CSE 574: Introduction to Machine Learning

Assignment 3

Team: Sphoorthi Keshannagari (50372699) skeshann@buffalo.edu
Ishita Aggarwal (50431309) iaggarwa@buffalo.edu
Bharath Reddy Nallu (50376364) bnallu@buffalo.edu

Logistic Regression

In this project we implemented Logistic Regression, Multi-class Logistic Regression and Support Vector Machines

1.1 **Logistic Regression:**

We are making use of MNIST dataset to implement Logistic Regression to classify hand-written digit images into correct corresponding labels. We have 10 classes, each class to recognise the digits 0-9. We build 10 binary classifiers to distinguish a class from all other classes using one-vs-all strategy.

Preprocess() function is already defined which loads the data and then divides it into training, validation and test data set.

Sigmoid(z) is already defined which returns f(z) value as below:

$$f(z) = \frac{1}{1 + exp(-z)}$$

The below functions were implemented:

1.1.1 blrObjFunction(X, w_k, y_k):

The input to the above functions are weight vector w_k of size (D + 1) x 1 , column vector y_k label vector (y_k) of size N x 1 and data matrix X i.e. train data of size N x D.
We add bias to the data matrix and compute the below equation 2 to obtain the error value.

$$E(\mathbf{w}) = -\frac{1}{N} \ln p(\mathbf{y}|\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} \{y_n \ln \theta_n + (1 - y_n) \ln(1 - \theta_n)\}$$

We then compute the gradient of the error function with respect to w, the equation is computed as below:

$$\nabla E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (\theta_n - y_n)\mathbf{x}_n$$

The error and error gradient are the outputs of the function blrObjFunction().

1.1.2 blrPredict(W,data):

This function takes the data matrix as input and takes the matrix of weight W as input. Each column in the weight matrix is the weight vector of a logistic regression classifier. We first compute the posterior probability of a class, using the formula below:

$$P(y = C_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$$

Then we calculate the argmax of posterior probability obtained and output a column vector label.

1.1.3 Final Results:

Accuracy of classification method on the handwritten digits test data –

Overall:
Training set Accuracy: 92.67 %
Training set Error: 7.33 %
Validation set Accuracy: 91.46 %
Validation set Error: 8.54 %
Testing set Accuracy: 91.94 %
Testing set Error: 8.06 %

Class-wise distribution on the basis of digits 0-9:

Class	Training Accuracy (%)	Training Error (%)	Testing Accuracy (%)	Testing Error (%)
0	97.83	2.17	98.17	1.83
1	97.91	2.09	98.24	1.76
2	91.12	8.88	88.95	11.05
3	89.55	10.45	91.09	8.91
4	93.80	6.20	93.28	6.72
5	87.99	12.01	85.32	14.68
6	96.30	3.70	94.68	5.32
7	94.23	5.77	92.22	7.78
8	87.45	12.55	87.06	12.94
9	89.15	10.85	89.10	10.90

We can infer that the training error is slightly less than the test error. It could be because the test data points for training sample are a lot more than that for test sample.

1.2 **Support Vector Machine:**

We implemented the Support Vector Machine tool in sklearn.svm.SVM to perform classification on our data set. We take only 10,000 random values from a set of 50,000 values since SVM models are known to be difficult to scale well to large dataset.

1.2.1 Using Linear Kernel:

We compute the accuracy of prediction with respect to training, validation and test data using kernel='linear' in the svm.SVC tool.

Accuracy of classification method with kernel = linear:

Training Accuracy: 99.59 %
Training Error: 0.41 %
Validation Accuracy: 91.59 %
Validation Error: 8.41 %
Testing Accuracy: 92.09 %
Testing Error: 7.91 %

1.2.2 Using Radial Basis Function with Gamma setting to 1:

We compute the accuracy of prediction with respect to training, validation and test data using kernel='rbf' and gamma = 1.0 in the svm.SVC tool.

Accuracy of classification method with kernel = 'rbf' and gamma= 1.0:
Training Accuracy: 100 %
Training Error: 0 %
Validation Accuracy: 10.13 %
Validation Error: 89.87 %
Testing Accuracy: 11.66 %
Testing Error: 88.34 %

The Validation and Testing Accuracy is low because of poor generalization of the model on validation and test data.

1.2.3 Using Radial Basis Function with Gamma setting to default:

We compute the accuracy of prediction with respect to training, validation and test data using kernel='rbf' and gamma = 'auto & scale' in the svm.SVC tool.
When gamma is '**auto**', it's value will be 1/n_features. Whereas, when it is **'scale'**, it's value will be 1/(n_features * X.var())

Accuracy of classification method with kernel = 'rbf' and gamma= '**auto**':

Training Accuracy: 92.53 %
Training Error: 7.47 %
Validation Accuracy: 92.10 %
Validation Error: 7.90 %
Testing Accuracy: 92.56 %
Testing Error: 7.44 %

Accuracy of classification method with kernel = ‘rbf’ and gamma= ‘**scale**:

Training Accuracy: 98.58 %
Training Error: 1.42 %
Validation Accuracy: 96.31 %
Validation Error: 3.69 %
Testing Accuracy: 96.37 %
Testing Error: 3.63 %

1.2.4 Using Radial Basis Function with value of gamma setting to default and varying value of C(1,10,20…100):

We calculate the accuracy of prediction over 11 C values (regularization parameter of error) [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100] with respect to training, validation and test data using kernel=’rbf’ and C = c in the svm.SVC tool.

Table containing the Accuracy values in % with gamma=**’auto’**:

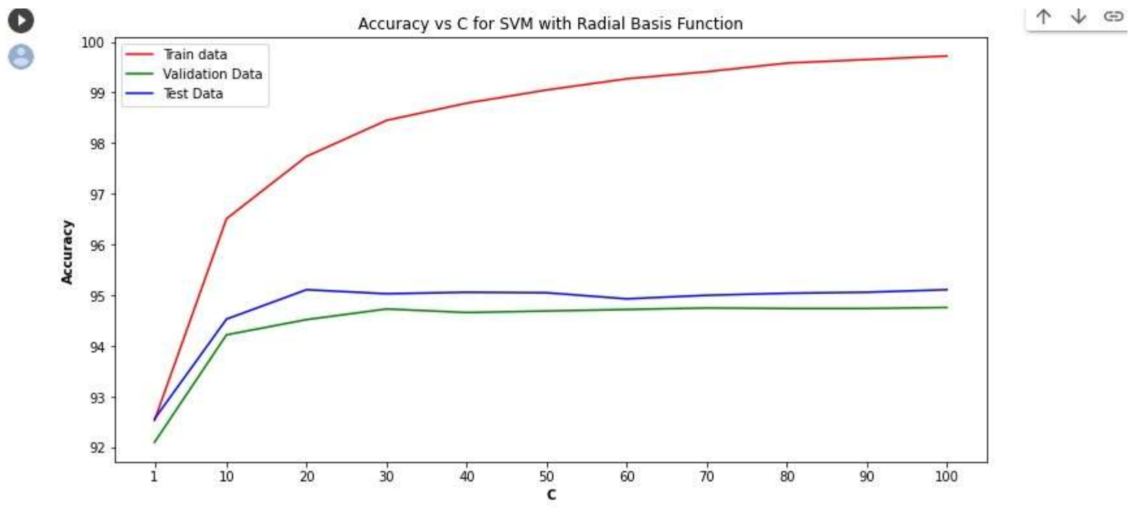
C	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)
1	92.53	92.10	92.56
10	96.51	94.22	94.53
20	97.74	94.52	95.11
30	98.45	94.73	95.03
40	98.79	94.66	95.06
50	99.05	94.69	95.05
60	99.27	94.72	94.93
70	99.41	94.75	95
80	99.58	94.74	95.04
90	99.65	94.74	95.06
100	99.72	94.76	95.11

Table containing the Accuracy values in % with gamma='**scale**':

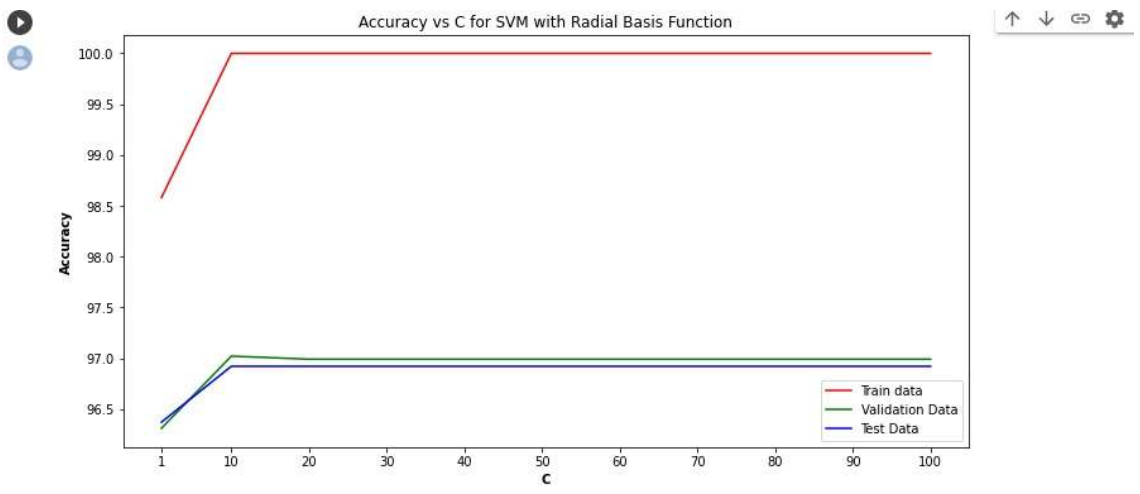
C	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)
1	98.58	96.31	96.37
10	100	97.02	96.92
20	100	96.99	96.92
30	100	96.99	96.92
40	100	96.99	96.92
50	100	96.99	96.92
60	100	96.99	96.92
70	100	96.99	96.92
80	100	96.99	96.92
90	100	96.99	96.92
100	100	96.99	96.92

1.2.5 <u>Output:</u>

Graph of Accuracy with respect to values of C [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100] over Training, Validation and Test Data and gamma = '**auto**':



Graph of Accuracy with respect to values of C [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100] over Training, Validation and Test Data and gamma = ‘**scale**’:



Best C = 10

We compute the accuracy of prediction with respect to training, validation and test data using kernel=’rbf’ and gamma = ‘auto & scale’ and C = 10 in the svm.SVC tool to run over the complete set and obtain accuracy results with best C.

Accuracy of classification method with kernel = ‘rbf’ and gamma= ‘**auto**’ and C = 10:

Training Accuracy: 97.13 %

Validation Accuracy: 96.18 %

Testing Accuracy: 96.1 %

Accuracy of classification method with kernel = ‘rbf’ and gamma= ‘**scale**’ and C = 10:

Training Accuracy: 99.988 %

Validation Accuracy: 98.45 %

Testing Accuracy: 98.34 %

1.3 **Multi-Class Logistic Regression:**

We implemented the Multi-Class Logistic Regression to solve multi-class classification. The benefit of doing so is that we do not need to build 10 classifiers like before. The functions implemented are as below:

1.3.1 mlrObjFunction(params, *args):

The input to the above functions are weight vector w_k of size (D + 1) x 10 , column vector y_k label vector of size N x 1 and data matrix x i.e. train data of size N x D.

We add bias to the data matrix.

We first compute the posterior probability (theta) with the below formula:

$$P(y = C_k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T\mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T\mathbf{x})}$$

We then calculate the error by taking the negative of the sum of log of theta values, by implementing the below formula:

$$E(\mathbf{w}_1, \cdots, \mathbf{w}_K) = -\ln\ P(\mathbf{Y}|\mathbf{w}_1, \cdots, \mathbf{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln\ \theta_{nk}$$

We then calculate the error gradient by taking the summation of the data X multiplied by the subtraction of theta value with the column vector, by implementing the below formula:

$$\frac{\partial E(\mathbf{w}_1, \cdots, \mathbf{w}_K)}{\partial \mathbf{w}_k} = \sum_{n=1}^{N}(\theta_{nk} - y_{nk})\mathbf{x}_n$$

The error and error gradient are the outputs of the function mlrObjFunction().

1.3.2 mlrPredict(W, data):

This function takes the data matrix as input and takes the matrix of weight W as input.
We add bias to the data matrix, then we compute the posterior probability using the below formula:

$$P(y = C_k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T\mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T\mathbf{x})}$$

Then we calculate the argmax of posterior probability obtained and output a column vector label.

1.3.3 Final Results:

Accuracy of classification method on the handwritten digits test data –

Overall:
Training set Accuracy: 93.18 %
Training set Error: 6.82 %
Validation set Accuracy: 92.46 %
Validation set Error: 7.54 %
Testing set Accuracy: 92.51 %
Testing set Error: 7.49 %

Class-wise distribution on the basis of digits 0-9:

Class	Training Accuracy (%)	Training Error (%)	Testing Accuracy (%)	Testing Error (%)
0	97.24	2.76	98.37	1.63
1	97.44	2.56	97.80	2.20
2	90.80	9.20	89.92	10.08
3	90.53	9.47	90.69	9.31
4	94.13	5.87	93.48	6.52
5	89.03	10.97	86.88	13.12
6	95.79	4.21	94.68	5.32
7	94.09	5.91	91.93	8.07
8	89.94	10.05	88.30	11.70
9	91.67	8.33	91.97	8.03

We can infer that the training error is slightly less than the test error. It could be because the test data points for training sample are a lot more than that for test sample.

1.3.4 Comparing the performance difference between multi-class strategy and one-vs-all strategy:

Set	BLR Accuracy (%)	MLR Accuracy (%)
Training	92.67	93.18
Validation	91.46	92.46
Testing	91.94	92.51

The probability of a digit belonging to a class is directly calculated in MLR and hence has a higher probability of belonging to a particular class. In BLR, this probability is calculated in a one-vs-rest approach which gives a relatively lesser probability and hence, lesser accuracy.