CSE 574 : Introduction to Machine Learning

Programming Assignment 3: Classification and Regression

Group 4

Shubham Sharma (Person No.: 50290293)
Aditya Sahay(Person No.: 50292761)
Katyayni Shankar Kaushik(Person No.: 50289158)

May 11, 2019

The dataset provided to us, i.e. *mnist_all.mat* (original dataset from MNIST has 10 matrices for testing set and 10 matrices for training set, corresponding to 10 digits. The training set data (60000 examples) is split into training (50000 examples) and validation data (10000 examples. Also, as a part of the pre-processing step we ignore the columns(features) which give no new information, i.e. for which there is no variation between the data points (standard deviation > 0.001).

Logistic Regression: one-vs-all strategy

A binary logistic regression classifier is implemented using $one - vs - all$ strategy to classify the images into one of the 10 classes. Cross-entropy error function is obtained by taking the negative logarithm of the log likelihood for each class. The Python scipy function *scipy.optimize.minimize* is used to solve the optimization problem of finding the weights to minimize the error function using the option: conjugate gradient descent using the error and error gradient value obtained for each class.

The Training set Accuracy obtained is 92.746%, Validation set Accuracy is 91.51% and Testing set Accuracy is 91.88%. Clearly, the test accuracy is lower than the training accuracy which is what is expected since our model is generalizing fairly well for unseen data i.e. test data.

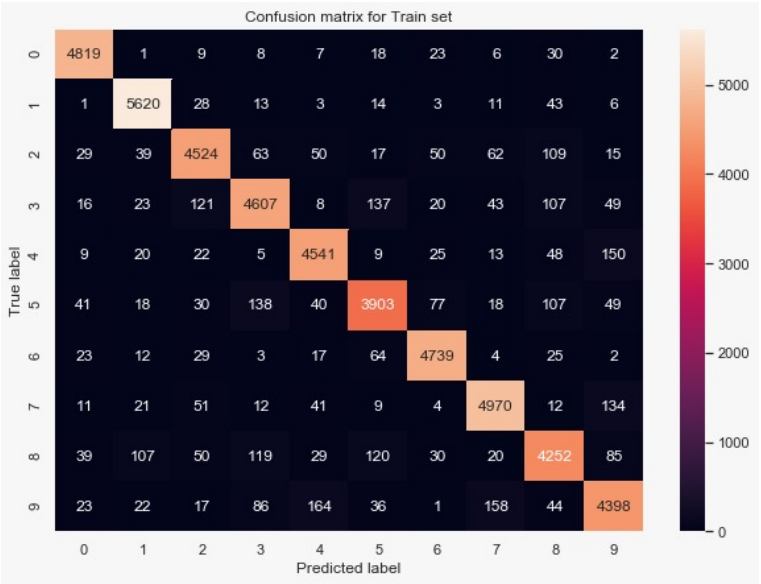

Figure 1: Confusion Matrix for training data set for binary logistic regression

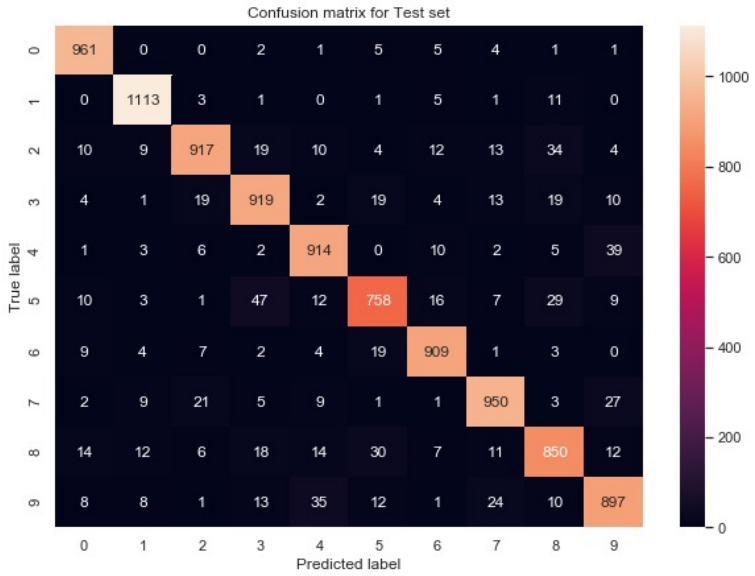

Figure 2: Confusion Matrix for test data set for binary logistic regression

	accuracy
0	0.980612
1	0.980617
2	0.888566
3	0.909901
4	0.930754
5	0.849776
6	0.948852
7	0.924125
8	0.872690
9	0.888999

Figure 3: Accuracy w.r.t each category in test data for binary logistic regression

	precision	recall	f1-score	support
0	0.94	0.98	0.96	980
1	0.96	0.98	0.97	1135
2	0.93	0.89	0.91	1032
3	0.89	0.91	0.90	1010
4	0.91	0.93	0.92	982
5	0.89	0.85	0.87	892
6	0.94	0.95	0.94	958
7	0.93	0.92	0.93	1028
8	0.88	0.87	0.88	974
9	0.90	0.89	0.89	1009
micro avg	0.92	0.92	0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

Figure 4: Classification report w.r.t each category in test data for binary logistic regression

The accuracies with respect to each category in test data is summarised in figure 3. Using the confusion matrices in figures 1 and 2 we can identify the error rate that has been incurred for each class. Thus, upon looking into the confusion matrix, we can arrive at a conclusion that for digit 8,5 and 2 the accuracy is low comparatively. This is because they have similar shapes and thus, the model is finding it difficult to separate them in different classes. The accuracy is best for digits 1 and 2. From the confusion matrix for training set (figure 1), we can see which digits, the system is misinterpreting as a wrong digit, e.g. 7 is misinterpreted as 9 in 134 cases out of 5265 true cases of 7. The figure 4 shows that precision and recall values are high for 0 and 1 while low for 8 and 5 which is consistent with our previous discussion.

Multi-class Logistic Regression

In order to do multi-class classification using logistic regression without the need to use the $one-vs-all$ strategy, we use softmax function to calculate the posterior probabilities. The Training set Accuracy obtained is 93.448%, Validation set Accuracy is 92.48% and Testing set Accuracy is 92.55%. The confusion matrix for training data set is shown in figure 5 which shows minimum correct classifications for the digit 5. In figure 6 the confusion matrix is shown for test data, accuracy for each digit for test data is shown in figure 7 and the classification report for the test data is shown in figure 8.

We observe that the test set accuracy for Multi-class logistic regression approach is slightly better than the $one-vs-all$ strategy and hence should be the preferred approach for multi-class classification using the logistic regression.

If the dataset can be learned approximately with linear hypothesis, it could be interesting to use multi-class logistic regression since in an $one-vs-all$ strategy, one could build 3 independent classifiers and, for an unseen instance, choose the class for which the probability is highest. Conversely, in multi-class logistic regression it learns all the classes directly. In this way, the parameters of each class are estimated interdependently and the model built may be more robust against outliers.

Multi-class classifiers can be a good option when we have many classes but would be slower during training as compared to binary classifiers. On the other hand, binary classifier would be good if classes are less and would also pose problems in case of class imbalance.

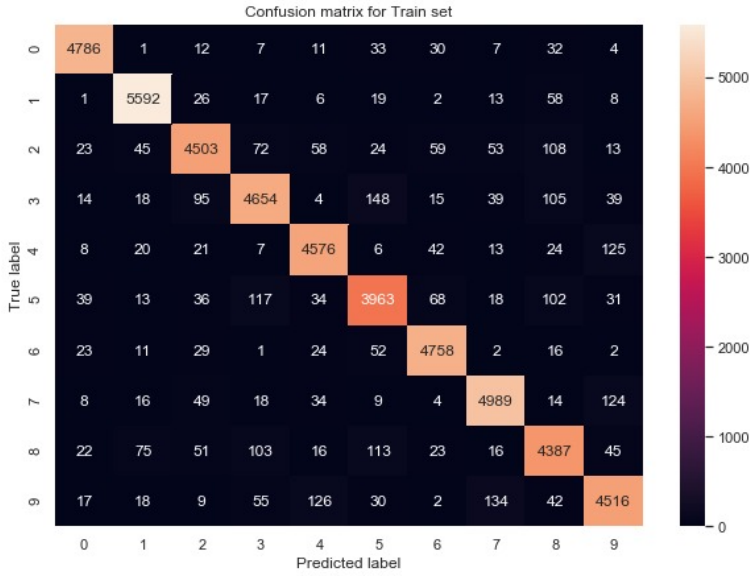

Figure 5: Confusion Matrix for training data set for multi-class logistic regression

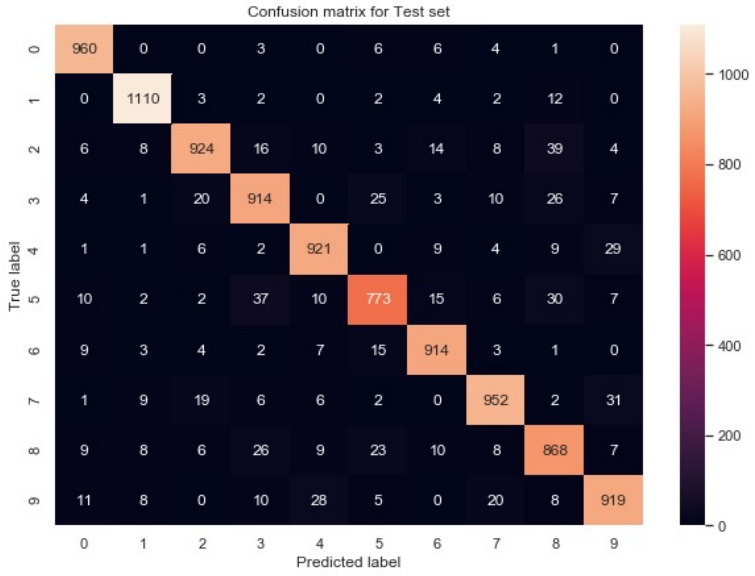

Figure 6: Confusion Matrix for test data set for multi-class logistic regression

	accuracy
0	0.979592
1	0.977974
2	0.895349
3	0.904950
4	0.937882
5	0.866592
6	0.954071
7	0.926070
8	0.891170
9	0.910803

Figure 7: Accuracy w.r.t each category in test data for multi-class logistic regression

	precision	recall	f1-score	support
0	0.94	0.98	0.96	980
1	0.96	0.98	0.97	1135
2	0.93	0.89	0.91	1032
3	0.89	0.91	0.90	1010
4	0.91	0.93	0.92	982
5	0.89	0.85	0.87	892
6	0.94	0.95	0.94	958
7	0.93	0.92	0.93	1028
8	0.88	0.87	0.88	974
9	0.90	0.89	0.89	1009
micro avg	0.92	0.92	0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

Figure 8: Classification report w.r.t each category in test data for multi-class logistic regression

Support Vector Machines

Support Vector Machines are implemented using the $sklearn.svm.SVC$ implementation to perform classification on our data set. As mentioned, linear and radial basis function are used to compare performances for different values of $Cost\ parameter : C$ and γ. The training data is sub-sampled for 10,000 entries to learn the SVM models and compute training, validation and test accuracy. The results are summarised for Linear kernel, Radial Basis Function with value of gamma setting to 1, and Radial Basis function with value of gamma setting to default in figure 9. As we see in figure 9 as well as figure 10, the classifier using RBF kernel with $\gamma = 1$ performs poorly for validation and test set data despite performing perfectly for the training data. This is because this model fits the training data too well, which is known as $overfitting$ and hence is not able to generalize well. Thus, we infer that the $\gamma = 1$ is not a good choice in this case. We observe in figure 12 that as we increase the value of C, the accuracies increase and then become constant.

Comparison of Linear and RBF Kernel:

Linear Kernel is used when the data is linearly separable, that is, it can be separated using a single Line. It is mostly used when there are a large number of features in a particular data set. RBF kernel is used to classify data that is not linearly separable.

Linear SVM is a parametric model, and RBF kernel SVM isn't, and the complexity of the latter grows with the size of the training set

C, a parameter for the soft margin cost function, plays a balancing role between how much value we are willing to concede and how much margin we want to have. It plays the same role as lambda (regularization parameter) in regression scenario where lambda balances training error complexity. The margin can be seen as the inverse of complexity. This trades off misclassification of training examples against simplicity of the decision surface, i.e. a balance between classification of training points accurately and a smooth decision boundary . Hence, it suggests the model to choose data points as support vectors. If C is large, a smaller margin hyperplane is chosen since the model chooses more data points as a support vector and vice versa for small value of C irresepective of the number of misclassifications for a larger margin i.e. fewer data points are chosen as a support vector leading to lower variance and high bias, .

The optimal value of C is found out to be 40.

The γ parameter defines how far the influence of a single training example reaches and it affects this inversely, can be considered as the training error that we are willing to accommodate. Thus, low values mean far away points carry more weights than nearer points and thus our decision boundary becomes

more like a straight line whereas high values mean our decision boundary will depend on points close to the decision boundary and nearer points carry more weights than far away points due to which our decision boundary becomes more wiggly. Hence, γ depicts the inverse of the radius of influence of samples selected by the model as support vectors. The optimal value for γ is default which is 1/no. of features i.e. 1/716 since for $\gamma = 1$ we see very low validation and test set accuracies.

On comparing the RBF and linear kernel, we observe that RBF kernel performs better than the linear kernel slightly. The results for the optimal parameters obtained have been shown in the last row of figure 10.

	Training	Validation	Test
Linear	99.68	92.00	91.96
RBF (Gamma 1)	100.00	15.82	17.60
RBF (Default)	92.79	91.98	92.45

Figure 9: Results Summary for Linear kernel, Radial Basis Function with value of gamma setting to 1, and Radial Basis function with value of gamma setting to default. All Values in %. This is for training set size = 10000 random samples

	Training	Validation	Test
Linear	97.286	93.64	93.78
RBF (Gamma 1)	100.000	15.48	17.14
RBF (Default)	94.294	94.02	94.42
RBF (C = 40)	98.706	97.23	97.19

Figure 10: Results Summary for Linear kernel, Radial Basis Function with value of gamma setting to 1, Radial Basis function with value of gamma setting to default and Radial Basis function with value of gamma setting to default and optimal C = 40. All Values in %. This is for training set size = 50000 samples

All computations and recording of run times is done on Macbook Pro (Core-i7 8th Generation 16 GB RAM)

C-Value	Training	Validation	Test
1	92.79	91.98	92.45
10	96.58	94.18	94.61
20	97.82	94.48	94.67
30	98.43	94.65	94.85
40	99.02	94.65	94.87
50	99.36	94.55	94.92
60	99.49	94.50	94.92
70	99.57	94.40	94.97
80	99.67	94.38	95.01
90	99.75	94.34	95.00
100	99.82	94.34	94.95

Figure 11: Training, validation and test set accuracies for different values of C using Radial Basis Function kernel. All Values in %. This is for training set = 10000 samples

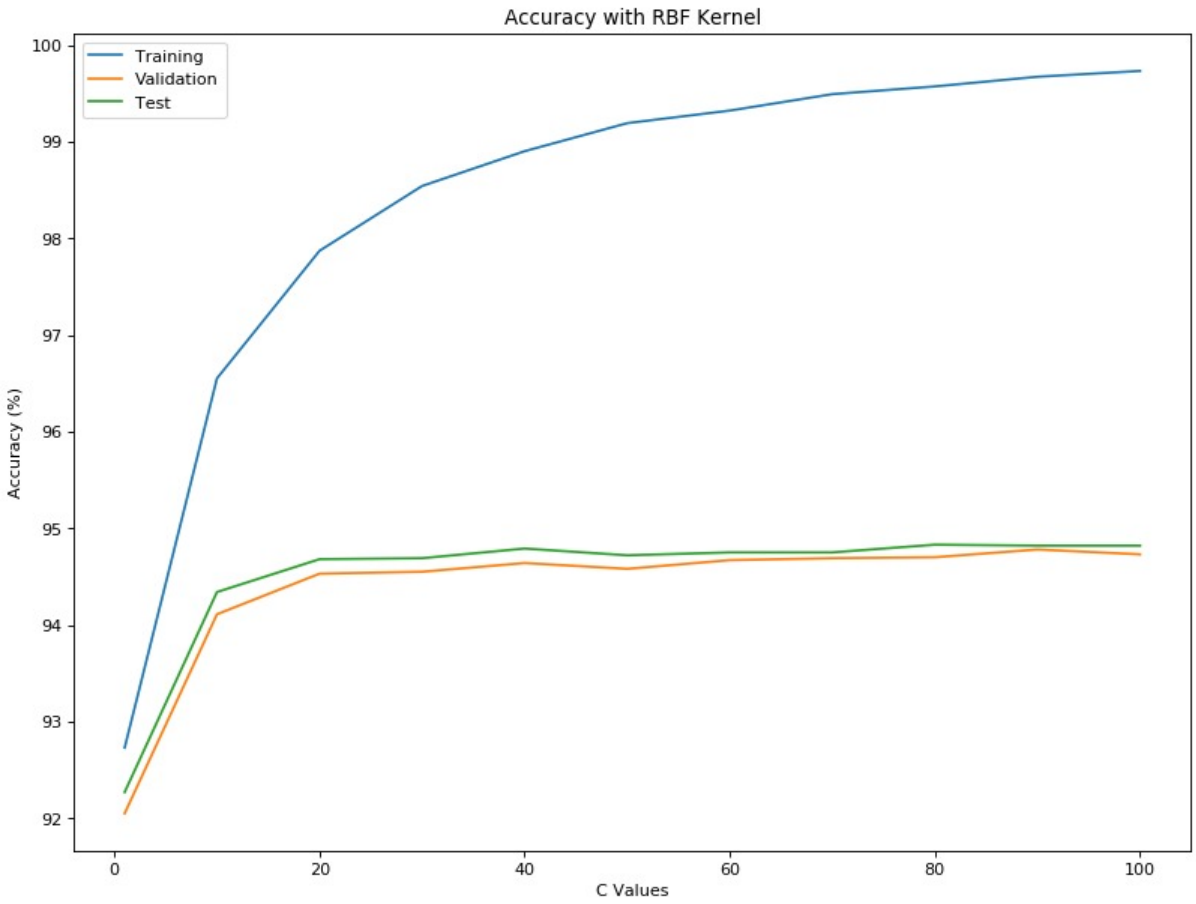

Figure 12: Plot for Training, validation and test set accuracies for different values of C using Radial Basis Function kernel. This is for training set = 10000 samples