Machine Learning (CSE 574) - 3

PROBLEM STATEMENT:

Implement Logistic Regression to classify hand-written digit images into correct corresponding labels. Build 10 binary-classifiers (one for each class) to distinguish a given class from all other classes. Implement multi-class Logistic Regression. Build 1 classifier that can classify 10 classes at the same time.

Below are the tasks to be performed: -

(i) Implement Logistic Regression and give the prediction results.

(ii) Use the Support Vector Machine (SVM) toolbox sklearn.svm.SVM to perform classification.

(iii) Implement the gradient descent minimization of multi-class Logistic Regression (using softmax function)

SOLUTION:

Dataset MNIST - (mnist all.mat) has been provided. The task is to implement Binary and Multinomial Logistic Regression and evaluate its performance in classifying handwritten digit images into correct corresponding labels.

Preprocessing has already been done on the data set to obtain the training, validation and the test data.

→ *Binary Logistic Regression (blrObjFunction() / blrPredict())*

"*blrObjFunction"* takes in 3 parameters as input. The bias is added at the beginning of the vector. The logistic regressor is trained using the given data X (Preprocessed feature vectors of MNIST data) with labels y. The output of the function is an error value and the gradient error.

The final accuracies obtained for blr function are:

'*Training Accuracy' - 92.734%*

'Validation Accuracy' - 91.52%

'Test Accuracy' - 92.0%

→ *BLR Confusion Matrix and Classification Report*

Training Set Confusion Matrix

BLR Training Confusion Matrix

```
[[4819    1    9    8    7   18   23    6   30    2]
 [   1 5620   28   13    3   14    3   11   43    6]
 [  29   39 4524   63   50   17   50   62  109   15]
 [  16   23  121 4607    8  137   20   43  107   49]
 [   9   20   22    5 4541    9   25   13   48  150]
 [  41   18   30  138   40 3903   77   18  107   49]
 [  23   12   29    3   17   64 4739    4   25    2]
 [  11   21   51   12   41    9    4 4970   12  134]
 [  39  107   50  119   29  120   30   20 4252   85]
 [  23   22   17   86  164   36    1  158   44 4398]]
```

In the above training set confusion matrix, the non-diagonal elements show the number of observations that were not predicted correctly against the true labels whereas the diagonal elements show the count of correctly predicted observations. For example, for class 0, 4819 values were correctly predicted whereas (1+9+8+7+18+23+6+30+2 =) 104 were not predicted correctly. Similarly, results can be predicted for other classes.

Training Set Classification Report

BLR Training Classification Report

	precision	recall	f1-score	support
0	0.98	0.96	0.97	5011
1	0.98	0.96	0.97	5883
2	0.91	0.93	0.92	4881
3	0.90	0.91	0.90	5054
4	0.94	0.93	0.93	4900
5	0.88	0.90	0.89	4327
6	0.96	0.95	0.96	4972
7	0.94	0.94	0.94	5305
8	0.88	0.89	0.88	4777
9	0.89	0.90	0.89	4890
micro avg	0.93	0.93	0.93	50000
macro avg	0.93	0.93	0.93	50000
weighted avg	0.93	0.93	0.93	50000

The above classification report shows a representation of the blr training set classification metrics on a per-class basis to give a better intuition of the classifier performance on the training data. The **best**

train results can be checked using the "***f1-score***" and were obtained for **class 0, 1 and 6**, while the **worst train results** were obtained for **class 5, 8 and 9.**

Test Set Confusion Matrix

```
BLR Test Confusion Matrix
[[ 961    0    0    2    1    5    5    4    1    1]
[   0 1113    3    1    0    1    5    1   11    0]
[  10    9  917   19   10    4   12   13   34    4]
[   4    1   19  919    2   19    4   13   19   10]
[   1    3    6    2  914    0   10    2    5   39]
[  10    3    1   47   12  758   16    7   29    9]
[   9    4    7    2    4   19  909    1    3    0]
[   2    9   21    5    9    1    1  950    3   27]
[  14   12    6   18   14   30    7   11  850   12]
[   8    8    1   13   35   12    1   24   10  897]]
```

In the above test set confusion matrix, for example, for class 0, 961 values were correctly predicted whereas (0+0+2+1+5+5+4+1+1=) 19 were not predicted correctly. Similarly, results can be predicted for other classes.

Test Set Classification Report

```
BLR Test Classification Report
              precision    recall  f1-score   support

           0       0.98      0.94      0.96      1019
           1       0.98      0.96      0.97      1162
           2       0.89      0.93      0.91       981
           3       0.91      0.89      0.90      1028
           4       0.93      0.91      0.92      1001
           5       0.85      0.89      0.87       849
           6       0.95      0.94      0.94       970
           7       0.92      0.93      0.93      1026
           8       0.87      0.88      0.88       965
           9       0.89      0.90      0.89       999

   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
```

The above classification report shows a representation of the blr test set classification metrics on a per-class basis to give a better intuition of the classifier performance on the test data. The **best test results** can be checked using the "***f1-score***" and were obtained for **class 0 and 1**, while the **worst test results** were obtained for **class 5, 8 and 9.**

→ *Multinomial Logistic Regression (mlrObjFunction() / mlrPredict())*

Logistic Regression can also be used to implement multi-class classification. Hence, "*mlrObjFunction"* is used to build 1 classifier that can classify 10 classes at the same time. The function takes in 3 parameters as input. The bias is added at the beginning of the vector. The logistic regressor is trained using the given data X (Preprocessed feature vectors of MNIST data) with labels y. The output of the function is an error value and the gradient error.

The final accuracies obtained for mlr function are:

'*Training Accuracy' - 93.166%*
'Validation Accuracy' - 92.41%
'Test Accuracy' - 92.51%

→ *MLR Confusion Matrix and Classification Report*

Training Set Confusion Matrix

```
MLR Training Confusion Matrix
[[4801    1   15    5    9   28   20    8   33    3]
 [   1 5636   26   12    4   17    2    9   26    9]
 [  25   52 4541   70   51   16   41   49   99   14]
 [  23   27  113 4636    4  135   15   32   96   50]
 [  11   28   31    3 4516    6   39   11   31  166]
 [  52   29   51  117   32 3898   58   16  127   41]
 [  32   16   43    0   29   53 4715    2   26    2]
 [   9   22   57   17   36   11    3 4925   15  170]
 [  26  124   57   92   19  100   18    9 4350   56]
 [  22   31   11   55  103   28    2  107   31 4559]]
```

In the above training set confusion matrix, the non-diagonal elements show the number of observations that were not predicted correctly against the true labels whereas the diagonal elements show the count of correctly predicted observations. For example, for class 0, 4801 values were correctly predicted whereas (1+9+8+7+18+23+6+30+2 =) 104 were not predicted correctly. Similarly, results can be predicted for other classes.

Training Set Classification Report

MLR Training Classification Report

	precision	recall	f1-score	support
0.0	0.96	0.98	0.97	4923
1.0	0.94	0.98	0.96	5742
2.0	0.92	0.92	0.92	4958
3.0	0.93	0.90	0.91	5131
4.0	0.94	0.93	0.94	4842
5.0	0.91	0.88	0.89	4421
6.0	0.96	0.96	0.96	4918
7.0	0.95	0.94	0.94	5265
8.0	0.90	0.90	0.90	4851
9.0	0.90	0.92	0.91	4949
micro avg	0.93	0.93	0.93	50000
macro avg	0.93	0.93	0.93	50000
weighted avg	0.93	0.93	0.93	50000

The above classification report shows a representation of the mlr test set classification metrics on a per-class basis to give a better intuition of the classifier performance on the training data. The **best train results** can be checked using the "***f1-score***" and were obtained for **class 0, 1 and 6**, while the **worst test results** were obtained for **class 5 and 8.**

Test Set Confusion Matrix

In the below test set confusion matrix, for example, for class 0, 964 values were correctly predicted whereas (0+1+3+0+4+3+4+1+0=)

16 were not predicted correctly. Similarly, results can be predicted for other classes.

```
MLR Test Confusion Matrix
[[ 964    0    1    3    0    4    3    4    1    0]
[   0 1120    4    1    0    1    3    2    4    0]
[   7   12  926   14    9    4   14    8   33    5]
[   5    2   20  914    0   22    4   10   24    9]
[   1    3   10    1  910    0    7    5    6   39]
[  10    3    4   43    9  761   12    6   36    8]
[  14    3    5    2    8   15  906    3    2    0]
[   1   11   21    6    7    1    0  943    2   36]
[   8   16    7   22    9   20   11    7  864   10]
[  11    9    1    8   23    6    0   14    8  929]]
```

Test Set Classification Report

```
MLR Test Classification Report
              precision    recall  f1-score   support

         0.0       0.94      0.98      0.96       980
         1.0       0.95      0.99      0.97      1135
         2.0       0.93      0.90      0.91      1032
         3.0       0.90      0.90      0.90      1010
         4.0       0.93      0.93      0.93       982
         5.0       0.91      0.85      0.88       892
         6.0       0.94      0.95      0.94       958
         7.0       0.94      0.92      0.93      1028
         8.0       0.88      0.89      0.88       974
         9.0       0.90      0.92      0.91      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
```

The above classification report shows a representation of the mlr test set classification metrics on a per-class basis to give a better intuition of the classifier performance on the test data. The **best test results** can be checked using the "***f1-score***" and were obtained for **class 0 and 1**, while the **worst test results** were obtained for **class 5 and 8.**

→ *Comparison (Multiclass vs One-vs-All Strategy)*

Below table shows the overall final accuracies comparison:

Strategy	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
Multiclass	93.166	92.41	92.51
One-vs-All	92.734	91.52	92.0

Below table shows the misclassification results (error count) and the corresponding error percentage in the training and test set of BLR and MLR functions:

Class	BLR Function Misclassification (Error)				MLR Function Misclassification (Error)			
	Training	Error (%)	Test	Error (%)	Training	Error (%)	Test	Error (%)
0	104	2.11	19	1.93	122	2.47	16	1.63
1	122	2.12	22	1.93	106	1.84	15	1.32
2	434	8.75	115	11.14	417	8.41	106	10.27
3	524	10.21	91	9.00	495	9.64	96	9.50
4	301	6.21	68	6.92	326	6.73	72	7.33
5	518	11.71	134	15.02	523	11.82	131	14.68
6	179	3.63	49	5.11	203	4.12	52	5.42
7	295	5.60	78	7.58	340	6.45	85	8.26
8	599	12.34	124	12.73	501	10.32	110	11.29
9	551	11.1	112	11.1	390	7.88	80	7.92

→ *Support Vector Machines (SVM)*

SVM models are difficult to scale well to a large dataset. Hence, the model is first fit by randomly sampling the training data set for 10000 records. This data is then further used to fit the model in 4 different ways:

→ *Linear Kernel*

Using the scikit learn SVM SVC model, fit the linear kernel and keep all the other parameters fixed. Fit the sampled 10000 training data to the model.

The accuracies obtained for Linear Kernel with **default Gamma** are:
'*Training Accuracy' - 99.68%*
'Validation Accuracy' - 91.16%
'Test Accuracy' - 91.93%

→ *Radial Basis function (RBF)* - ***Gamma set to 1***

Similar to above, fit the 'rbf' kernel with gamma value set to 1.
The accuracies obtained for RBF kernel are:
'*Training Accuracy' – 100.0%*
'Validation Accuracy' – 10.0%
'Test Accuracy' – 11.35%

The above results show that the model suffers from overfitting.

→ *Radial Basis function (RBF)* – ***Gamma set to default***

On fitting the radial basis model with default gamma.
The accuracies obtained for RBF kernel are:
'*Training Accuracy' - 92.92%*
'Validation Accuracy' - 91.8%
'Test Accuracy' - 92.39%

The above results show that although the model still suffers from overfitting but performs better than when Gamma was set to 1.

Kernel	Gamma	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
Linear	default	99.68	91.16	91.93
RBF	1	100.0	10.0	11.35
RBF	default	92.92	91.8	92.39

→ *Radial Basis function default Gamma setting and varying C values*

Now, learn the data by varying the cost values. Select a range of 1 to 100 with an increment of 10.
Below are the different accuracies that obtained using the different C values.

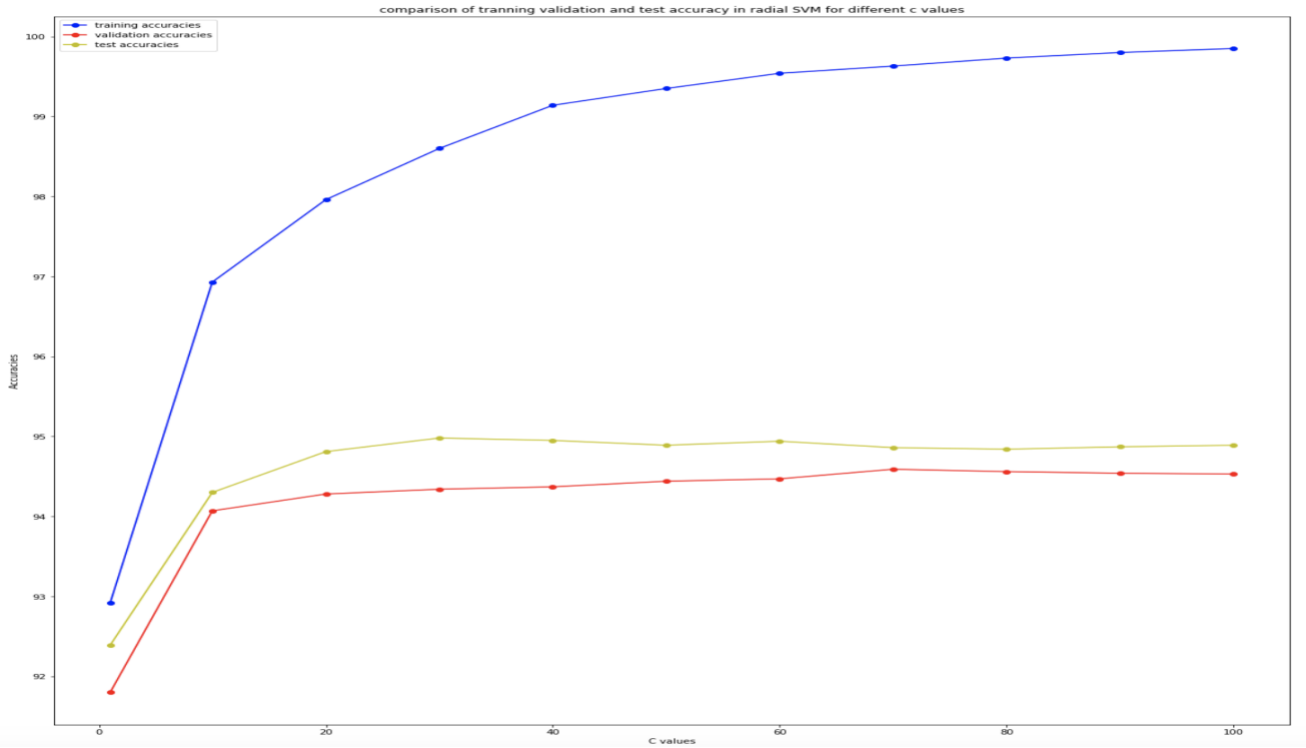
Cost Values	Train Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
1	92.92	91.8	92.39
10	96.93	94.07	94.3
20	97.96	94.28	94.81
30	98.6	94.34	94.98
40	99.14	94.37	94.95
50	99.35	94.44	94.89
60	99.54	94.47	94.94
70	99.63	94.59	94.86
80	99.73	94.56	94.84
90	99.8	94.54	94.87
100	99.85	94.53	94.89

→ *Training SVM – Full Data Set with best C-Value – RBF kernel*

The accuracies obtained using full data set for RBF when **optimum C=70** are:

'Training Accuracy' - 91.956%
'Validation Accuracy' - 91.8%
'Test Accuracy' - 92.39%

→ *Plot to show the variation of accuracies with C values*



It can be observed that the value of C has an impact over the accuracies of the model. This is because with the variation of C values the width of the margin varies. With a lower C value, the margin width is bigger as the weighed error is low. Thus, a lower penalty is set on the misclassification error. Whereas upon increasing the C values the margin width decreases as a result of which the misclassification also decreases and a higher accuracy is obtained.

But this might not always be the same as there is also a decrease in accuracy as well after an optimum C value.

CONCLUSION:

As it can be observed from the SVM models, the best results are obtained from the Radial Kernel with a **C value** set to **70**.

Finally, training the model with the **complete data set** - 50000 records below are the **accuracies** obtained:
Validation - 91.8% ; Test - 92.39%

To conclude, with high values of gamma the model over fits very badly. Also, for this dataset, radial models outperform the linear model. The variation of C values as explained above also have a considerable effect on the classification of the model. Having too high C-value can cause over-fitting while small C-value can lead to under-fitting.

Logistic regression works in just classifying the data by learning a line which might not work well with a different data set while SVM models try to learn the best line that can classify the data. It learns a margin as well. Using the concept learning there is a good chance that this line and margin will also be able to work with a different data set as well. When the input dimensions are quite high it becomes very complex for the Logistic regression model to learn all the parameters. This is not the case for SVM models.

The difference between training and test error could be because the training error is obtained when the trained model is run back on the training data but while test error is obtained when the trained model is run on the data that has never been exposed to before.