Introduction to Machine Learning, Assignment 1

Diederik Vink

July 12, 2017

Contents

1 Problem 1 **1**

2 Problem 2 **2**
2.1 Question 2a . . . 2
2.2 Question 2b . . . 2
2.3 Question 2c . . . 3
2.4 Question 2d . . . 4

3 Problem 3 **4**
3.1 Problem 3a . . . 4
3.2 Problem 3b - Calculating Test Error . . . 6
3.2.1 Problem 3b1 . . . 6
3.2.2 Problem 3b2 . . . 10
3.2.3 Problem 3b3 . . . 12
3.2.4 Problem 3b4 . . . 13
3.3 Problem 3c - Handwritten Text Classification . . . 18
3.3.1 Problem 3c1 . . . 18
3.3.2 Problem 3c2 . . . 19
3.3.3 Problem 3c3 . . . 20
3.3.4 Problem 3c4 . . . 21

Pledge

I, Diederik Adriaan Vink, 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 Problem 1

When answering Problem 1, the first thing to consider is what a confidence level entails. According to [Berman, 2017] a confidence level "refers to the percentage of all possible samples that can be expected to include the true population parameter".

The confidence level for a specific situation can be found using the one-sided Hoeffding's inequality. The one-sided Hoeffding's inequality is derived in two parts from equation (1).

$$P(|\nu - \mu| \geq \epsilon) \leq 2e^{-2n\epsilon^2} \tag{1}$$

From equation (1), both variations of the one-sided Hoeffding's inequalities can be derived. The variation of the inequality that is applicable to Problem 1 can be seen in equation(2)

$$P(-\nu + \mu \geq \epsilon) \leq e^{-2n\epsilon^2} \tag{2}$$

which is simplified to equation (3):

$$P(\mu \geq \nu + \epsilon) \leq e^{-2n\epsilon^2} \tag{3}$$

where n = sample size, ν = sample mean, μ = population mean and $\epsilon \geq 0$ [?].

After considering the equation above, this one sided inequality can be interpreted as the level of significance α for a confidence around θ of size ϵ. As a result equation (3) can be rewritten as equation (4).

$$P(\mu \geq \nu + \epsilon) \leq e^{-2n\epsilon^2} \leq \alpha \tag{4}$$

Equation (5) explicitly states the implication of equation (4). Both equations demonstrates that we are trying to find the probability that μ is outside of the range of $\nu + \epsilon$, and use this as a lower bound for our level of significance.

$$P(\mu \notin [\nu + \epsilon]) \leq e^{-2n\epsilon^2} \leq \alpha \tag{5}$$

Solving equation (5) provides the number of observations n needed in order for $\nu = \mu - \epsilon$ with a $1 - \alpha$ confidence level.

In the situation that has been presented in Problem 1, it is stated that the population mean $\nu = 0.45$. Furthermore, we want to know the confidence level that $\mu \geq 0.5$, as that is the condition upon which Brexit will occur. So referring to equation (5) it can be deduced that $\epsilon = 0.05$. Furthermore, it is stated that $n = 2052$.

When applying these values to the equation (4), the following working and result are produced.

$$\begin{aligned} P(\mu \geq \nu - \epsilon) \leq e^{-2(2052)(0.05)^2} &\leq \alpha \\ P(\mu \geq 0.5) \leq e^{-2(2052)(0.05)^2} &\leq \alpha \\ \therefore 0.00003500568784 &\leq \alpha \end{aligned}$$

As a result, it can be concluded that the confidence level is $1 - \alpha \geq 0.99996499431$

2 Problem 2

Problem 2 was a lot less theoretical and a lot more practical working than problem 1.

2.1 Question 2a

Question 2a) asked to show that $\rho > 0$ in equation (6).

$$\rho = min_{i \in \{1,\ldots,n\}} y_i w_*^T x_i \tag{6}$$

If $w_t = w_*^T$ then we know that $sign(w_*^T x_t) = y_t$. From this we can deduce that $y_i w_*^T x_i$ will always be positive as y_i and $w_*^T x_i$ will always have the same sign. As a result we can guarantee that $\rho > 0$.

2.2 Question 2b

Question 2b) aimed to prove

$$w_*^T w_t \geq t\rho \tag{7}$$

for any t before the algorithm stopped.
Following the hint provided on the sheet, the first step was to show that $w_*^T w_t \geq w_*^T w_{t-1} + \rho$. This was done starting with:

$$w_t = w_{t-1} + y_{t-1} x_{t-1} \tag{8}$$

We know from question 2a) and equation (6), that $\rho = min_{i \in \{1,\ldots,n\}} y_i w_*^T x_i$, meaning that at all times:

$$\rho \leq w_*^T y_{t-1} x_{t-1} \tag{9}$$

Using this information, we can multiply equation (8) by w_*^T, resulting in equation (10).

$$w_*^T w_t = w_*^T w_{t-1} + w_*^T y_{t-1} x_{t-1} \tag{10}$$

Using equation (10) and equation (9), we can show that:

$$w_*^T w_t \geq w_*^T w_{t-1} + \rho \tag{11}$$

From here we can proceed to use induction to prove that $w_*^T w_t \geq t\rho$.
At first we prove that $w_*^T w_t \geq t\rho$ works for when $t = 1$:

$$\begin{aligned} w_*^T w_1 &\geq w_*^T w_{1-1} + \rho \\ &\geq w_*^T w_0 + \rho \\ &\geq \rho \end{aligned}$$

considering $w_0 = 0$ as stated in the question.
Next we assume that $w_*^T w_t \geq t\rho$ is true for when $t = k$:

$$\begin{aligned} w_*^T w_k &\geq w_*^T w_{k-1} + \rho \\ &\geq k\rho \end{aligned}$$

Finally we show that $w_*^T w_t \geq t\rho$ works for $t = k+1$ to inductively prove out equation:

$$\begin{aligned} w_*^T w_{k+1} &\geq w_*^T w_k + \rho \\ &\geq k\rho + \rho \\ &\geq (k+1)\rho \\ &\underline{QED} \end{aligned}$$

and knowing that $t = k+1$

$$w_*^T w_t \geq t\rho$$

2.3 Question 2c

Question 2c) aimed to prove that:

$$\|w_t\|^2 \leq tR^2 \tag{12}$$

where $R = max_{i \in \{1,\dots,n\}} \|x_i\|$. This question was answered by first showing that:

$$\|w_t\|^2 \leq \|w_{t-1}\|^2 + \|x_{t-1}\|^2 \tag{13}$$

Equation (13) was formed multiplying equation (8)'s transpose by itself and simplifying it, as shown below:

$$\begin{aligned} w_t &= w_{t-1} + y_{t-1} x_{t-1} \\ w_t^T w_t &= (w_{t-1} + y_{t-1} x_{t-1})^T w_{t-1} + y_{t-1} x_{t-1} \\ \|w_t\|^2 &= \|w_{t-1}\|^2 + \|x_{t-1}\|^2 + w_{t-1}^T (y_{t-1} x_{t-1}) + w_{t-1} (y_{t-1} x_{t-1})^t \\ &= \|w_{t-1}\|^2 + \|x_{t-1}\|^2 + 2w_{t-1}^T (y_{t-1} x_{t-1}) \\ &\leq \|w_{t-1}\|^2 + \|x_{t-1}\|^2 \end{aligned}$$

After acquiring equation (13), we can use induction to prove equation (12). The first step is proving equation (12) works for the case where $t = 1$, as shown below:

$$\begin{aligned} \|w_1\|^2 &\leq \|w_0\|^2 + \|x_0\|^2 \\ \|w_1\|^2 &\leq \|x_0\|^2 \\ \|w_1\|^2 &\leq (1)R^2 \end{aligned}$$

considering that $R \geq \|x_0\|^2$. Secondly, we assume that equation (12) works for the case where $t = k$, resulting in:

$$\begin{aligned} \|w_k\|^2 &\leq \|w_{k-1}\|^2 + \|x_{k-1}\|^2 \\ \|w_k\|^2 &\leq kR^2 \end{aligned}$$

Finally we aim to prove that equation (12) works for the case where $t = k + 1$, as shown below:

$$\begin{aligned}\|w_{k+1}\|^2 &\leq \|w_k\|^2 + \|x_k\|^2 \\ &\leq kR^2 + \|x_k\|^2 \\ &\leq kR^2 + R^2 \\ &\leq (k+1)R^2 \\ &\underline{QED}\end{aligned}$$

2.4 Question 2d

Question 2d) aimed to show that $t \leq \frac{R^2\|w_*\|}{\rho^2}$ and conclude that the algorithm ended after no more than $\frac{R^2\|w_*\|}{\rho^2}$ iterations.
To answer question 2d) equations (7) and (12) need to be used together. First, equation (7) can be rewritten as follows using the Cauchy-Schwarz inequality:

$$\begin{aligned}w_*^T w_t &\geq t\rho \\ \|w_*\|\|w_t\| &\geq t\rho \\ \|w_*\|^2\|w_t\|^2 &\geq t^2\rho^2\end{aligned}$$

Now, combining the modified version of equation (7) and applying that to equation (12) which is mutliplied by $\|w*\|^2$ it becomes possible to solve the problem as follows:

$$\begin{aligned}tR^2\|w_*\|^2 &\geq \|w_t\|^2\|w_*\|^2 \\ &\geq t^2\rho^2 \\ \frac{R^2\|w_*\|^2 t}{\rho^2} &\geq t^2 \\ \frac{R^2\|w_*\|^2}{\rho^2} &\geq t\end{aligned}$$

The conclusion that the algorithm will end after no more than $\frac{R^2\|w_*\|}{\rho^2}$ iterations can easily be made, it is clear that $\frac{R^2\|w_*\|}{\rho^2}$ is the upper bound on the number of iterations t. As a result, t will never exceed $\frac{R^2\|w_*\|}{\rho^2}$.

3 Problem 3

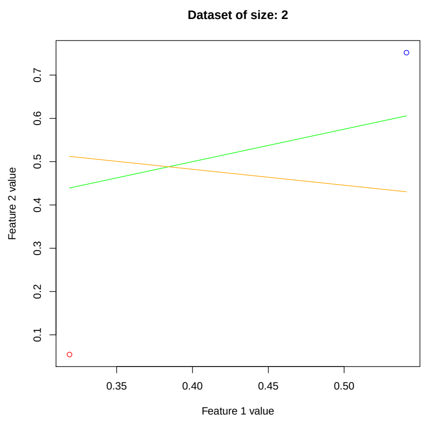
3.1 Problem 3a

The Perceptron algorithm that has been implemented in the attached code, and has been fully commented to make it more understandable. As a result the fine details of the code will not be discussed here. The Perceptron algorithm that was provided on the question sheet is the one that was implemented in the code.

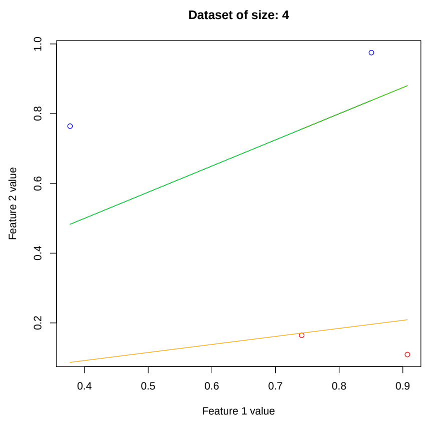
When running the Perceptron algorithm, the next point to update was selected as the first point to fail when comparing $h(x_i) = sign(w^T x_i)$ to y_i. Such a decision can produce a different result for w than if the point to update was selected in another way (such as randomly, the last point or any other method). There are multiple viable results for w_* that are feasible (as will be shown in Question 3b4) because the hypothesis class is infinite. As a result, all of these methods are viable methods, but can lead to different values of w_*. The test error will vary accordingly, as this is produced as a direct result from w.

The dataset was generated using a uniform distribution between 0 and 1. The exact details of this implementation have been annotated within the attached code. The algorithm in the code works for both Problem 3a and Problem 3b. The generation algorithm ensures that half of the data points are uniformly distributed to the left of the target line and half of the data points are uniformly distributed to the right of the target line. This is the most obvious in Figure 1d. Due to this separation, the data points are guaranteed to be linearly separable, and as a result, if the number of iterations is set high enough, the perceptron algorithm should always arrive at a w_*. As a result when testing the algorithm, if it is unable to reach a feasible value of w_*, there is clearly a flaw in the algorithm. The upper bound on iterations can get very high if there is a very high density of points. This upper bound is calculated and presented later on in the report.

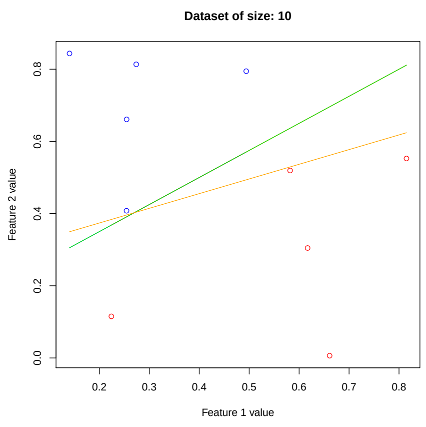
The graphs presented in Figure 1 demonstrate how the Perceptron algorithm functioned with varying dataset sizes. In these graphs, the green line represents the line produced from the w_* values and the orange line represents the original line used to classify the points. The original line follows the equation of $y = 0.75x + 0.2$. The blue dots are all the points with $y = 1$ and all the red dots are the points with $y = -1$. It is clear that as the size of the dataset increases, the Perceptron algorithm manages to produce a better estimate of the target (original) line. This is because the points are more densely packed and will sit closer to the target line. Therefore, the Perceptron algorithm will have a lower error margin and have a higher chance of producing a better estimate. This will be reaffirmed in Problem 3b3 using both theory and data analysis. Furthermore, the more points that are present, the higher the average number of iterations will be to find a value for w_*. This will be justified in Problem 3b4.



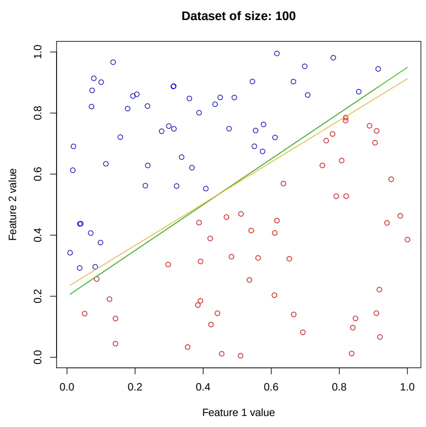
(a) *Size* $= 2$



(b) *Size* $= 4$



(c) *Size* $= 10$



(d) *Size* $= 100$

Figure 1:
Blue dot = 1
Red dot = -1
Orange = Original line
Green = Perceptron-generated line

3.2 Problem 3b - Calculating Test Error

Problem 3b aims to calculate the test error that is produced by the perceptron algorithm within the closed region of the unit square. The points are uniformly distributed across the square using the same algorithm as in 3a, where the data points are distributed across the region $K_\gamma = (x_1, x_2) \in [0,1]^2 : |x_2 - x_1 - 0.1| > \gamma$ for $\gamma = 0.3, 0.1, 0.01, 0.001, 0$ in the same way as in Problem 3a. The implementation of this is annotated in the attached code.

3.2.1 Problem 3b1

Problem 3b1 required finding a closed for formula for the test error of a linear separator $ax + b$ in the situation prescribed for Problem 3b. The test error is equivalent to the area between the linear separator and the target area of K_γ within the unit square. The generic equation that calculates this area is portrayed in equation (14):

$$Area = (\int_{lim_{1.1}}^{lim_{1.2}} y_1 \; dx + \int_{lim_{1.2}}^{lim_{1.3}} \; dx) - (\int_{lim_{2.1}}^{lim_{2.2}} y_2 \; dx + \int_{lim_{2.2}}^{lim_{2.3}} \; dx) \tag{14}$$

In this equation, either y_1 or y_2 is the separator line and the other is the lower and upper bound of the region K_γ. When $\gamma = 0$ this value is just the line of $x + 0.1$. y_1 is always the line that is above y_2 within the unit square, and the method of deciding whether the separator line is y_1 or y_2 will be discussed later on in this report. As this equation is an integral, this is not yet a closed form equation, but this is merely the area between two lines within the unit square, when the limits are correctly set. A closed form of this equation can be derived as follows:

$$\begin{aligned}
Area_1 &= (\int_{lim_{1.1}}^{lim_{1.2}} y_1 \; dx + \int_{lim_{1.2}}^{lim_{1.3}} \; dx) && (15)\\
&= (\frac{1}{2} * a * (lim_{1.2}^2 - lim_{1.1}^2)) + ((b-1) * lim_{1.2}) - (b * lim_{1.1}) + lim_{1.3} && (16)\\
Area_2 &= (\int_{lim_{2.1}}^{lim_{2.2}} y_2 \; dx + \int_{lim_{2.2}}^{lim_{2.3}} \; dx) && (17)\\
&= (\frac{1}{2} * c * (lim_{2.2}^2 - lim_{2.1}^2)) + ((d-1) * lim_{2.2}) - (d * lim_{2.1}) + lim_{2.3} && (18)\\
Area &= Area_1 - Area_2 && (19)
\end{aligned}$$

Equation (19) is the closed form equation that will calculate the area between two straight lines where $y_1 = ax + b$ and $y_2 = cx + d$. The limits $lim_{1.1}$, $lim_{1.2}$, $lim_{1.3}$, $lim_{2.1}$, $lim_{2.2}$ and $lim_{2.3}$ are determined depending on the relative position of y_1 and y_2 to the unit square. The pseudocode below demonstrates how limits $lim_{1.1}$, $lim_{1.2}$, $lim_{2.1}$ and $lim_{2.2}$ are determined.

Algorithm 1 Determine $limit_{1.1}$

```
procedure SET $limit_{1.1}$
    if $y_{1,xintercept} \leq 0$ then
        $lim_{1.1} \leftarrow 0$
    else if $(y_{1,xintercept} > 0)$ && $(y_{1,xintercept} < 1)$ then
        $lim_{1.1} \leftarrow y_{1,xintercept}$
    else if $(y_{1,xintercept} \geq 1)$ then
        return $(Area = 0)$
    end if
end procedure
```

Algorithm 2 Determine $limit_{1.2}$

```
procedure Set $limit_{1.2}$
    if $y_{1,(y=1)intercept} \leq 0$ then
        $lim_{1.2} \leftarrow 0$
    else if $(y_{1,(y=1)intercept} > 0)$ && $(y_{1,(y=1)intercept} < 1)$ then
        $lim_{1.2} \leftarrow y_{1,(y=1)intercept}$
    else if $(y_{1,(y=1)intercept} \geq 1)$ then
        $lim_{1.2} \leftarrow 1$
    end if
end procedure
```

Algorithm 3 Determine $limit_{2.1}$

```
procedure Set $limit_{2.1}$
    if $y_{2,xintercept} \leq 0$ then
        $lim_{2.1} \leftarrow 0$
    else if $(y_{2,xintercept} > 0)$ && $(y_{2,xintercept} < 1)$ then
        $lim_{2.1} \leftarrow y_{2,xintercept}$
    else if $(y_{2,xintercept} \geq 1)$ then
        $lim_{2.1} \leftarrow 1$
    end if
end procedure
```

Algorithm 4 Determine $limit_{2.2}$

```
procedure Set $limit_{2.2}$
    if $y_{2,(y=1)intercept} \leq 0$ then
        return $(Area = 0)$
    else if $(y_{2,(y=1)intercept} > 0)$ && $(y_{2,(y=1)intercept} < 1)$ then
        $lim_{2.2} \leftarrow y_{2,(y=1)intercept}$
    else if $(y_{2,(y=1)intercept} \geq 1)$ then
        $lim_{2.2} \leftarrow 1$
    end if
end procedure
```

Algorithms 1, 2, 3 and 4 demonstrate how to determine $lim_{1.1}$, $lim_{1.2}$, $lim_{2.1}$ and $lim_{2.2}$, using the x-coordinates of where either y_1 or y_2 intersect with the x-axis or the line $y = 1$. This pseudocode is realized in the language r in the code inside the function "case".

After the limits $lim_{1.1}$, $lim_{1.2}$, $lim_{2.1}$ and $lim_{2.2}$ have been defined, the final limits of $lim_{1.3}$ and $lim_{2.3}$ need to be determined. This is done by knowing where y_1 and y_2 intersect. There are three cases of intersection, each one leading to a different values of $lim_{1.3}$ and $lim_{2.3}$. We will first define how the cases are defined, and then how they assign a value to $lim_{1.3}$ and $lim_{2.3}$.

Case 1 is classified by having $x_0 \leq 1$ or $y_0 \leq 1$, where x_0 and y_0 are the respective x and y coordinates of the intersection point between the two lines, as can be seen in Figure 2:

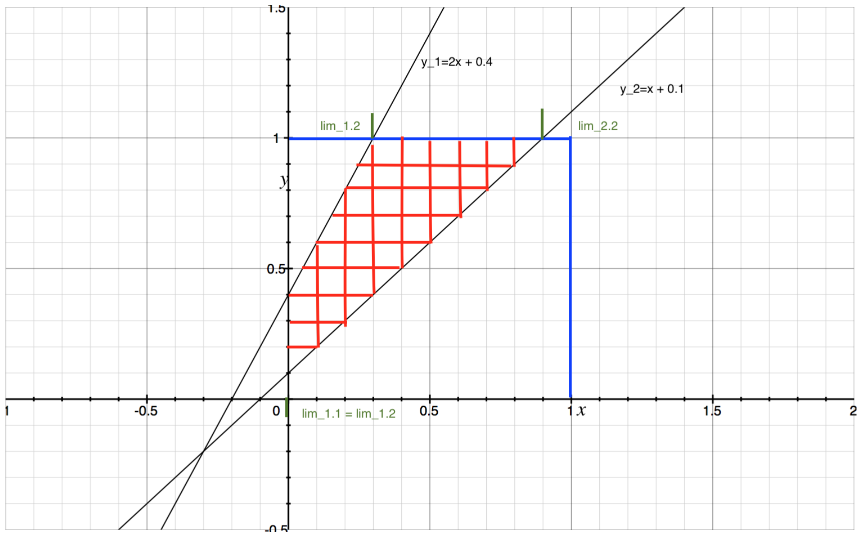


Figure 2: Case 1

The region within the blue lines indicates the unit square. The region in red indicates the error margin and the area that will be calculated. As can be seen here, the fixed equation that was used to classify the points is currently y_2. If the separator line were to have a smaller gradient than the fixed line, then the classification line would become y_1. This shows the versatility of the algorithm. The values of the limits $lim_{1.1}$, $lim_{1.2}$, $lim_{2.1}$ and $lim_{2.2}$ are determined by the algorithms discussed above.

Case 3 is classified by having $x_0 > 0$, $y_0 \geq 0$ and either $x_0 \geq 1$ or $y_0 \geq 1$, where x_0 and y_0 are again the respective x and y coordinates of intersection point. An example of Case 3 is shown below in the Figure 3:

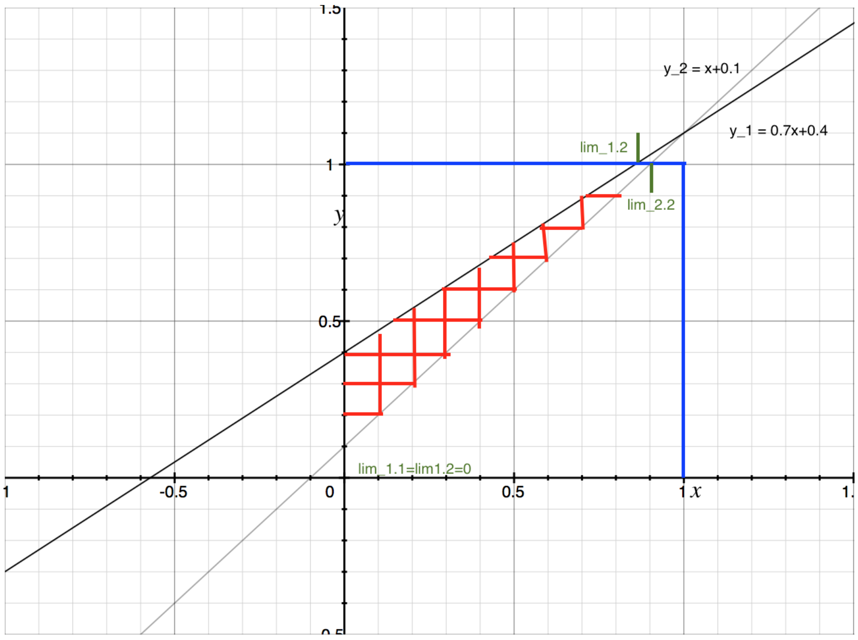


Figure 3: Case 3

Again, the blue lines indicate the unit square and the region in red the error margin area that is calculated. And just like in Case 1, the fixed line can be either y_1 or y_2 depending on whether it is the top or the bottom line within the unit square.

Case 2 is classified by having x_0 and y_0 within the unit square. An example of Case 2 is show in the Figure 4:

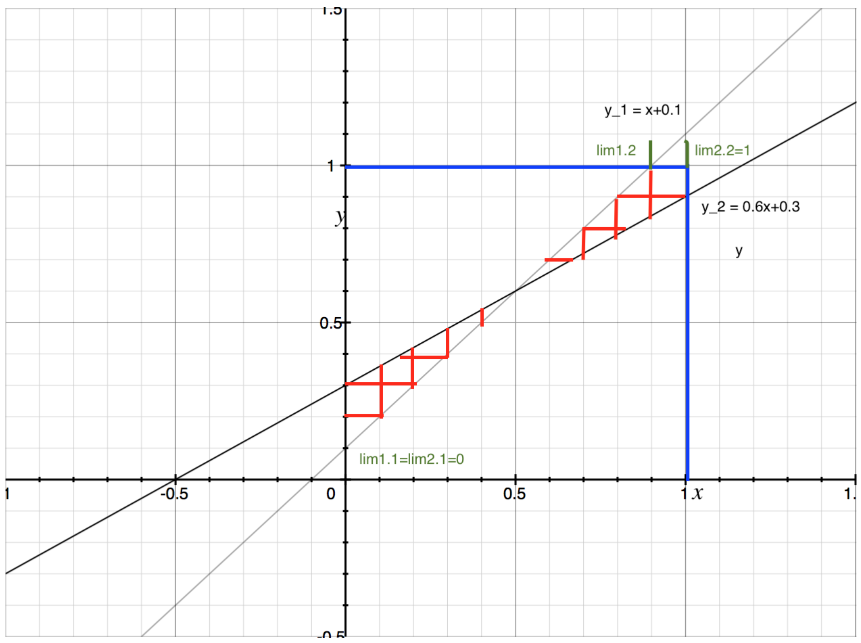


Figure 4: Case 2

Here as well, the blue lines indicate the unit square and the region in red the error margin area that is calculated. Just like before, to ensure that the general equation still works, the fixed line can be y_1 (like in this situation) or y_2.

Now that all of the cases have been described, it is possible to assign a value to $lim_{1.3}$ and $lim_{2.3}$. In both Cases 1 and 3 $lim_{1.3} = lim_{2.3} = 1$, as the area to be calculated is within the unit square. On the other hand, for Case 2, there are two areas to be calculated. The first area is the one of left of the intersection, where $lim_{1.2} = lim_{1.3} = lim_{2.2} = lim_{2.3} = x_0$ and the second area is the one on the right, where $lim_{1.1} = lim_{2.1} = x_0$, $lim_{1.3} = lim_{1.2}$ and $lim_{2.3} = lim_{2.2}$. The total area for Case 2 as a result is the sum of these two areas. For a code implementation, please refer to the attached code to the various ways in whic the function “line_integral” is called from the function “case”.

As mentioned before, in the closed form formula, it is left open as to which line (out of the separator and the fixed line) is considered to be y_1 and which line is y_2. This is only done when it is known which line is placed above the other within the unit square. As a result, the formula becomes more generic. All that is needed is the follow the pseudocode below to determine which line is the upper line.

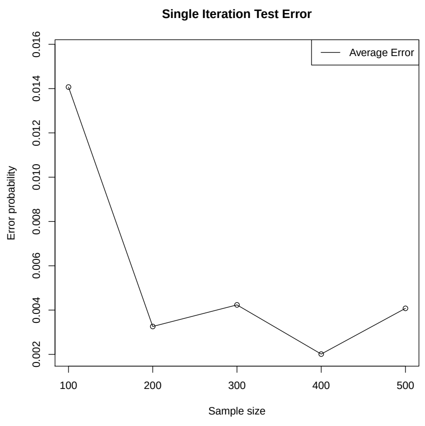
Algorithm 5 Determine y_1 and y_2

```
procedure FIND TOP LINE
    if $(case == 1) \|(case == 2)$ then
        if $line_1, (y = 1)intercept < line_2, (y = 1)intercept$ then
            $y_1 \leftarrow line_1$
            $y_2 \leftarrow line_2$
        else if $line_1, (y = 1)intercept > line_2, (y = 1)intercept$ then
            $y_1 \leftarrow line_2$
            $y_2 \leftarrow line_1$
        end if
    else if $(case == 3)$ then
        if $line_1, xintercept < line_2, xintercept$ then
            $y_1 \leftarrow line_1$
            $y_2 \leftarrow line_2$
        else if $line_1, xintercept > line_2, xintercept$ then
            $y_1 \leftarrow line_2$
            $y_2 \leftarrow line_1$
        end if
    end if
end procedure
```

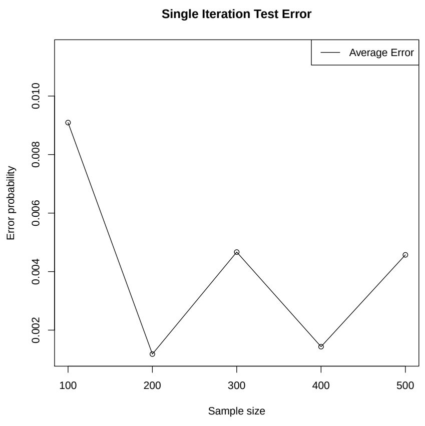
Using Algorithm 5 to determine the top line, one can appropriately call the closed frame formula with the correct values for all appropriate variables. This will calculate the test error that is created between the perceptron algorithm line and the line $x+0.1$. This formula will also work for Problem 3b4 for the region K_γ. The only change that needs to occur, is that the equation must first be called on the top boundary line, then on the bottom boundary line, and then the appropriate area values must be added together. This will be discussed in more detail in the relevant area of the report.

3.2.2 Problem 3b2

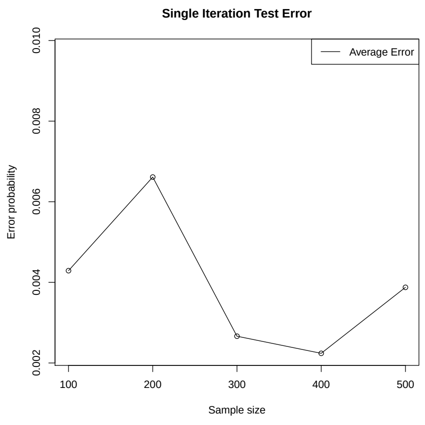
For Problem 3b2, the results of running this test are presented and discussed below. All of this data can be recreated using by calling "assignment_dav114.r" and viewing the graph in "iter1-gamma0-testerror.pdf". The raw data for this trail can also be found in the file "iter1-gamma0-values.out". Below Figure 5 demonstrates the results from six runs of training the algorithm with datasets of 100, 200, 300, 400 and 500 points.



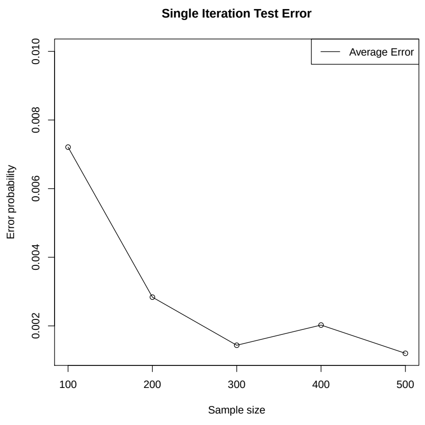
(a) $Trail1$



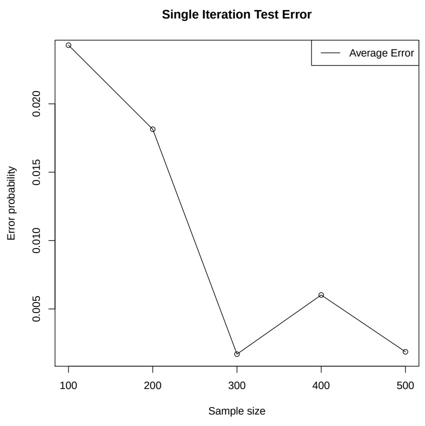
(b) $Trail2$



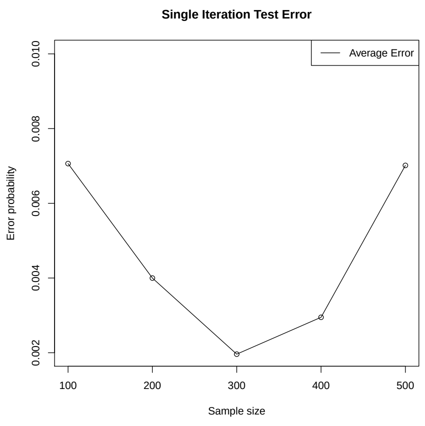
(c) $Trail3$



(d) $Trail4$



(e) $Trail5$



(f) $Trail5$

Figure 5: Single iteration test error graphs from 6 independent trails

As can be clearly noticed form the Figure 5, the error values and graph shape are seemingly random. The error probability change as sample size increases does not seem to follow any clear trend. The only thing that stays consistent it the scale of magnitude of the error. The value for the error was calculated as described in Problem 3b1. This is just a graphical representation of how this error varies over time.

3.2.3 Problem 3b3

For Problem 3b3, the same code was utilized as for Problem 3b2, except now the graphs display the average error over 100 trials, rather than just one trail. Furthermore, both the empirical 90% confidence interval and the 90% confidence interval calculated from Hoeffding's inequality are plotted as well. Figure 6 demonstrates that data.

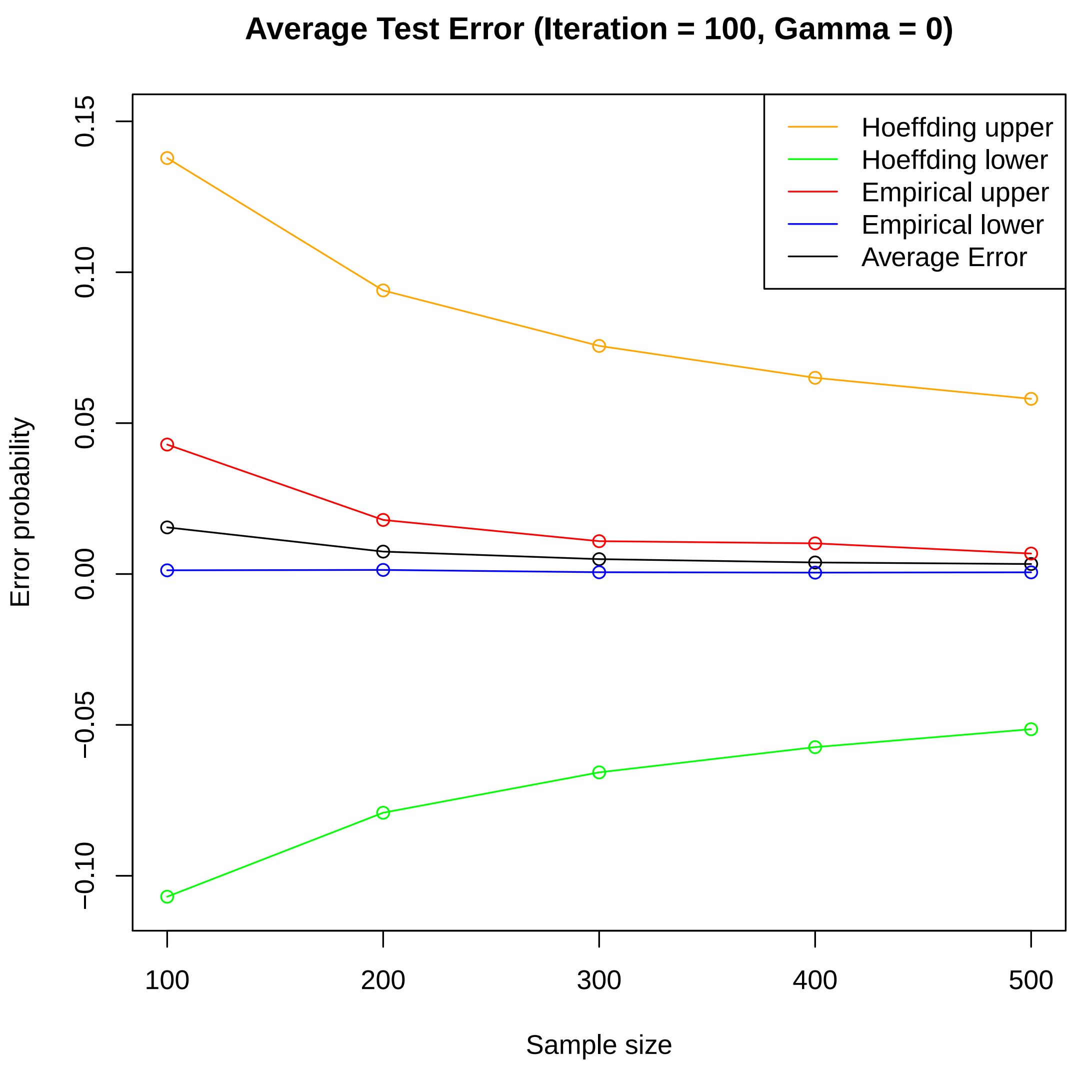


Figure 6: Test error graph from 100 independent trails

All graphs containing Average Test Errors have the same layout. The yellow line is the theoretical upper bound (calculated using Heoffding's) on the error probability of the 90% confidence interval. The red line is the empirical upper bound of the of 90% confidence interval. The green line is the theoretical lower bound (calculated using Heoffding's) on the error probability of the 90% confidence interval. The blue line is the empirical lower bound of the of 90% confidence interval. The black line is the average test error probability.

As can be seen from Figure 6, the average test error now follows a clear trend of a decreasing average error as the number of iterations increases. This goes to show, that even though on individual trials there does not seem to be a clear trend for the error values, when taking the average across a larger amount of trails, there is a clear trend. This decreasing trend also reaffirms what was mentioned in Problem 3a, that an increased sample size results in a smaller test error probability. The average test error is always within empirical 90% confidence interval (as is to be expected). Furthermore, the 90% confidence interval that is calculated using Hoeffding's inequality is clearly far above the empirical 90% confidence interval. This should always be the case, as Hoeffding's inequality provides an upper bound for the 90% confidence interval, so the empirical values should never exceed the Hoeffding 90% interval. Beyond just the graph, all of this information can be found in the corresponding raw data file iter100-gamma0-values.out, located in the ./results/Q3b/ folder. If the emperical confidence intervals exceed the bounds calculated by Hoeffding's inequality, there has clearly been a mistake in the test error calculation.

3.2.4 Problem 3b4

Problem 3b4 is a variation upon 3b1,3b2 and 3b3. The only difference is that the data is no longer uniformly distributed in the unit square and classified along the line $x + 0.1$, but now the data is uniformly distrbuted across the region $K_\gamma = (x_1, x_2) \in [0,1]^2 : |x_2 - x_1 - 0.1| > \gamma$ for $\gamma = 0.3, 0.1, 0.01, 0.001$. This means that there will be a zone where no points will be placed. An example of the data distribution from Problem 3b4 is shown below to give and idea of what this distribution is to look like.

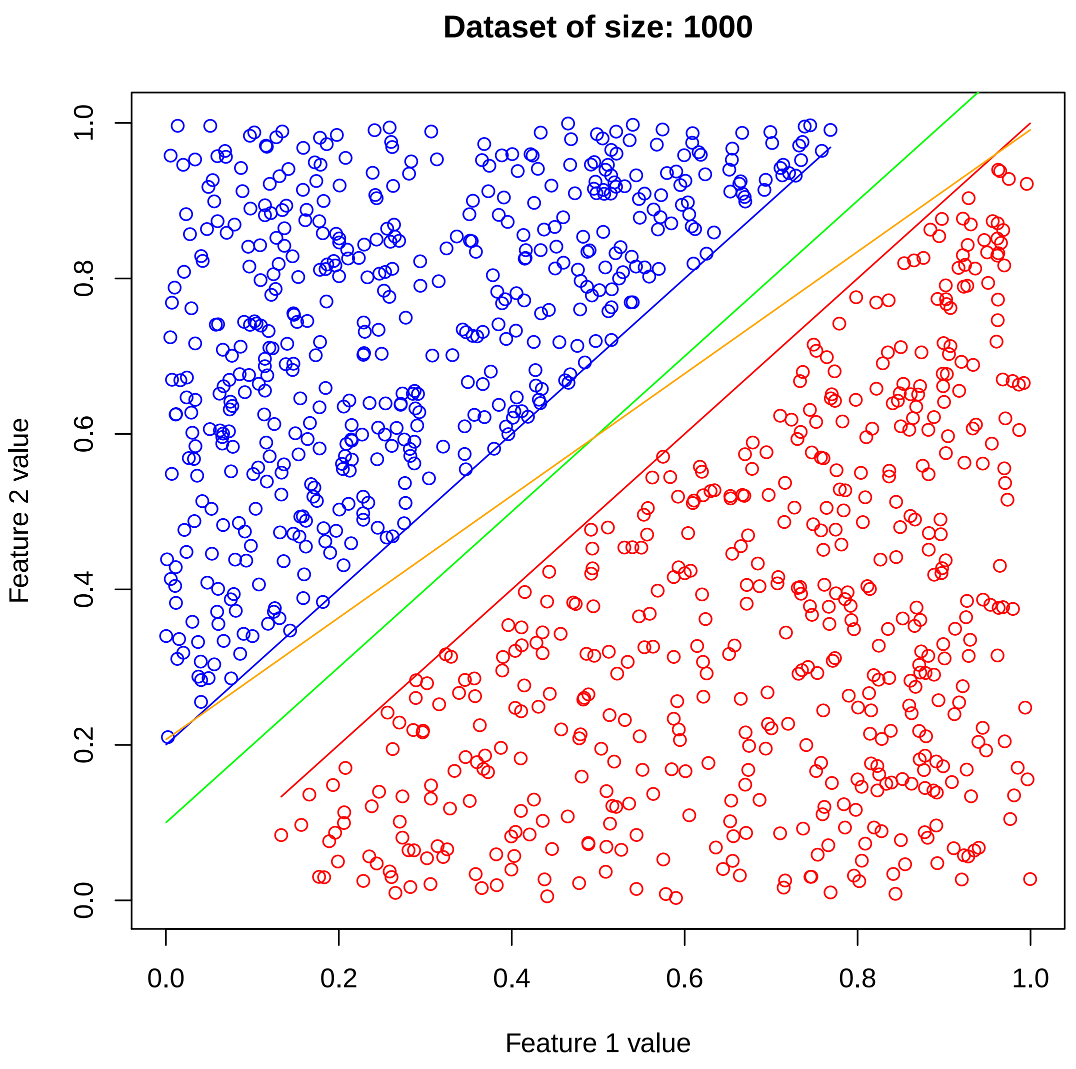


Figure 7: Distribution of 1000 points, with $\gamma = 0.1$

Figure 7 shows the distribution of 1000 points uniformly distributed outside of the region of of K_γ. The green, blue and red lines represent $x + 0.1$, $x + 0.1 + \gamma$ and $x + 0.1 - \gamma$ respectively, the blue dots represent the $+1$ classified points and the red dots represent the -1 classified points. The orange line represents the line generated by the perceptron algorithm.

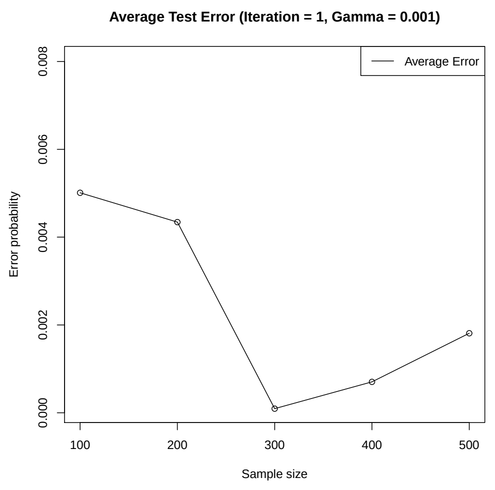
The algorithm to calculate the test error for this problem was very similar to Problem 3b1, with only a few minor changes. In Problem 3b1 the algorithm was first provided with two lines, then they were identified as y_1 and y_2 and finally the corresponding areas were calculated. Now the error needs to be calculated relative to a region rather than a specific line. The following algorithm considers each of the cases discussed in 3b1 and calculates the correct area for each case.

Algorithm 6 Calculate area relative to line from K_γ

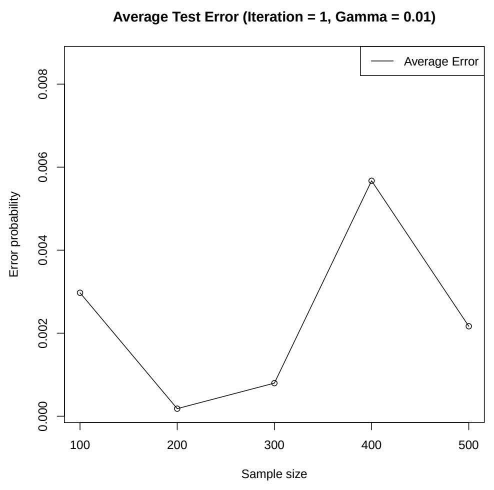
```
procedure FIND AREA RELATIVE TO $x + 0.1 \pm \gamma$
    if $(case == 1) \| (case == 3)$ then
        $Area \leftarrow Area_1 - Area_2$
        if $\gamma == 0$ then
            return Area
        else
            if considering $x + 0.1 + \gamma$ then
                if $y_2$ is $x + 0.1 + \gamma$ then
                    return Area
                else
                    return 0
                end if
            else if considering $x + 0.1 - \gamma$ then
                if $y_1$ is $x + 0.1 - \gamma$ then
                    return Area
                else
                    return 0
                end if
            end if
        end if
    else if $(case == 2)$ then
        $Area_{left} \leftarrow Area_{1,left} - Area_{2,left}$
        $Area_{right} \leftarrow Area_{1,right} - Area_{2,right}$
        $Area \leftarrow Area_{left} + Area_{right}$
        if $\gamma == 0$ then
            return Area
        else
            if considering $x + 0.1 + \gamma$ then
                if $y_2$ is $x + 0.1 + \gamma$ then
                    return $Area_{right}$
                else if $y_1$ is $x + 0.1 + \gamma$ then
                    return $Area_{left}$
                end if
            else if considering $x + 0.1 - \gamma$ then
                if $y_2$ is $x + 0.1 - \gamma$ then
                    return $Area_{left}$
                else if $y_1$ is $x + 0.1 - \gamma$ then
                    return $Area_{right}$
                end if
            end if
        end if
    end if
end procedure
```

As can be seen from Algorithm 6, Case 1 and Case 3 behave in the same way. They both only return an area if y_2 is the upper boundary of the region K_γ and only return the calculated area if y_1 is the lower boundary for the region K_γ. This ensures that any area that is calculated that falls inside the region K_γ is not included in the error calculation. Case 2 is approached in a very similar fashion as Cases 1 and 3, but due to the nature of where the lines intersect, it is necessary to consider which of the calculated areas must be returned. If this algorithm is followed, the area calculated will always be the area outside of the region K_γ. When calculating the overall area, it is important to remember that the total area is sum of the area produced between the perceptron line and the upper boundary and the area produced between the perceptron line and the lower boundary, eg: $Area_{total} = Area_{upperboundary} + Area_{lowerboundary}$.

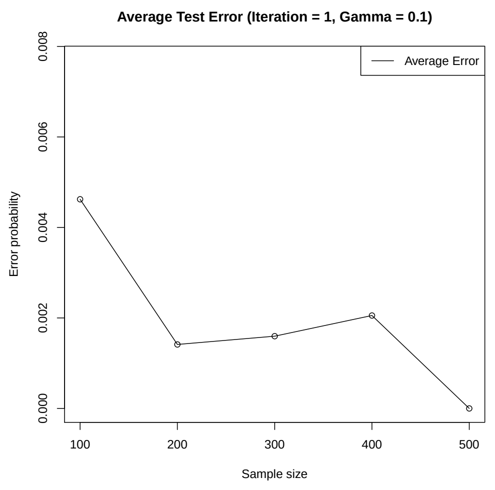
It can clearly be seen in from Figure 7 that the test error is going to be extremely small. When recreating the situation from Problem 3b2, the following graphs were produced.



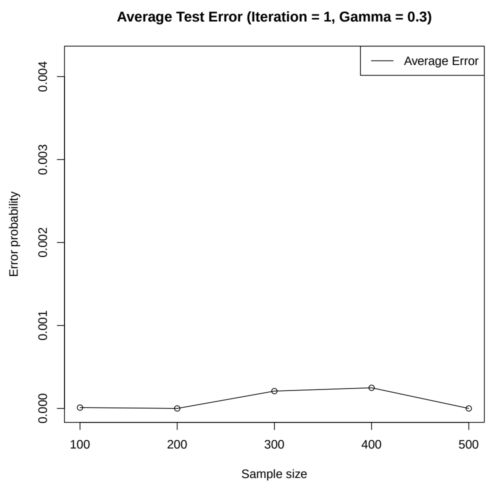
(a) $\gamma = 0.001$



(b) $\gamma = 0.01$



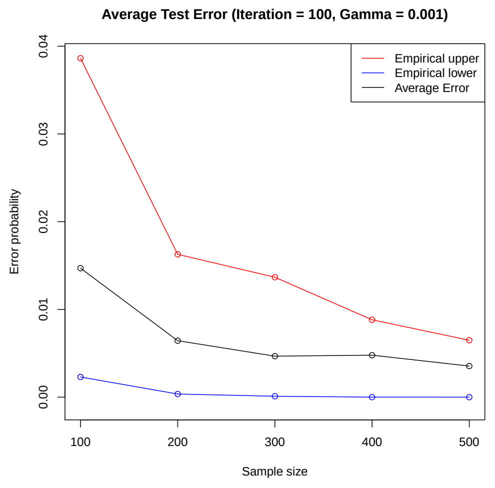
(c) $\gamma = 0.1$



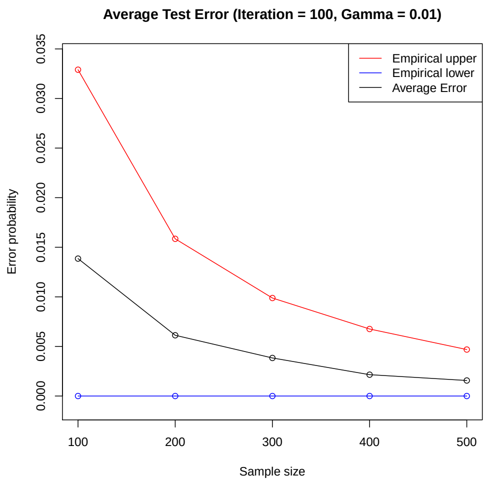
(d) $\gamma = 0.3$

Figure 8: Average test error for varying values of γ

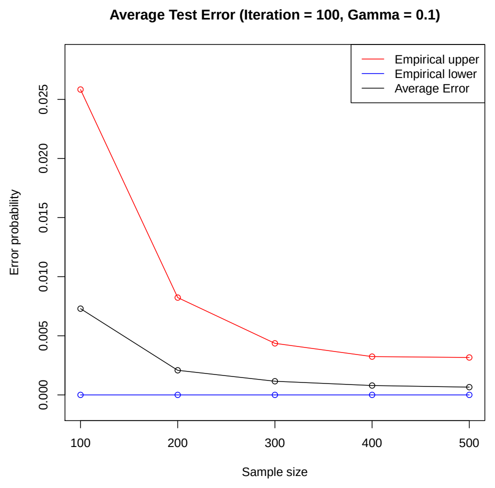
All raw data associated to these graphs can be found in the corresponding files in the folder ./results/Q3b/. The graphs presented in Figure 8 demonstrate the same characteristics as in Problem 3b2. They again seem to follow no real discernible pattern and are seemingly random. If multiple individual trails were run, the same result of finding a trend would be achieved. There is one detail that is worth noting, and that is change in the magnitude of the error. As γ increases, the magnitude of the error decreases correspondingly as was mentioned in the discussion of Problem 3a. This is because a large region of the error relative to the line $x + 0.1$ still falls within the K_γ region. The larger this region, the smaller the test error will be. The decrease in error with the increase of γ becomes a lot clearer when plotting the average test error over 100 trails, emulating Problem 3b3 but with the region K_γ.



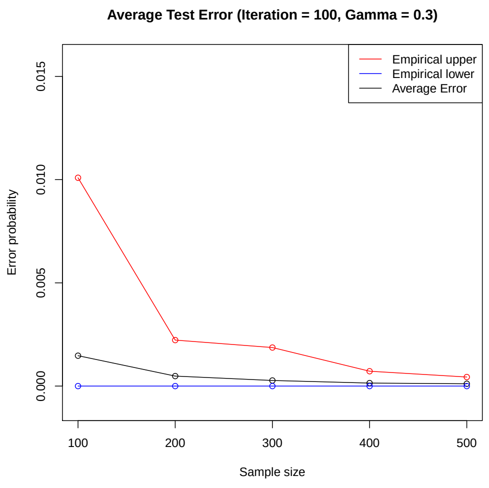
(a) $\gamma = 0.001$



(b) $\gamma = 0.01$



(c) $\gamma = 0.1$



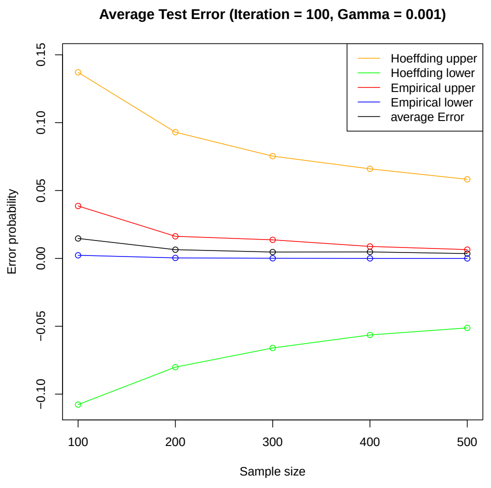
(d) $\gamma = 0.3$

Figure 9: Average test error for varying values of γ

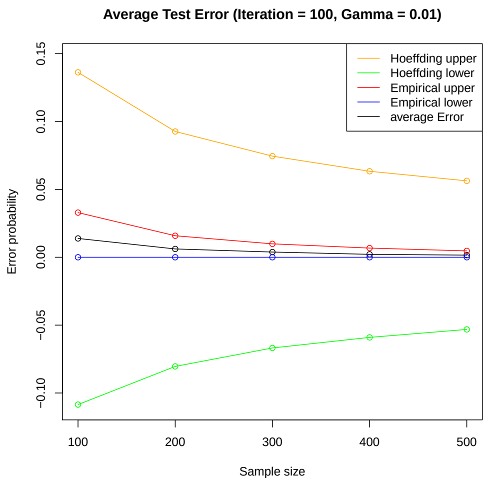
The graphs in Figure 9 follow the same setup as Figure 6.

The graphs in Figure 9 do not include the confidence interval produced by the Hoeffding inequality. This reason for this is that if they are included, they would scale the graphs in such a way that the information becomes unreadable. As a result they have been added in Figure 10, but for discussion Figure 9 will be used. Figure 9 confirms the explanation above that as γ increases, the error value will drop. The value of the average error and the empirical 90% confidence intervals decrease very proportionally to the increase of gamma. Furthermore, the behavior that was displayed in Problem 3b3 that an increasing sample size decreases the test error probability still holds, which was exactly what was expected.

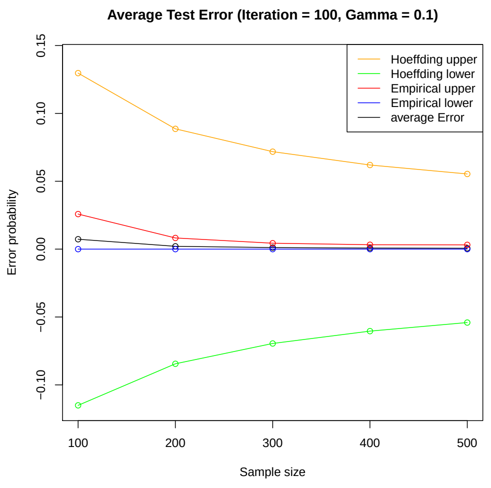
When running the code, the figures in Figure 10 will have a similar file name as Figure 9, except with an extra "_full" at the end. Again, all of the raw data can be found in the corresponding ".out" files.



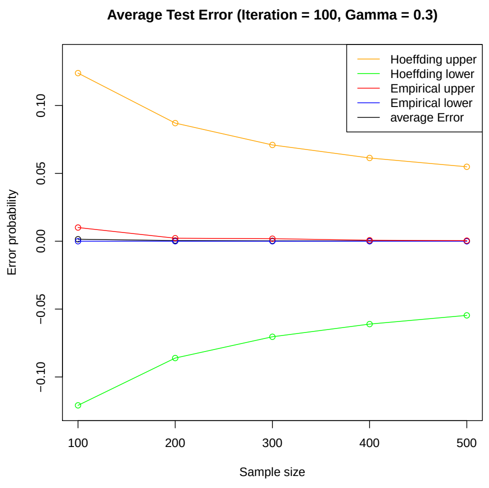
(a) $\gamma = 0.001$



(b) $\gamma = 0.01$



(c) $\gamma = 0.1$



(d) $\gamma = 0.3$

Figure 10: Average test error for varying values of γ

The graphs in Figure 9 follow the same setup as Figure 6.

Finally, Problem 3b4 requests to find viable choices of w_* for ρ. Every time the code is run, all values of w_* are stored in the appropriate output files. These file contains the two viable choices of w_* for every data set. A w_* is considered viable when $\rho > 0$. An example of some viable w_* values are found in the table below. Of course these values will change every time the program is run because the generated data is not the same each time is is run. This is due to the fact that the data is a random uniform distribution. Values from calling "assingment1_dav114.r" are shown below:

Iterations	100	200	300	400	500
$w_*[1]$	-1	-1	-1	-2	-1
$w_*[2]$	-13.236147	-8.639746	-8.15402	-22.00328	-8.112491
$w_*[3]$	12.511313	8.815217	8.514444	22.04205	8.407176

Table 1: First set of viable w_* values for rho, $\gamma = 0.1$

Iterations	100	200	300	400	500
$w_*[1]$	-1	-2	-1	-2	-1
$w_*[2]$	-8.639746	-22.003279	-7.19991	-21.61702	-8.702345
$w_*[3]$	8.815217	22.042055	7.643841	21.41518	8.880008

Table 2: Second set of viable w_* values for rho, $\gamma = 0.1$

Above are the viable values for w_* for rho where $\gamma = 0.1$. Similar results can be found for the other values of γ in the raw data that is produced in the folder ./results/Q3b/.

The tables below show a comparison between the theoretical bound on the number of iterations, and the empirical data that has been collected.

Sample size	100	200	300	400	500
Iterations	10.18	10.82	11.53	11.67	11.75
Theoretical bounds	4914196.4	1361870.8	11806861.4	619161.2	561029.5

Table 3: $\gamma = 0.3$

Sample size	100	200	300	400	500
Iterations	40.78	49.12	52.54	55.11	56.33
Theoretical bounds	4977219284	195206926	212848081	245635682576	10278824102

Table 4: $\gamma = 0.1$

Sample size	100	200	300	400	500
Iterations	298.15	298.39	390.14	347.27	463.65
Theoretical bounds	4.272567e+09	5.316163e+11	9.990827e+10	1.416764e+11	1.122514e+12

Table 5: $\gamma = 0.01$

Sample size	100	200	300	400	500
Iterations	519.92	1241.97	1782.05	2463.35	3012.07
Theoretical bounds	1.567373e+13	8.087088e+12	4.984224e+12	3.824600e+12	5.657813e+11

Table 6: $\gamma = 0.001$

As can be seen in the tables above, there is a massive gap between the theoretical bound and the practical number of iterations that were required to complete the algorithm. This is because the theoretical bound considers the worst possible route that could be taken to end up with a value of w_*. Even though this is extremely unlikely to ever happen, the upper bound must always account for such situations, justifying why it is such a large value, and why it is such and order of magnitude larger than the actual number of iterations.

3.3 Problem 3c - Handwritten Text Classification

Problem 3c, unlike Problem 3b, did not depend on random, uniformly generated data. Instead there were grayscale classifications of handwritten numbers. These were not necessarily linearly separable datasets like in Problem 3b.

3.3.1 Problem 3c1

The original perceptron algorithm would always output w_* at the point where there are no more failures or the iteration limit is reached. Because this is not linearly classifiable data, the iteration limit is always reached and the w returned by the algorithm is not necessarily the best result, just the w at the iteration limit. On the other hand, the modified perceptron algorithm keeps track of and returns the lowest value of w, rather than the most recent one. This modified perceptron algorithm will still works for linearly separable, because if the data is linearly separable, then w_* is the lowest value of w.

3.3.2 Problem 3c2

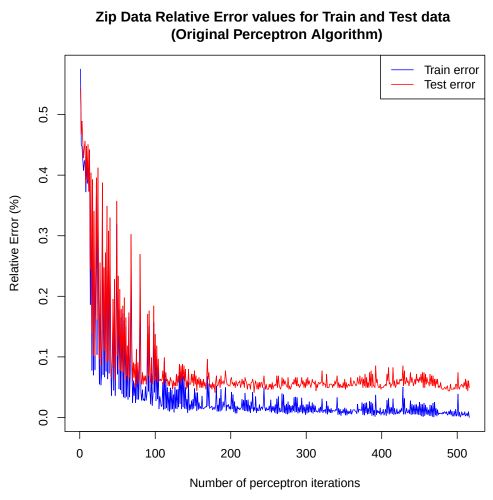
For this problem, the following data was produced:

Zip data training error (Original algorithm)	0
Zip data test error (Original algorithm)	18
Zip data training error (Modified algorithm)	0
Zip data test error (Modified algorithm)	18
Feature data training error (Original algorithm)	479
Feature data test error (Original algorithm)	147
Feature data training error (Modified algorithm)	286
Feature data test error (Modified algorithm)	106
Feature data training error using linear regression (Original algorithm)	537
Feature data test error using linear regression (Original algorithm)	166
Feature data training error using linear regression (Modified algorithm)	288
Feature data test error using linear regression (Modified algorithm)	106

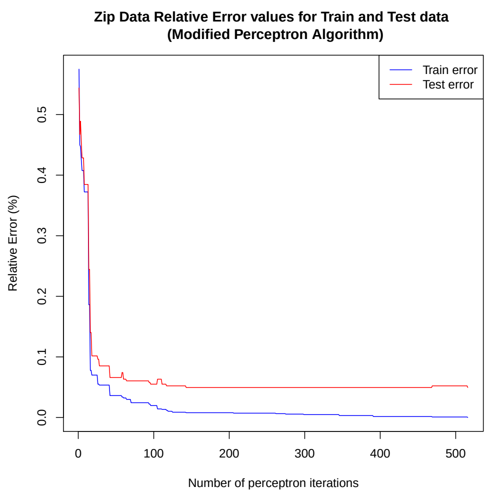
Table 7: Train and test data values

The table above shows the average error values for the different data sets using both algorithms, both with and without linear regression as staring value for w. The values in this table are not relative, but aboslute errors. They indicate that the zip training data is indeed linearly classifiable unlike the feature training data. This is further confirmed by all the graphs in Problem 3c1.

In the Figure 11 below you can see the relative error of the training versus test data. Because the zip data had a training error of zero, it is clear that this is a linearly classifiable set of data. The featured data on the other hand was not linearly classifiable and as the training error was not zero. Looking at the way it ran through the non-modified perceptron algorithm, it can also be show that the graph is noticeably more volatile. and does not tend towards zero like the zip data does.

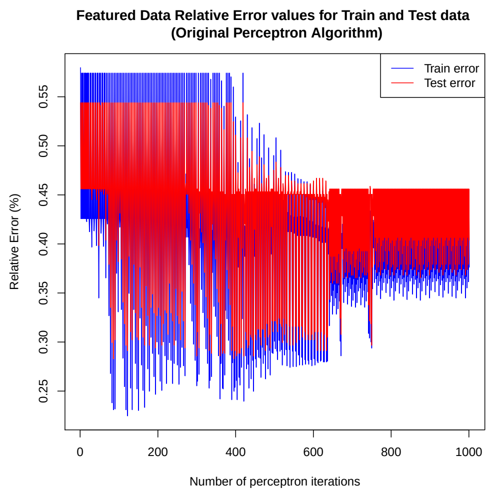


(a) $\gamma = 0.001$

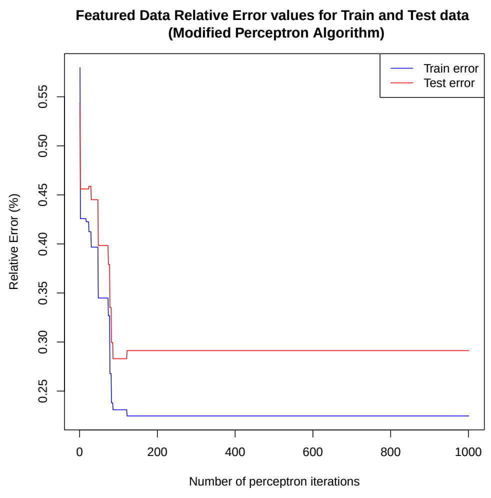


(b) $\gamma = 0.01$

Figure 11: Relative Error for Training vs Test data



(a) $\gamma = 0.001$

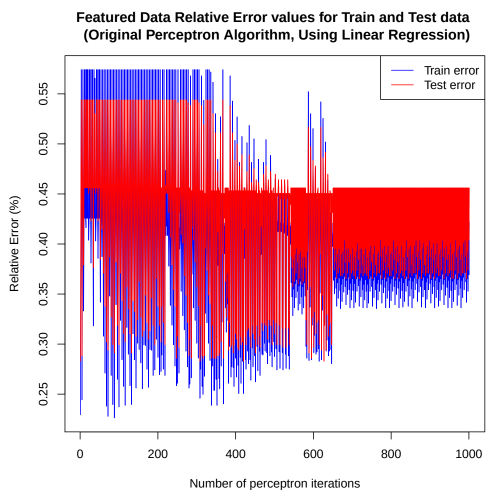


(b) $\gamma = 0.01$

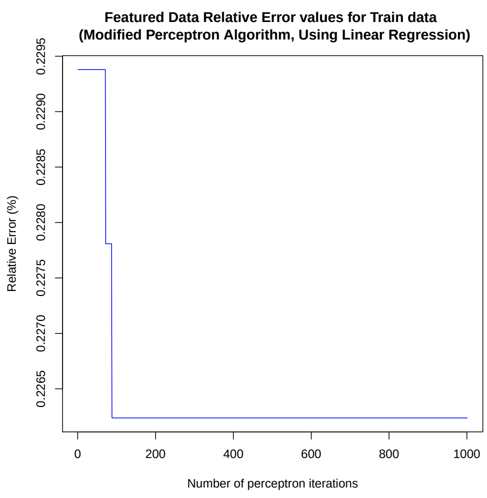
Figure 12: Relative Error for Training vs Test data

3.3.3 Problem 3c3

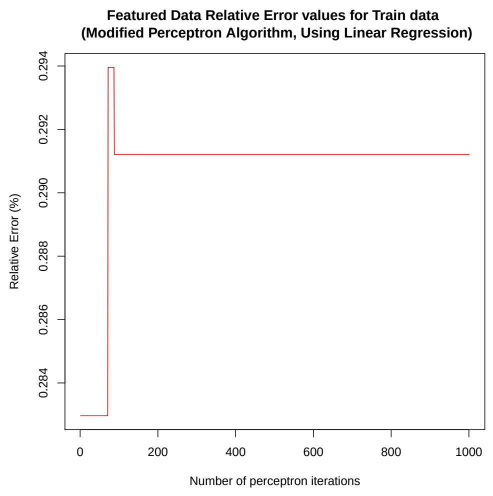
The use of linear regression for has clearly not changed the final value for the relative error. This can be deduced from the fact that the values represented in the new graphs in Figure 13 end up at the same value. The only thing that the linear regression has done was speed up the process of arriving at the same value. In the end, the preceptron will still run in the same fashion, and almost always end up with the same value, as in this case, that is the best value. Of course if the number of iterations is too low, and the lowest value of w is not found before the iteration limit is reached, then this will make a difference, as the perceptron algorithm will follow a different path. But all in all, if the perceptron algorithm is allowed to iterate enough times, then using an initial w found through linear regression only speeds up finding the lowest value of w, it does not change it.



(a) $\gamma = 0.001$



(b) $\gamma = 0.01$



(c) $\gamma = 0.01$

Figure 13: Relative Error for Training vs Test data

3.3.4 Problem 3c4

To find the upper bound on the difference of the emperical test error and the ideal test error, the formula provided in lecture 2 slide 4 can be used. This formula gives and upper bound to $R(g) - R(h^*)$, where h^* is the ideal test error and g is the best empirical hypothesis. This upper bound is officially formulated as $R(g) - R(h^*) \leq \sqrt{\frac{\log(\frac{2|H|}{\delta})}{2n}}$, with a probability of $1 - \delta$. We are currently aiming for a probability of 95%, so $\delta = 0.05$. Furthermore, we need to determine the size of the hypothesis class $|H|$. When looking at the graphs in Figures 13,11 and 12 we can see the values seem iterate over roughly the same area. This would indicate that the uniqueness of w had gone, and we have reached the end of our reachable hypothesis class. This value is between 700-1000 iterations. Referring back to the tables from Problem 3b3, we can see that this would result in a theoretical bound of a value in the order of magnitude of 10^12. As a result, this value will be used for $|H|$.

We will first deal with the situation where the data set is the zip data. $n_{feat} = 1273$. We will first deal with the situation where the data set is the feature data. $n_{feat} = 364$. This results in an upper bound of 0.4148341425. For the zip data set, $n_{zip} = 1273$ and the upper bound is 0.2218252519. As a result we can say with a 95% certainty that $R(g) - R(h^*) \leq 0.4148341425$ for the featured data set and $R(g) - R(h^*) \leq 0.2218252519$ for the zip data set. The bound for the zip data set is tighter, which agrees with what has been mentioned through out this report that if there are more data points, the test error value will be lower on average. Furthermore, when looking at the plots of any value of test error data in this report, the difference in values between a ideal and empirical errors never exceed the bounds which was been calculated here.

References

Harvey Berman. *Confidence Level: Definition*, 2017. http://stattrek.com/statistics/dictionary.aspx?definition=confidence_level [Accessed: 29/01/2017].