ECE521

Inference Algorithms and Machine Learning

A2: Linear and Logistic Regression

Due date: March 16

Ze Yang 1000593930
Siyi Wu 1000430759
Wen Ying Zhang 1000404207

Contribution Table:

Name	Student Number	Contribution
Ze Yang	1000593930	33.3%
Siyi Wu	1000430759	33.3%
Wen Ying Zhang	1000404207	33.3%

Package Version:

The following code was developed and tested with python 3.6 and tensorflow 1.4.0.

Part 1: Linear Regression

Note:

Ypredict = $W^T X + b$

We initialize the weight matrix using ***tf.truncated_normal(shape=[784,1], stddev=0.5)*** and initialize b to **0**.

1. Tuning the learning rate

Learning Rate	MSE
0.005	**0.09645746**
0.001	0.2533125
0.0001	1.1037968

Table 1.1.1.1 MSE for different learning rates on test dataset after 20000 iterations (linear regression, SGD, λ = 0, B = 500)

```
Learning Rate 0.005: 0.09645746
Learning Rate 0.001: 0.2533125
Learning Rate 0.0001: 1.1037968
```

Figure 1.1.1.1 Console Output for **Table 1.1.1.1**

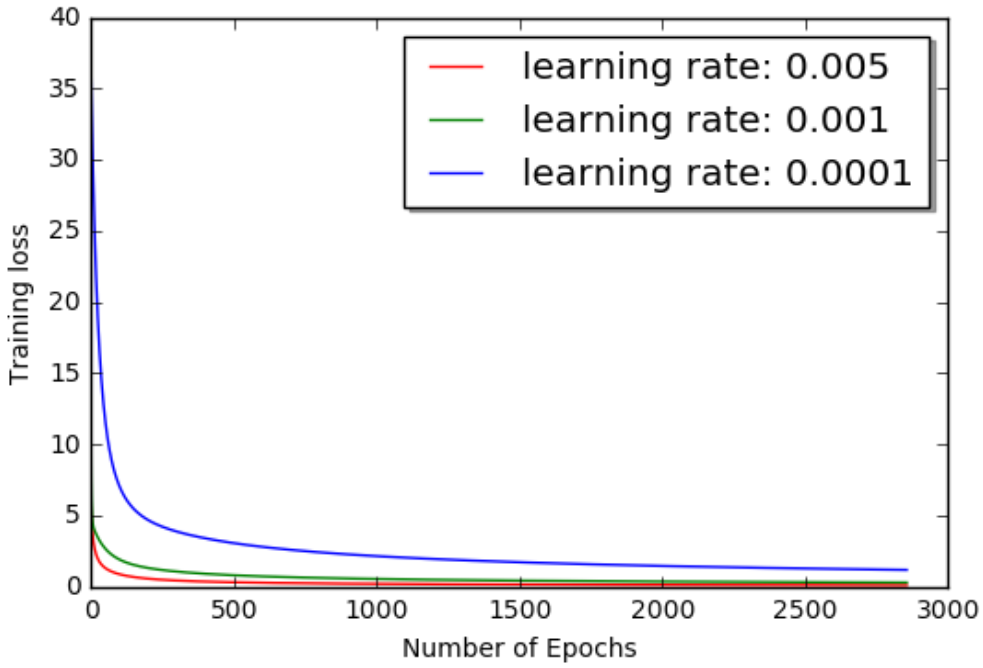

Figure 1.1.1.2 Training Loss vs Number of Epochs for Tuning Learning Rate (iterations = 20000, λ = 0, B = 500)

We found that the highest **learning rate of 0.005** has the fastest convergence and the lowest final MSE. With a higher learning rate, the step size of each iteration is larger in our SGD algorithm. As long as the step size is sufficiently small, we would prefer a larger step size which will converge faster to the desired result.

2. Effect of the mini-batch size

We've selected the best learning rate of **0.005** from part 1 to run our SGD algorithm, the following table displays our results:

Mini-batch size	**Final MSE**	**Training Time**
500	**0.0676885**	**23.632 s**
1500	0.0664233	73.587 s
3500	0.066439	251.967 s

Table 1.1.2 Table of Final MSE and Training Time for Different Mini-batch sizes (learning rate = 0.005, iterations = 20000, $\lambda = 0$)

Since the training data amount is 3500, and is not divisible by 1500. With the mini-batch size of 1500, we wrap around for the last round of iteration for each epoch, we use the last 500 data points, and then the first 1000 data points. and for the next immediate iteration, we pick the start index from where we left off from last iteration. We do this to ensure that we use all the data points in our training data equally without outweighing some data points over the others.

From **Table 1.1.2**, we found that the training time increases with the size of our mini-batch. However, the final MSE is similar for all mini-batch sizes. Therefore, for best time trade off, we should **choose a small mini-batch size**. If we look for low final MSE, the best results are also not directly related to a large or small mini-batch size.

3. Generalization

Note: In this section, we compare different initial value for W matrix and we found that when we set stddev=0.1, we get a better result compare to stddev=0.5.

Since the true output y only contains the value of 0 or 1, to determine accuracy, we check the predicted y output by seeing if it is greater than 0.5. If it greater, we give it a label of 1. If it is smaller, we give it a label of 0. We then compare the label with the true output to calculate accuracy. We use this method for all two-class notMNIST dataset calculations.

stddev=0.5

Weight Decay Coefficient	Valid Data Accuracy	Test Data Accuracy
0	89.0%	89.65517241379311%
0.001	92.0%	89.65517241379311%
0.1	**98.0%**	**97.24137931034483%**
1	97.0%	96.55172413793103%

Table 1.1.3.1 Table of Valid Data and Test Data Accuracy for Different Weight Decay (stddev=0.5**,** learning rate = 0.005, iterations = 20000, batch size = 500)

```
Weight Decay Coefficient 0: valid data accuracy: 89.0% test data accuracy: 89.65517241379311%
Weight Decay Coefficient 0.001: valid data accuracy: 92.0% test data accuracy: 89.65517241379311%
Weight Decay Coefficient 0.1: valid data accuracy: 98.0% test data accuracy: 97.24137931034483%
Weight Decay Coefficient 1: valid data accuracy: 97.0% test data accuracy: 96.55172413793103%
```

Figure 1.1.3.1 Console Output for **Table 1.1.3.1**

stddev=0.1

Weight Decay Coefficient	Valid Data Accuracy	Test Data Accuracy
0	98.0	96.55%
0.001	98.0%	97.24%
0.1	**98.0%**	**97.24%**
1	97.0%	96.55%

Table 1.1.3.2 Table of Valid Data and Test Data Accuracy for Different Weight Decay (stddev=0.1**,** learning rate = 0.005, iterations = 20000, batch size = 500)

```
Weight Decay Coefficient 0: valid data accuracy: 98.0% test data accuracy: 96.55172413793103%
Weight Decay Coefficient 0.001: valid data accuracy: 98.0% test data accuracy: 97.24137931034483%
Weight Decay Coefficient 0.1: valid data accuracy: 98.0% test data accuracy: 97.24137931034483%
Weight Decay Coefficient 1: valid data accuracy: 97.0% test data accuracy: 96.55172413793103%
```

Figure 1.1.3.2 Console Output for **Table 1.1.3.2**

From **Table 1.1.3.2**, we observed that when we set the standard deviation to 0.5, the best Weight Decay Coefficient is 0.1. When the standard deviation is 0.1, the valid data accuracies are higher, and the same value for weight decay coefficient = 0, 0.01 and 0.1. This is reasonable since the weight decay loss penalizes weight vectors with large magnitudes. In general, we conclude that the **best weight decay coefficient λ** is **0.1**. The corresponding **valid data accuracy** is **98.0%**, and the **test data accuracy** is **97.24%.**

We need to be careful choosing the reasonable weight decay coefficient λ. We do not want to choose a large λ that might penalize the magnitude of the weight vector too much and it dominates the loss function instead of our training dataset. Also, we do not want to pick a too small weight vector such that the penalization function is too negligible and does not have effect.

We need to look at valid data accuracy instead of training data accuracy because our weight vector and bias are **tailored to best fit the training data only**, it cannot be generalized to other data points lying outside the training dataset. However, our model will be used to all general data points, so it is necessary to decide based on valid test data which is not included in the train data.

4. Comparing SGD with normal equation

Note: We use stddev=0.5 for initializing W matrix, batch size=500,learn rate=0.005 for SGD.

TrainData Comparison

	Final MSE	TrainData Accuracy	Running time
SGD	0.0676885	89.0%	23.632s
Normal Equation	0.0094179	99.31%	0.479s

Table 1.1.4.1 Table of Final MSE, Train Data Accuracy, and Running Time for SGD and Normal Equation for Train Data (stddev=0.5**,** learning rate = 0.005, iterations = 20000, batch size = 500)

TestData Comparison

	Final MSE	TestData Accuracy	Running time
SGD	0.0977744	89.66%	23.632s
Normal Equation	0.0259504	95.86%	0.479s

Table 1.1.4.2 Table of Final MSE, Test Data Accuracy, and Running Time for SGD and Normal Equation for Test Data (stddev=0.5**,** learning rate = 0.005, iterations = 20000, batch size = 500)

From **Table 1.1.4.2**, we can see that **for both Test Data and Train Data, Normal equation yield a better result compared to SGD**, i.e.Train Data (99.31% vs 89.0%) and Test Data (95.86% vs 89.66%). Since Normal equation is guaranteed to get the optimal weight and bias using Training Data. Therefore the result should be better compare to SGD if Train Data and Test Data were shuffled properly. In addition to that, Normal equation (0.479s) has a shorter running time compared to SGD (23.632s). But running time will increase significantly as the dimension of input data increase.

When SGD is more practical than normal equation?

Although normal equation has a shorter running time compare to SGD. However, in the practical case, we may encounter a large dataset such as N=1000000. At that time, normal equation will require much more computation time and computation memory. Such as calculating inverse matrix and matrix multiplication compare to SGD. For SGD, you usually choose a small batch size which means less memory and computation time were required. Furthermore, SGD could take the advantage of parallelization using Multiprocessor or distributed computing systems.

Part 2: Logistic Regression

2.1 Binary cross-entropy loss

1. Learning

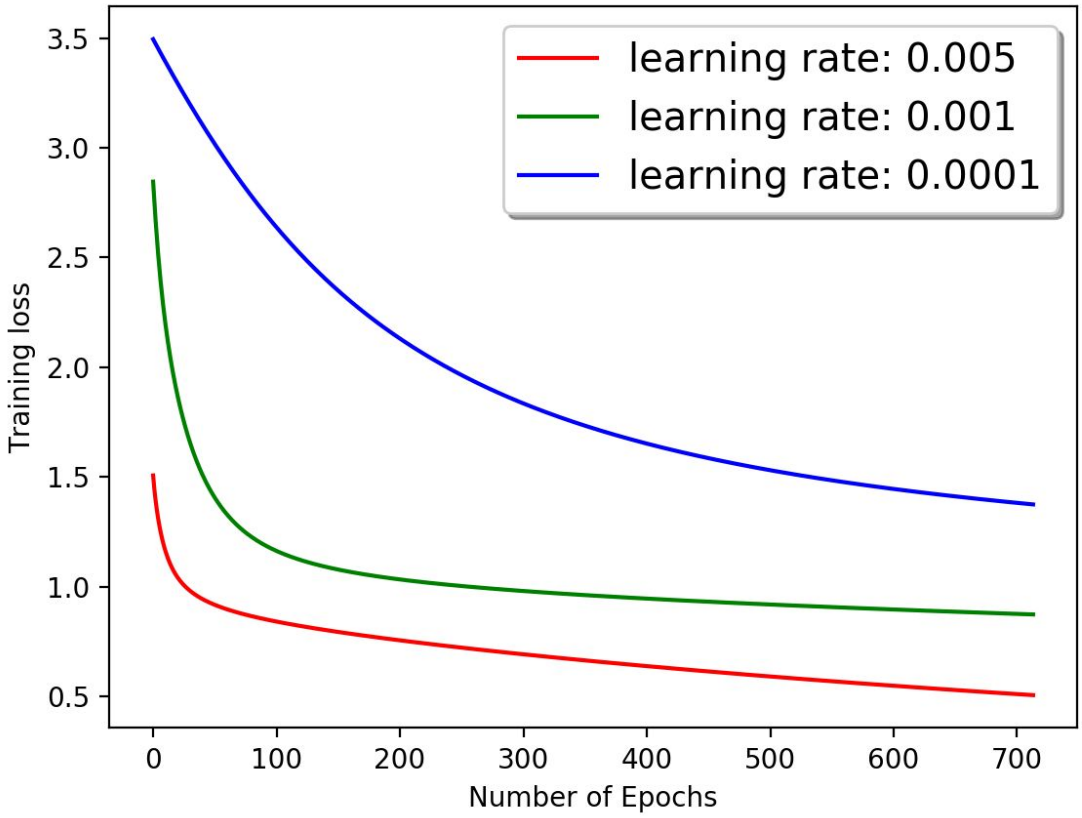

Figure 2.1.1.1 Training Loss vs Number of Epochs for Tuning Learning Rate (iterations = 5000, λ = 0.01, B = 500, with logistic regression)

To tune the learning rate, we tried a learning rate of 0.005, 0.001 and 0.0001 with logistic regression of training and the given configurations of iterations = 5000, λ = 0.01, B = 500. From **Figure 2.1.1.1** we found that **a learning rate of 0.005 converges to the lowest training loss fastest**. So we will use a learning rate of 0.005 for finding best training and validation curves for both cross-entropy loss and classification accuracy vs. the number of epochs.

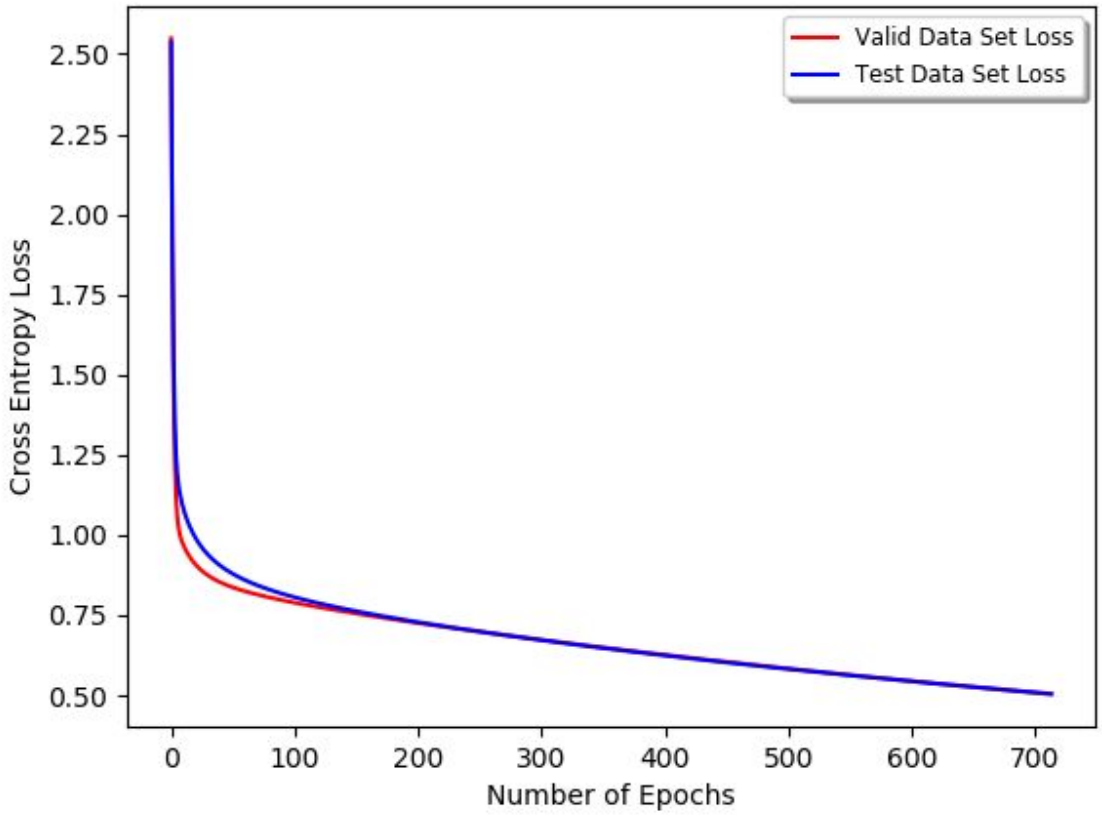

Figure 2.1.1.2 Cross Entropy Loss vs Number of Epochs for Valid and Test Data set (iterations = 5000, λ = 0.01, B = 500, learning rate = 0.005 with logistic regression)

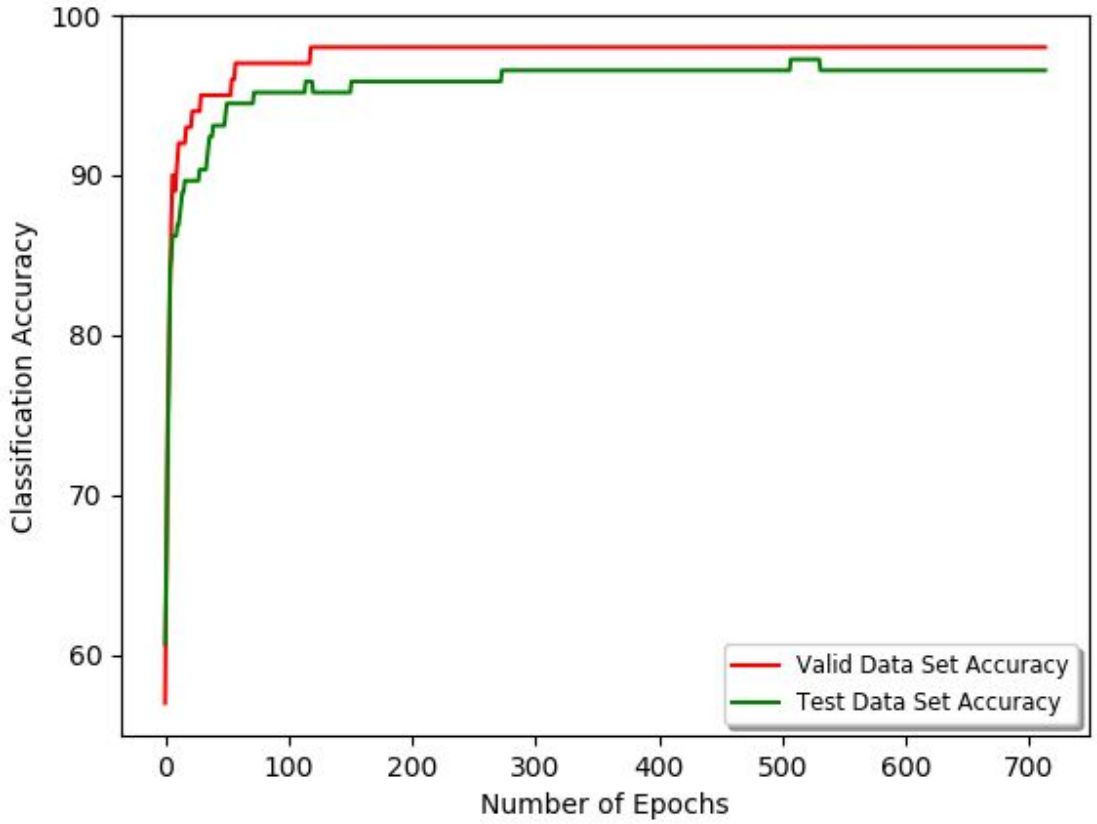

Figure 2.1.1.3 Classification Accuracy vs Number of Epochs for Valid and Test Data set (iterations = 5000, λ = 0.01, B = 500, learning rate = 0.005 with logistic regression)

```
Best test accuracy: 97.9310344828
Best valid accuracy: 98.0
```

Figure 2.1.1.4 Console Output For Best Test and Valid Accuracy

From **Figure 2.1.1.3** and our terminal output, the **best test classification accuracy is: 97.9310344828%**

2. Beyond plain SGD

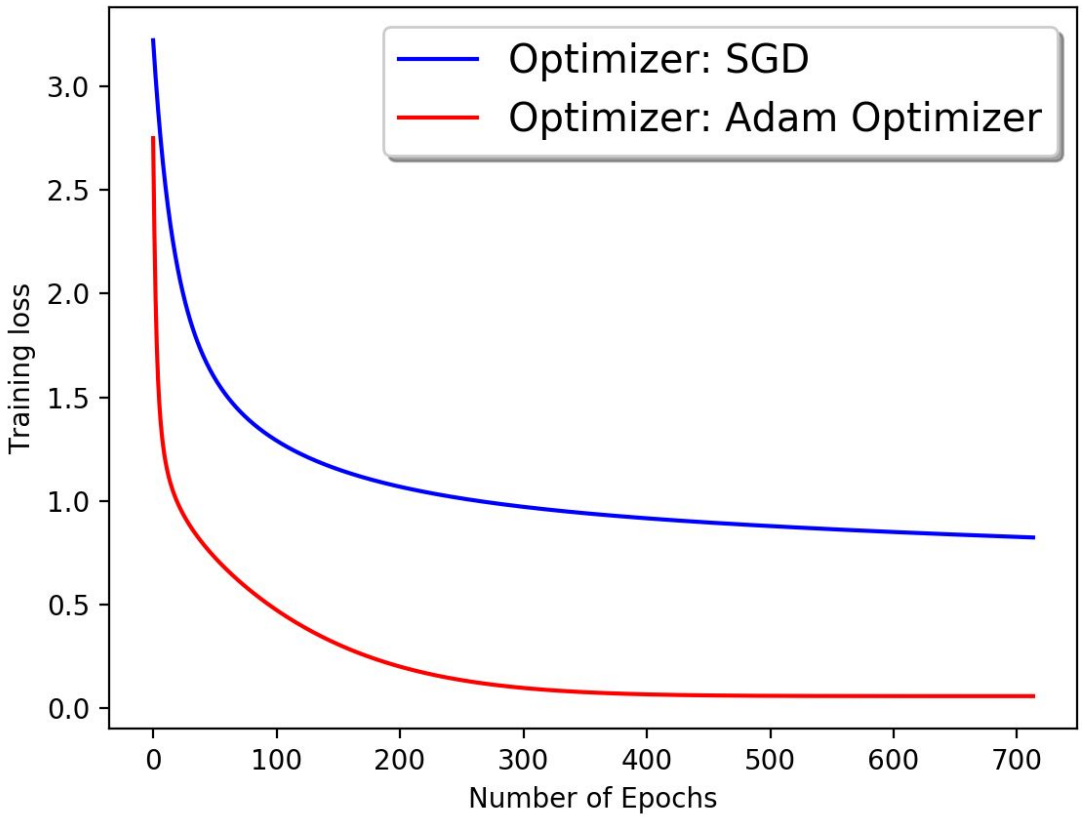

Figure 2.1.2 Training Loss vs Number of Epochs for SGD and Adam Optimizer (learning rate = 0.001, iterations = 5000, λ = 0.01, B = 500 with logistic regression)

From **Figure 2.1.2** we found that training the logistic regression model with the **Adam optimizer results in a lower cross entropy loss** than training with SGD optimizer.

From research we found that Adam optimizer uses moving averages of parameters. The algorithm will be able to converge to a large effective step size without fine tuning. However, more computation will be needed in each training step. More state will also be needed to store average and variance for each parameter. When we want lower training loss with a fixed learning rate, Adam optimizer is the better choice.

3. Comparison with linear regression

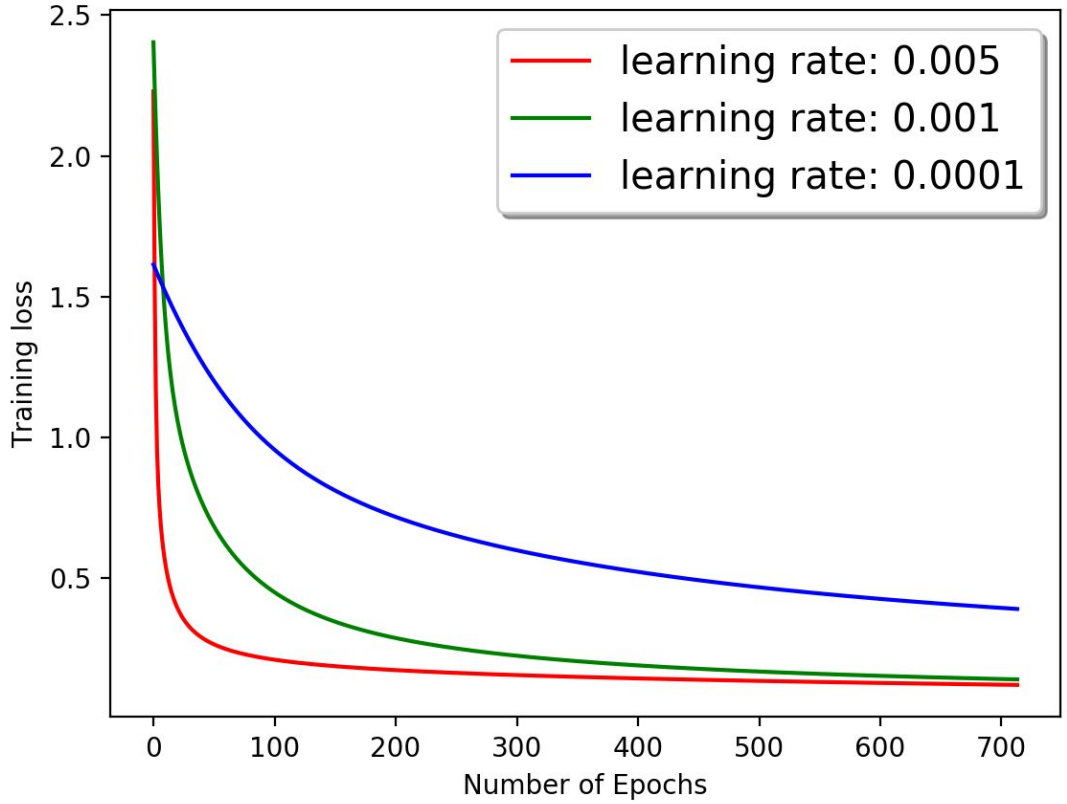

Figure 2.1.3.1 Training Loss vs Number of Epochs for Tuning Learning Rate (iterations = 5000, λ = 0, B = 500, with logistic regression)

	Optimal Logistic Regression without Weight Decay	Linear Regression Using Normal Equation
Train Data Accuracy	96.51428571428572%	99.31428571428572%
Valid Data Accuracy	98.0%	95.86206896551724%
Test Data Accuracy	98.6206896551724%	97.0%

Table 2.1.3.1 Accuracy for Varying Data for Logistic Regression Without Weight Decay vs Linear Regression Using Normal Equation
(learning rate = 0.05, iterations = 5000, λ = 0, B = 500)

```
logistic_train_accuracy 96.51428571428572
logistic_valid_accuracy 98.0
logistic_test_accuracy 98.62068965517241
Y_norm_train_accuracy 99.31428571428572
Y_norm_Test_accuracy 95.86206896551724
Y_norm_Valid_accuracy 97.0
```

Figure 2.1.3.2 Console Output for **Table 2.1.3.1**

From **Table 2.1.3.1**, we conclude that **both the linear regression and logistic regression achieve quite satisfying accuracies**, and linear regression through normal equation can achieve a better accuracy for the training data set while for valid and test data set, logistic regression would have a better accuracy. We think the observation is reasonable since the optimal weight vector found through normal equation is guaranteed to be optimal for the training data set only. It does not guarantee the performance over valid data and test data.

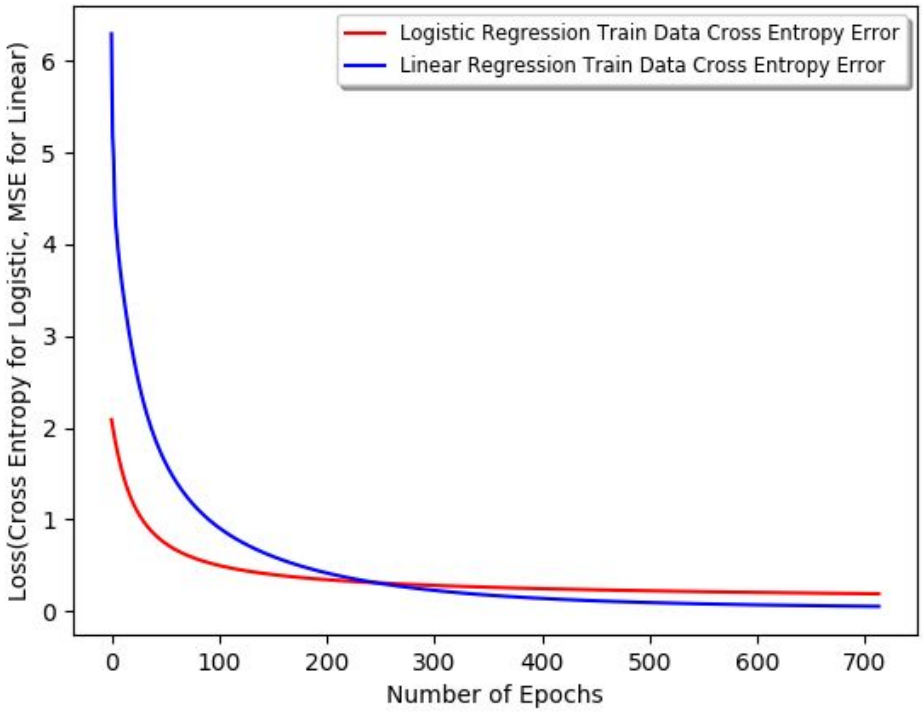

Figure 2.1.3.3 Loss vs Number of Epochs for Logistic Regression (Cross Entropy Loss) and Linear Regression (MSE) over Training Data Set
(learning rate = 0.001, iterations = 50 00, λ = 0, B = 500)

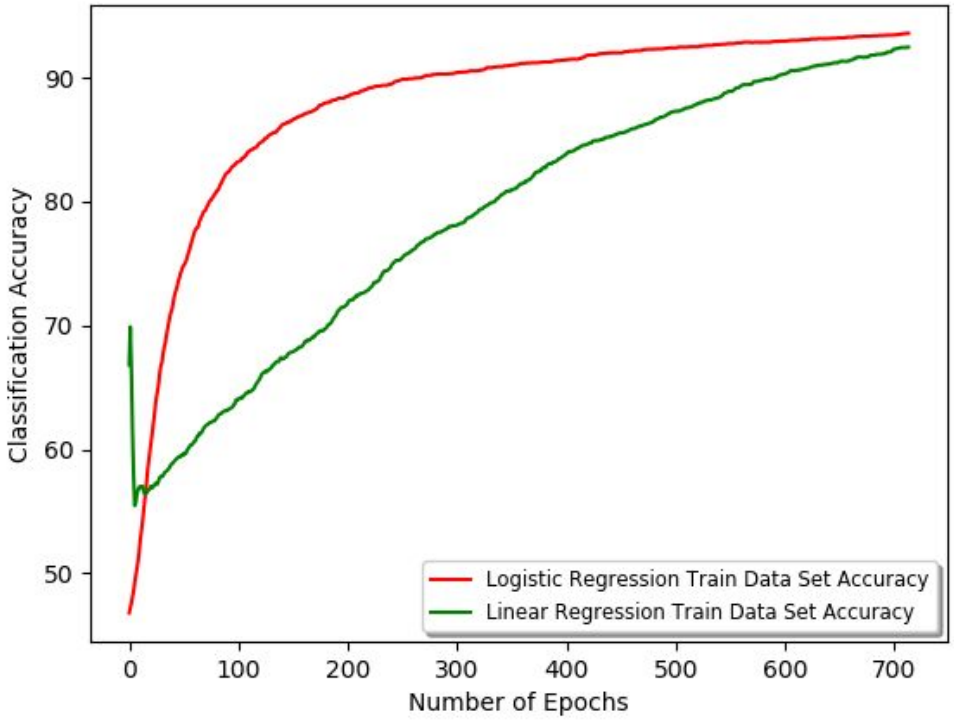

Figure 2.1.3.4 Classification Accuracy vs Number of Epochs for Logistic Regression and Linear Regression over Training Data Set
(learning rate = 0.001, iterations = 5000, λ = 0, B = 500)

From **Figure 2.1.3.3** and **Figure 2.1.3.4**, we can conclude that the **cross entropy loss helps the regression converges much faster** than the mean square error loss.

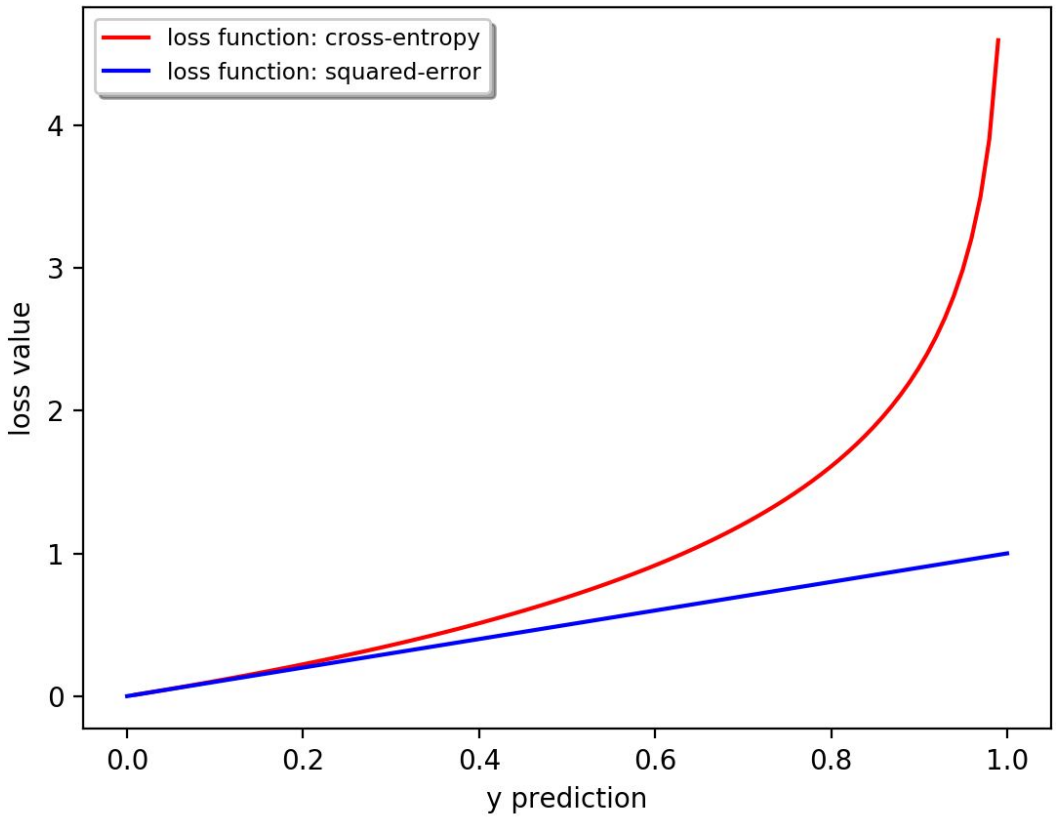

Figure 2.1.3.5 cross-entropy loss vs squared-error loss as a function of y prediction within the interval [0, 1] and a dummy target y = 0 in 1-D
(using 100 data points from 0 to 1 for y prediction)

From **Figure 2.1.3.5** we can see that cross entropy error increases much more significantly with a large difference between y prediction and y target. This verifies our observation from last part that cross entropy loss helps the regression to converge much faster.

In **Figure 2.1.3.5**, when y prediction is 1.0, the loss value should be a really large number since all of our true labels are 0. The loss of linear regression with squared error is a very small number less than 1, which means it still needs a lot more iterations to get to the actual loss value (i.e. a really large number). However, for logistic regression with cross entropy loss, the loss value obtained is much closer to the actual loss.

2.2 Multi-class classification

Note about Accuracy Calculation for part 2.2 (permuted version of notMNIST DataSet and FaceScrub Dataset). We use tf.one_hot to encode the Real target, and compare with our training target using matrix manipulation and comparison.

notMNIST Dataset

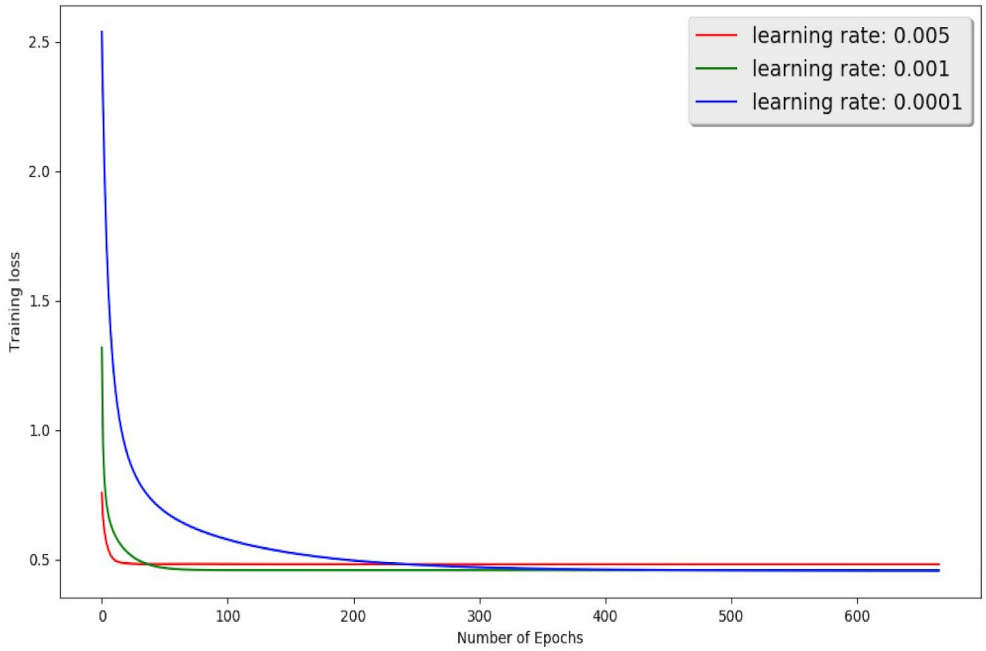

Figure 2.2.1.1 Cross-Entropy loss for different learning rate with fixed weight-decay coefficient : 0.01 for Multi-class Logistic Regression.

To tune the learning rate, we tried a learning rate of 0.005, 0.001 and 0.0001 with Multiclass logistic regression of training and the given configurations of iterations = 20000, λ = 0.01, B = 500. Based on the result of **Figure 2.2.1.1**, **the best learning rate is 0.001**. Although learning rate 0.005 has the fastest converge time, the final loss is higher compare to others. Learning rate 0.001 and 0.0001 have almost the same loss, but 0.001 converge faster than 0.0001. Therefore we choose learning rate **0.001** to plot the training and validation curves.

Learning Rate	Cross-Entropy-Loss
0.005	0.433416
0.001	**0.427732**
0.0001	0.42451

Table 2.2.1.1 Cross Entropy Loss for Different Learning Rates
(iterations = 20000, λ = 0.01, B = 500)

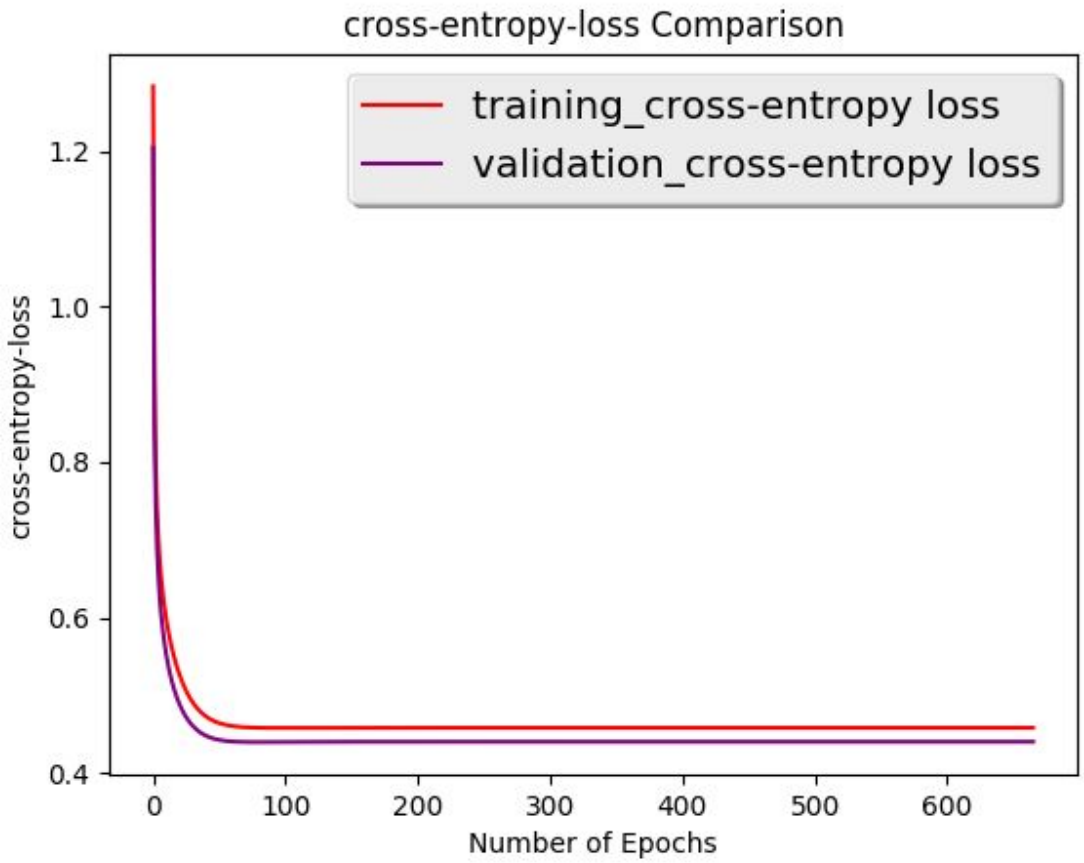

Figure 2.2.1.2 Training Data Cross-Entropy Loss vs Validation Data Cross Entropy Loss
(learning rate = 0.001, batch size = 500, iteration = 20000, λ = 0.01)

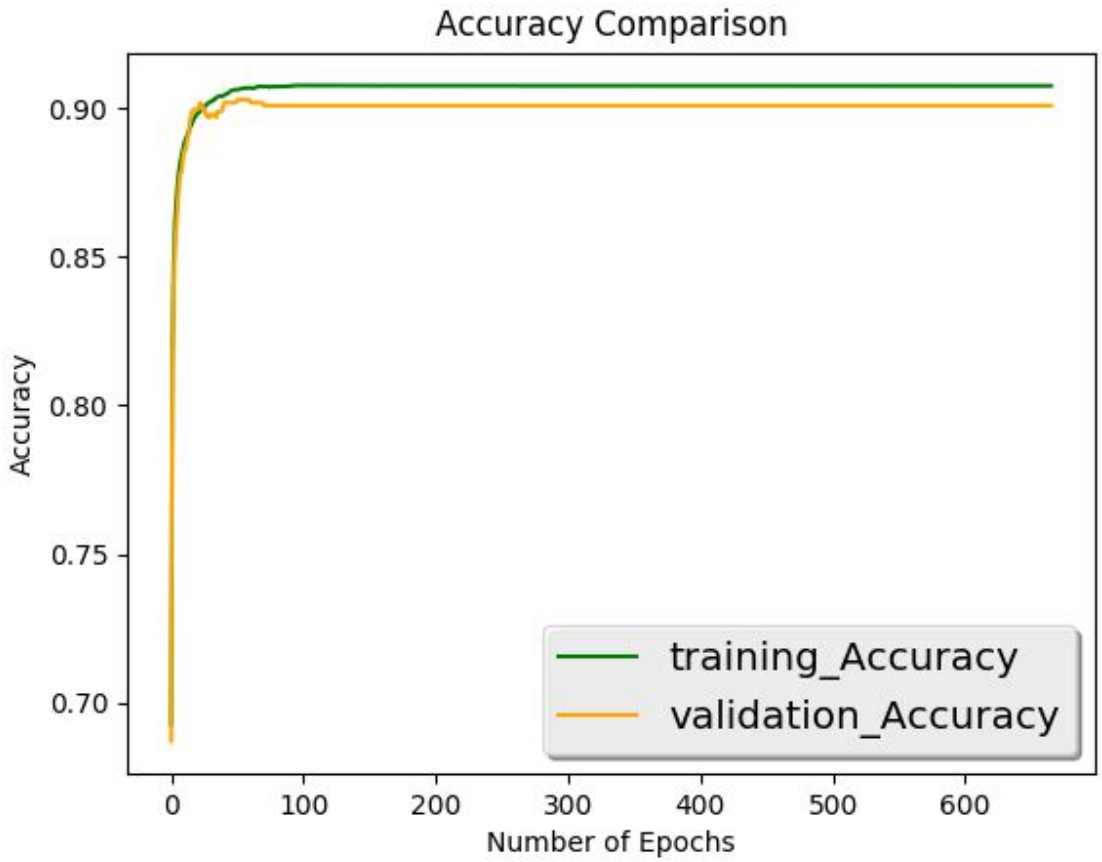

Figure 2.2.1.3 Training Data Accuracy vs Validation Data Accuracy (learning rate = 0.001, batch size = 500, iteration = 20000, λ = 0.01)

Learning Rate	Multi-Test accuracy	Binary-Test accuracy
0.005	88.91%	96.44133123214348%
0.001	**89.39%**	96.24137931034483%
0.0001	89.46%	97.54133222313183%

Table 2.2.1.2 Comparison of Test Data accuracy using different Learning Rate.

The Best test accuracy is 89.46% obtained using learning rate 0.001. But as we discuss above, learning rate 0.001 has a faster converge time, and it has 89.39% accuracy which is similar to learning rate 0.0001.

Comparison between Binary Logistic Regression and Multi-Class Logistic Regression:

Based on the result from **Table 2.2.1.2**, the performance of multi-classification is worse than binary-classification.The reason is that for multi-classification, you have more output nodes (10 for this case), but the complexity of algorithm remain the same. Therefore, the result of multi-classification is worse than binary case. In order to improve the accuracy for multi-case, you may want to increase the size of your database or using more complicate model such as Neural network.

FaceScrub Dataset

Learning Rate	**Weight Decay Coefficient**	**Valid Data Accuracy**
0.005	0	70.6522%
0.005	0.001	72.8261%
0.005	0.1	70.6522%
0.005	1	53.2609%
0.001	0	72.8261%
0.001	0.001	72.8261%
0.001	0.1	73.913%
0.001	1	55.4348%
0.0001	0	69.5652%
0.0001	**0.001**	**75.0%**
0.0001	0.1	71.7391%
0.0001	1	55.4348%

Table 2.2.2.1 Table of Valid Data Accuracy for Varying Learning Rate and Weight Decay
(iterations = 10000, B = 300)

```
accuracy for learn rate 0.005 weight decay 0.0:   0.706522
accuracy for learn rate 0.005 weight decay 0.001:   0.728261
accuracy for learn rate 0.005 weight decay 0.1:   0.706522
accuracy for learn rate 0.005 weight decay 1:   0.532609
accuracy for learn rate 0.001 weight decay 0.0:   0.728261
accuracy for learn rate 0.001 weight decay 0.001:   0.728261
accuracy for learn rate 0.001 weight decay 0.1:   0.73913
accuracy for learn rate 0.001 weight decay 1:   0.554348
accuracy for learn rate 0.0001 weight decay 0.0:   0.695652
accuracy for learn rate 0.0001 weight decay 0.001:   0.75
accuracy for learn rate 0.0001 weight decay 0.1:   0.717391
accuracy for learn rate 0.0001 weight decay 1:   0.554348
```

Figure 2.2.2.1 Console Output for **Table 2.2.2.1**

From **Table 2.2.2.1,** we found the best performance for valid data accuracy is with **learning rate 0.0001 and weight decay coefficient 0.001.**

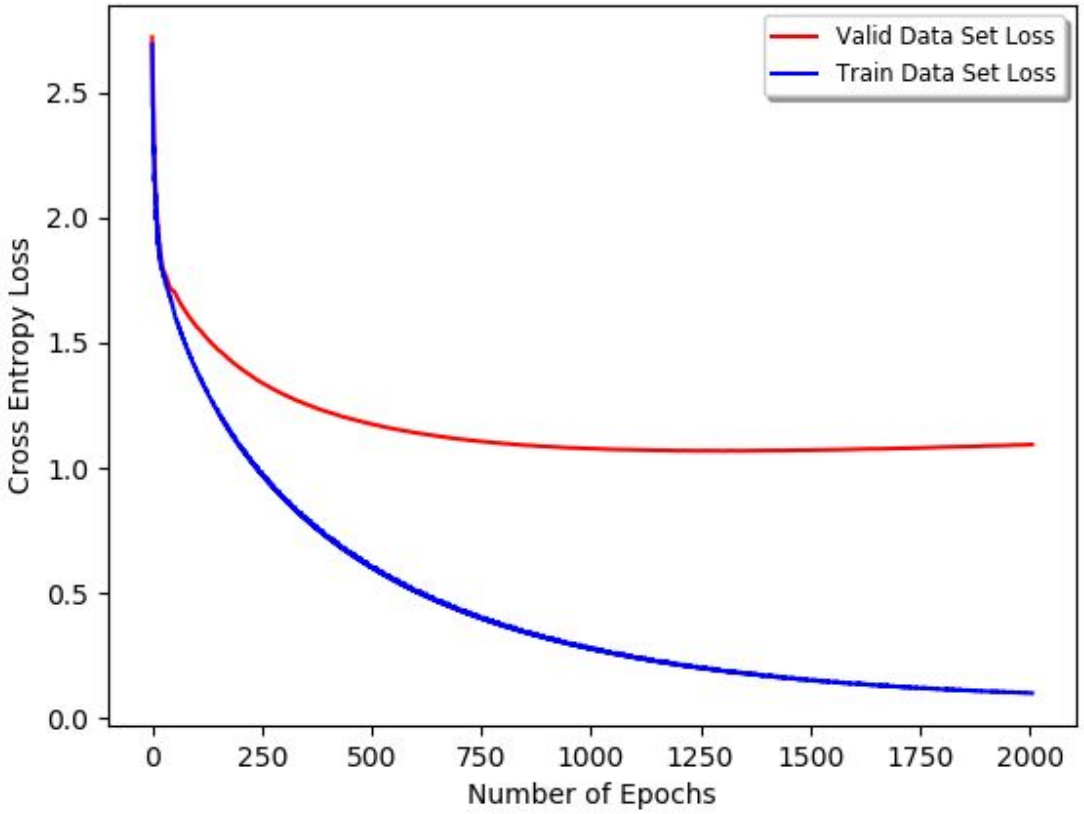

Figure 2.2.2.2 Cross Entropy Loss vs Number of Epochs over Valid and Training Data (learning rate = 0.0001, batch size = 300, iteration = 5000, λ = 0.001)

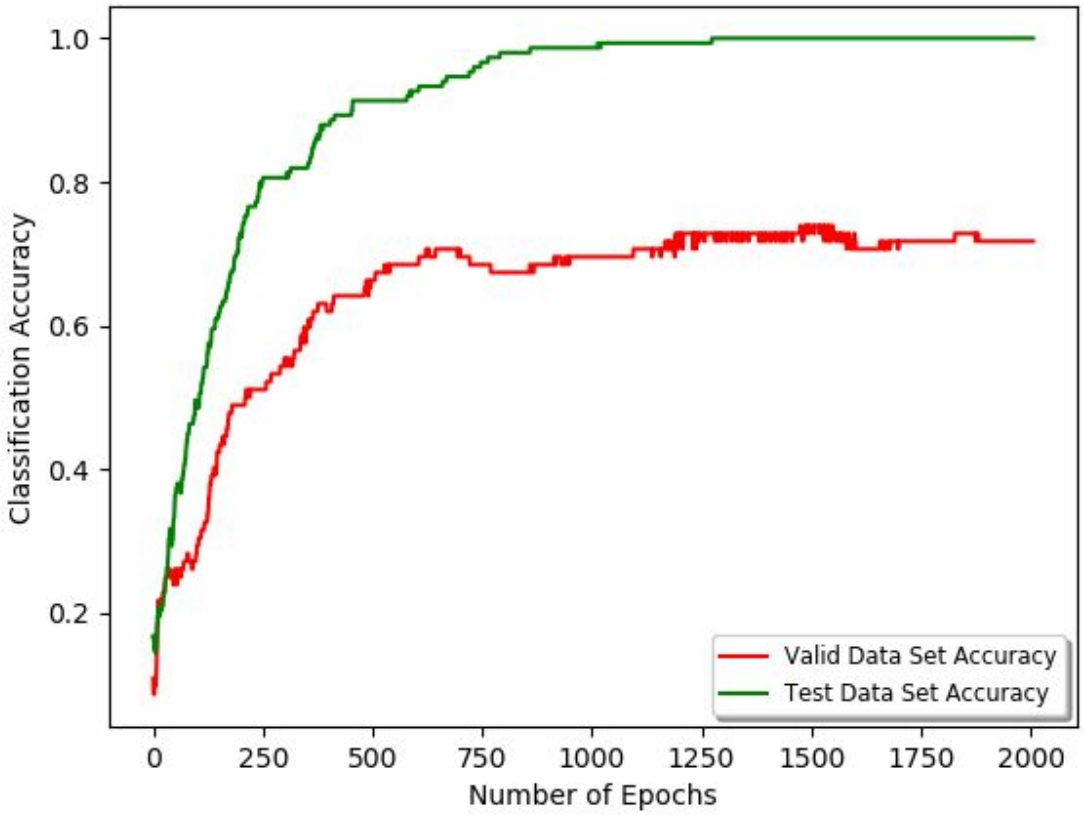

Figure 2.2.2.3 Classification vs Number of Epochs over Valid and Training Data (learning rate = 0.0001, batch size = 300, iteration = 5000, λ = 0.001)

k	Accuracy for the validation set
1	66.30434782608695 %
5	56.52173913043478 %
10	55.434782608695656 %
25	56.52173913043478 %
50	53.2608695652174 %
100	46.73913043478261 %
200	31.521739130434785 %

Table 2.2.2.2 Accuracy of k-NN for Validation Data Using Name ID Targets (from A1)

```
accuracy for test data is : 0.784946
```

Figure 2.2.2.4 Console Output for Logistic Regression Accuracy over Test Data

Comparing our logistic regression accuracy with the k-NN accuracy for validation data, we found that logistic regression achieved **a better performance of 78.4946%** compared to 66.30434782608695%.