Report

Handwritten Digits Classification

CSE 574 Assignment 3

Submitted By:-

Name	**UBIT name**	**Person #**
Pratik Bhat	pratiksu	50096806
Anirudha Karwa	akarwa	50097742
Andreina Uzcategui	auzcateg	50056733

Logistic Regression:

Logistic Regression is a supervised learning classification technique used for classifying data. In traditional way, logistic regression is used for binary classification. We have implemented 2 regression techniques for classifying handwritten (0-9). Data set comprised of 50000 training samples and 10000 validation and testing samples each.

1. Binary Logistic regression
2. Multiclass logistic regression

1. Binary Logistic Regression (BLR)

BLR is implemented using 2 techniques namely; Gradient descent and Newton Raphson Iterative Method. For both approaches, we have built 10 classifiers, one for each digit (0-9).

(A) Logistic Regression using Gradient Descent:

Each individual classifier is trained for 200 iterations.

Entropy error is calculated by

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

Error Gradient is calculated by

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

We have used *fmincg* to compute gradient descent and hence calculate optimal weight.

(B) Logistic Regression using Newton Raphson:

Each individual classifier is trained for 5 iterations.

Weights are updated by

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}).$$

Where,

H → Hessian Matrix

Delta E → Error gradient.

Error Gradient is calculated by

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi_n = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{y}-\mathbf{t})$$

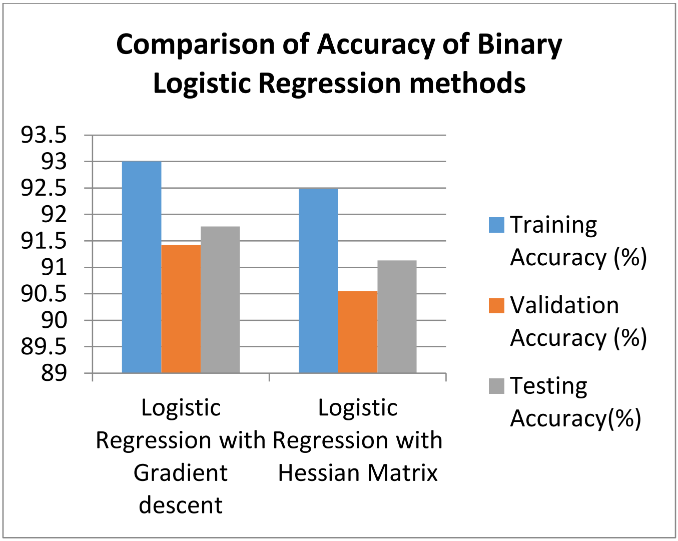
Hessian Matrix is calculated by

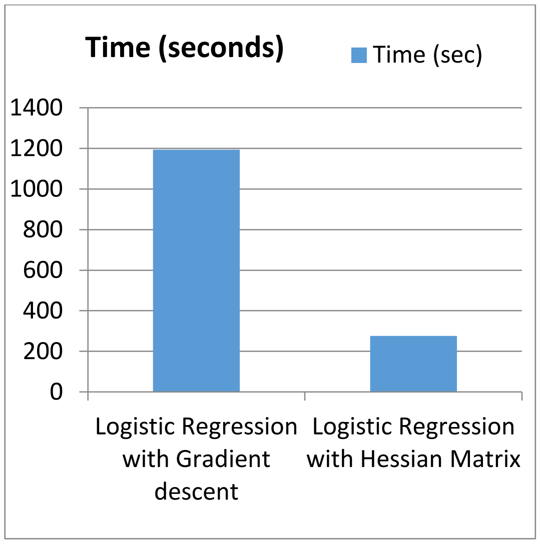
$$\mathbf{H} \;=\; \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N} y_n(1-y_n)\phi_n\phi_n^{\mathrm{T}} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{R}\mathbf{\Phi}$$

Observations for BLR:

BLR with Newton Raphson converges faster than Logistic Regression with Gradient Descent. The accuracy obtained with both the techniques is almost same.

	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)	Time (sec)
Logistic Regression with Gradient descent	93	91.42	91.77	1193.944
Logistic Regression with Hessian Matrix	92.48	90.55	91.13	275.9526





Multiclass Logistic Regression (MLR)

In MLR, we have implemented a single classifier that classifies 10 digits at the same time. MLR is also implemented using 2 methods, Gradient descent and Newton Raphson Iterative method. For both the approaches, we have used single classifier.

(A) Multiclass Logistic Regression using Gradient Descent:

Data set comprised of 50000 training samples and 10000 validation and testing samples each. The classifier is trained for 200 iterations.

Entropy error is calculated by

$$E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

Error Gradient is calculated by

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\, \boldsymbol{\phi}_n$$

We have used *fmincg* to compute gradient descent and hence calculate optimal weight.

(B) Multiclass Logistic Regression using Newton Raphson:

Data set comprised of 2000 training samples and 1000 validation and testing samples each. The classifier is trained for 5 iterations.

Weights are updated by using

$$\mathbf{w}^{(\mathrm{new})} = \mathbf{w}^{(\mathrm{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}).$$

Where,

H → Hessian Matrix

Delta E → Error gradient.

Error Gradient is calculated by using

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\, \boldsymbol{\phi}_n$$

Hessian Matrix is calculated by using

$$\nabla_{\mathbf{w}_k}\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N} y_{nk}(I_{kj} - y_{nj})\boldsymbol{\phi}_n\boldsymbol{\phi}_n^{\mathrm{T}}.$$

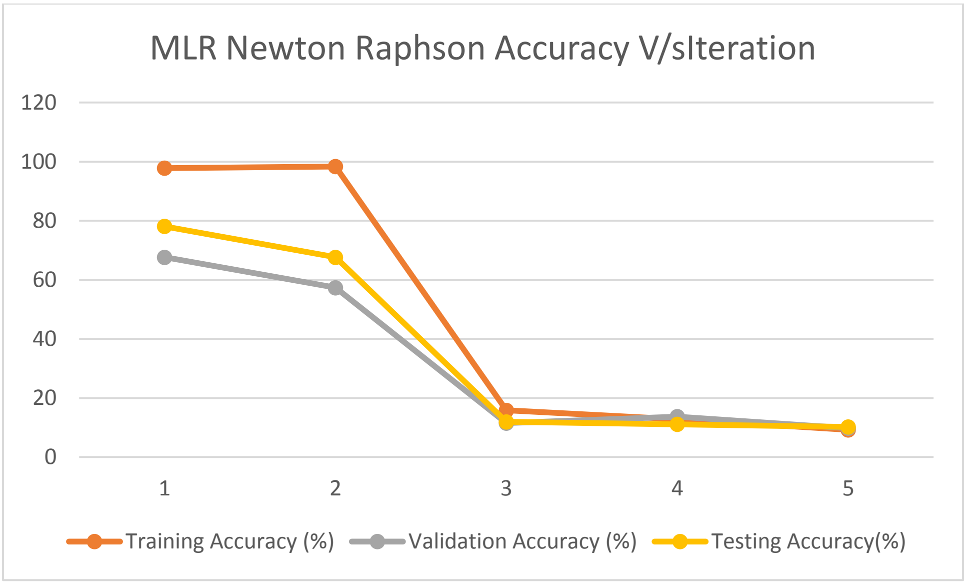
This will give us a single block of (D+1)*(D+1). As we are now classifying all different classes at a single point of time, the final hessian matrix will consists of 100 such blocks.

Observation for MLR

- ***In Newton Raphson method, we need to calculate inverse of final hessian matrix to update weights. As the final hessian matrix is of ((D+1)*10) by ((D+1)*10) i.e. 7160*7160, calculating its inverse is very time consuming. Hence, multiclass logistic regression with gradient descent converges faster as well as computationally less intensive than multiclass Newton Raphson.***
- ***As MLR with Newton Raphson method has been trained with only 2000 training samples, although the training accuracy is good, the validation and testing accuracy is comparatively less than that observed for MLR with Gradient Descent***.

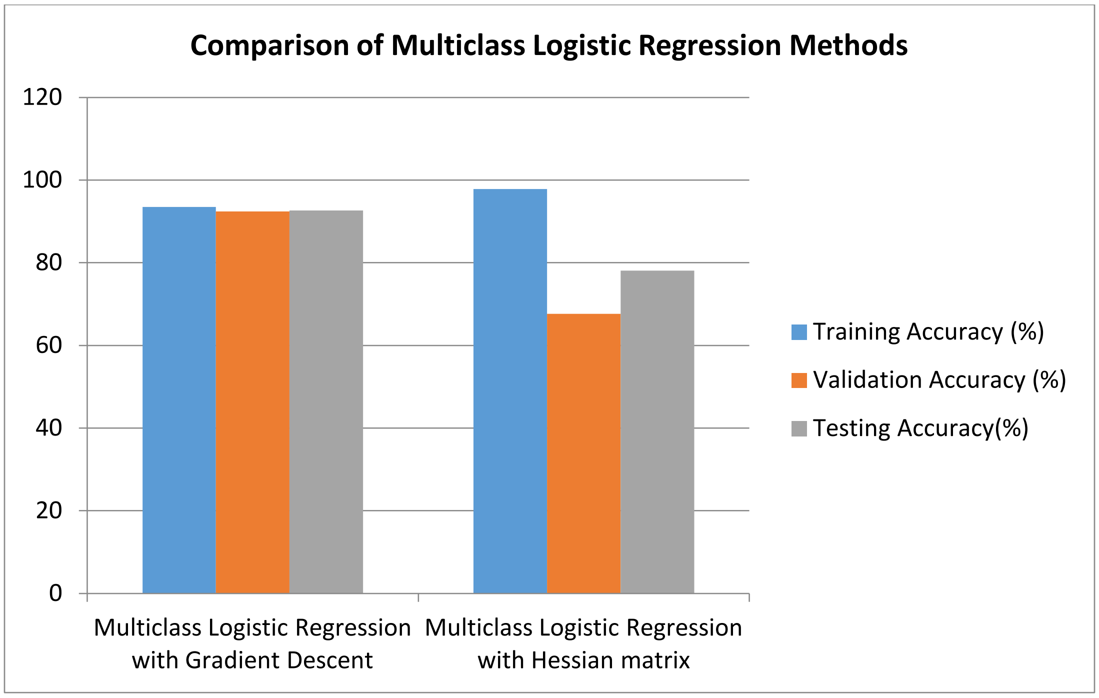
Here is the reading for Newton Raphson method

Iteration	**Training Accuracy (%)**	**Validation Accuracy (%)**	**Testing Accuracy (%)**
1	97.8	67.6	78.1
2	98.35	57.3	67.6
3	15.8	11.5	11.9
4	12.8	13.7	11.1
5	9.2	9.8	10.2



	Training Accuracy (%)	**Validation Accuracy (%)**	**Testing Accuracy (%)**	**Time**
Multiclass Logistic Regression with Gradient Descent	93.514	92.37	92.66	170.107
Multiclass Logistic Regression with Hessian matrix	97.8	67.6	78.1	8161.48

***Note: For Newton Raphson Method, the reading mentioned above is for 1st iteration only (after which it diverges giving very low accuracy).**



Conclusion: From all the above readings and observation, we conclude that Multi Class Logistic Regression using Gradient Descent approach outperforms all other regression methods in the given settings on the basis of Accuracy (high) and execution time (low).

	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy(%)	Time (sec)
Logistic Regression with Gradient descent	93	91.42	91.77	1193.944
Logistic Regression with Hessian Matrix	92.48	90.55	91.13	275.9526
Multiclass Logistic Regression with Gradient Descent	**93.514**	**92.37**	**92.66**	**170.107**
Multiclass Logistic Regression with Hessian matrix	97.8	67.6	78.1	8161.48

SVM

SVM was done using a data set consisting of 10000 training samples and validation and testing was performed on 2000 samples each. The *svmtrain* function was used to train the data and create a model. This model was used to predict digits on validation and testing data using the *svmpredict* function.

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

   Options used:

   ```
   model_linear = svmtrain (train_label , train_data , '-t 0') ;
   ```

   The accuracy obtained is as below:

   Training Accuracy = 99.84% (9984/10000) (classification)
   Validation Accuracy = 91.15% (1823/2000) (classification)
   Testing Accuracy = 90.85% (1817/2000) (classification)

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

   Options used:

   ```
   model_rbf_1 = svmtrain (train_label , train_data , '-t 2 -g 1') ;
   ```

   The accuracy obtained is as below:

   Training Accuracy = 100% (10000/10000) (classification)
   Validation Accuracy = 10% (200/2000) (classification)
   Testing Accuracy = 15.75% (315/2000) (classification)

   As we can observe, setting these as parameters causes high over fitting for training data. Increasing gamma increases the flexibility of decision boundary and thus high values of gamma cause over fitting. (gamma = 1)

3. **Using radial basis function with value of gamma setting to default (all other parameters are kept default)**
   Options used:

   ```
   model_rbf_default = svmtrain (train_label , train_data , '-t 2') ;
   ```
   The accuracy obtained is as below:

   Training Accuracy = 93.66% (9366/10000) (classification)
   Validation Accuracy = 90.35% (1807/2000) (classification)
   Testing Accuracy = 90.2% (1804/2000) (classification)

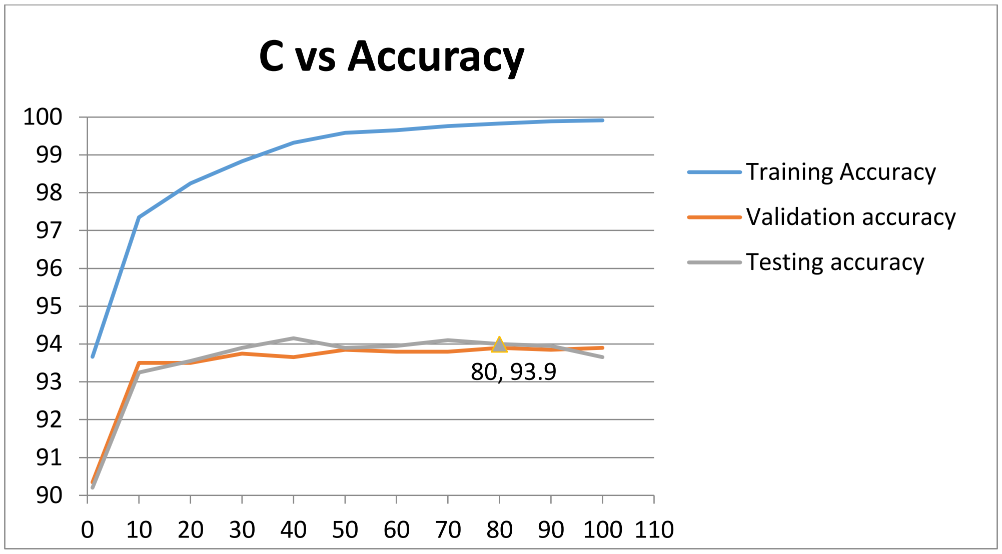
   As gamma is set to default, that is (1/716) we observe that there is no over fitting of training data anymore.

4. **Using radial basis function with value of gamma setting to default and varying value of C (1;10;20;30; ..;100)**

```
model_rbf_C = svmtrain (train_label , train_data , '-t 2 -c 80') ;
```

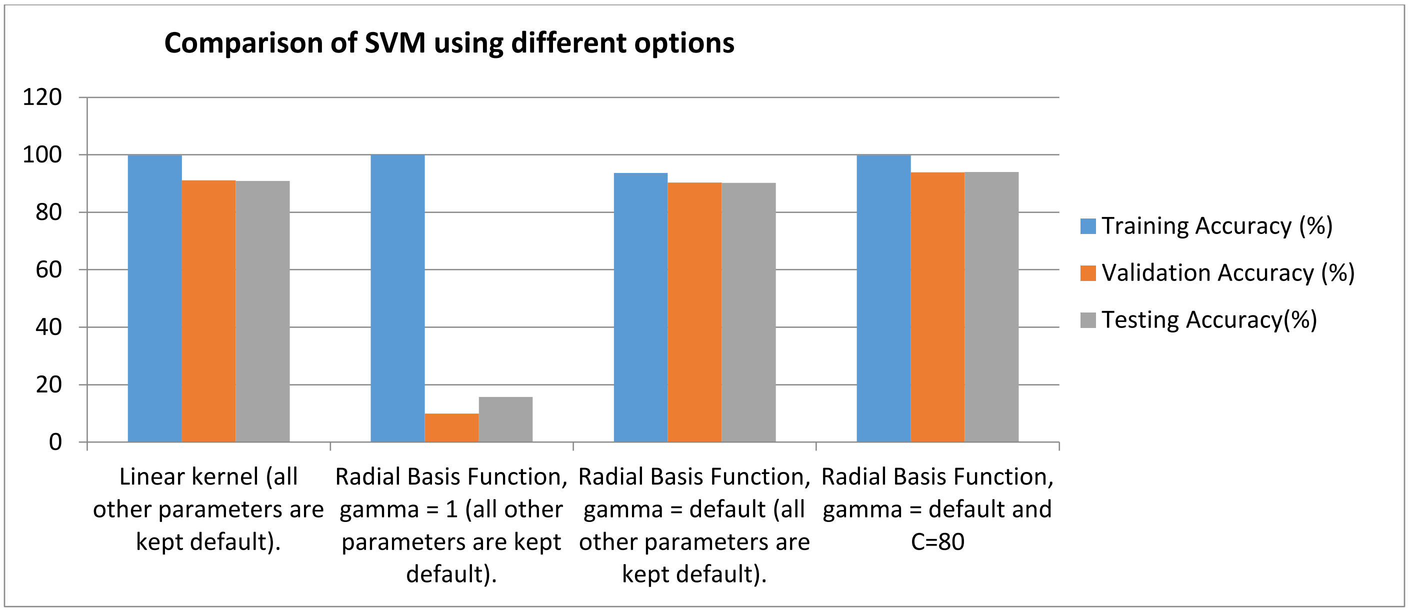
- The C parameter essentially acts like a regularization parameter which controls how much outliers are taken into account in calculating Support Vectors.
- As we increase the value of C, the training accuracy increases. But if we increase C too much we risk losing the generalization properties of the classifier, because it will try to fit all the training points (including the noisy data in the dataset)
- The optimum value of C is the one which gives highest validation accuracy. If we get the same validation accuracy for more than 2 values of C, then we consider the one with lowest C, as we have seen that C should be kept as low as possible. Also higher value of C increases the running time, due to which a lower value of C with good validation accuracy is preferred.

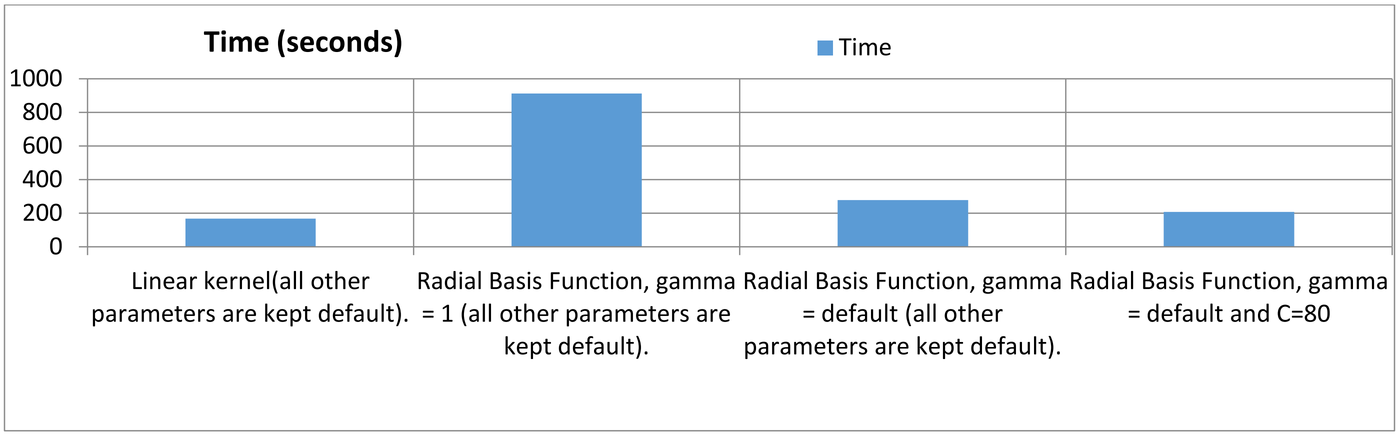
C	Training Accuracy %	Validation accuracy %	Testing accuracy %
1	93.66	90.35	90.2
10	97.35	93.5	93.25
20	98.25	93.5	93.55
30	98.83	93.75	93.9
40	99.32	93.65	94.15
50	99.58	93.85	93.9
60	99.65	93.8	93.95
70	99.76	93.8	94.1
80	99.83	93.9	94
90	99.89	93.85	93.95
100	99.91	93.9	93.65



Comparison of SVM methods

	Training Accuracy (%)	**Validation Accuracy (%)**	**Testing Accuracy (%)**	**Time**
Linear model (all other parameters are kept default).	99.84	91.15	90.85	167.53904
Radial Basis Function, gamma = 1 (all other parameters are kept default).	100	10	15.75	911.11045
Radial Basis Function, gamma = default, (all other parameters are kept default).	93.66	90.35	90.2	276.8881
Radial Basis Function, gamma = default and C=80	99.83	93.9	94	206.02875





Conclusion: From all the above readings for different SVM techniques, we conclude that

- Time perspective: Linear model performs in least time
- Accuracy perspective: Radial Basis Function, gamma = default and C=80 gives highest accuracy.

Comparison of Regression and SVM methods

As both the methods are trained on different datasets with different dataset size, we cannot compare both the methods directly on time and accuracy basis.

Still taking validation accuracy readings into consideration, ***Radial Basis Function, gamma = default and C=80*** gives highest accuracy among both the methods and across all the readings.

#Important Note: All the observations, comparison and conclusions are based on validation accuracy.