CSE 474
INTRO TO MACHINE LEARNING

Classification and Regression

Syed Ali Hasan
Yousef Jaber
Ahmad Temoor

Group Number: 23

Binary Logistic Regression (BLR)

BLR is implemented in *script.py* using the *blrObjFunction()* and *blrPredict()* methods. Following are the results for the **blrPredict()** method:

Training Set Accuracy	86.264%
Validation Set Accuracy	85.47%
Testing Set Accuracy	85.44%

Executing the BLR methods on the given training data, our model was able to accurately classify the labels of the feature vectors, with approximately only **13%** of labels misclassified. After training the model with the training data it was tested with a set of validation and test data, for which the model only misclassified approximately **14%** and **15%** respectively. We can see that after employing the ***one-vs-all strategy*** we were able to build 10 binary classifiers, one for each class, in order to predict the correct labels of all of the feature vectors.

Multi-class Logistic Regression (MLR)

MLR is implemented in *script.py* using the *mlrObjFunction()* and *mlrPredict()* methods. Following are the results for the **mlrPredict()** method:

Training Set Accuracy	93.07%
Validation Set Accuracy	92.37%
Testing Set Accuracy	92.51%

- Through multi-class classification, our model was able to accurately label approximately **93%** of the data it was trained on. The validation and test data was marginally less successful than the training data, classifying the individual data sets with an accuracy of **92.37%** and **92.51%** respectively, nevertheless still more accurate than BLR by approximately **7%**.

- When comparing the results above to those of our **one-vs-all strategy*,*** we observe a clear performance advantage both in speed and accuracy. Using multi-class logistic regression, we were able to classify the data more accurately in approximately **2 minutes**, compared to the runtime of approximately **10 minutes** when using the one-vs-all strategy implemented through BLR.

Support Vector Machines (SVM)

- Using Linear Kernel (all other parameters are kept default)

Training Set Accuracy	97.286%
Validation Set Accuracy	93.64%
Testing Set Accuracy	93.78%

- RBF Kernel with Gamma = 1 and all other parameters default

Training Set Accuracy	100.0%
Validation Set Accuracy	15.48%
Testing Set Accuracy	17.14%

- RBF Kernel with Gamma set to default and all other parameters default

Training Set Accuracy	94.294%
Validation Set Accuracy	94.02%
Testing Set Accuracy	94.42%

- RBF Kernel with Gamma set to default and C = 1.0 and all other parameters default

Training Set Accuracy	94.294%
Validation Set Accuracy	94.02%
Testing Set Accuracy	94.42%

- RBF Kernel with Gamma set to default and C = 10.0 and all other parameters default

Training Set Accuracy	97.132%
Validation Set Accuracy	96.18%
Testing Set Accuracy	96.1%

- RBF Kernel with Gamma set to default and C = 20.0 and all other parameters default

Training Set Accuracy	97.952%
Validation Set Accuracy	96.9%
Testing Set Accuracy	96.67%

- RBF Kernel with Gamma set to default and C = 30.0 and all other parameters default

Training Set Accuracy	98.372%
Validation Set Accuracy	97.1%
Testing Set Accuracy	97.04%

- RBF Kernel with Gamma set to default and C = 40.0 and all other parameters default

Training Set Accuracy	98.706%
Validation Set Accuracy	97.23%
Testing Set Accuracy	97.19%

- RBF Kernel with Gamma set to default and C = 50.0 and all other parameters default

Training Set Accuracy	99.002%
Validation Set Accuracy	97.31%
Testing Set Accuracy	97.19%

- RBF Kernel with Gamma set to default and C = 60.0 and all other parameters default

Training Set Accuracy	99.196%
Validation Set Accuracy	97.38%
Testing Set Accuracy	97.16%

- RBF Kernel with Gamma set to default and C = 70.0 and all other parameters default

Training Set Accuracy	99.34%
Validation Set Accuracy	97.36%
Testing Set Accuracy	97.26%

- RBF Kernel with Gamma set to default and C = 80.0 and all other parameters default

Training Set Accuracy	99.438%
Validation Set Accuracy	97.39%
Testing Set Accuracy	97.33%

- RBF Kernel with Gamma set to default and C = 90.0 and all other parameters default

Training Set Accuracy	99.542%
Validation Set Accuracy	97.36%
Testing Set Accuracy	97.34%

- RBF Kernel with Gamma set to default and C = 100.0 and all other parameters default

Training Set Accuracy	99.612%
Validation Set Accuracy	97.41%
Testing Set Accuracy	97.40%

Observations Derived from Hyperparameters

Using a linear kernel function (with all other parameters kept default) assumes that the data is linearly separable. Having the first SVM kernel set to linear, we observe that the accuracy is less than the accuracy of the SVM kernel set to RBF. Hence, we derive that the data isn't perfectly separable by a line.

Changing the value of the gamma parameter alters the variance of the Gaussian. When the gamma value is set to 1, the variance is lower, thus the SVM performs poorly when classifying the test and validation data. By comparison, the default gamma value is always less than 1 and variance is higher. This is because the default gamma value is **1/n**, where n is the number of features[1]. The default gamma value provides a much higher accuracy in classification:

- RBF with gamma = **1**
    - **15.48%** on validation data
    - **17.14%** on test data

- RBF with gamma = **default**
    - **94.42%** on validation data
    - **94.02%** on test data

[1] reference: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

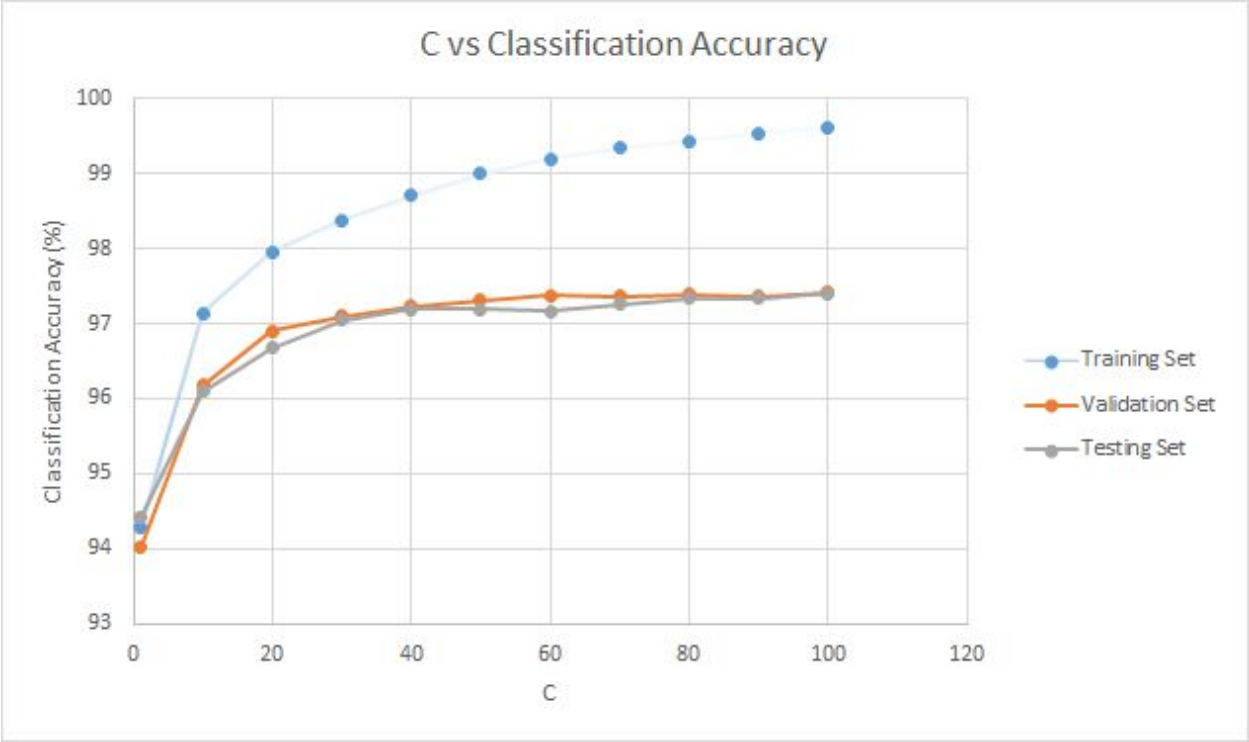

Figure 1: Classification Accuracy vs C for all data sets

- Figure 1 shows a plot of 11 SVMs using a radial basis function with value of gamma setting to default and varying value of **c** where c ϵ C={1, 10, 20, 30, $\cdots$, 100} against the training, validation and testing accuracy.

By incrementing the value of **c**, the soft margin cost function parameter, we observe an increase in classification accuracy. This can be seen in the plot above when comparing accuracy on varying values of c. For example, comparing **c=1** and **c=10**:

- we observe an increase in accuracy of **5.318%** on the training data, and approximately **3.4%** for the test and validation data.

This is due to the fact that altering **c** allows for a change in the margin size of the hyperplane which in effect varies the influence of individual support vectors to obtain a *better fit*. As our value for **c** increases, the percentage of misclassifications decreased since a higher **c** value prioritizes correct classification to a larger margin.[2]

[2] reference: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

From the results obtained above, it is evident that in all RBF cases except where gamma is set to **1**, the SVMs using radial basis function kernels outperform the linear kernel in terms of classification accuracy. Nevertheless, the linear kernel was able to execute within a smaller and thus more efficient runtime. This is due to the fact that the data is not linearly separable, thus is better off separated by a nonlinear kernel function (Gaussian).