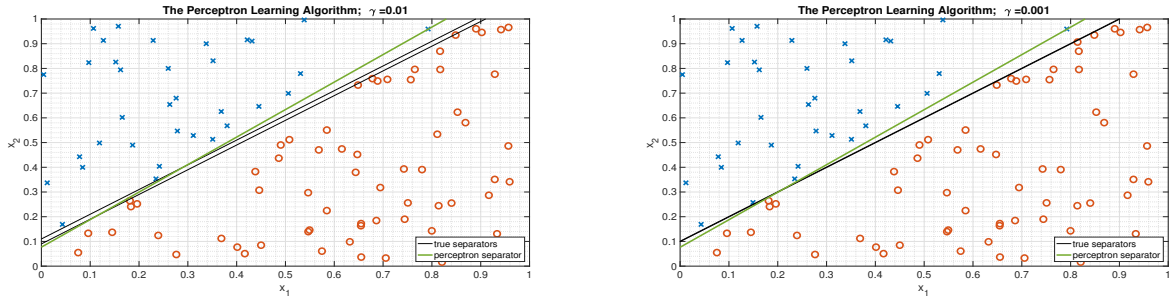


*Figure 6: Perceptron Learning Algorithm with $N = 100$ and $\gamma = 0.01$ and $\gamma = 0.001$*

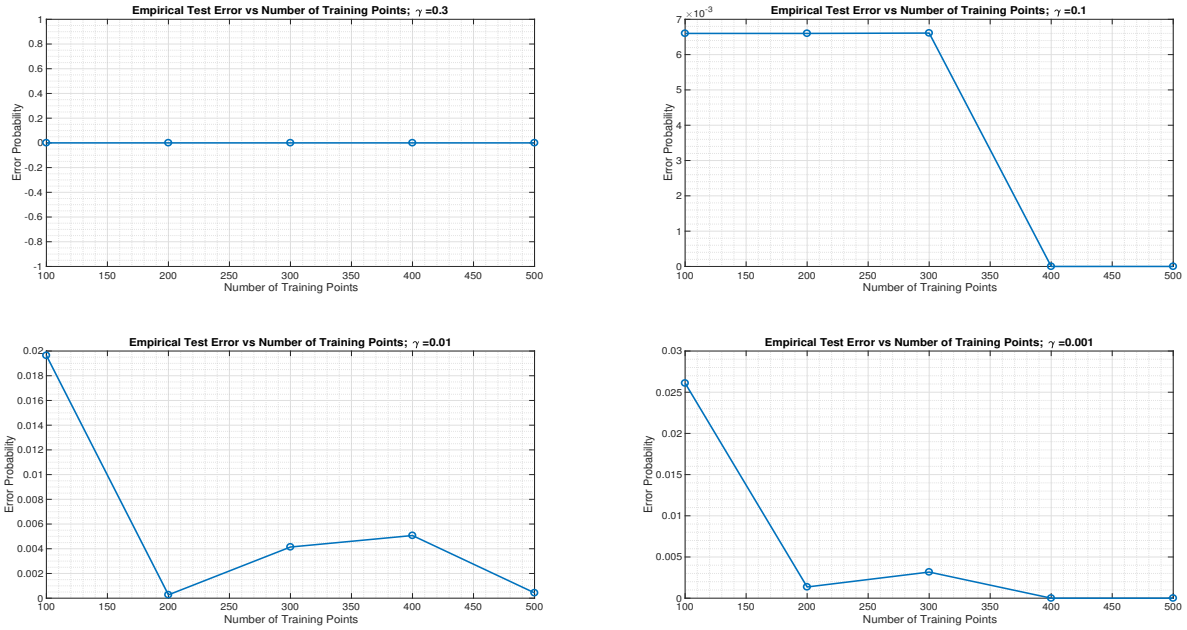


*Figure 7: Empirical Error for Different Training Data Margins*

Figure 7 shows the error probability results that were collected for a single training data set, whereas Figure 8 shows the average error probability results across $100 \times 5 = 500$ different trials. These results show that the error probability, in nominal terms, is much higher for the smaller values of $\gamma$. In fact, a margin specified by $\gamma = 0.3$ provides the perceptron algorithm with generous headroom to not only converge almost immediately, but to also converge in a manner that almost completely eliminates any intersections with the true classifier lines.

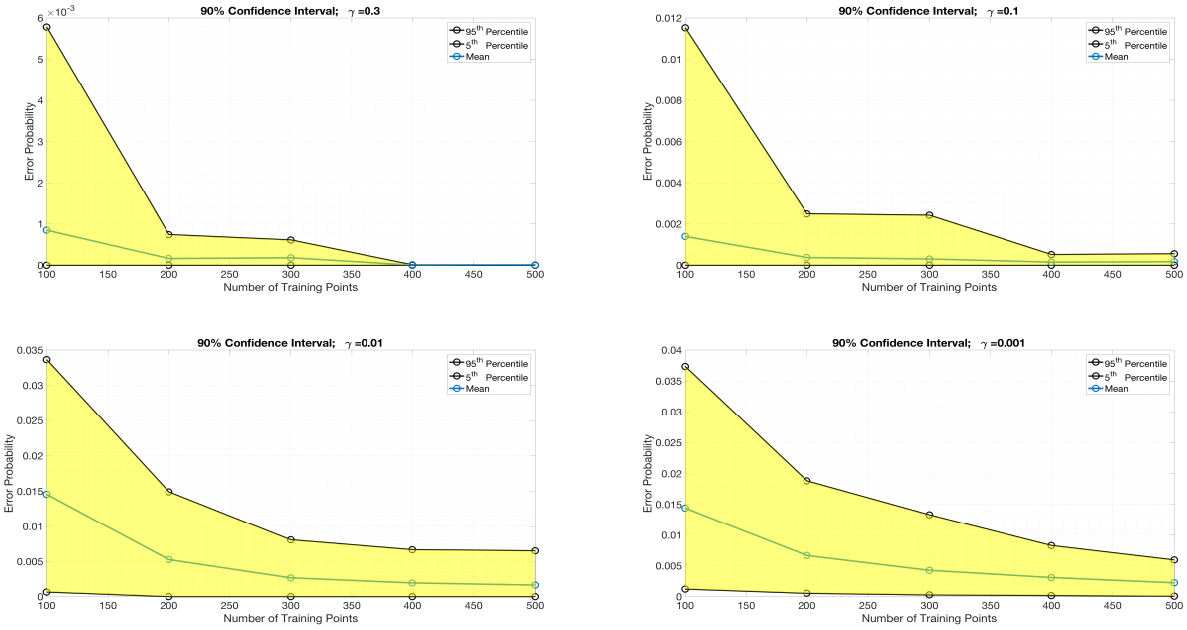


*Figure 8: Average Empirical Error for Different Training Data Margins*

Figure 9 illustrates four different plots, each corresponding to a particular value of $\gamma$, that depict the number of both actual perceptron iterations/updates as well as the theoretical maximum against the number of training data points. As expected, we observe an increasing number of both quantities with a decreasing margin width. Note that these are averaged values plotted on a log-linear scale.

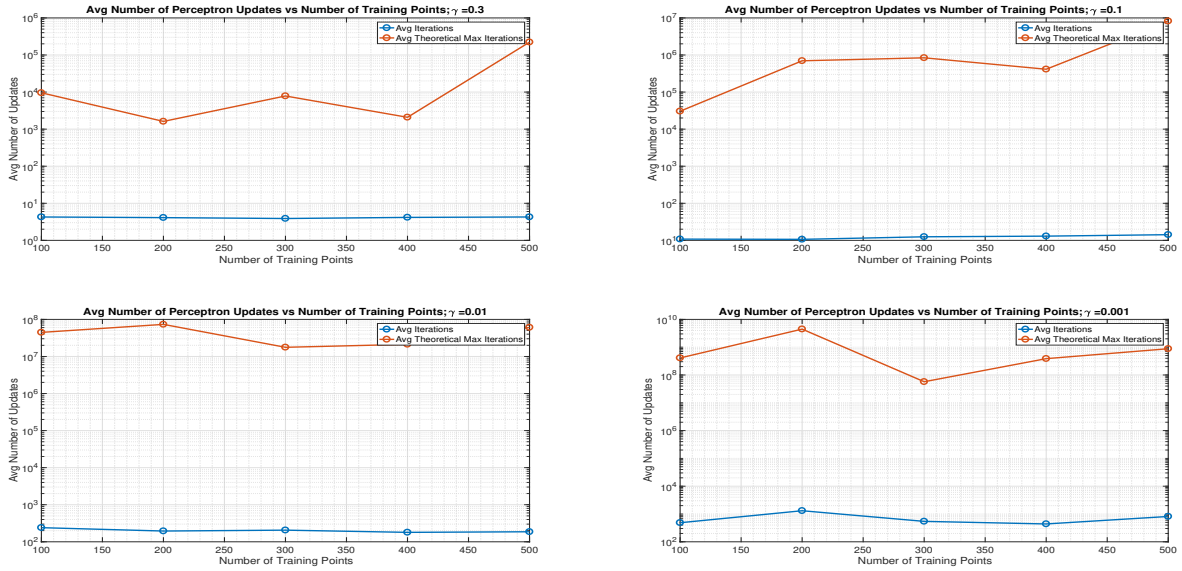


*Figure 9: Average Number of Perceptron Updates for Different Training Data Margins*

## Part C - Handwritten Digit Classification

### (1) PerceptronPro

Our attempt at improving the original perceptron algorithm can be categorised into two. First, we include a simple stopping condition that exits the function in case the algorithm is provided with input data that is not linearly-separable. Secondly, we have modified the way in which we choose a specific feature vector from the set of all feature vectors that are wrongly classified at any given iteration.

The original perceptron function would iterate through the data set from the first index to the last, using any misclassified points found in that process to iteratively update the weight vector. Our modification entails pre-computing the product $y^j \mathbf{w}_{k-1}^T \mathbf{x}_{k-1}^j$ for $j = 1, 2, ..., N$ at iteration $k$, since the numerical value associated with each scalar product represents the extent to which that feature vector is correctly or incorrectly classified.

Our initial approach involved always selecting the feature vector that was most incorrectly classified - i.e. the input vector that produces the largest negative value for the product $y^j \mathbf{w}_{k-1}^T \mathbf{x}_{k-1}^j$ for $j = 1, 2, ..., M$, where $M \leq N$ represents the number of misclassified points. When tested on the raw handwriting data, we observed that this approach reduced the number of updates carried out in classifying the raw data from 441 to 364.

However, despite the fact that the 2-D feature data that has been provided is not linearly separable without a positive training error, we chose to modify our aggressive strategy since it did not perform well on the 2-D data set. Furthermore, using the worst case input vector to update the weights is more likely to lead to a scenario where the algorithm oscillates between two extreme points, potentially producing a weight vector (when the function exits at the maximum number of iterations) that seems to have forgone any classification attempt.

Therefore, we have ultimately settled on using the feature vector corresponding to the *median* $y^j \mathbf{w}_{k-1}^T \mathbf{x}_{k-1}^j$ value, as opposed to using the *minimum* value to update the weights. In the rest of the section, `perceptronPro` will refer to a modified version of the perceptron algorithm that uses the median approach.

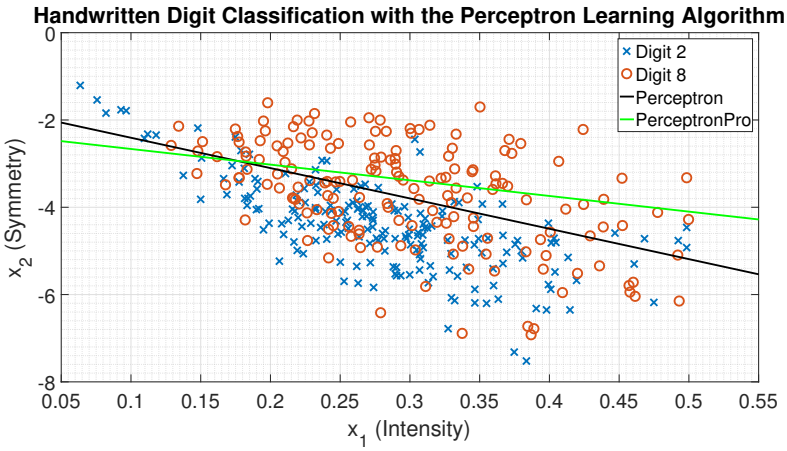


*Figure 10: Handwritten Digit Classification with the Perceptron Learning Algorithm*

Figure 10 illustrates the lines produced by both the original and modified perceptron algorithm. Note that the markers in the background correspond to test data. The table below illustrates the best-case training and test error results that were obtained with our modified perceptron algorithm.

| Type of Data | Training Error (%) | Testing Error (%) |
|---|---|---|
| **256 Dimensional Raw Data** | 0.00 | 4.40 |
| **2 Dimensional Feature Vector** | 25.22 | 30.22 |

*Table 1: Training and Test Errors for Two Cases*

### (2) Training and Test Errors

Figure 11 illustrates a plot of the training error and the test error values that are logged at each iteration of the original and modified perceptron algorithm. Figure 11 shows results for the case where both algorithms are evaluated against raw data, whereas Figure 12 shows results for the case where the 2-D (linearly inseparable) dataset is used instead.

The graphs in Figure 11 depict the difference our simple modification has made on the rate of convergence. As expected, the test error is almost always higher than the training error at each iteration. Figure 12 shows some interesting results for two algorithms that are unable to converge to a zero value for the training error. The error values for the original function are seen to oscillate frequently whereas those logged for the modified function do not demonstrate such regularity.

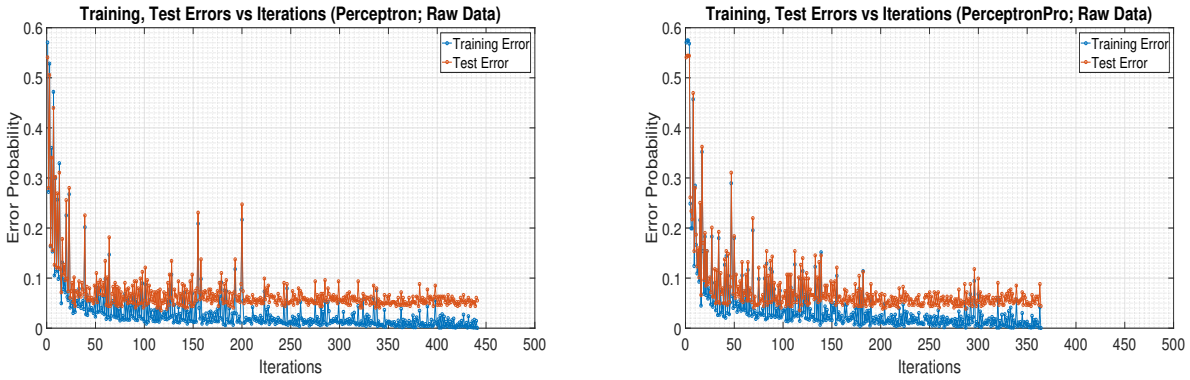


*Figure 11: Training and Test Errors vs Number of Iterations for Raw Data*

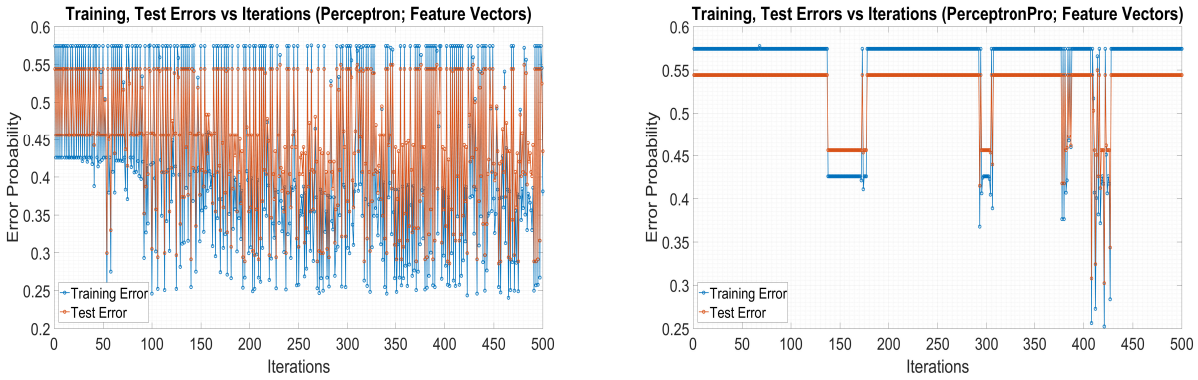


*Figure 12: Training and Test Errors vs Number of Iterations for 2-D Features Data*

## (3) Optimal Linear Regression Weights

In this section, we reproduce Figure 11 and Figure 12 for the 2-D feature vector case when the perceptron algorithms are initialised with optimal linear regression weights. The resulting training and test error graphs are shown in Figure 13. We do not observe any noticeable differences in the overall evolution of the test and training error when using linear regression weights to initialise the perceptron algorithm.

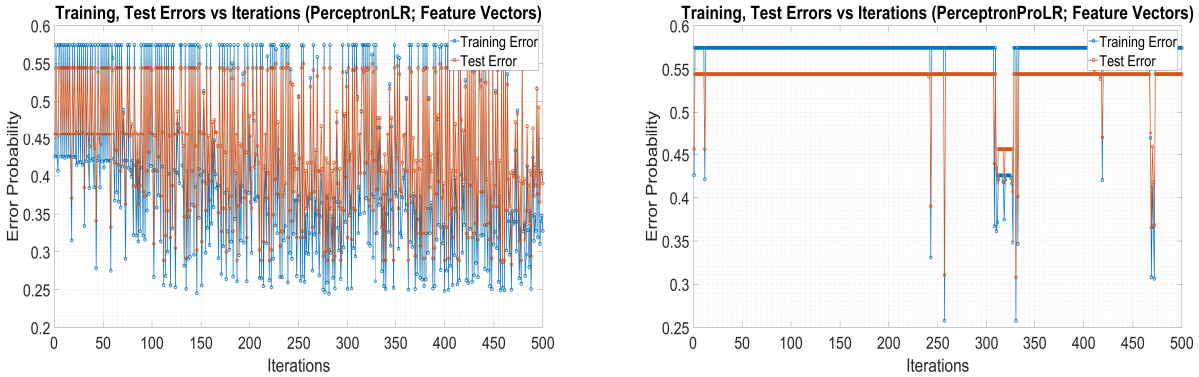


*Figure 13: Training and Test Error Data for perceptron initialised with Linear Regression weights*

## (4) Probability bounds

We would like to compute a high-probability upper bound on the difference between the empirical test error against true test error. We will invoke Hoeffding's inequality as shown in Equation 4 in order to provide this bound.

$$\Pr\left(\left|\bar{X} - \mathbb{E}(\bar{X})\right| \geq \epsilon\right) \leqslant 2 \cdot \exp\left(-2n\epsilon^2\right) \tag{4}$$

Equation 5 represents an alternative representation of the same inequality. In this form, Hoeffding's inequality can be applied to our own problem where we would like to find out a high probability (RHS) upper bound $\epsilon$ on the difference between an empirical quantity and its theoretical mean.

$$\Pr\left(\left|\bar{X} - \mathbb{E}(\bar{X})\right| < \epsilon\right) \leqslant 1 - 2 \cdot \exp\left(-2n\epsilon^2\right) \tag{5}$$

Setting n = 1273 (the number of training data points) and solving for $\epsilon$ such that the upper bound holds with 90% certainty produces $\epsilon = 3.43\%$.