HW3: Logistic Regression

Due: **Wednesday, 29 March 2017**, 11:59PM UTC-12:00 on T-Square

In this assignment, you will implement logistic regression for classification of digits on an image dataset, MNIST. You will also experiment with changing parameters and training set sizes, and evaluate how the behaviour of the model is affected.

Instructions

- You must complete this assignment on your own, not in a group.
- All your code must be written in the R programming language.
- You need to turn in 2 files for this assignment:
    - Code: R script file named exactly **`hw3.R`** with code for the different problems.
    - Report: PDF file named exactly **`hw3_report.pdf`** with written responses.
- Each problem includes a Submit section that clearly mentions what you need to put in the Code and/or Report file.
- Some problems include a Notes section which clarifies any ambiguities you may encounter in any part of the problem.
- Make sure you include your GT account name at the top of both files (the one you use to log into T-Square or Passport), not in the filename.
- Total points in this assignment: 100.

Data

Two CSV (comma-separated value) files are provided for this assignment:
`mnist_train.csv` and **`mnist_test.csv`** (also available under Piazza > Resources)

Each CSV file contains <785 rows x n columns>, where n is the number of sample images in the file. Each column represents one 28x28 image of a digit stacked into a 784 length vector followed by the class label (0, 1, 3 or 5).

We have chosen a subset of the original MNIST dataset, where the images of only four digits have been included (0, 1, 3 and 5). As a further simplification, we will only attempt binary classification in this assignment. You will need to separate the data into two classes for your training and testing: 0, 1 samples and 3, 5 samples. You will then build two models - one to distinguish between 0 and 1, and another to distinguish between 3 and 5.

Problems

0. Data Preprocessing

Carry out the following steps to prepare the data for modeling:

a. Download the CSV files for the provided dataset.
b. Read `mnist_train.csv` and `mnist_test.csv` separately.
   Note: There is no header row in the data; set `header = FALSE` when reading, e.g.:
   `train <- read.csv('mnist_train.csv', header = FALSE)`
c. Partition the training set for classification of 0, 1 and 3, 5 classes based on the class label (last row 785): `train_0_1`, `train_3_5`.
d. Do the same for the test set: `test_0_1`, `test_3_5`.
e. Separate the class label from all the partitions created (remove row 785 from the actual data and store it as a separate vector).
f. You will finally have two sets of data and their corresponding true labels:
   - for classification of digits 0, 1: (`train_0_1`, `test_0_1`) and (`true_label_train_0_1`, `true_label_test_0_1`);
   - For classification of digits 3, 5: (`train_3_5`, `test_3_5`) and (`true_label_train_3_5`, `true_label_test_3_5`);
g. Visualize 1 image from each class to ensure you have read in the data correctly. You will have 4 images corresponding to 0, 1, 3 and 5. You need to convert the 1D image data into 2D for visualisation.

Notes:
- To can convert 1D data to 2D, you can cast it into a `matrix` and specify the number of rows/columns.
- To plot 2D data, you can use the `image()` function with `col=gray.colors(256)`.
- Digits should be upright and not mirrored.

Submit:
- Report: 4 sample images, one from each class, along with their class labels to demonstrate you've read the data correctly.
- Code: All the code you used to read in and preprocess the data.

1. Theory

a. Write down the formula for computing the gradient of the loss function used in Logistic Regression. Specify what each variable represents in the equation.
b. Write pseudocode for training a model using Logistic Regression.
c. Calculate the number of operations per gradient descent iteration. (Hint: Use variable **n** for number of examples and **d** for dimensionality.)

Notes:
- You can use a loss function with labels 0/1 or -1/+1.
- Ensure your answers for 1b and 1c are consistent with your choice of loss function.

Submit:
- Report: Include your responses to each part in the report.
- Code: This is a theory question, no code is required.

2. Implementation

Implement Logistic Regression using Gradient Descent.

Notes:
- You can use any gradient descent technique (batch or stochastic).
- Choose any set of initial parameters for gradient descent. You will use the same for question 3.
- Choose any convergence criteria. You will use the same for question 3.
- Try to vectorize code by avoiding for loops.

Submit:
- Code: R code consisting of a logistic regression function and gradient descent function with brief comments clarifying different steps/components.

3. Training

Train and evaluate the code from question 2 on MNIST dataset.

a. Train 2 models, one on the train_0_1 set and another on train_3_5, and report the training and test accuracies.
b. Repeat 3a 10 times, i.e. you should obtain 10 train and test accuracies for each set. Calculate the average train and test accuracies over the 10 runs, and report them.
c. For 0,1 and 3,5 cases, explain if you observe any difference you in accuracy. Also, explain why do you think this difference might be.
d. This assignment deals with binary classification. Explain what you would do if you had more than two classes to classify, using logistic regression.

Notes:

- When training multiple times, if you're getting the exact same result every time, try randomly shuffling the data.

Submit:

- Code: R code of a predict function to calculate the predictions, accuracy function to calculate accuracy, calls to these function for 3a and 3b.
- Report: For 3a, train and test accuracies for 0/1 and 3/ 5 classification. For 3b, average test and train accuracies for 0/1 and 3/5 classification. For 3c and 3d, a short paragraph answering the questions.

4. Evaluation

In this question, you will experiment with different sets of parameters and observe how your model performs. This should be done ONLY for 3,5 classification.

a. Experiment with different initializations of the parameter used for gradient descent . Clearly mention the initial values of the parameter tried, run the same experiment as 3b using this initialization, report the average test and train accuracies obtained by using this initialization, mention which is set of initializations is the better.
b. Experiment with different convergence criteria for gradient descent. Clearly mention the new criteria tried, run the same experiment as 3b using this new criteria, report average test and train accuracies obtained using this criteria, mention which set of criteria is better.

Notes:

- You have [initial_parameters_baseline, convergence_criteria_baseline] in your first set of results 3b and these will serve as your baseline results.
- For 4a, just change the initial parameters and keep the convergence criteria same [initial_parameters_experiment, convergence_criteria_baseline] and obtain the results.
- For 4b, just change the convergence criteria and keep parameter initialization same [initial_parameters_baseline, convergence_criteria_experiment] and obtain results.
- Experiment with ONLY one different initialization and ONLY one different convergence criteria than baseline.
- You can either code new functions for question 4 or you can re-use your previous functions with different function arguments if your code is generic and modular. For e.g. in Gradient Descent function, you can have "theta" as an argument and call it with different initial values instead of creating a new Gradient Descent function for a different initial value of "theta". Implementation details are up to you.

Submit:

- Report: For 4a, mention how you obtained a new set of initial parameters, report accuracies obtained and how they are different from your baseline implementation.

For 4b, mention the new convergence criteria tried, report the accuracies obtained and how they are different from your baseline implementation.
- Code: R code with functions for 4a and 4b if you have created new functions OR function calls to previous functions (see notes for details).

5. Learning Curves

In this question, you will experiment with different training set sizes and see how the accuracy changes with them. This question should be done for both 0,1 and 3,5 classification.

a. For each set of classes (0,1 and 3,5), choose the following sizes to train on: 5%, 10%, 15% … 100% (i.e. 20 training set sizes). For each training set size, sample that many inputs from the respective complete training set (i.e. train_0_1 or train_3_5). Train your model on each subset selected, test it on the corresponding test set (i.e. test_0_1 or test_3_5), and graph the training and test set accuracy over each split (you should end up with TWO graphs - one showing training & test accuracy for 0,1 and another for 3,5 set). Remember to average the accuracy over 10 different divisions of the data each of the above sizes so the graphs will be less noisy. Comment on the trends of accuracy values you observe for each set.
b. Repeat 5a, but instead of plotting accuracies, plot the logistic loss/negative log likelihood when training and testing, for each size. Comment on the trends of loss values you observe for each set.

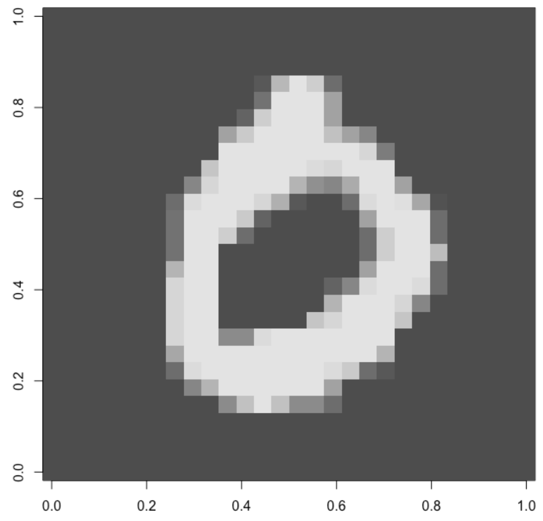
Notes:
- You can choose to select the % of training data in any manner. If you choose random divisions, make sure you extract the correct corresponding labels for each image.
- For each training set size (5%, 10%, etc.), you should run your model 10 times and average the accuracy. If 10 times is too computationally intensive, you can run 5 times. Clearly specify the number of times you have averaged on.

Submit:
- Report: For 5a, two graphs, one for each classification set (0,1 and 3,5), showing trends of train and test accuracies over different training set sizes and comments on trends observed. For 5b, Two graphs, one for each classification set(0,1 and 3,5) showing trends of train and test negative log-likelihood over different training set sizes and comments on trends observed.
- Code: R code for 5a and 5b, code for extraction of training data and creating plots.

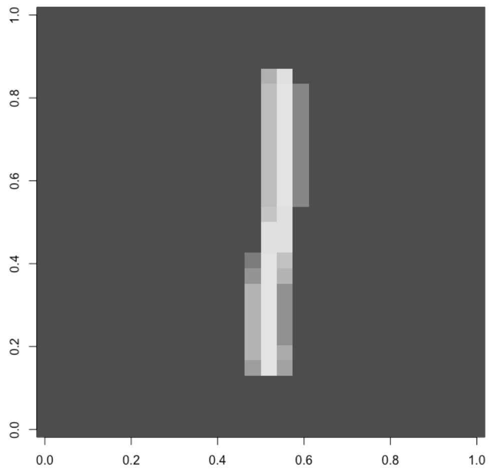
GT Account Name: tbobik3
Tyler Bobik

HW3: Logistic Regression

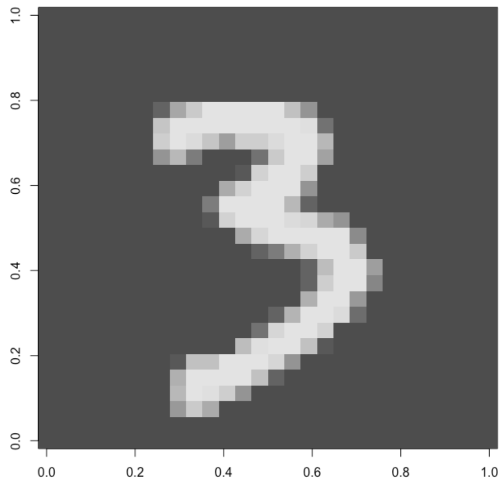
Problem 0



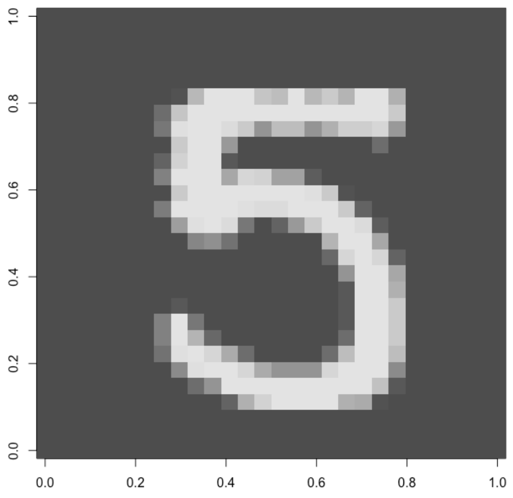
Class Label: 0



Class Label: 1



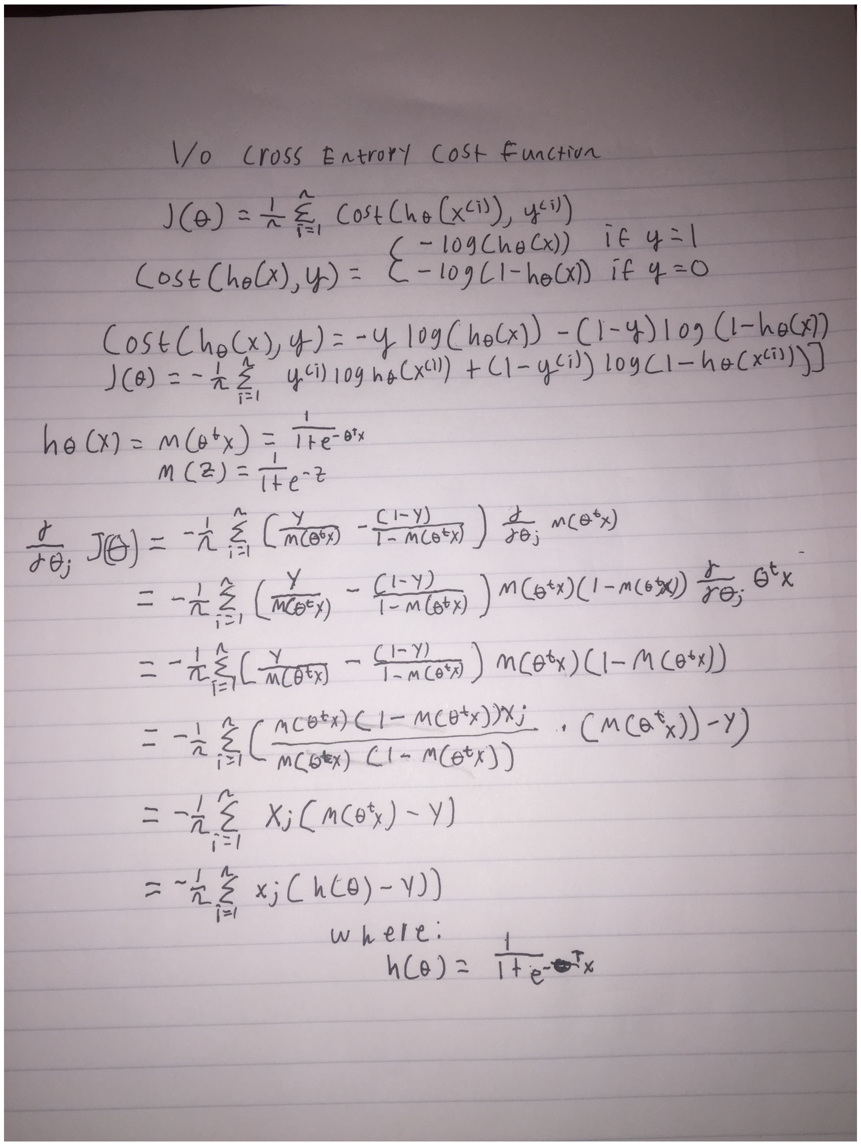
Class Label: 3



Class Label: 5

Problem 1

a)



1/0 Cross Entropy Cost Function

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}$$

$$\text{Cost}(h_\theta(x), y) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

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

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

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

$$\frac{\partial}{\partial\theta_j} J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y}{m(\theta^t x)} - \frac{(1-y)}{1-m(\theta^t x)}\right)\frac{\partial}{\partial\theta_j} m(\theta^t x)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y}{m(\theta^t x)} - \frac{(1-y)}{1-m(\theta^t x)}\right) m(\theta^t x)(1-m(\theta^t x))\frac{\partial}{\partial\theta_j}\theta^t x$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y}{m(\theta^t x)} - \frac{(1-y)}{1-m(\theta^t x)}\right) m(\theta^t x)(1-m(\theta^t x))$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left(\frac{m(\theta^t x)(1-m(\theta^t x))x_j}{m(\theta^t x)(1-m(\theta^t x))}\cdot(m(\theta^t x))-y\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n} x_j(m(\theta^t x)-y)$$

$$= -\frac{1}{n}\sum_{i=1}^{n} x_j(h(\theta)-y))$$

where:

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

b)

```
SET theta to all 0's
SET cost_hist = TRUE
SET increments = 0
WHILE increments < iterations & cost_hist == TRUE
    SET cost equal to Cost Function
    SET cost_hist_1 equal to the sum of the absolute value of Cost_hist
    SET theta equal to theta minus alpha multiplied by the cost
    INCREMENT increments
    IF cost_hist_1 < threshold THEN
        cost_hist = FALSE
RETURN theta
```

CostFunction
Grad equals transpose(trainx) %*% sig(trainx %*% (dot product) theta) minus trainy
RETURN Grad

SigFunction
RETURN (1 / (1 + exp(-1 * hypothesis)))
c)

1. Grad Decent:Trainx : n (d+1)
2. Costfn (transpose(trainx)) n (d +1)
3. Costfn/sig: sig(trainx %*% (dot product) theta): 2n (d+1)
4. Costfn: subtract y: (d + 1)
5. Grad DecentFn: cost_hist_1 equal: n
6. Grad Decent: theta equal to theta minus alpha multiplied by the cost 3(d + 1)
7. n(d+1) + n(d+1) + 2n(d+1) + (d+1) + n + 3(d+1)
8. dn + n + dn + n + 2dn + 2n + d + 1 + n + 3d + 3
9. 4dn + 3d + 5n + 4

Per a post on piazza I decided to not normalize the gradient of my cost function because it doesn't make a difference since it's a constant and can be assimilated in the learning rate

Problem 3

a)

Set	Threshold	Iterations	Alpha	Train Accuracy	Test Accuracy
0/1	.0001	200	.0001	99.87%	99.95%
3/5	.0001	200	.001	96.04%	96.21%

Theta: Initialized to all 1's

b)

a repeated 10 times and averaged

Set	Threshold	Iterations	Alpha	Train Accuracy	Test Accuracy
0/1	.0001	200	.0001	99.88%	99.95%

3/5	.0001	200	.0001	96.02%	96.28%

Theta: Initialized to all 1's

c) For set 0/1 I thought it was a little shocking to see that the test accuracy is higher than the train accuracy, even though both are very high, my assumption is that the model isn't too specific to the training set. My thoughts on why 0/1 has higher train and test accuracy is the numbers 0 and 1 are easier to differentiate than the numbers 3 and 5 which are a bit more similar, causing some misclassification.

d) If I had more than two classes to classify, using logistic regression I would use multinomial logistic regression. A main difference with multinomial regression is that we will use a softmax function instead of a sigmoid. I would use the linear model to output logits (my score) and I will use the linear model to update the weights in the training phase. I then will pass on the logits to a softmax function that finds the probabilities given the score. The probabilities of a softmax function is in the range from 0 to 1 and the sum of all the probabilities is one. After this I would use cross entropy to find the similarity distance between the probabilities from the softmax function and the target one-hot-encoding matrix. Next, I would use an iterated process to calculate the loss function and continue until the loss function's value is decreased.

Problem 4

a) For this problem I decided to have theta initialized randomly from 0 to 1 instead of just all 1's like I did in 3b.

Set	Threshold	Iterations	Alpha	Train Accuracy	Test Accuracy
3/5	.0001	200	.0001	95.52%	95.54%

We can see here that randomly initializing theta results in lower train AND test accuracies than 3b.

b)
Here I initialized theta to 1
For b I changed the way my gradient decent converges by calculating the sum of the absolute value of new theta calculated subtracted by the last theta calculated. If the sum ends up being less than the threshold than the iterations end.

Set	Threshold	Iterations	Alpha	Train Accuracy	Test Accuracy

3/5	.0001	200	.0001	95.95%	96.17

Here I decreased alpha from .0001(in 3b) to .00001

Set	Threshold	Iterations	Alpha	Train Accuracy	Test Accuracy
3/5	.0001	200	.00001	94.56%	95.20%

Here I increased alpha from .0001(in 3b) to .01

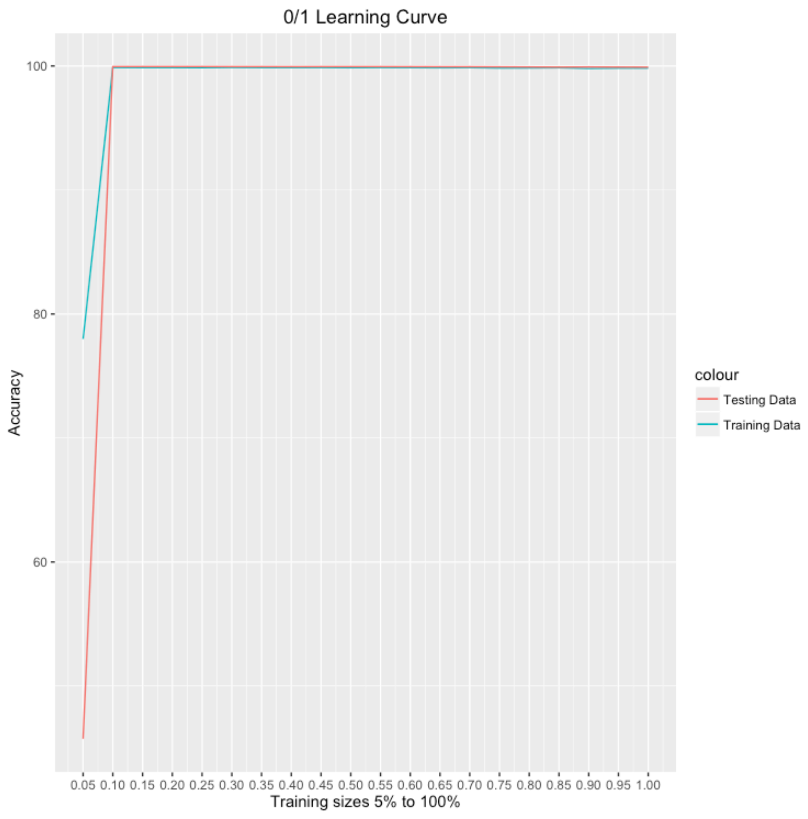
Set	Threshold	Iterations	Alpha	Train Accuracy	Test Accuracy
3/5	.0001	200	.1	85.03%	85.20%

We can see that the new convergence criteria improved the accuracy by a bit, but lowering or increasing the value of alpha does not increase the accuracy.

Problem 5

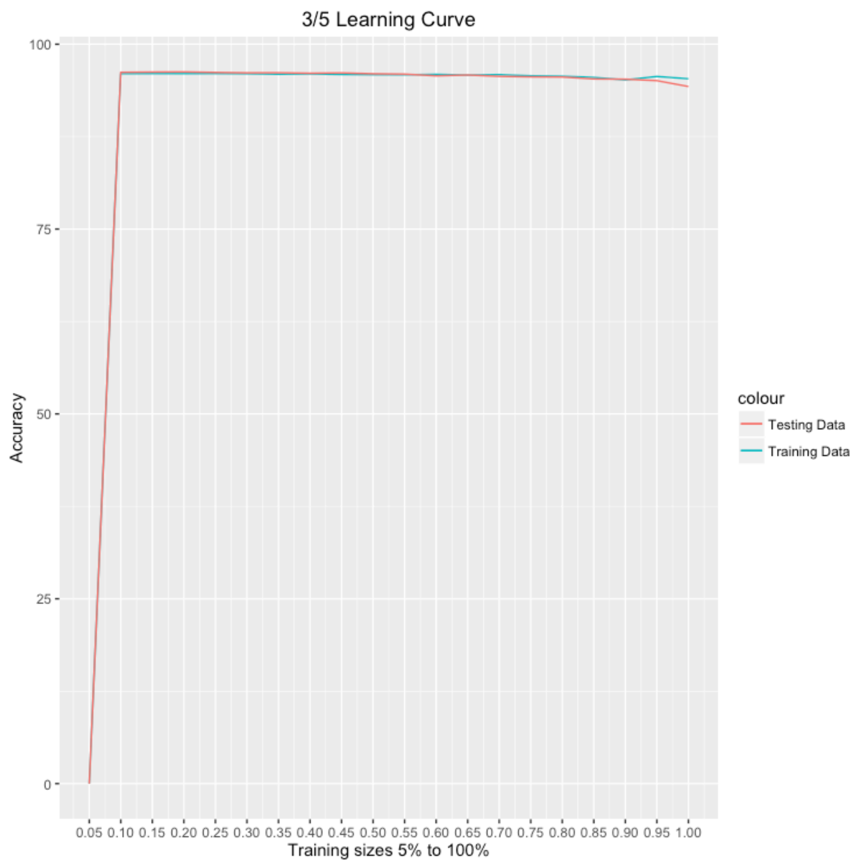
a)

10 runs from 5% to 100% of the training set.
Learning Curve 0/1



Set	Threshold	Iterations	Alpha
0/1	.0001	200	.0001

10 runs from 5% to 100% of the training set.
Learning Curve 3/5



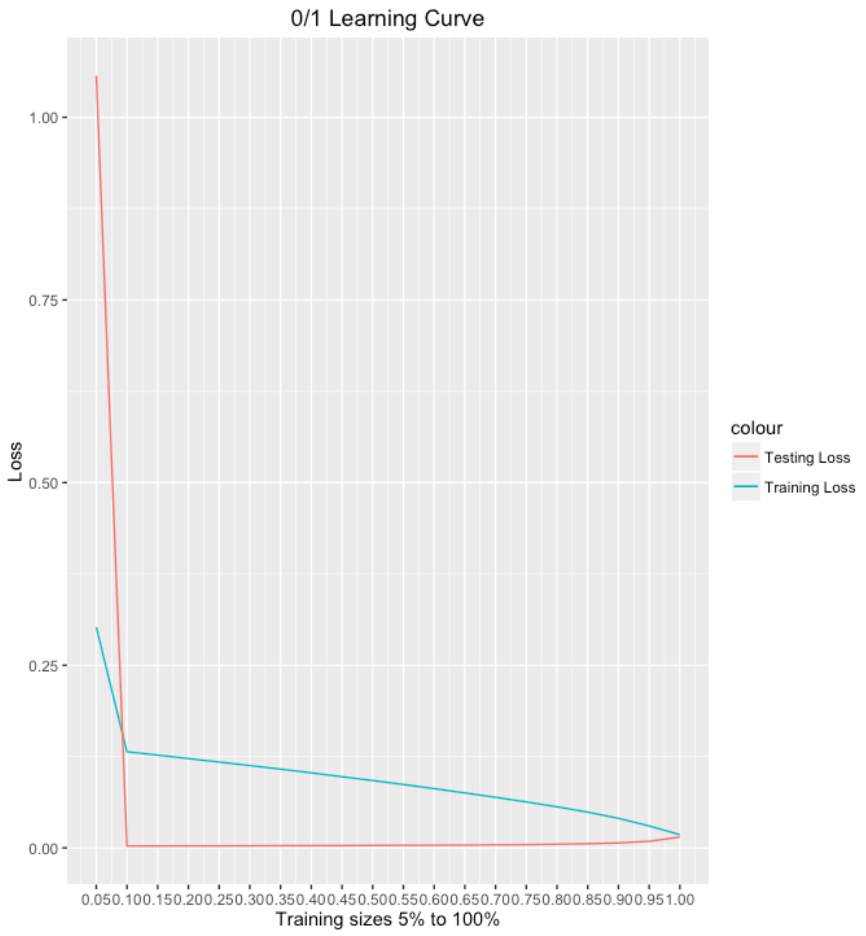
Set	Threshold	Iterations	Alpha
3/5	.0001	200	.0001

For both problem sets 0/1 and 3/5 you can see that it takes just 10% of the training data for gradient decent to converge.

b)

10 runs from 5% to 100% of the training set.

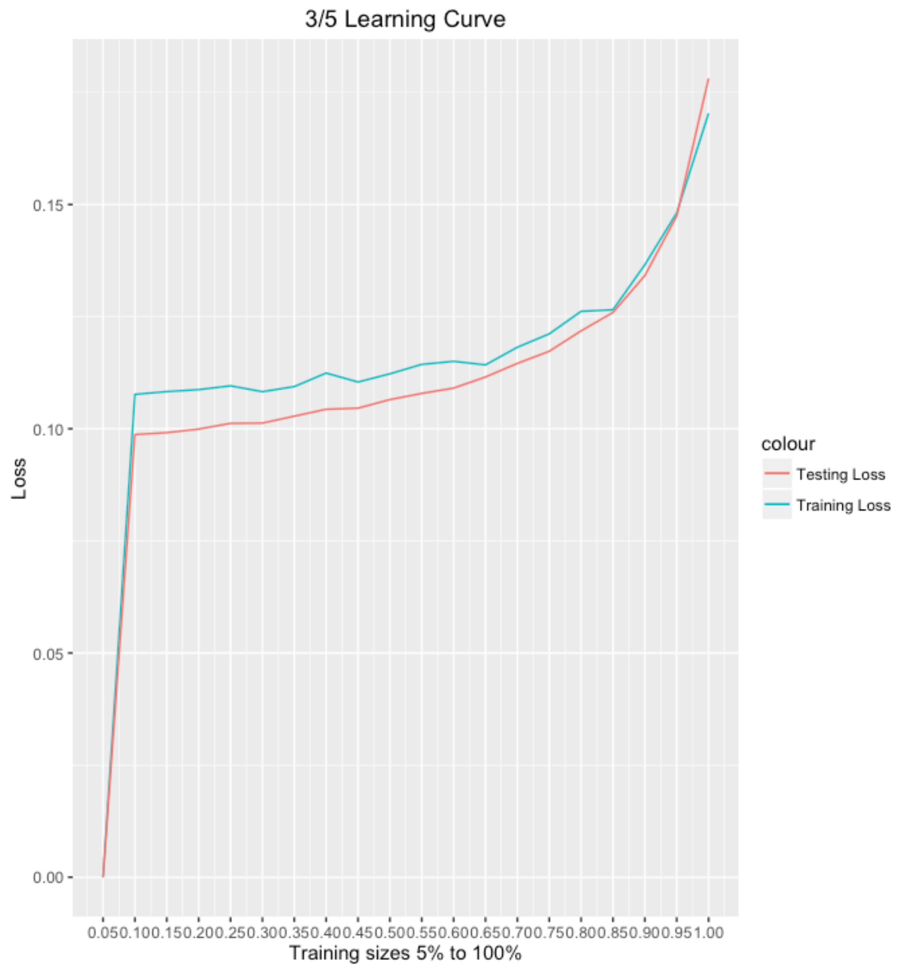
Learning Curve (Loss) 0/1



Set	Threshold	Iterations	Alpha
1/0	.0001	200	.0001

10 runs from 5% to 100% of the training set.

Learning Curve (Loss) 3/5



Set	Threshold	Iterations	Alpha
3/5	.0001	200	.0001

It is interesting that for the set 0/1 the testing loss is almost zero while the training loss is higher and eventually decreases as the training size goes to 100%. In the 3/5 set we can see that the training and testing loss increase as the training size increases which is interesting because the accuracy stays almost stagnant after 10% of the training data, this could be a sign of over fitting, but more testing would have to be done to confirm.