CSE574 Introduction to Machine Learning

Programming Assignment 3

Classification and Regression

Team-60

Himal Dwarakanath (himaldwa)
Manish Kasireddy (manishka)
MuthuPalaniappan Karuppayya (muthupal)

Logistic Regression:

Logistic Regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome.

Binary Logistic Regression is used for classification of data into one of the two binary outcomes. The MNIST dataset consists of ten output classes (i.e. numbers 0 to 9) into which the input data is classified. Thus, to find the class of a given input, we apply binary logitic regression to the training data in ten iterations, where each iteration results in a set of weights trained for one of the ten classes. We then used these trained weights for predicting the class of an image, in such a way that the class resulting in maximum posterior probability is chosen as the predicted class for that image.

The training of weights is done using gradient descent which aims to minimize the generated error.

The below formula is used for calculating error in the training phase:

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

The error gradiance is evaluated using the below formula:

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

Accuracies obtained:

Training set Accuracy: 86.21%
Validation set Accuracy: 85.32%
Testing set Accuracy: 85.27%

Direct Multi-class Logistic Regression is a variant of Logistic Regression where we train a single classifier for all the 10 classes at one go instead of using 10 separate classifiers. The posterior probabilities are given by a softmax transformation of linear functions of the feature variables:

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

The error for the k(=10) classes is calculated as follows:

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

The below formula is used for evaluating the gradiance:

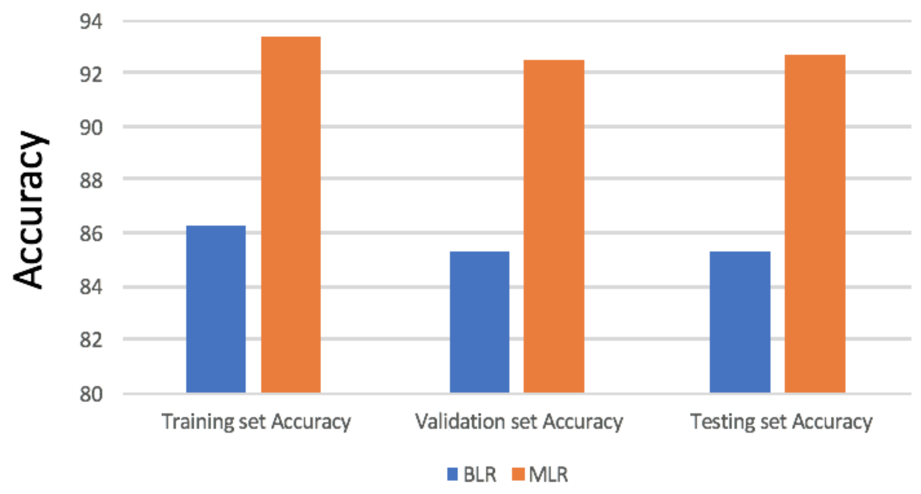
$$\frac{\partial E(\mathbf{w}_1, \cdots, \mathbf{w}_K)}{\partial \mathbf{w}_k} = \sum_{n=1}^{N}(\theta_{nk} - y_{nk})\mathbf{x}_n$$

Accuracies obtained:

Training set Accuracy: 93.39%
Validation set Accuracy: 92.43%
Testing set Accuracy: 92.67%

Comparison of Binary and Multi-class Logistic Regression:

We notice that we get a higher accuracy while using multi-class logistic regression as compared to binary logistic regression. In general separate logistic models tend to have larger standard errors. The multi-class logistic regression can be expected to perform at least as good as the binomial logistic regression as it consists of a set of binary logistic models. Binary logistic Regression can be termed as a "one-vs-rest" approach while multi-class logistic Regression can be termed as a "one-vs-all" approach, this implies that better accuracy is obtained by the one-vs-all approach.



Support Vector Machines:

SVM constructs a hyper-plane in a high-dimensional space, used for classification or regression. A good separation is achieved for a hyper plane that has the largest distance to the nearest training data point of any class; since larger the margin, the lower the generalization error of the classifier. The implementation learns the SVM model and makes predictions with respective accuracies for training, validation and test data based on following parameters:

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

b. Using radial basis function with gamma set to 1 and other parameters set to default.

c. Using radial basis function with all parameters set to default values and C = 1.

d. Using radial basis function with gamma set to default and value of C varying uniformly from 10 to 100.

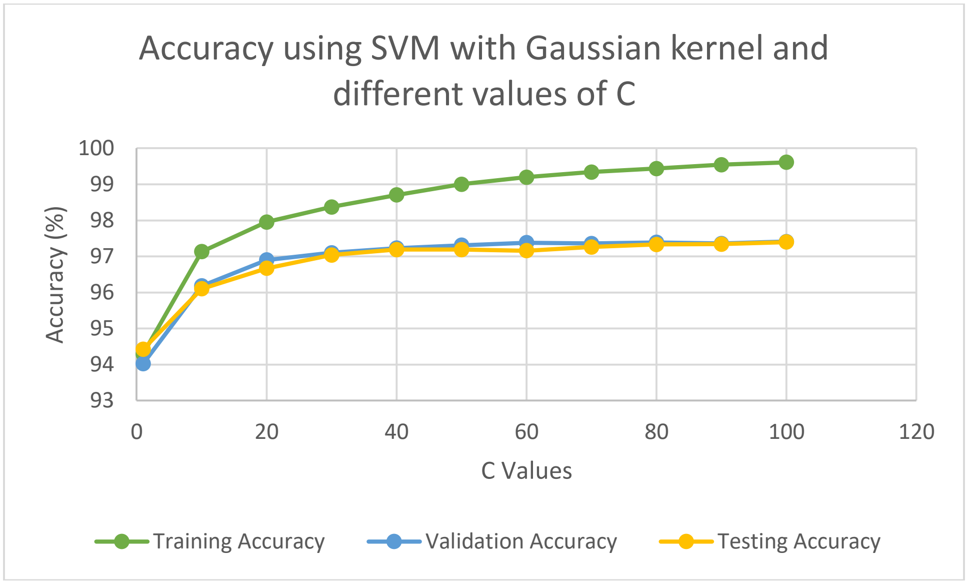
Linear SVM:

Training set Accuracy:97.286%
Validation set Accuracy:93.64%
Testing set Accuracy:93.78%

The accuracies for above analysis were observed as follows:

	Kernel = ‘rbf’											
	γ=1	γ = default										
	C=1	C=1	C=10	C=20	C=30	C=40	C=50	C=60	C=70	C=80	C=90	C=100
Training	100%	94.29%	97.13%	97.95%	98.37%	98.706%	99.002%	99.19%	99.34%	99.43%	99.54%	99.61%
Validation	15.48%	94.02%	96.18%	96.9%	97.1%	97.23%	97.31%	97.38%	97.36%	97.39%	97.36%	97.41%
Testing	17.14%	94.42%	96.1%	96.67%	97.04%	97.19%	97.19%	97.16%	97.26%	97.33%	97.34%	97.4%

SVM gives higher accuracy when using a Gaussian kernel, that maps the input features into an infinite dimensional space. In most cases, a non-linear kernel works better than a linear one, despite a longer training time. The parameter C determines the impact of the error on the training examples, and as a consequence, controls the complexity of the learned hyperplane: a lower C gives a larger marginal hyperplane, while higher C value give a smaller marginal.



As it can be seen from the plot, we have a higher accuracy for higher values of C. That is because C controls the penalty for the error term on each training example.

When C is low, the weight of each error term is low as well: therefore, a larger error value is accepted during the training phase. Consequently, a larger margin hyperplane is created in the expense of having more samples misclassified. Conversely, when C increases, the weight of each error term increases as well, so lower error values will be accepted. As a result, a smaller margin hyperplane will be built, but less number of points will be misclassified, so the accuracy of data classification increases. Based on this explanation, there is a high risk of overfitting for larger values of C. It is not evident from the plot, that shows increasing accuracy for all 3 data sets. However, we can see how the accuracy on the training set increases considerably when C grows, while the accuracy on validation and test set saturates pretty fast (from 40 to 60 it actually decreases for the test set) and this is a clear sign that increasing the complexity of the hyperplane may give overfitting.

Comparison of linear kernel vs RBF:
It is observed that higher accuracy for test data is obtained when we use RBF as compared to the linear kernel. We may give two supporting reasons for this result:

1. In general real-time data fits into a Gaussian distribution, since RBF follows a Gaussian kernel, it proves to give better accuracy.
2. Another reason why RBF performs better is because in our dataset we have the amount of data much greater than the number of features. Gaussian kernels give performance gains in such scenarios.