Logistic Regression &

Support Vector Machine:

Handwritten Digits Classification

AUTHORS:
ANGAD GADRE
HARISH MANGALAMPALLI
RAJARAM RABINDRANATH

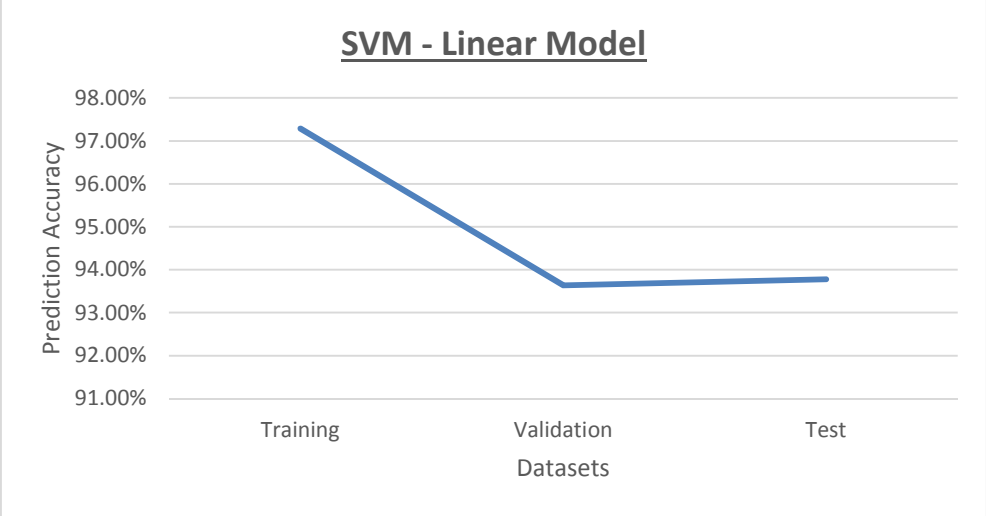
Contents

SUPPORT VECTOR MACHINES

SVM Classifier with Linear Kernel

Using the SVM MATLAB library to implement a Linear Kernel and all other parameters kept default.

Linear Model	
Dataset	Accuracy
Training	97.29%
Validation	93.64%
Test	93.78%



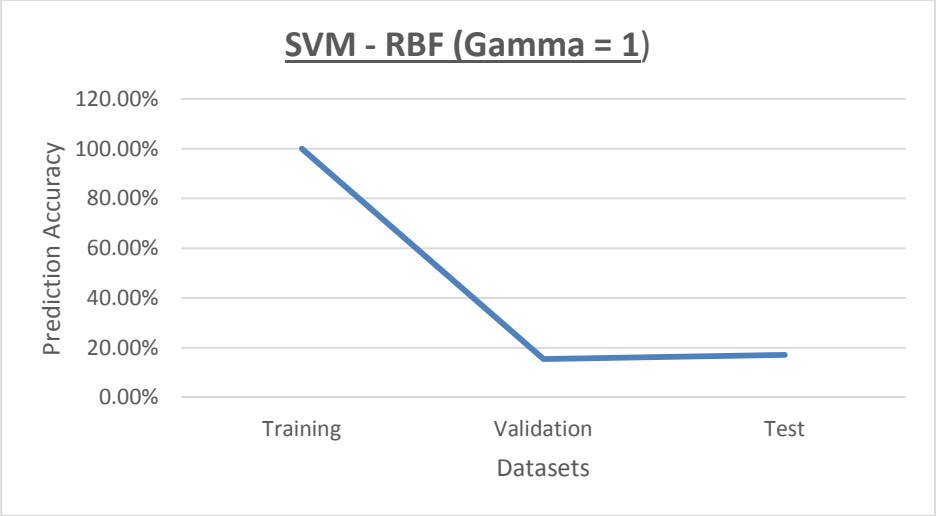
In this experiment we are trying to learn a linear classifier which shall rightly classify the data points as belonging each of the 10 classes. This is the simplest SVM that one could learn.

SVM Classifier with RBF Kernel & γ = 1

If suppose the dataset we are dealing with is not linearly separable we have to learn a non-linear surface to classify our data points, for this purpose we move to a higher dimension space. We use the RBF kernel to do this. RBF kernels map the data points to an infinite dimension space. Here we have to deal with a hyper-parameter Gamma; Gamma parameter is the = 1/(2 variance).

In the first stage choose Gamma to be 1 and try and learn a classifier, following are the results that we get.

RBF Kernel γ = 1	
Dataset	Accuracy
Training	100.00%
Validation	15.48%
Test	17.14%



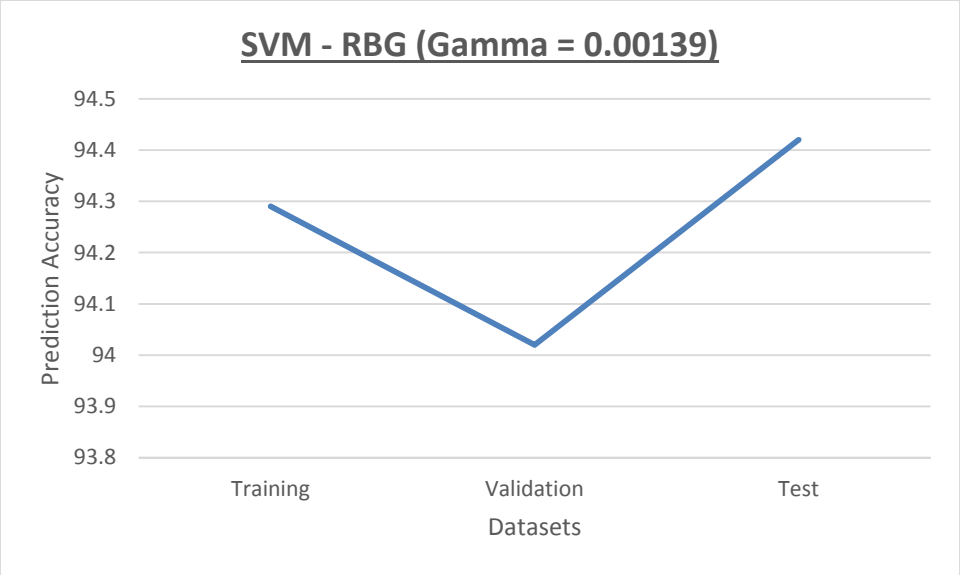
Inferences:

1. We see that our predictions are great with respect to training data but the learned model performs poorly with respect to the validation and test dataset. This is due to over fitting that happens vis-à-vis the training data
    a. Reason being the choice of Gamma – Note as we had mentioned earlier that it is inversely proportional to the Variance. By choosing a large value of Gamma we are making an assumption that the variance is very low. The variance here is the spread of the data points in the training dataset which makes the model learn a very complex surface, for each class we are learning multiple decision boundaries with respect to the training data points. For correctly classifying test data points those points must lie within either of these decision boundaries this is too strict a classification requirement and therefore results in poor accuracies w.r.t test and validation data.

SVM Classifier with RBF Kernel & γ = (1/num_features) = 0.00139

In this experiment we use the default value 'γ', which happens to be 1/num_features. We then proceed to learn the decision surface and use it for predictions. Following are the results that we get.

RBF Kernel Params = Default	
Dataset	Accuracy (%)
Training	94.29
Validation	94.02
Test	94.42



Inferences:

Having decided to use the default value of Gamma, we notice that the prediction accuracies are good for all datasets (Train, Test & Validation). This is purely due to change in the value of Gamma. Following is our understanding:

A very small value of gamma implies high variance, which translates to a smoothed decision boundary. What is happening is that by choosing a very small value of Gamma we are relaxing the strict classification requirements, we set in the previous case, and make it more accommodating due to the higher variance. The decision surface that we learn in this case is a less complex than the one learn in the previous exercise but still non-linear.

Consequently the model performs better when predicting the classes for the test and validation datasets, because the high variance allows for each data point to be assigned to its corresponding right class. But one must be careful about reducing the value of Gamma and consequently increasing the variance. A very high variance shall result it a higher misclassification rate. Given our dataset is very clean we can do better.

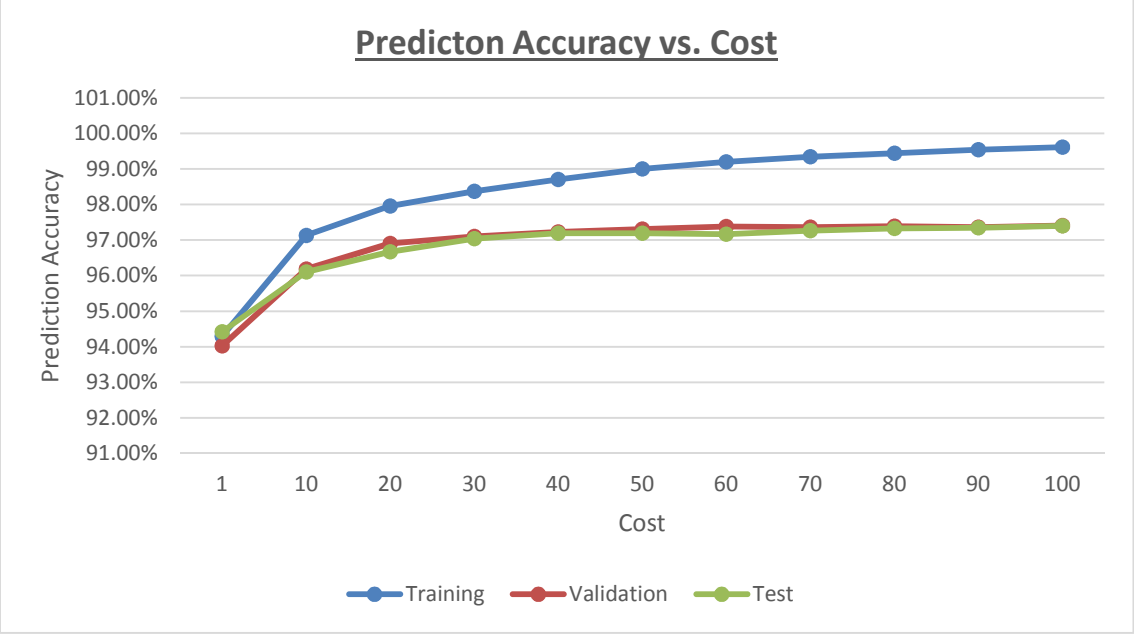
Large Gamma means → Low variance → complex surface (over fitting)

Small Gamma means → High variance → smoothed decision surface

SVM Classifier with RBF Kernel with γ = (1/num_features) = 0.00139 & introducing new Hyper-parameter C

We now proceed to introduce another hyper-parameter 'COST' AKA penalty, which accounts for the slack in soft-SVM. 'COST' is inversely proportional to the Slack which results in wider margins and is analogous to the regularization parameter that we used in other classification techniques.

RBF Kernel γ = Default			
Cost	Accuracy (%)		
	Training	**Validation**	**Test**
1	94.29%	94.02%	94.42%
10	97.13%	96.18%	96.10%
20	97.95%	96.90%	96.67%
30	98.37%	97.10%	97.04%
40	98.71%	97.23%	97.19%
50	99.00%	97.31%	97.19%
60	99.20%	97.38%	97.16%
70	99.34%	97.36%	97.26%
80	99.44%	97.39%	97.33%
90	99.54%	97.36%	97.34%
100	99.61%	97.41%	97.40%



Inferences:

We observe that adding this 'COST' gives us a much better model. And we also notice that for higher values of COST we get better predictions. In the previous exercise we learned that higher variance is better than lower variance (i.e Small Gamma is good), this is however only true to a certain point resulting in greater misclassifications for very small values of Gamma. The 'Cost' parameter helps in mitigating the adverse effect of large variance.

Large 'Cost' means → Large penalty on misclassification → small margin hyper-plane

Small 'Cost' means → More room for error → large margin for hyper-plane

What we have inferred is the impact of Gamma and Cost independently on classification; however it is the combination of gamma and cost that we look for each classification problem/ dataset can have multiple combinations of Gamma and Cost which give similar prediction accuracy.

LOGISTIC REGRESSION:

Binary Logistic Regression – Using gradient descent

Logistic regression: One vs. the others classifier, for learning how to classify a data point given multiple classes K, we shall learn K 1 vs. the other classifiers. We try to minimize the classification error using gradient descent.

***Results**:*

BLR - Gradient Descent	
Dataset	Accuracy
Train	92.91%
Validation	91.47%
Test	91.86%

Binary Logistic Regression – Using Newton-Raphson

Logistic Regression: Given K classes learning K 1 vs. the other classifiers using Newton-raphson method. Here we are using numerical methods to minimize classification error

Results:

BLR - Newton Raphson	
Dataset	Accuracy
Train	92.48%
Validation	90.55%
Test	91.13%

Multi-class Logistic Regression – Using Gradient Descent

Multiclass logistic regression: Learning a single classifier that classifies all the datapoints correctly given K classes to classify.

Results:

MLR - Gradient Descent	
Dataset	Accuracy
Train	93.65%
Validation	92.39%
Test	92.78%

Multi-class Logistic Regression – Using Newton-Raphson

Logistic Regression: Using numerical methods to find the right weight vectors that give us the minimum classification errors.

Results:

MLR - Newton Raphson	
Dataset	Accuracy
Train	86.35%
Validation	84.85%
Test	86.51%

SUMMARY & CONCLUSIONS

PREDICITION ACCURACY					
	Train Accuracy	**Validation Accuracy**	**Test Accuracy**	**Learning Time in sconds**	**Learning Time in Minutes**
BLR - Gradient Descent	92.91%	91.47%	91.86%	2532.3	42.2
BLR - Newton Raphson	92.48%	90.55%	91.13%	318.6563	5.3
MLR - Gradient Descent	93.65%	92.39%	92.78%	415.1094	6.9
MLR - Newton Raphson	86.35%	84.85%	86.51%	2907.8	48.5
SVM	99.61%	97.41%	97.40%	NA	NA

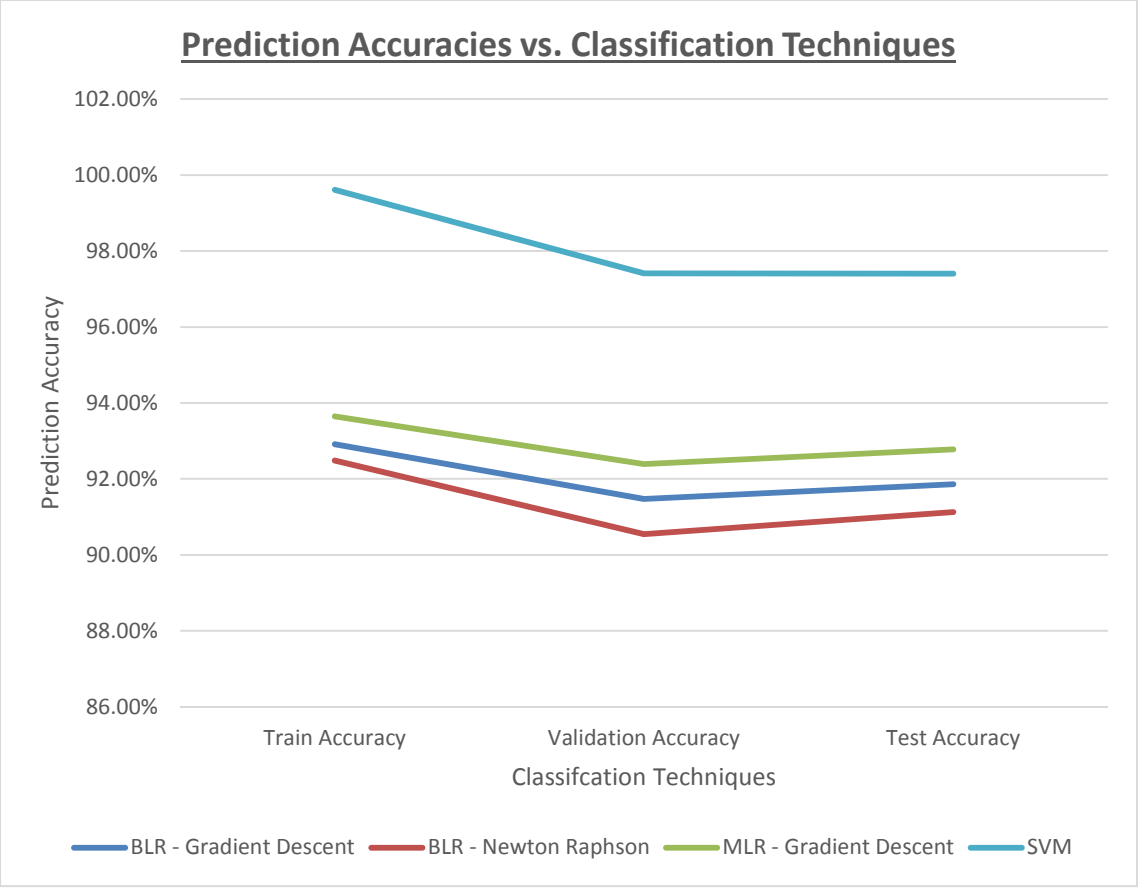
It is evident from the consolidated result set above that SVM outperforms every other classifier in terms of prediction accuracy. However; one must note that the process of learning the model requires one to find the right values for the hyper-parameters, this does take quite some time. In all of the classification techniques we relied on the error due to misclassification as a compass for finding the right model.

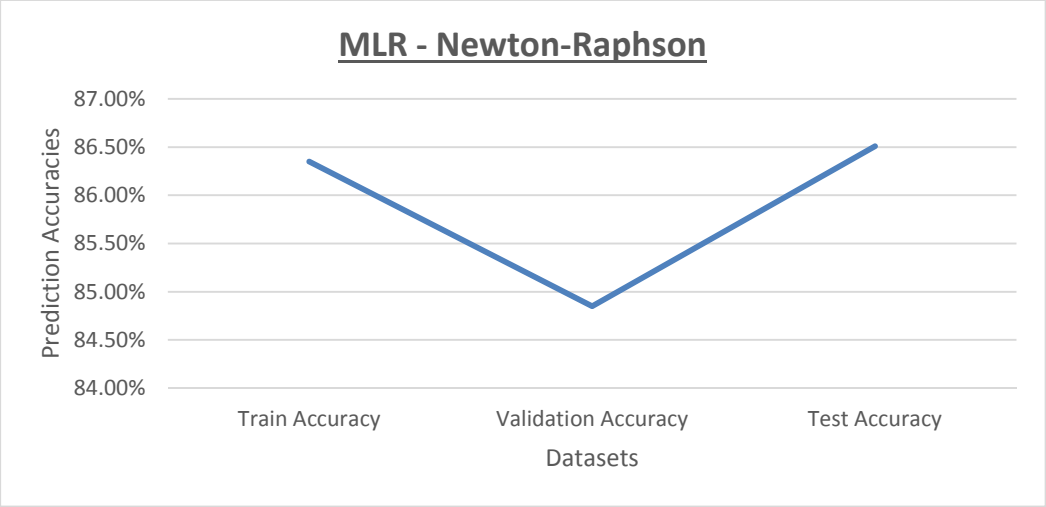
- Logistics regression (BLR & MLR) – we minimized the error by finding the right weight vector
- Support Vector Machines – We find the right model by tuning the hyper-parameters Gamma and Cost, since the error of misclassification is dependent on these factors

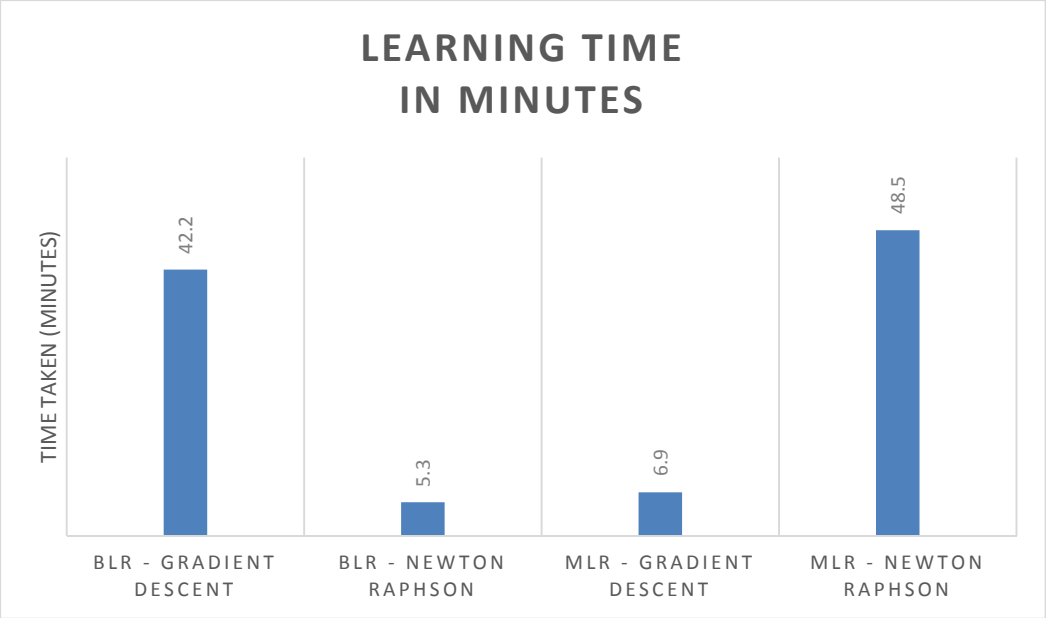
Some inferences that we can draw from these results:

- BLR newton-raphson takes less time compared to BLR – gradient descent but gives comparable prediction accuracies.
- MLR Gradient Descent gives comparable accuracy and only takes as much time as BLR- newton-raphson.
- MLR Gradient descent takes less time than BLR – Gradient descent; the reason being that BLR learn 10 classifiers as opposed to MLR that needs to learn just 1 classifier.
- The MLR – newton-raphson seems to take a long time to learn and does not perform as well as the other techniques in terms of predictions; however we cannot pass judgments on the time taken as we are using for loops in our computations.

If we were to suggest one technique for logistic regression, we shall go with Multi-class logistic regression with gradient descent.




MLR - Newton-Raphson
87.00%
86.50%
86.00%
85.50%
85.00%
84.50%
84.00%
Prediction Accuracies
Train Accuracy
Validation Accuracy
Test Accuracy
Datasets


LEARNING TIME IN MINUTES
TIME TAKEN (MINUTES)
42.2
5.3
6.9
48.5
BLR - GRADIENT DESCENT
BLR - NEWTON RAPHSON
MLR - GRADIENT DESCENT
MLR - NEWTON RAPHSON