CSE 574 – Machine Learning

Assignment 3

Group 28

Haripriya Iyer

Priya Karadi

Problem 1: Implementation of Logistic Regression

Binary Logistic Regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. It transforms its output using the sigmoid function and returns a probability value which is mapped to two or more distinct classes.

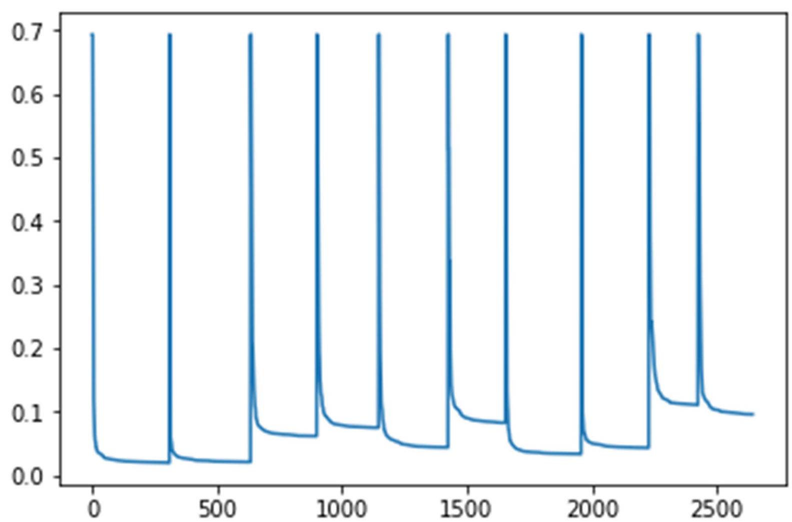
Observations:

Objective function Errors:

Plot of the errors:

Here we can see that the objective function is calculating error for each of the 10 classes, and each of them is minimized to a small value.

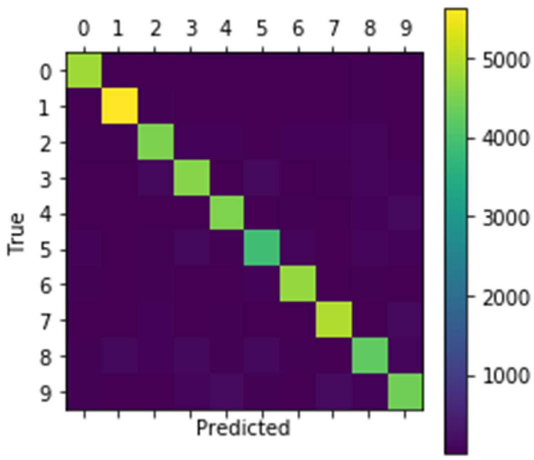
Thus for every class (Here we can see 10 shoots and dips each for 0-9 categories of the MNIST data) we can see the error is decreased and we get the optimum value from the objective function.



Error in prediction for the Training data set –

Confusion Matrix -

```
[[4820    1    9    7    8   19   21    7   29    2]
 [   1 5626   28   10    3   18    3   11   34    8]
 [  32   38 4520   63   50   17   49   65  110   14]
 [  18   19  124 4602    6  145   18   44  104   51]
 [  10   19   22    8 4543   10   25   12   49  144]
 [  46   21   32  126   39 3900   81   21  109   46]
 [  24   14   30    3   22   61 4737    2   23    2]
 [  10   22   51   12   42   10    3 4960   13  142]
 [  39  111   54  119   28  118   35   19 4245   83]
 [  24   22   13   82  157   35    1  156   45 4414]]
```



Here you can see that for individual categories or classes the correctly predicted values are along the diagonal. And the values not on the diagonal represent the wrongly classified value for the particular class.

Error for each class

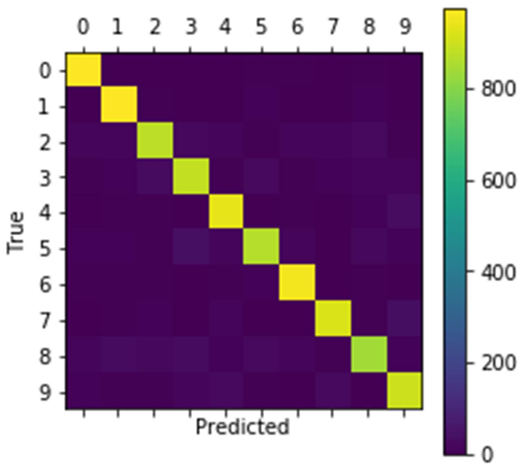
Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
4.060%	4.5%	7.4%	8.5%	7.2%	9.9%	4.7%	6.3%	10.8%	10%

The total training data error: 50000 - 46367 = 3633

Training set Accuracy: 92.73400000000001%

Error in prediction for the Validation data set –

```
[[977   0   1   2   0   6   5   1   6   2]
 [  0 974   4   1   1   9   0   1   8   2]
 [ 12  14 878  22  13   5  12  13  25   6]
 [  4   9  28 889   4  25   4  11  13  13]
 [  1   4   7   2 939   0   7   0   9  31]
 [  9   9   7  41  18 868  17   2  21   8]
 [  7   4   6   0   5  11 958   2   7   0]
 [  3   5  10   1  15   2   0 923   4  37]
 [ 16  28  20  28   9  25  19   4 842   9]
 [ 10   4   5  19  23   5   1  26   3 904]]
```



Here again, you can see that for individual categories or classes the correctly predicted values are along the diagonal. And the values not on the diagonal represent the wrongly classified value for the particular class.

Error for each class

Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
5.9%	6%	9.1%	11.5%	8.5%	9.2%	6.3%	6.1%	10.23%	10.67%

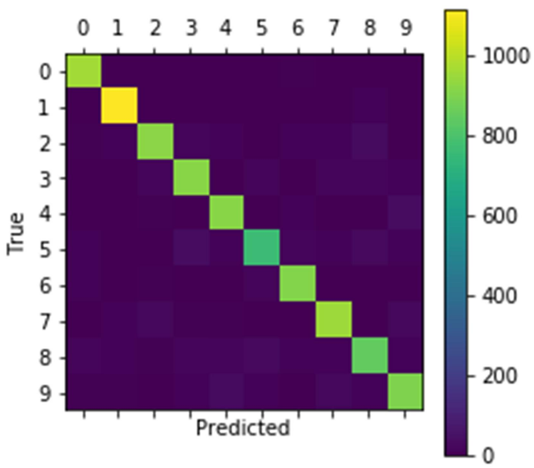
The total Validation data error: 10000 - 9152 = 848

Validation set Accuracy: 91.52%

Error in prediction for the Testing data set –

Confusion Matrix –

```
[[ 961    0    1    2    1    4    5    4    1    1]
 [   0 1115    3    1    0    1    4    1   10    0]
 [   8    9  920   19   11    4   12   13   32    4]
 [   4    1   20  919    2   20    4   14   17    9]
 [   1    2    6    3  917    0    9    3    4   37]
 [  10    3    1   38   11  764   17   10   29    9]
 [   9    4    7    2    4   20  908    1    3    0]
 [   2    9   22    5    8    2    1  951    2   26]
 [  14   13    7   20   14   28    9   10  847   12]
 [   8    8    1   12   34   12    1   23   12  898]]
```



Again for individual categories or classes the correctly predicted values are along the diagonal. And the values not on the diagonal represent the wrongly classified value for the particular class.

Error for each class

Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
5.5 %	4.2%	6.8%	9.9%	8.4%	10.6 %	6.3%	7.6%	11.4%	9.8%

The total Test data error: 10000 - 9200 = 800

Testing set Accuracy: 92.0%

Conclusion:

The training accuracy is 92.734%

The testing accuracy is 92%.

Upon training the data in binary logistic objective function, we get the optimum hyper parameter to use. In this case we are trying to find out the optimum hyper parameter that is the penalty term lambda when trying to optimize algorithm using gradient descent.

After finding out the optimum value of lambda on our model we will get a stable prediction on the test dataset.

The total training and test errors are nearly equal. But there is lot of error variation in the individual classes of the dataset. One possible reason could be that variance is not properly explained in the training set i.e. the data is too simple, causing underfitting.

Problem 2: Support Vector Machine

A Support Vector Machine is a supervised machine learning algorithm which can be used for both classification and regression problems. It uses the kernel trick to project the data into higher dimensions to identify the decision boundaries while performing calculations in a lower dimension. SVM tries to find the best hyperplane that separates the different classes of datapoints. It does so by maximizing the distance between the sample points and the hyperplane.

SVM Hyperparameters:

Kernel: It provides either a ‘linear’ or ‘non-linear’ separation hyperplane. It can be ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’, or a callable. The default value is ‘rbf’.

Gamma: This parameter is the kernel coefficient for non-linear hyperplanes. It defines the influence of a single training example. A higher gamma value indicates that the model tries to fit the training data exactly. This can lead to overfitting. The default value is ‘auto’ i.e. 1/num_features.

C (cost): This is a penalty or regularization parameter of the error term. It therefore controls the trade-off between smooth decision boundaries (intuitively, maximizing decision margin) and correct classification of training data. A lower value of C encourages a larger margin, therefore simple decision function, at the cost of training accuracy. A higher value of C may lead to overfitting. The default value is 1.

a) Using linear kernel (all other parameters are kept default).

Training set Accuracy : 0.9283

Validation Set Accuracy: 0.9128

Test Set Accuracy : 0.9167

b) Using radial basis function with value of gamma setting to 1 (all other parameters are kept default).

Training set Accuracy : 0.34054

Validation Set Accuracy: 0.1542

Test Set Accuracy : 0.1725

c) Using radial basis function with value of gamma setting to default (all other parameters are kept default).

Training set Accuracy : 0.9202

Validation Set Accuracy: 0.9204

Test Set Accuracy : 0.924

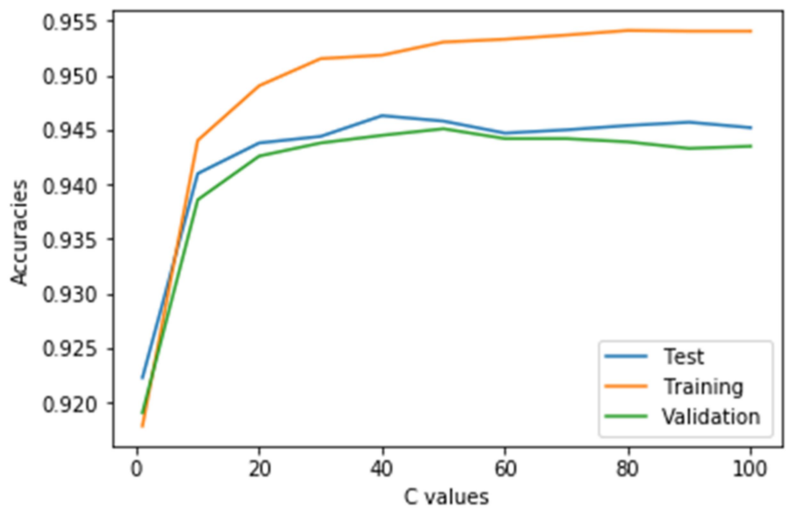
Performance Comparison:

SVM	**Training Set**	**Validation Set**	**Test Set**
Linear	0.93022	0.9135	0.9204
RBF gamma = 1	0.34054	0.1542	0.1725
RBF gamma=default	0.9202	0.9204	0.9240

- SVM with linear kernel performs marginally better on training data than the rbf kernel (with default gamma). Performance on the validation and test set is roughly the same.
- RBF kernel with gamma = 1 indicates that we want exact classification of the data. This leads to overfitting during training. Overall, it has a very poor accuracy in this experiment.

d) Using radial basis function with value of gamma setting to default and varying value of C

Cost Value	**Training Set**	**Validation Set**	**Test Set**
1	0.9178	0.9191	0.9223
10	0.94404	0.9386	0.9419
30	0.95152	0.9438	0.9444
40	0.95184	0.9445	0.9463
50	0.95304	0.9451	0.9458
60	0.95330	0.9442	0.9447
70	0.95358	0.9442	0.9450
80	0.95410	0.9439	0.9454
90	0.95404	0.9433	0.9457
100	0.95404	0.9435	0.9452



Plot of C values vs Accuracies

The value of C from the table and the graph shown is around 40, for which the accuracies are the best.

Applying the SVM on complete training dataset, using the best parameters –

- C = 40
- Kernel = ‘rbf’
- Gamma = default

The results of fitting SVM model using the optimum parameters are –

Training Set Accuracy : 0.98706

Validation Set Accuracy: 0.9723

Test Set Accuracy : 0.9719

Conclusion: The result from incorporating the optimal parameters found gives us the best possible accuracy in the SVM model.

Problem 3: Multi class Logistic Regression

Multi class classification is implemented by training multiple logistic regression classifiers, one for each of the K classes/categories in the training dataset.

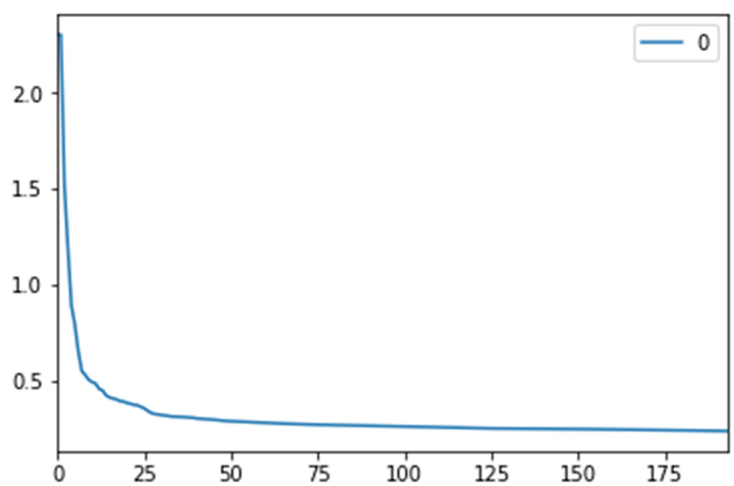
The Logistic Regression procedure produces all predictions at the individual case level, regardless of how the data are entered and whether or not the number of covariate patterns is smaller than the total number of cases, while the Multinomial Logistic Regression procedure internally aggregates cases to form subpopulations with identical covariate patterns for the predictors, producing predictions.

Below are confusion matrices for training validation and testing datasets for the MNIST dataset multiclass logistic regression, the correctly predicted labels are along the diagonals and the incorrectly predicted labels are elsewhere, with each column representing each digit.

Observations:

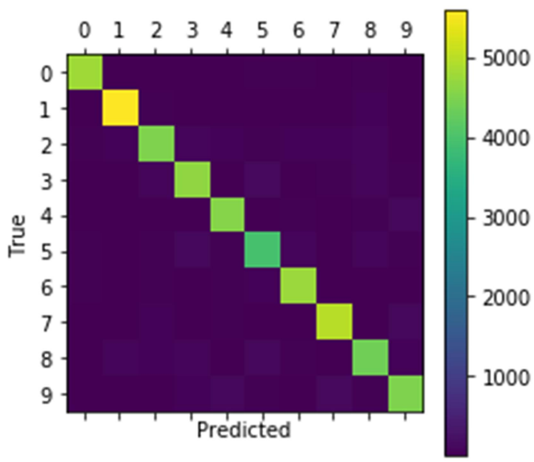
After 193 iterations, the least error value we get from the objective function in case of multinomial logistic regression: 0.237878

Below is a plot which shows the error is which is decreasing after each iteration finally converging at the 190th iterations



Training Dataset Error : 50000 – 46751 = 3249

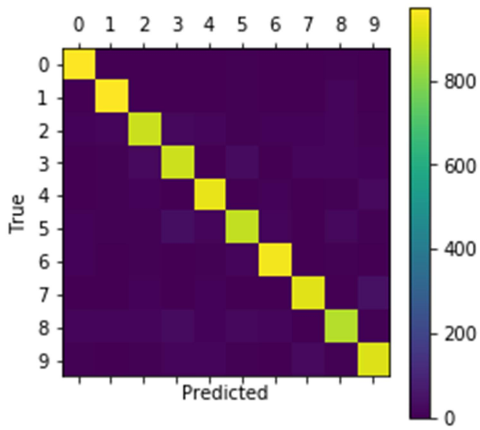
```
Training set Accuracy:93.448%
[[4786    1   12    7   11   33   30    7   32    4]
 [   1 5592   26   17    6   19    2   13   58    8]
 [  23   45 4503   72   58   24   59   53  108   13]
 [  14   18   95 4654    4  148   15   39  105   39]
 [   8   20   21    7 4576    6   42   13   24  125]
 [  39   13   36  117   34 3963   68   18  102   31]
 [  23   11   29    1   24   52 4758    2   16    2]
 [   8   16   49   18   34    9    4 4989   14  124]
 [  22   75   51  103   16  113   23   16 4387   45]
 [  17   18    9   55  126   30    2  134   42 4516]]
```



Class0	Class1	Class2	Class3	Class4	Class5	Class6	Class7	Class8	Class9
3.13%	3.43%	6.78%	7.85%	6.40%	9.87%	4.89%	5.58%	10.24%	7.96%

Validation Dataset Error: 10000 – 9248 = 752

```
 Validation set Accuracy:92.47999999999999%
[[975   0   1   3   2   7   3   2   6   1]
 [  0 972   3   2   1   5   0   2  13   2]
 [ 10  13 896  22  13   4  11   9  18   4]
 [  1   7  23 902   3  28   2  12  13   9]
 [  1   4   8   3 941   1  10   2   7  23]
 [  9   4   6  37  17 884  14   2  22   5]
 [  9   2   4   1   7  12 957   1   6   1]
 [  2   3   9   0   9   1   0 931   3  42]
 [ 13  17  19  27   9  20  19   2 868   6]
 [  4   3   5  14  19   4   1  24   4 922]]
```

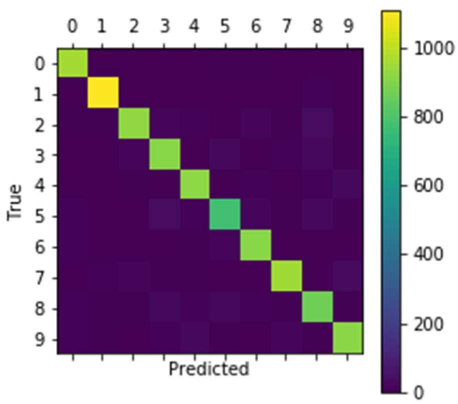


Error for each class:

Class0	Class1	Class2	Class3	Class4	Class5	Class6	Class7	Class8	Class9
4.78%	5.17%	8.00%	10.78%	7.83%	8.48%	5.89%	5.67%	9.58%	9.16%

Testing Dataset Error : 10000 – 9255 = 745

```
Testing set Accuracy:92.55%
[[ 960    0    0    3    0    6    6    4    1    0]
 [   0 1110    3    2    0    2    4    2   12    0]
 [   6    8  924   16   10    3   14    8   39    4]
 [   4    1   20  914    0   25    3   10   26    7]
 [   1    1    6    2  921    0    9    4    9   29]
 [  10    2    2   37   10  773   15    6   30    7]
 [   9    3    4    2    7   15  914    3    1    0]
 [   1    9   19    6    6    2    0  952    2   31]
 [   9    8    6   26    9   23   10    8  868    7]
 [  11    8    0   10   28    5    0   20    8  919]]
```




```
Training set Accuracy:93.448%

Validation set Accuracy:92.47999999999999%

Testing set Accuracy:92.55%
```

Error for each class:

Class0	Class1	Class2	Class3	Class4	Class5	Class6	Class7	Class8	Class9
13.59%	3.47%	6.09%	10.21%	7.06%	9.48%	6.25%	6.39%	12.85%	8.46%

Conclusion:

The total training and test errors are nearly equal. But there is lot of error variation in the individual classes of the dataset. One possible reason it could be is that variance is not properly explained in the training set i.e. the data is too simple, causing underfitting.

The performance of Multinomial and Binary/ One vs All Logistic Regression is almost the same, the latter may contain larger standard errors due to which there are difference observed normally, and hence Multinomial Logistic Regression normally performs better but here in this case they're almost the same.