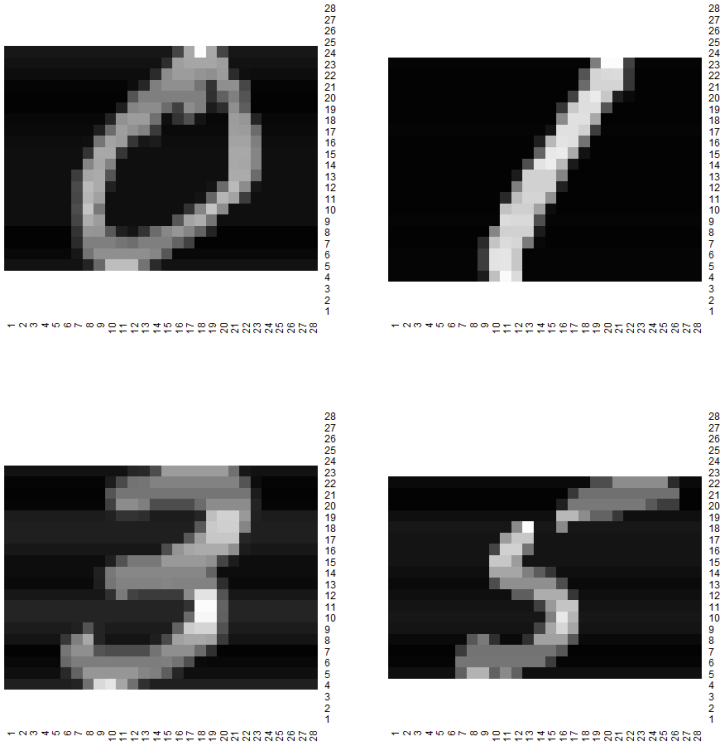
HW3: Logistic Regression

Part 0: Data Pre-processing:

The following are sample visualizations of each class labeled correctly:



The following code have been used to preprocess data and generate the visualizations above

```
library(ggplot2)
library(GGally)
setwd("C:/Users/AMD/Desktop/Classes/GaTech/Data Visualization/HWs/HW3")

train <- read.csv('mnist_train.csv', header = FALSE)
test <- read.csv('mnist_test.csv', header = FALSE)

#Partitioning Training Set
train_0_1 = train[,train[785,]==0 | train[785,]==1]
train_3_5 = train[,train[785,]==5 | train[785,]==3]

#partitioning Testing Set
test_0_1 = test[,test[785,]==0 | test[785,]==1]
test_3_5 = test[,test[785,]==5 | test[785,]==3]

#Isolate label vectors
true_label_train_0_1 = train_0_1[785,]
train_0_1 = train_0_1[1:784,]

true_label_train_3_5 = train_3_5[785,]
train_3_5 = train_3_5[1:784,]

true_label_test_0_1 = test_0_1[785,]
test_0_1 = test_0_1[1:784,]
```

```
true_label_test_3_5 = test_3_5[785,]
test_3_5 = test_3_5[1:784,]

#creating image matrices
idx0 = which(true_label_train_0_1==0)[1]
class_0_image = apply(matrix(train_0_1[,idx0],28,28), 2, rev)

idx1 = which(true_label_train_0_1==1)[1]
class_1_image = apply(matrix(train_0_1[,idx1],28,28), 2, rev)

idx3 = which(true_label_train_3_5==3)[1]
class_3_image = apply(matrix(train_3_5[,idx3],28,28), 2, rev)

idx5 = which(true_label_train_3_5==5)[1]
class_5_image = apply(matrix(train_3_5[,idx5],28,28), 2, rev)

#plotting image matrices
heatmap(class_0_image,Rowv=NA,Colv=NA,col=paste("gray",1:99,sep=""))
heatmap(class_1_image,Rowv=NA,Colv=NA,col=paste("gray",1:99,sep=""))
heatmap(class_3_image,Rowv=NA,Colv=NA,col=paste("gray",1:99,sep=""))
heatmap(class_5_image,Rowv=NA,Colv=NA,col=paste("gray",1:99,sep=""))
```

Part 1: Theory:

a. Following Stanford Professor Andrew Ng's notes on logistic regression:

We start by defining our hypothesis $h_\theta(x)$ as follows:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where $\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$ and $x_0 = 1$
By denoting $\theta^T x$ as z we can re-write our hypothesis as a function of z as follows:

$$g(z) = \frac{1}{1 + e^{-z}}$$

The function above is just a sigmoid function which has a derivative that is product of the function multiplied by one minus the function as shown below:

$$g'(z) = g(z)(1 - g(z))$$

We will base our solution on the following assumption where y can have a binary value (1,0) instead of (1,-1):

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

Our cost function is basically the log of the above two functions.
We can combine the above functions to find the probability of y given x and theta resulting in:

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

Given that we have m independent training examples, the likelihood $L(\theta) = p(\vec{y} \mid X; \theta)$ can be written as the product of the likelihoods of the individual training examples:

$$L(\theta) = \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}$$

We then would want to maximize our likelihood which can be done by maximizing the $\log L(\theta)$ which we will call $\ell(\theta)$ and will be equal to:

$$\ell(\theta) = \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

Maximization can be done using gradient ascent. If we look at a single example, we can calculate the derivative, while keeping this property $g'(z) = g(z)(1 - g(z))$ in mind, needed for gradient ascent as follows:

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x)$$

Which simplifies to:

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = (y - h_\theta(x))\, x_j$$

Now our update rule will simply be:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

The above step is repeated until convergence where we do simultaneous for our theta j while iterating over all sample for 'n' iterations which is stopped based on a stopping criteria . Also, This maximization problem can simply be turned to a minimization problem by multiplying our log sigmoid function by -1 and this will simply change our problem to a minimization problem.

b. The following can be a pseudo code for logistic regression:

```
def logisticRegression(trainingSamples, traininglabel, learningRate,
numberOfIterations){
        X = trainingSamples (where each row is a sample and columns are features)
        X0 = vector of ones equal to number of samples
        X <- addColumn(X ,X0) (add bias column to X)
        numberOfSamples = number of rows in X
        thetas = random vector of values between -1 & 1 and has dimension equal to
                number of features of X
        iteration = 1
        while(iteration<= numberOfIterations){
                h = 1/(1+exp(-(X*thetas)))
                for (i=1; i++; i<=numberOfSamples){
                        y = traininglabel[i]
                        xi = X[i]
                        thetas = thetas + learningRate*(y-h[i])*xi
                }
                Iteration = iteration + 1
        }
        Return (thetas)
}
```

c. The number of operations in the code above per gradient descent iteration is:
n: is number of samples; d: is number of features in every n (without the bias)

- When calculating h X*thetas will have (n*(d+1)) multiplications and n exponential calculations
- In *thetas = thetas + learningRate*(y-h[i])*xi* we have (d+1) additions and 2*(d+1) multiplications for n times because of the loop resulting n*(d+1) additions and 2n*(d+1) multiplications.

2. Code for logistic Regression

```
logisticRegression2 = function(trainingSamples, labels, LR, threshold=1e-09,
numIterations=100,shuffle=0,portions=1){
  #Add Bias to training set
  X = insertRow(trainingSamples, rep(1,ncol(trainingSamples)),1)
  thetas = runif(nrow(X), min = -1, max = 1)
  #thetas = rep(0,nrow(X))
  #transpose set so that each row is a sample
  X = t(X)
  labels = t(labels)
  if (shuffle==1){
    numRows = floor(portions*nrow(X))
    if (numRows<=1){
      numRows = 2
    }
    selectedSamples = sample(nrow(X), numRows)
    X = X[selectedSamples, ]
    labels = labels[selectedSamples,]
  }
  labels = unname(labels)
  numSamples = nrow(X)
  iteration = 1
  while (iteration<=numIterations){
    h = 1/(1+exp(-(X%*%(thetas) )) )
    for (i in seq(1,numSamples)){
      y = labels[i]
      Xi =X[i,]
      grad = (y-h[i])
      thetas = thetas + LR*grad*Xi

      #   Stopping criterion. is the number of iterations
    }
    iteration = iteration+1
  }
  return (thetas)

}
```

3. Answers to question 3

The measure of accuracy is the percent of correctly classified instances.

a. Comparison of accuracies between the classes

	Classes 0_1	Classes 3_5
Train Accuracy	99.5%	88.9%
Test Accuracy	99.8%	89.9%

b. These values are for alpha = 0.001 after 10 runs on random shuffles of 85% of the training data:

```
mean(train_1_0_accuracies) = 0.9951994
mean(train_3_5_accuracies) = 0.8896035
mean(test_1_0_accuracies)  = 0.998156
mean(test_3_5_accuracies)  = 0.8991588
```

c. It was noted that when the overall accuracy of 1_0 is higher than that of 3_5. This could be attributed to the shapes being compared. In the case of 0 and 1 the shapes are quite different where one is straight and the other is curved. Whereas, shapes 3 and 5 look slightly similar especially the curved part at the bottom of both. In other words, the overlap between them is higher than the overlap between 0 and 1. Also, the stopping criteria for both sets could be different where we might need more iterations for the 3_5 class.

d. we can use a one vs. all approach. In our original approach we only had two classes which we can solve using the implemented logistic regression classifier. We want however to be able to classify into more than two classes. Let's assume we have 4 classes (1,0,3,5). We will turn our problem into 4 separate two-class classification problems (4 is the number of classes). We will start with one of the classes for example "1" where we will create a new fake training set that labels "1" as the positive class and all other classes will be labeled with the same label and be assigned to the negative class. We will fit a classifier to the data set that is denoted by $h_\theta^{(1)}(x)$ the (1) superscript corresponds to the first class "1". We will do the same for "0" class by assigning all samples with label "0" to the positive class and everything else to the negative class and we will also have a new classifier $h_\theta^{(2)}(x)$ here the (2) superscript also stands for the second class which is "0". We will do the same for the other classes where each will have its classifier. We will end up with 4 classifiers (same as number of classes).

What each classifier calculates is the probability of a sample to be part of that class. So to make a prediction on a new input x, we choose the class i that maximizes $\max_i h_\theta^{(i)}(x)$ which is simply calculating the probabilities using all classifiers and choose the class that has the largest value.

4. Answers to question 4

a. the different parameter for initialization is the value of alpha or the learning rate which was changed from 0.001 to 0.01. I also tried changing initial values of theta. Below are the average results for 10 runs:

Accuracy	Alpha = 0.01 Theta = zeros	Alpha = 0.001 Theta = zeros	Alpha = 0.00000000001 Theta = zeros	Alpha =0.001 Theta = random number between -1 & 1
Train 0_1	0.9951678	0.9951994	0.5527438	0.9921674
Train 3_5	0.8810336	0.8896035	0.4705332	0.8598511
Test 0_1	0.9976832	0.998156	0.5551773	0.9951773
Test 3_5	0.8854364	0.8991588	0.4666141	0.8648265

We can see that there is no significant difference when we changed our initialization values for alpha from 0.001 to 0.01. However, if alpha was too large it could overshoot and not find the global maxima When alpha was set too small to 0.00000000001 the learning will require a larger number of iterations to achieve high accuracy and this is why for the current number of iterations we achieved poor results.

I noticed a change when the initial values of my thetas were changed from zeros to random values between -1 and 1, there has been a drop in performance that is noticeable in the 3_5 classes.

b. By changing the termination criteria from 10 iterations over all samples to 1 iteration over all samples while keeping the same value of alpha = 0.001

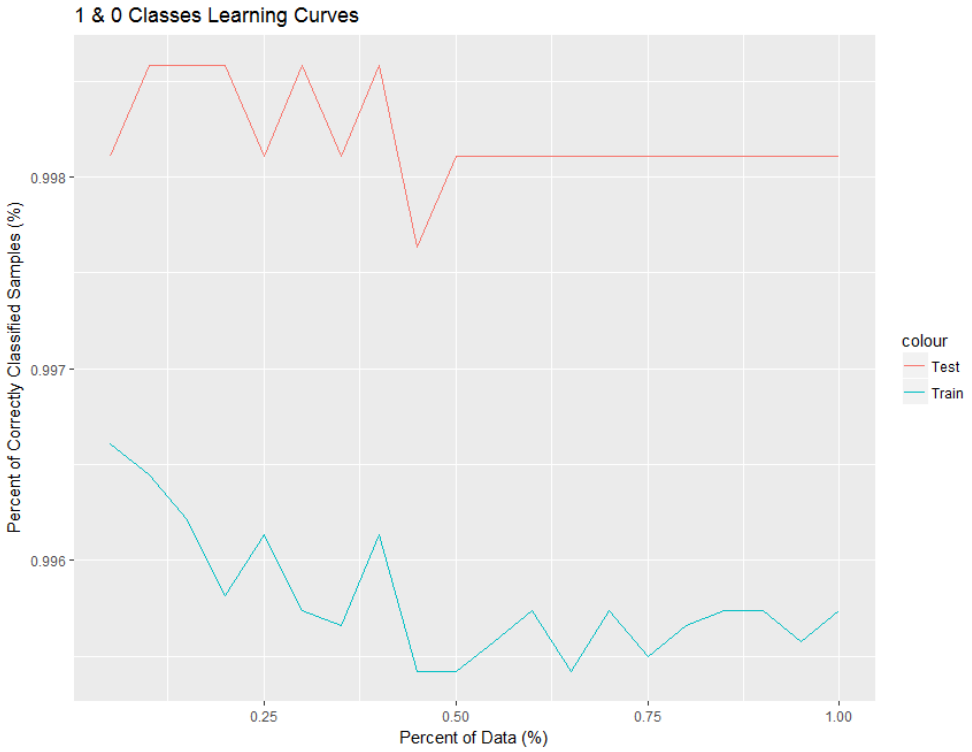
Accuracy	1 iteration	10 iterations
Train 0_1	0.9904303	0.9951994
Train 3_5	0.5755194	0.8896035
Test 0_1	0.9956028	0.998156
Test 3_5	0.5832808	0.8991588

Class 1_0 did not suffer see much improvement in accuracy. However, class 3_5 saw a really big change which even improved further when the number of iterations was set to 100 at around **93%** accuracy for the 3_5 testing data.
Since the 3 and 5 classes have more overlapping similarity than 1 and 0, they are expected to need more iterations to learn. By increasing the number of iterations (epochs), the classifier was able to generalize better and the overall accuracy increased.

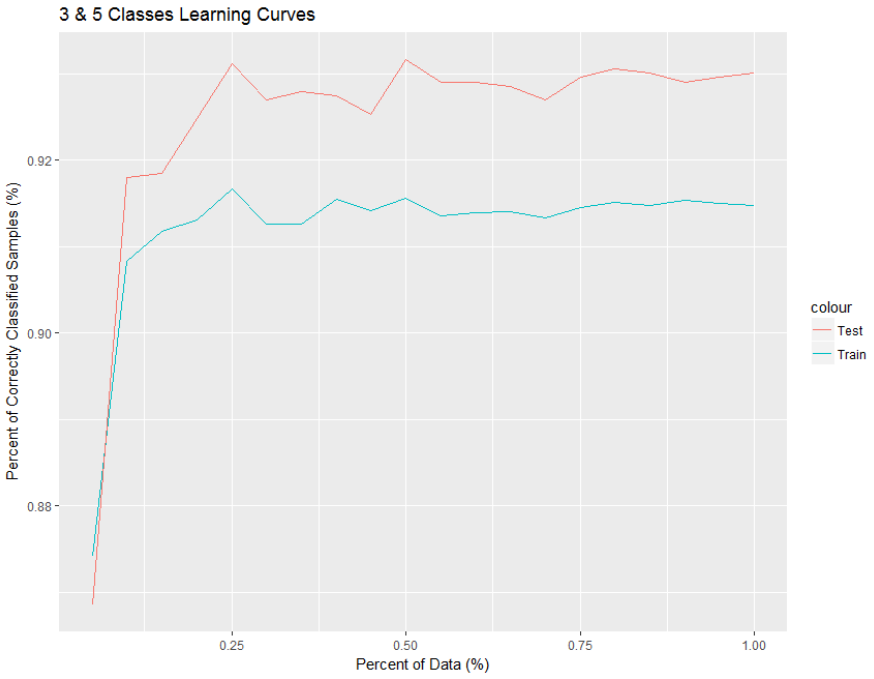
A possible improvement to the algorithm is to have another measure that is called tolerance which is simply a threshold that checks if there has been any significant changes in the gradient that are larger than that threshold value. If no changes are present, the algorithm terminates. By combining this criteria with the number of iteration termination criteria we can improve the training time and avoid over training.

5. Answers to question 5

A. The following are the learning curves:



As can be seen above there appears to be variance between the curves but this is only because it is zoomed in beyond 0.99. As for bias, since the accuracy of our learner is high, we can conclude that the achieved results are acceptable. We can see that distinguishing between 0 and 1 is not that difficult as we only need a small portion of the training data so we can say that 0 and 1 do not resemble each other.



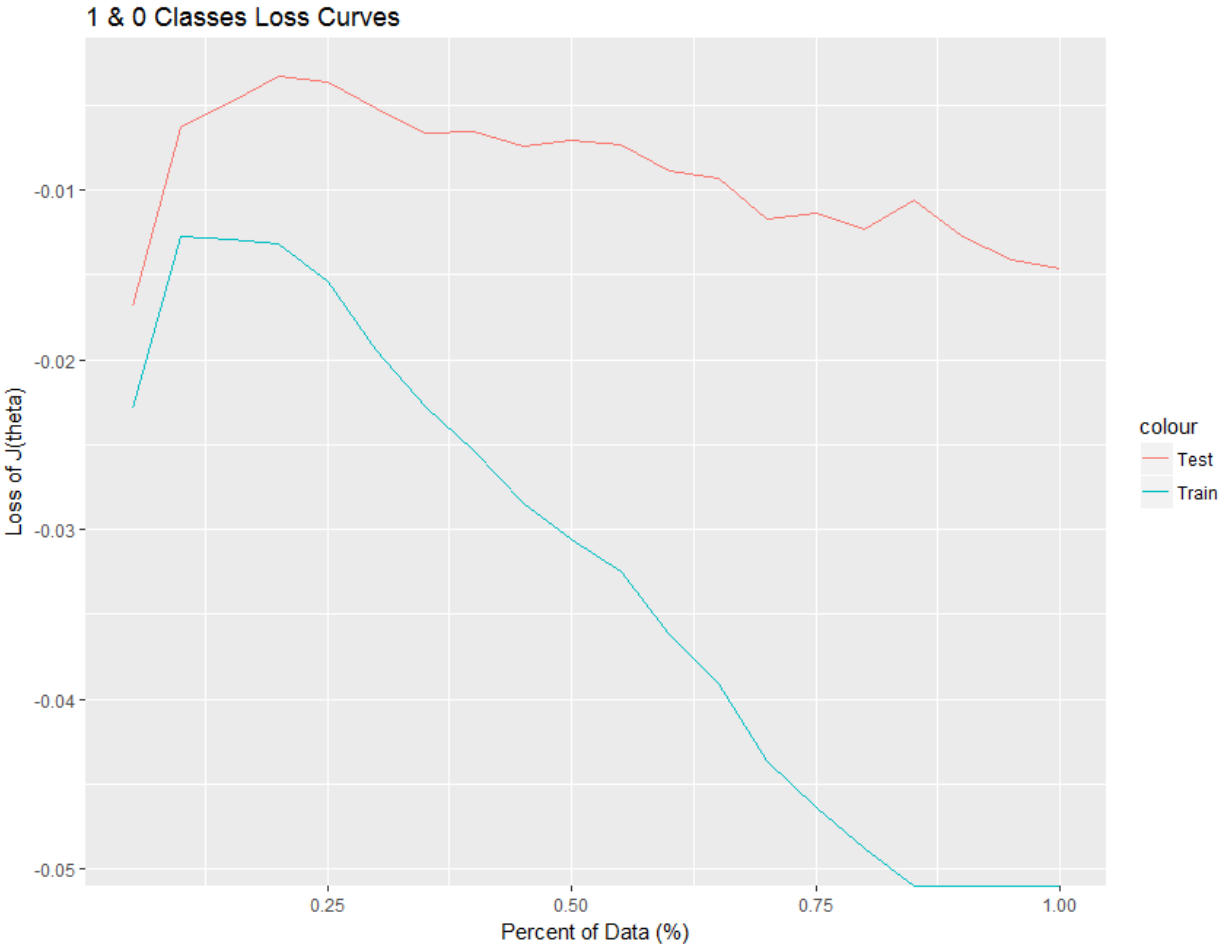
We can see from the image above that being able to distinguish between 3 and 5 requires more data than 0 and 1. This is due to 3 and 5 resembling each other in some portions. This variance can be reduced by increasing the number of training iterations. One possible solution is to reduce the dimensions in the image.

B. logistic loss plots

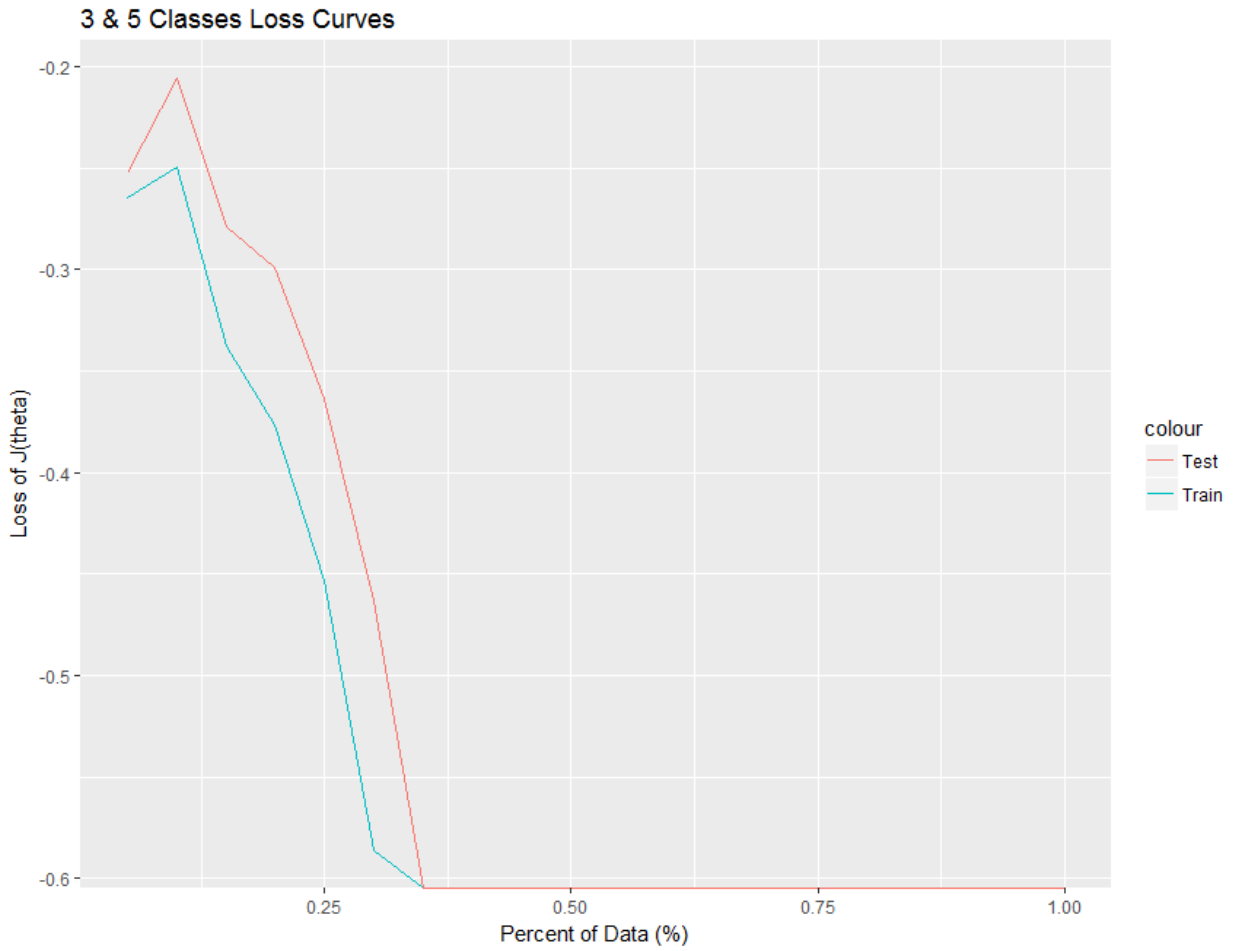
The cost function J(theta) can be written as (for labels 0 & 1):

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)})\log\left(1-h_\theta\left(x^{(i)}\right)\right)]$$

The main problem that might happen is when we have a zero inside the log which will be evaluated as NaN by R. When calculating it I checked whether y = 1 or 0 and only calculated the part related to the label because one part will be deleted depending on the label value and it was that part that caused the issues.



It seems that the values are monotonically decreasing as the percent of data increases. The scale difference or a bit off due to normalization.



The second plot shows both of them eventually reaching the same value where we see a much faster decay with respect to the number of samples.

In both cases, I had issues when running multiple iterations and storing the values so I had to decrease the runs to 2 and averaged their results.