HW3: Logistic Regression [CSE-6242 - Data & Visual Analytics]

Saad Khan (skhan315@gatech.edu)
GT Account Name: **skhan315**

March 29, 2017

0 Data Preprocessing

Parts (a to f) : Code for parts 'a' to 'f' is covered in the `hw3.R` file.

Part (g) : Visualization of a single upright image from each class, i.e. 4 images corresponding to 0, 1, 3 and 5 are shown below in Figure 1 with descriptive text showing respective class labels [code included in `hw3.R`].

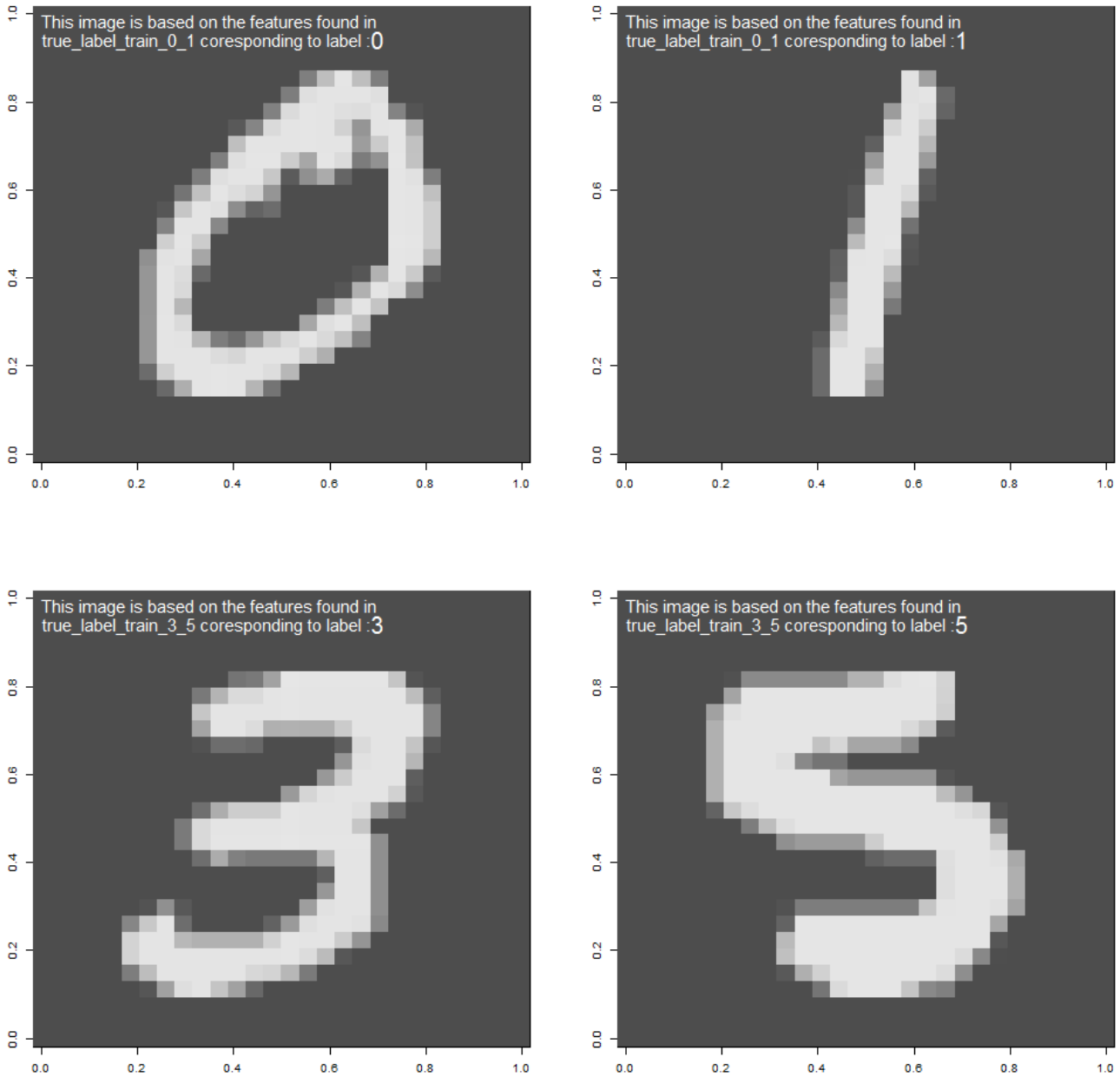


Figure 1: 4 sample images, one from each class, along with their class labels

1 Theory

1 (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?*

Derivation of the gradient of the loss function

A very well known method for training probabilistic classifiers (such as binary logistic regression in this case) is the maximum likelihood estimator (MLE). The resulting classifier provides a decision boundary in order to classify the predicted output as (0/1) or (-1/+1). In this case, all the derivations going forward are based on the loss function with labels (-1/+1).

The θ that maximizes the MLE can be represented as shown below in expression (1), as the product of conditional probabilities of the labels 'Y' given the feature vectors 'X'. As log is a concave function, i.e. it will not change the optimum and will make the derivation more easier converting the product of conditional probabilities to sum of log terms (logarithmic function here is natural log), the same expression in (1) can be converted in log-sum form as shown in expression (2).

$$\theta_{MLE} = argmax_\theta \quad p_\theta(Y = y^{(1)}|X = x^{(1)})...p_\theta(Y = y^{(n)}|X = x^{(n)}) \tag{1}$$

$$\theta_{MLE} = argmax_\theta \quad log(p_\theta(Y = y^{(1)}|X = x^{(1)})) + ... + log(p_\theta(Y = y^{(n)}|X = x^{(n)})) \tag{2}$$

$$p(Y = y|X = x) = \frac{1}{1 + e^{(y\langle\theta,x\rangle)}} \text{ , where } y \text{ is either +1 or -1} \tag{3}$$

Expression (3) is the formula for logistic regression which when substituted in MLE equation (2), results in the log(1/a) expression on the left side of (4) [i.e. the expression with the log(1/(1 + e)) term]. As log(1/a) can be written as -log(a), applying this to the left side of (4) we get the (-log) expression on the right side of 4 [i.e. the expression with the -log(1 + e) term]. To remove the minus(-) from this expression **argmax** can be converted to **argmin** to get the expression in (5).

$$\theta_{MLE} = argmax_\theta \sum_{i=1}^{n} log\frac{1}{1 + e^{(y^{(i)}\langle\theta,x^{(i)}\rangle)}} = argmax_\theta \sum_{i=1}^{n} -log(1 + e^{(y^{(i)}\langle\theta,x^{(i)}\rangle)}) \tag{4}$$

NOTE : The formula in (5) for computing the gradient of the loss function is based on the comments by the Professor in the post **Start of Week 10**, discussions in post **More Notes for HW3** and page 11 of **Brent's Notes L4**. The starting point of the actual derivation is the loss function (equation 8) from section 5 of **logreg.pdf** as shown here.

$$\theta_{MLE} = argmin_\theta \sum_{i=1}^{n} log(1 + e^{(y^{(i)}\langle\theta,x^{(i)}\rangle)}) \tag{5}$$

Now because of **argmin**, the task is to obtain vector θ that minimizes this function instead of maximizing it. This can be done by using the gradient descent method, i.e. by applying partial derivative w.r.t. θ. Following equations show the step by step procedure to compute the gradient of this loss function. Equation (5) can now be expressed in partial derivative form as shown in the equation on the left side of (6).

$$\frac{\partial \sum_{i=1}^{n} log(1 + e^{(y^{(i)}\langle\theta,x^{(i)}\rangle)})}{\partial\theta} \quad \text{is of the form} \quad \frac{\partial log(1 + e^{ab\theta})}{\partial\theta} \tag{6}$$

As the partial derivative is to be calculated w.r.t. to θ so variables x and y are considered constants. These can be represented as a and b respectively as in the form shown on right side of (6). **NOTE :** I have left out the $\sum$ for ease of mathematical manipulation as it is based on i rather than θ and makes no difference in the partial derivative calculations.

$$\textbf{Chain Rule :} \ \frac{df\,(\theta)}{dx} = \frac{df}{du} \cdot \frac{du}{dx} \text{ , Let } u = 1 + e^{ab\theta} \text{ and by substituting we get : } \frac{\partial log(u)}{\partial u} . \frac{\partial log(1 + e^{ab\theta})}{\partial\theta} \tag{7}$$

Now using chain rule for differentiation and substitution of u, we can express the partial derivative as shown on right side of (7) [i.e. the product of partial ∂u and $\partial\theta$]. As this is natural log, by solving this expression above, we get the solution as shown in (8). Now, we bring the $\sum$ w.r.t. i, substitute x &y back to generate (9), the actual formula for computing the gradient of the loss function used in Logistic Regression.

$$\frac{\partial log(u)}{\partial u} = 1/u \text{ and } \frac{\partial log(1 + e^{ab\theta})}{\partial\theta} = abe^{ab\theta} \text{ , substituting } u \text{ from (7) we get } = \frac{abe^{ab\theta}}{1 + e^{ab\theta}} \tag{8}$$

$$\textbf{Formula for gradient of the loss function :} \ \sum_{i=1}^{n} \frac{y^{(i)}.x^{(i)}.e^{(y^{(i)}\langle\theta,x^{(i)}\rangle)}}{1 + e^{(y^{(i)}\langle\theta,x^{(i)}\rangle)}} \tag{9}$$

Complete gradient descent update rule

The expression in (9) alone can not be used to minimize the vector θ. In order to do that, a learning rate α is combined to form the gradient descent update rule as shown in (10). Here the $x^{(i)}$ now has subscript j and becomes $x_j^{(i)}$ because the j^{th} value of x comes out of the partial derivative w.r.t. to θ_j while all other non-j elements are 0 for a single weight component.

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{n} \frac{y^{(i)}.x_j^{(i)}.e^{(y^{(i)}\langle\theta, x^{(i)}\rangle)}}{1 + e^{(y^{(i)}\langle\theta, x^{(i)}\rangle)}} \text{ , where } j = 1, \ldots \ d \text{ (no. of dimensions)} \tag{10}$$

Once, the change in θ for each j is less than a certain threshold ϵ, θ can then be used as the parameter vector for the logistic regression classifier. To avoid running into ∞ values (based on the piazza post **1030**) the expression in (10) can further be simplified by multiplying and dividing with the term $e^{(-y^{(i)}\langle\theta_j, x_j^{(i)}\rangle)}$, which results in a simplified expression shown in (11). This expression is used for the final implementation of the batch gradient descent update function.

$$\textbf{Equation for } \theta \textbf{ update : } \theta_j = \theta_j - \alpha \sum_{i=1}^{n} \frac{y^{(i)}.x_j^{(i)}}{e^{(-y^{(i)}\langle\theta, x^{(i)}\rangle)} + 1} \text{ , where } j = 1, \ldots \ d \text{ (no. of dimensions)} \tag{11}$$

Description of the variables in gradient equation

- θ **:** also known as the **parameter** vector, describes the classifier that will be generated after applying the gradient descent rule. This vector has length **d** (no. of features in the dataset) where a single element of this vector can be denoted by θ_d.
- **x :** is the 2D matrix that describes the **measurements**. For a dataset having **d** features, x_d can represent a single feature while $x^{(i)}$ represents the entire feature vector for a particular sample **i** in the form $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \ldots, x_d^{(i)})$.
- **y :** is the response variable or the **label**. For a binary classification task, as is the case here, y can take values (1, -1). For a measurement row $x^{(i)}$, label can be represented as $y^{(i)}$.
- **i :** is the i^{th} sample of the dataset which is used to perform summation on all samples and goes up to the last sample n.
- $\boldsymbol{\alpha}$: is the learning rate, which helps with convergence and minimizes the changes in the vector θ over time.

1 (b) : *Write pseudo code for training a model using Logistic Regression?*

Pseudo-code for training the model using Logistic Regression

Algorithm 1 Gradient_Descent()

```
θ ← weights [d]                                   ▷ initialization
for t = 1 → z do                                  ▷ t iterations to check if convergence criteria is less than a certain threshold
    for j = 1 → d do                              ▷ iterating over the no. of dimensions d
        sigma ← 0
        for i = 1 → n do                          ▷ iterating over the no. of samples n
            sigma ← sigma + y^(i).x_j^(i) / (e^(−y^(i)⟨θ,x^(i)⟩) + 1)   ▷ gradient of the loss function computation
        end for
        θ[j] ← θ[j] − (α  x  sigma)               ▷ updating the gradient descent weights based the learning rate
    end for
    if convergence criteria < epsilon then break  ▷ checking for the convergence criteria to break the outer loop
end for
return(θ)                                         ▷ returning the complete updated weight vector
```

The pseudo code is based on the expression in equation (11) for the gradient_descent() function which will be implemented later on in code for Section 2. The pseudo-code is based on the gradient of the loss function covered in section 1(a), i.e.

where output labels are (-1/+1). Firstly, the pseudo code shows the initialization of θ that will be required for training the model using logistic regression. The first **For** loop is the iteration loop, only used to limit the maximum times the function updates the θ vector based on a convergence criteria. The 2 nested **For** loops are the ones actually computing the gradient descent and updating the θ vector.

NOTE : The pseudo code here is loop based, however, the implementation in Section 2 is vectorized as confirmed in the piazza post **952**.

1 (c) : *Calculate the number of operations per gradient descent iteration?*

Following table covers all the different operations per the gradient descent iteration. To be consistent, the operation calculations in this section are based on the pseudo-code in section 1(b). Although, the actual code is vectorized, the number of operations remain still the same as answered in piazza post **1081**.

Task	Description	No. of operations
Initializations [θ , sigma]		
Array Initialization (θ)	intializing weights for θ with d dimensions	**d**
Variable Initialization (sigma)	intialized once every dimension: column sum per row	**t x d**
Loops [convergence , dimesnions , samples]		
Convergence - For Loop (t)	iterates until convergence criteria is less than a delta threshold	**t**
Dimension - For Loop (d)	iterates over all of the dimensions in the dataset	**d**
Samples - For Loop (n)	iterates over all of the samples in the dataset	**n**
Algebric Operations [multiplications , additions , dividions , exponent]		
Multiplications (numerator)	loss function numerator: multiplying y and x	**t x d x n**
Multiplications (dot product)	inside exponent: dot product multplications between θ and x	**t x d x n**
Additions (dot product)	inside exponent: dot product additions (n - 1) operation	**t x d x n**
Exponent term (y . <θ, x>)	exponent operation	**t x d x n**
Addition (denominator)	loss function denominator: addition of 1	**t x d x n**
Divisions (numerator/denominator)	Before updating sigma: numerator divided by denominator	**t x d x n**

2 Implementation

Code for this section is covered in the `hw3.R` file. Functions included for this section are as following:

- **gradient_descent() :** Batch gradient descent is implemented using the vectorized form of the pseudo code presented in 1(b). This function takes training samples (x), training labels (y), initial θ vector, initial α, convergence criteria threshold ϵ and number of iterations as inputs and generates the converged θ vector as the output.
- **logistic_regression() :** takes in testing samples (x) (train or test) and the converged θ vector as the input and outputs a probability vector based on the formula for logistic regression as shown in equation (3) in section 1(a).

3 Training

Before training and testing the logistic regression method implemented in section 2, two additional methods were created. Code for these methods is in the `hw3.R` file. Functions included for this section are as following:

- **predict() :** this method takes the probability vector as the input and outputs a label vector based on the logistic regression formula where p > 0.5 gets +1, p < 0.5 gets -1 label while p $= 0.5$ is assigned (-1/+1) label randomly.
- **accuracy() :** takes actual and predicted labels as inputs and computes the percentage of correctly classified labels.

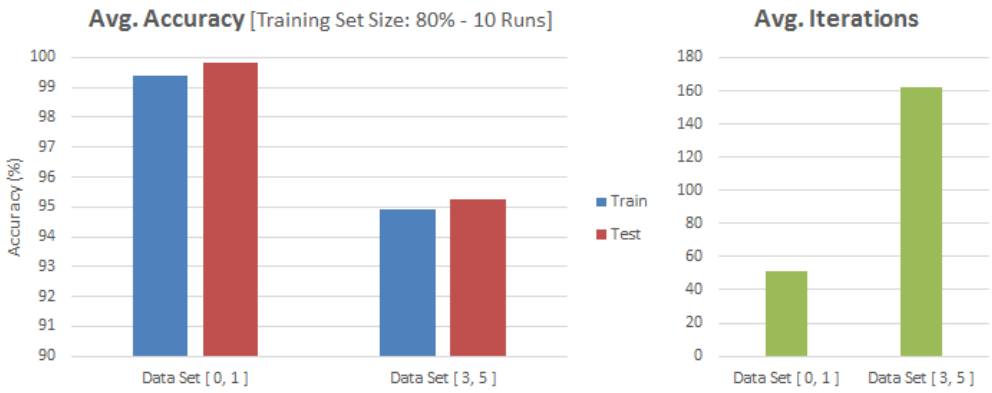
3 (a) : *Train 2 models, one on the train_0_1 set and another on train_3_5, and report the training and test accuracies?*

For this part of the homework, complete(100%) training sets, train_0_1 and train_3_5 were used to train 2 separate models. Parameters used to train both these models are shown in Figure 2[RIGHT]. For this part, θ was initialized randomly between 0 and 1 and convergence used was L1-Norm. Accuracy results for both the training and the test sets with a single run for the respective models are shown in Figure 2[LEFT]. It can clearly be seen that [0 ,1] model performs better compared to the [3,5] model in terms of accuracy and θ also takes less iterations to converge as shown by no. of iterations required for convergence in Figure 2[CENTER]. It can also be seen that test set accuracy for both [0,1] and [3,5] models is better than the training set accuracy, which is not the case normally when training and testing classification models.

Parameters	
Training Set Size	100%
Initial θ	runif(784)
Initial α	0.0001
Conv. Criteria	L1-Norm
threshold ε	0.25

Figure 2: [LEFT] : Single Run Accuracy, [CENTER] : Iterations, [RIGHT] : Parameters

3 (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?*



Parameters	
Training Set Size	100%
Initial θ	runif(784)
Initial α	0.0001
Conv. Criteria	L1-Norm
threshold ε	0.25
Sampling	without replacement

Figure 3: [LEFT] : Average Accuracy of 10 runs, [CENTER] : Average Iterations, [RIGHT] : Parameters

Figure 3[LEFT] shows the average accuracy values for 10 runs when randomly sampled (without replacement) 80% of train_0_1 and train_3_5 sets were used to train 2 separate models. Initial θ vector selected for this part of the assignment was same as section 3(a). Again, it can be seen that model trained for [0,1] cases performs better than the [3,5] model and also requires lesser no. of iterations on average to converge. Accuracy values of individual runs for each model are reported in Figure 4 where it can again be observed that test accuracy for both models is better than the training accuracy.

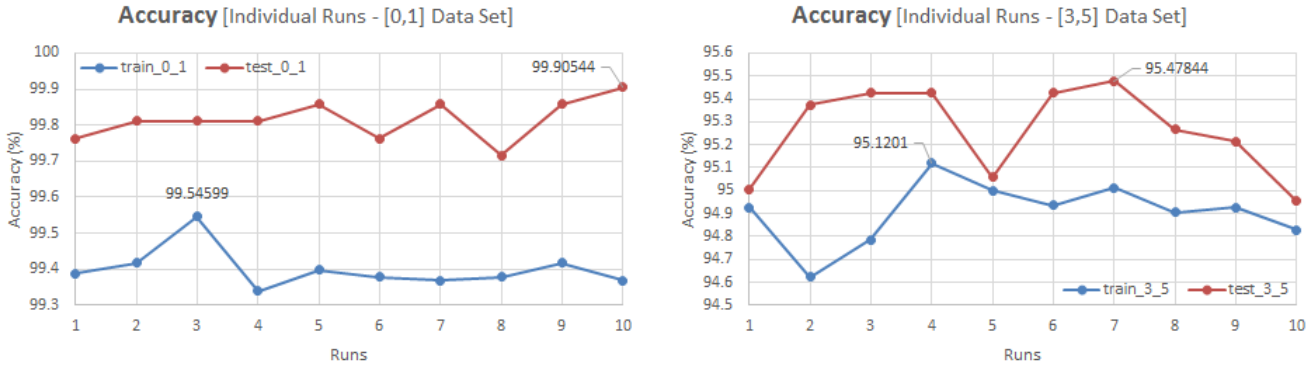


Figure 4: [LEFT] : [0,1] model - Accuracy over 10 runs, [RIGHT] : [3,5] model - Accuracy over 10 runs

3 (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?*

There are 2 main observations based on the results in 3(a) and 3(b). These are addressed in the following subsections.

1. Training accuracy less than Test accuracy

For both [0,1] cases and [3,5] cases, it can clearly be seen that test set accuracy is better than the training set accuracy. This is the case in both section 3(a) and 3(b). Following are some of the reasons that could give such results:

- **test set size is small :** it is possible to have higher test accuracy if the test set is too small compared to the training set and the model fits it better than the training set. This is in fact the case here as test_0_1 and test_3_5 are about only 14% of their respective complete datasets. A higher percentage of test set might show test accuracy less than training set accuracy.
- **noise and data split :** in addition to being quite small, it is also possible that the test set has less noise compared to the training set. As the model here is generated based on images, it is possible that the way train and test sets were split, 'cleaner' (less noisy) images went to the test set while 'dirtier' (more noisy) images went to the training set.
- **model generalization :** this is in tandem with the 2nd point. It is also possible that the images in the test set are generally easier to predict, i.e. having less variance and therefore are not influenced by something that is not captured by the model.

2. Accuracies for 0,1 cases are higher than the 3,5 cases

Both train and test accuracy values for the [0,1] cases are higher than the [3,5] cases. This can be explained by the images shown for all 4 cases in Figure 5. For the 0 and 1 case (images on the left), areas marked in red are the ones where generally pixels from both images would be hard to detect while the areas marked in yellow are the ones where both images can be identified separately. The 3 and 5 case (images on the right) have the similar areas marked as well.

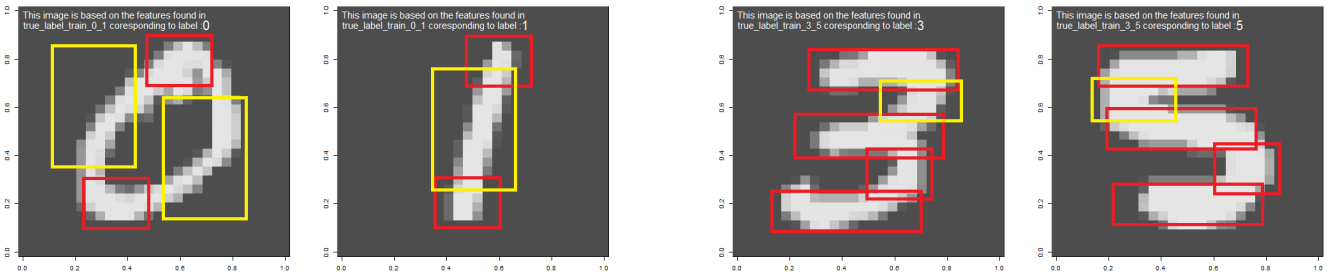


Figure 5: Differences and similarities between images

Looking at the images, it can clearly be seen that in general, the model would get less confused for the 0,1 case compared to the 3,5 case as the 0,1 case has fewer number of pixels (marked in red) that could lead the model to predict incorrectly between 0 and 1 while, on the other hand, there are much more areas marked in red in the 3,5 case which could lead the model to predict incorrectly much more compared to the 0,1 case, hence the gap between the accuracies(around 99% for the [0,1] case and around 95% for the [3,5] case) as shown in Figure 2 and 3.

3 (d) : *This assignment deals with binary classification. Explain what you would do if you had more than two classes to classify, using logistic regression?*

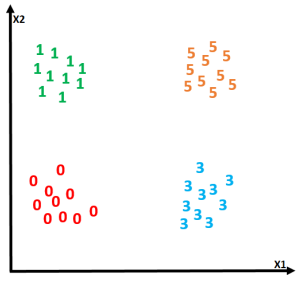


Figure 6: 4 classes in 2-D

To explain how logistic regression would work for more than 2 classes, MNIST training dataset can be considered as an example, i.e. containing data for all 4 classes (0, 1, 3 and 5). For ease of understanding, visualization in just 2 dimensions

is shown by the graph in Figure 6 with respective class labels. To apply logistic regression to a single problem to classify 4 images we can create 4 separate 2 class (binary) problems. This can be done by creating 4 pseudo training sets where the first problem will have +1s for image(0) and -1s all other images(1, 3, 5). Similar activity can be performed for the other 3 classes. An illustration of each such separate binary problem is shown in Figure 7.

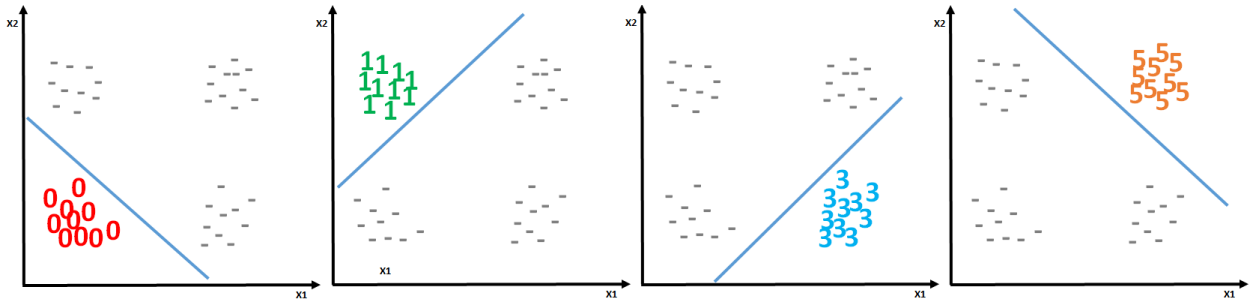


Figure 7: 4 separate binary classification problems

If 4 different logistic regression models are trained using these 4 pseudo training sets, classification decision boundaries similar to the ones shown in Figure 7 could be imagined. These trained models can be named as $model^{(0)}$, $model^{(1)}$, $model^{(3)}$ and $model^{(5)}$ respectively where general classification model can be written as shown by expression (12).

$$model_{\theta}^{(d)}(x) = p(Y = d|x;\theta) \text{ , where } d \text{ is 0, 1, 3 or 5} \tag{12}$$

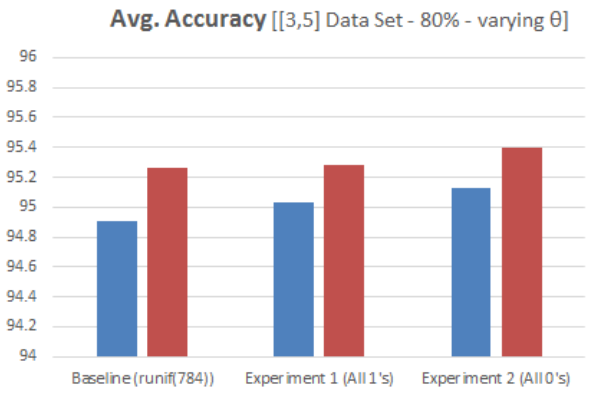
When predicting the label in the multi-class scenario for a given training example (x), all 4 classifiers are run and x is assigned the class of the model which gives the highest probability $[max_i \quad model_{\theta}^{(d)}(x)]$. For example: for a training example(x) if the probabilities are [$model^{(0)}$: 0.71, $model^{(1)}$: 0.34, $model^{(3)}$: 0.85, $model^{(4)}$: 0.27] for all 4 classifiers then (x) can be classified as image 3. This way a binary logistic regression classifier can be used to handle multi-class classification problems. The way a binary logistic classifier was converted to handle more than 2 classes is called *one vs. all* or sometimes *one vs. rest* method.

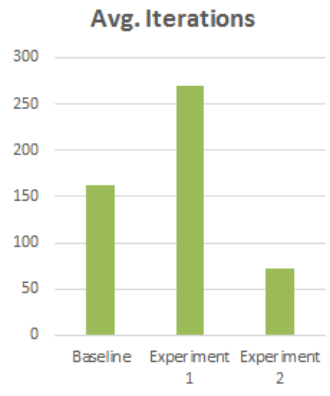
4 Evaluation

4 (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?*

4 (a) - 1 : Changing Theta (θ) - [Baseline : random(0,1), Experiment 1: All 1's, Experiment 2: All 0's]

For this section, only different initial values of the θ vectors were experimented (different from baseline, i.e. section 3(b)) based on the homework instructions [initial_parameters_experiment, convergence_criteria_baseline]. Parameter table in Figure 8[RIGHT] shows the 2 different initializations of θ. In contrast to randomly initializing θ [baseline 3(b)] first I initialized θ to all 0's and then to all 1's. The average train/test accuracies based on 10 runs were recorded only for the 3,5 set as shown in Figure 8[LEFT]. It can clearly be seen that experiment 2 where θ was initialized as all 1's (rep(1, 784)) is the best performing configuration as it resulted in best average train (95.13%) and test (95.39%) accuracy and also helped the model converge in least no. of iterations, i.e. less than 100 iterations while models for baseline (θ = runif(784)) and experiment 1 (θ = rep(0, 784)) both took more than 150 iterations to converge.



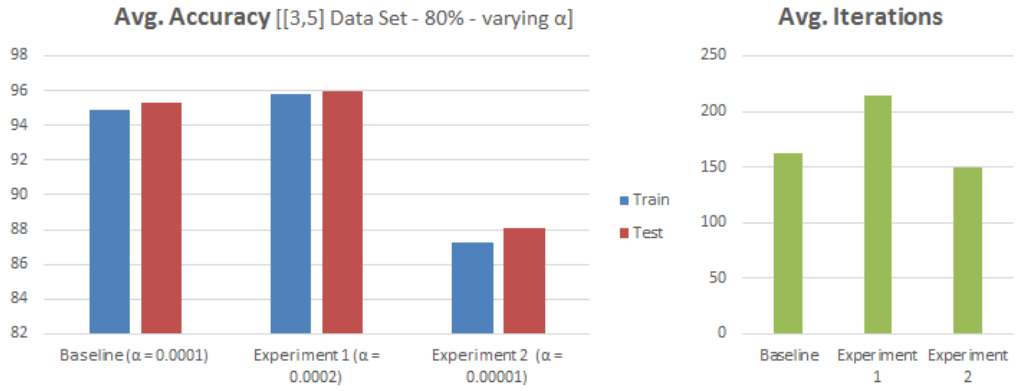


Parameters	
Training Set Size	100%
Baseline θ	runif(784)
Experiment 1 θ	rep(1, 784)
Experiment 2 θ	rep(0, 784)
Initial α	0.0001
Conv. Criteria	L1-Norm
threshold ε	0.25
Sampling	without replacement

Figure 8: [LEFT] : Average Accuracy of 10 runs, [CENTER] : Average Iterations, [RIGHT] : Parameters

4 (a) - 2 : Changing Alpha (α) - [Baseline : 0.0001, Experiment 1: 0.0002, Experiment 2: 0.00001]

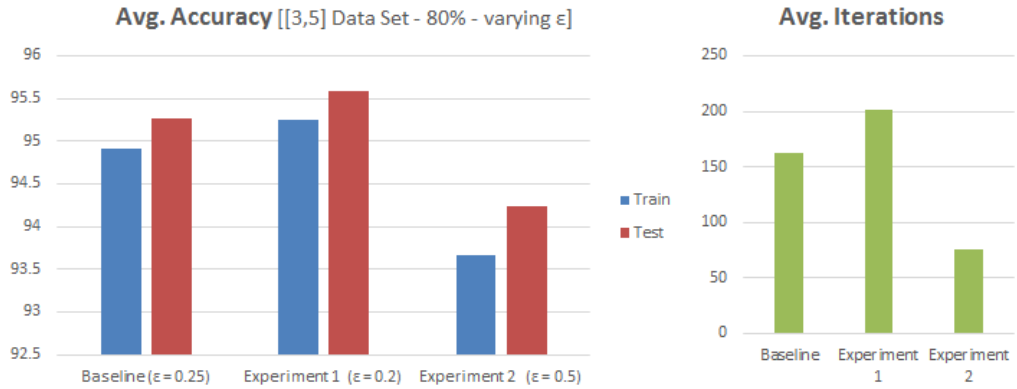
For this experiment, learning rate (α) was changed while all of the other parameters were kept same as the baseline as shown in Figure 9[RIGHT]. 2 experiments were conducted with values for α set at 0.0002 and 0.00001 while baseline was 0.0001. By looking at the results of the average accuracies for these experiments in Figure 9 [LEFT], it can be determined that lowering the learning rate does not help the model fully converge as the average train/test accuracies with $\alpha = 0.00001$ are well below 90%. On the other hand, with $\alpha = 0.0002$, there is a slight improvement in the average train/test accuracy compared to baseline but at the cost of much more iterations (above 200) where as baseline model converged in around 150 iterations on average. so, taking this fact into consideration, it can be said that the baseline model with $\alpha = 0.0001$ is the best performer. **NOTE :** I also tried to use α greater than 0.0002 but model was taking too long to converge.



Parameters	
Training Set Size	100%
Initial θ	runif(784)
Baseline α	0.0001
Experiment 1 α	0.0002
Experiment 2 α	0.00001
Conv. Criteria	L1-Norm
threshold ε	0.25
Sampling	without replacement

Figure 9: [LEFT] : Average Accuracy of 10 runs, [CENTER] : Average Iterations, [RIGHT] : Parameters

4 (a) - 3 : Changing Epsilon (ϵ) - [Baseline : 0.25, Experiment 1: 0.2, Experiment 2: 0.5]



Parameters	
Training Set Size	100%
Initial θ	runif(784)
Initial α	0.0001
Conv. Criteria	L1-Norm
Baseline ε	0.25
Experiment 1 ε	0.2
Experiment 2 ε	0.5
Sampling	without replacement

Figure 10: [LEFT] : Average Accuracy of 10 runs, [CENTER] : Average Iterations, [RIGHT] : Parameters

The third parameter that was changed was the convergence threshold (ϵ). Again, 2 experiments were conducted with $\epsilon = 0.2$ and 0.5 to compare the results with baseline $\epsilon = 0.25$. All other parameters were kept constant as shown in the table in Figure 10[RIGHT] and the model was trained 10 times with randomly chosen 80% of the training set. As expected, with $\epsilon = 0.5$, average train/test accuracies and iterations to converge were lower than baseline, while on the other hand, model trained with $\epsilon = 0.2$ has slightly better train/test accuracies than baseline (around 0.3% better) but took around 200 iterations to converge. Keeping the no. of iterations as a factor into consideration, overall the baseline configuration still performed better than the 2 experiments.

4 (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?*

4 (b) - 1 : Changing Convergence Criteria - [Baseline : L1-Norm, Experiment : L2-Norm]

For all the models trained up till now (including baseline (3b) and all models in 4(a)), I used L1-Norm as the convergence criteria. The formula for L1-Norm used is $sum(abs(old\_theta - current\_theta))$. For each theta update, sum of old_theta vector is compared to the sum of the current and if the absolute difference is less than ϵ, learning is stopped.

In this experiment, the models were trained based on the L2-Norm, which is $sqrt(sum((old\_theta - current\_theta)^2))$. Again 2 experiments were conducted, one with L2-Norm as the new convergence criteria (keeping all other parameters

constant (as suggest in the homework assignment [initial_parameters_baseline, convergence_criteria_experiment] and the other experiment with keeping the L2-Norm as the convergence criteria but modifying just ϵ in order to maximize the performance of the model. For this purpose, as suggested in the homework instructions, a new gradient descent method *gradient_descent_4b*$_1$() incorporating L2-Norm instead of L1-Norm was used to train the model. By looking at the accuracies in Figure 11[LEFT], it can clearly be seen that average accuracies for both train and test set based on baseline are still better than the 2 experiments. Experiment 1 with L2-Norm (convergence criteria) and keeping all initial parameters same as 3(b) resulted in a poor performance as it can be seen that accuracies dropped below 90%. Experiment 2, with lower value for ϵ, took a lot of iterations (more than 230 on average) to converge with train/test accuracies still not surpassing the baseline accuracies. For this scenario, clearly baseline with convergence criteria L1-Norm performed better than the experiments.

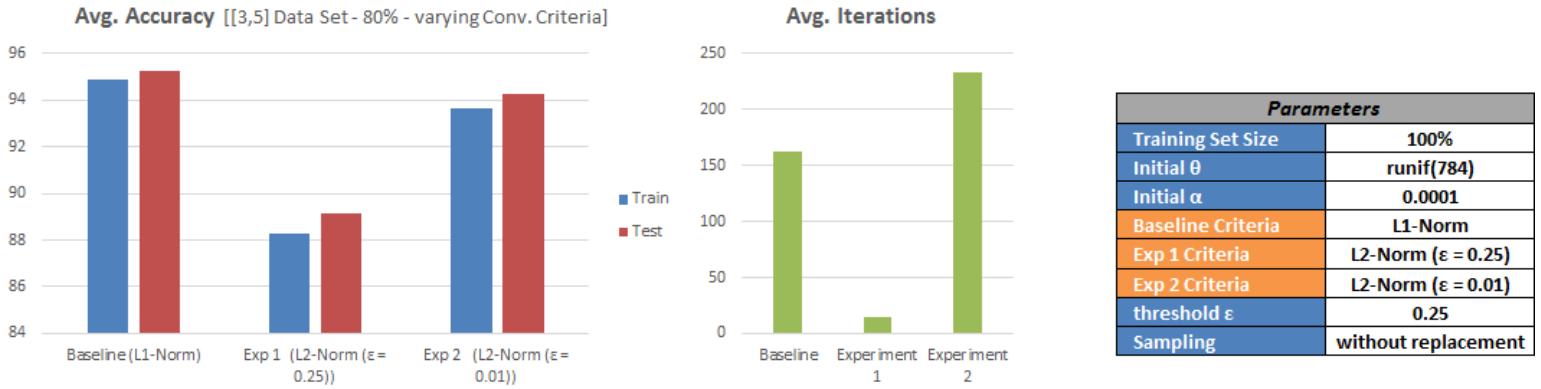


Figure 11: [LEFT] : Average Accuracy of 10 runs, [CENTER] : Average Iterations, [RIGHT] : Parameters

4 (b) - 2 : Changing Convergence Criteria - [Baseline : L1-Norm, Experiment : Comparing $\theta_{(j)}$ to $\theta_{(j-1)}$]

Another set of experiments was run by deploying a different convergence criteria based on difference between individual θ values in the weight vector. For this purpose, a modified gradient descent method *gradient_descent_4b_2*() was created. Using this criteria and keeping all of the initial parameters unchanged, experiment 1 was run, which showed very poor results. The model stopped in exactly 4 iterations for all 10 runs and as a result the train/test accuracies were around 60% as shown in Figure 12[LEFT]. However, lowering ϵ to 0.002 for experiment 2 (instead of baseline value of 0.25) resulted in a good performance with train/test accuracies slightly better than baseline. Although, this tweaked configuration took 198 iteration to train on average but still had almost balanced train and test accuracy values. In this case, baseline configuration is still better than the experimental configuration purely on the basis of no. of iterations it took to train. This convergence criteria (comparing $\theta_{(j)}$ to $\theta_{(j-1)}$) was the pick of the lot in terms of comparable train and test accuracies. Using this criteria, the model trained was able to generate train and test accuracy results which were very close to each other compared to the experiments conducted earlier.

Figure 12: [LEFT] : Average Accuracy of 10 runs, [CENTER] : Average Iterations, [RIGHT] : Parameters

SUMMARY: Overall, the best performing configuration amongst all the different experiments conducted as part of question 4 was where θ was set to all 0's while all the other parameters were kept constant as baseline 3(b). This resulted in both average train/test accuracy to be above 95% and the model on average took 72 iteration to train.

5 Learning Curves

5 (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?*

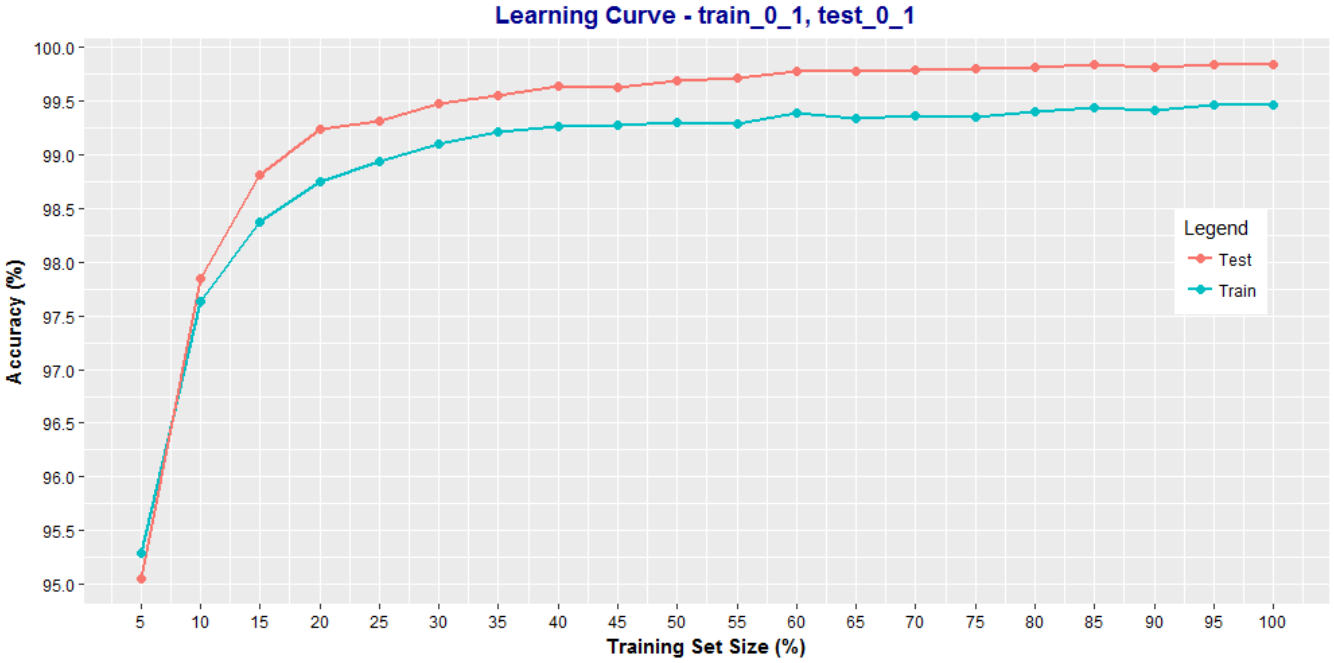


Figure 13: Learning Curve : [0,1] Data Set

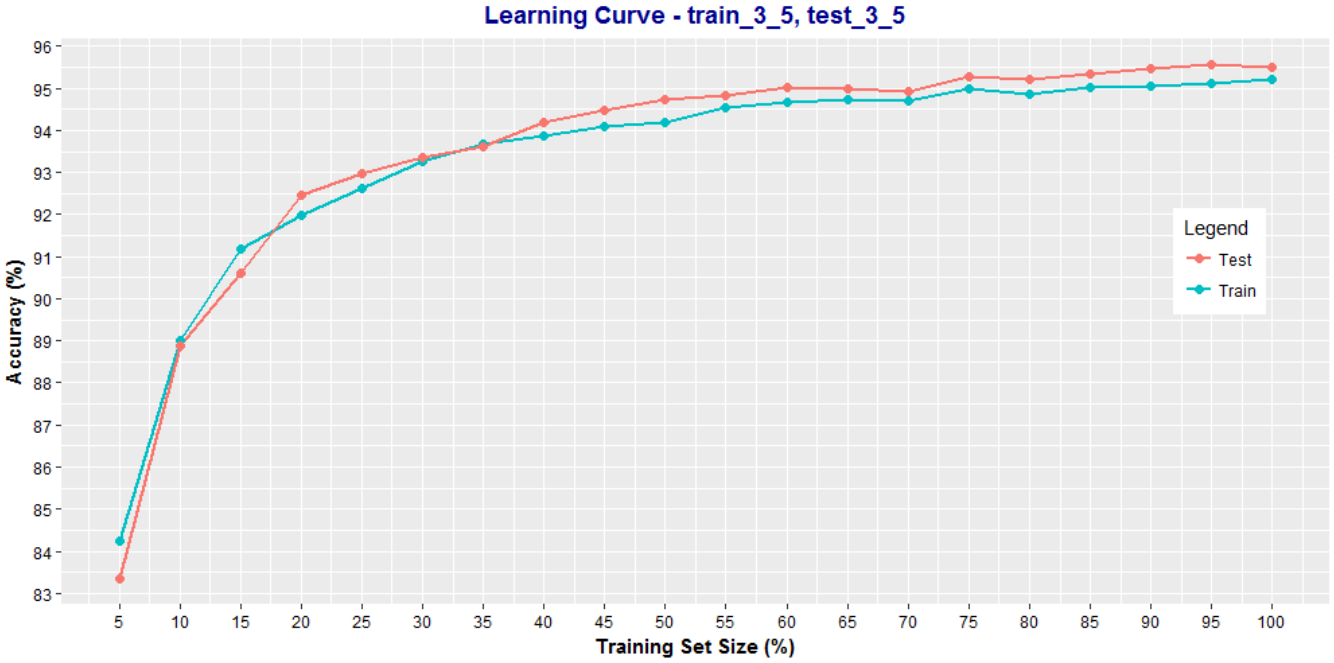


Figure 14: Learning Curve : [3,5] Data Set

Figure 13 and 14 show the learning curves for data set [0,1] and [3,5] respectively. Learning curves were generated using the baseline configuration 3(b) for both the data sets. It can clearly be seen that as the % training set size increases, so does the accuracy for both the data sets. Both learning curves also highlight the fact, which was determined in the earlier sections of this report, that test set accuracy is better than training set accuracy. Although for the [3,5] data set train and test accuracy values are very close to each other for most part of the plot but still test accuracy is better than the training accuracy.

As highlighted in the answer for question 3(c), there can be multiple reasons as to why the test accuracy is better than training accuracy, however, by looking at the trend we can say that both models for [0,1] and [3,5] are neither overfitting nor good fit to the data.

As the training set size increases, closely packed lines for both training and test data in case of [3,5] data set signify that there is low variance in the data. On the other hand, the curves for the [0,1] case are still closely packed but with a relatively bigger gap compared to the [3, 5] learning curve. This is understandable as in the [0,1] case, the images to predict are clearly distinguishable (0 and 1) while not quite easy to distinguish for the [3,5] case (explained in section 3(d)).

Overall, accuracy values for both the datasets are consistent (curves are almost flat for training set size greater than 50%) with what was observed in the earlier sections, i.e. greater than 99% (train and test) for the [0,1] data set and around 95% (train and test) for the [3,5] data set.

5 (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?*

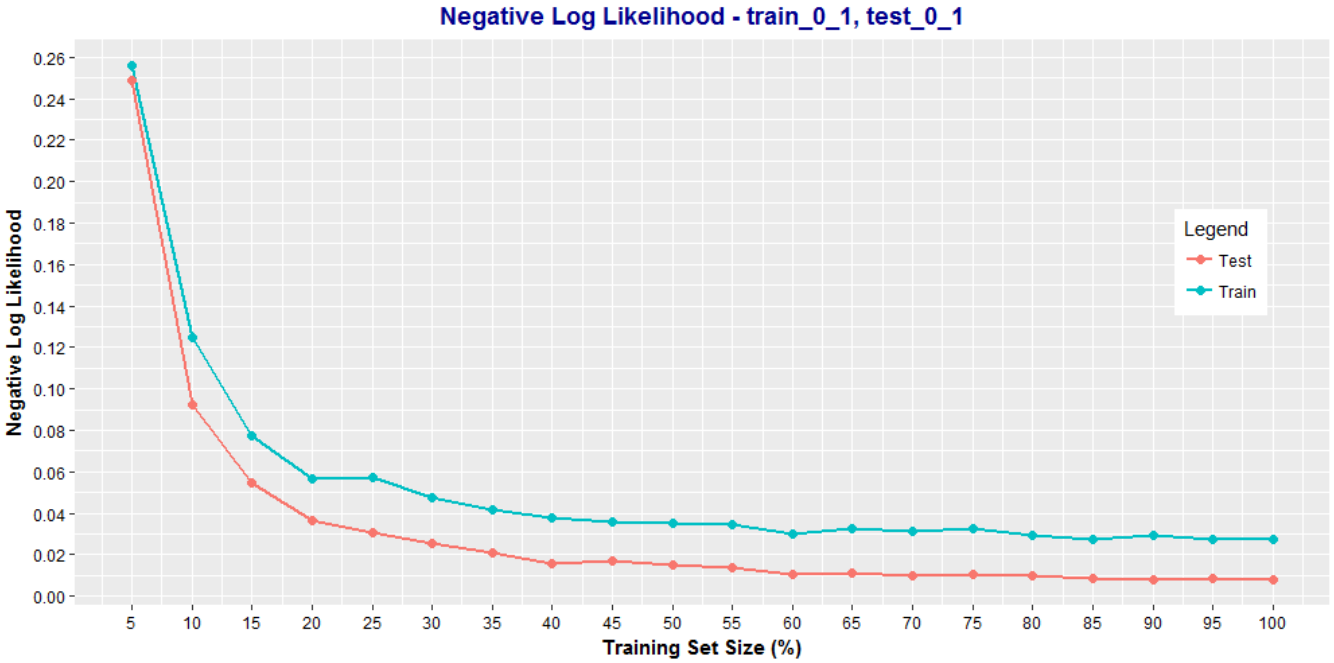


Figure 15: Negative Log Likelihood Curve : [0,1] Data Set

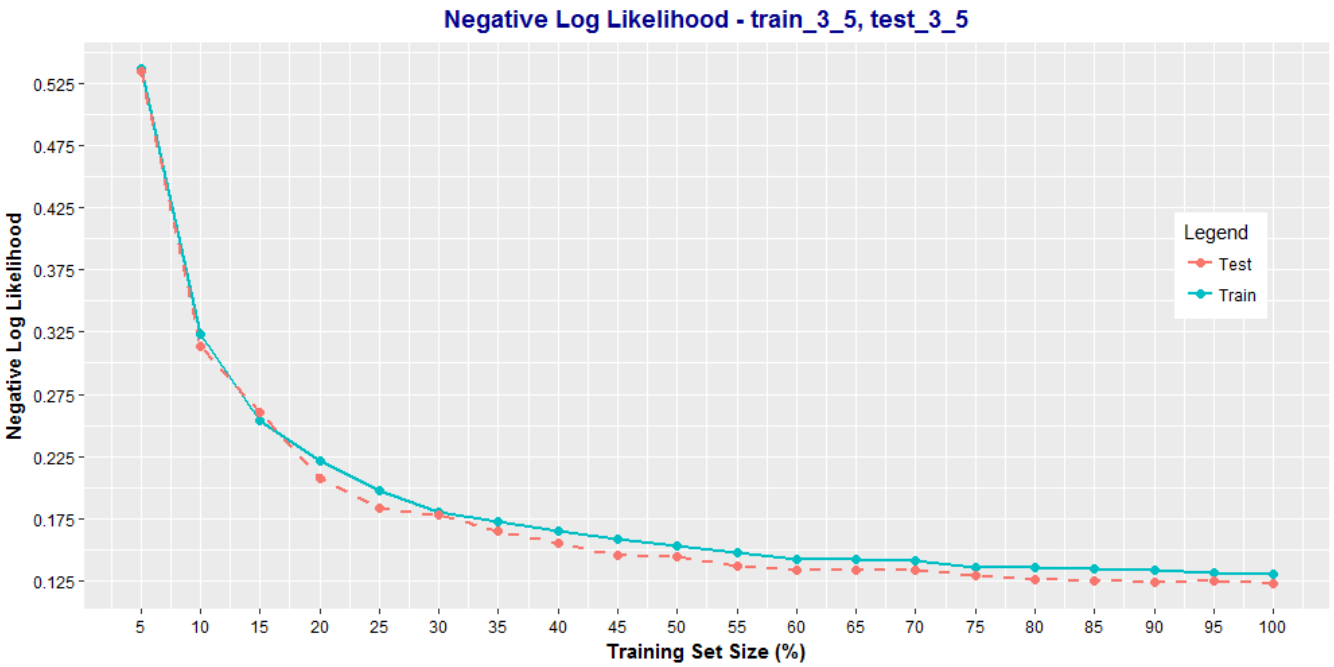


Figure 16: Negative Log Likelihood Curve : [3,5] Data Set

Configuration for the negative log likelihood plots is same as the learning curves and is based on the baseline in 3(b). As the main concept behind logistic regression is to minimize the negative log likelihood, this can be extended further to these

negative log likelihood curves and the fact that as the % training set size increases the negative log likelihood keeps on getting smaller and rightly so as negative log likelihood tries to determine the amount by which the predicted labels deviate from the true labels.

At first glance, negative log likelihood plots for both [0,1] and [3,5] seem like mirror images of the respective learning curves in section 5(a) which highlights the fact that as the model is improving more and more with increasing training set size, it is also getting better at reducing the deviation between predictions and actual values.

As log is a concave function (has to have a single global minimum), it seems that for smaller training set sizes (less than and up to 15%), the model does not find the required global optima, hence has more loss. This is evident in both the [0,1] and the [3,5] plots and can be related to the learning curves in section 5(a) where the corresponding accuracy values are also low.

Similar to the learning curves in 5(a), the negative log likelihood curves also goes flat for training set size greater than 50% with final values for the [0,1] set at around test = 0.008 and train = 0.027 while for the [3,5] set at around test = 0.123 and train = 0.131. The bigger gap between the test and train values for the [0,1] set and overall low cost compared to the [3,5] set highlights that 0 and 1 are easy to detect compared to 3 and 5.

NOTE : Another thing to notice is that test set curves for both [0,1] and [3,5] cases are lower than the train set curve, similar to the accuracy learning curve where test accuracy curves were higher than the training accuracy curves.

Additional Analysis [Visualizing the Images to detect outliers]

Being curious as to why test accuracy is better than the training accuracy, I took rowSums of all the samples for individual numbers and averaged them out. Then I visualized the averaged out numbers using conditional formatting in MS Excel and found out that the way training set was created, it had a lot of noise in it while test set was pretty clean. This is one of the points I mentioned in question 3(c). Possible solutions to this issue can be to re-evaluate data splitting method, add more data or possibly change the performance metrics other than accuracy.

Figure on Page 13 below shows the outliers for both the training and test sets and it is clearly evident that the training set data exhibits a lot of noise (highlighted by the outliers marked in red), hence the variation in accuracy.

Conclusion

This project was very interesting and helped me get good insight on logistic regression, parameter testing, etc. Special thanks to all of the fellow course mates on piazza for posting and answering diligently.

References

[1] **Deriving gradient of logistic regression**

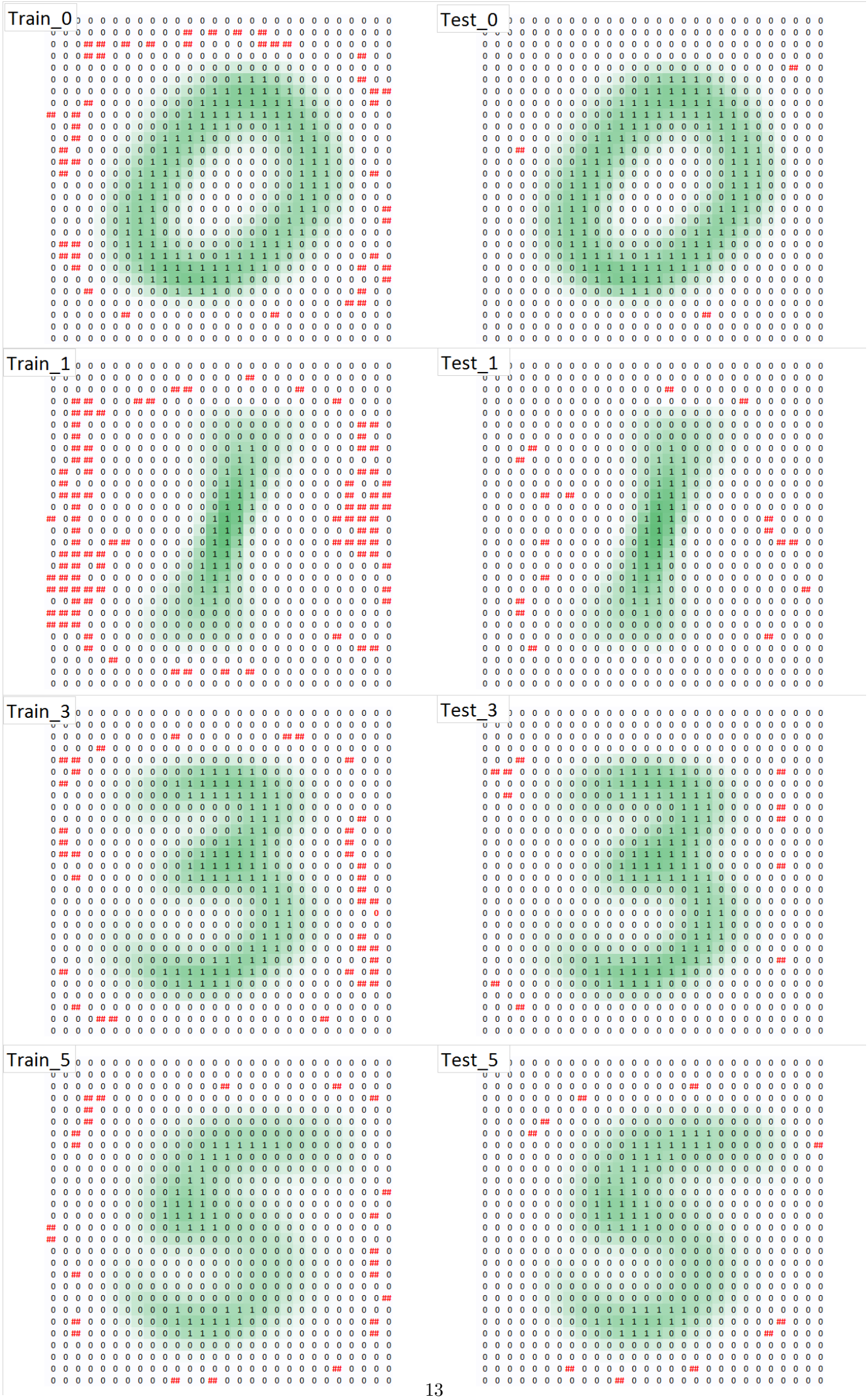
https://www.coursera.org/learn/ml-classification/lecture/QcofN/very-optional-deriving-gradient-of-logistic-regression-log-trick

[2] **Is it possible to have a higher train error than a test error in machine learning?**

https://www.quora.com/Is-it-possible-to-have-a-higher-train-error-than-a-test-error-in-machine-learning

[3] **3.3.1 Logistic Regression - Multiclass Classification (One vs all)**

https://www.youtube.com/watch?v=vNNcFTd_630&list=PL0Smm0jPm9WcCsYvbhPCdizqNKps69W4Z&index=38


Train_0
Test_0
Train_1
Test_1
Train_3
Test_3
Train_5
Test_5