Programming Assignment 1 : Part A

EE15B025 : Ganga Meghanath
August 31, 2017

1 SYNTHETIC DATASET CREATION

A synthetic data set for the classification task is to be generated using two classes with 20 features each. Each class is given by a multivariate Gaussian distribution, with both classes sharing the same covariance matrix that has been generated randomly and multiplied with it's transpose to make it positive semidefinite. Hence the covariance matrix is not spherical, i.e., that it is not a diagonal matrix, with all the diagonal entries being the same. ie, For a randomly generated matrix A,

$$Covariance\ Matrix = (A)^T(A) \tag{1.1}$$

A mean vector is generated at random.Then we define a standard deviation vector by taking the square root of the diagonal elements of the covariance matrix (from the idea of correlation). This vector is then scaled and added and subtracted from the mean vector to get the mean vectors for the 2 multivariate gaussian distributions. The scaling factor is chosen such that the centroids for the classes are close enough such that there is some overlap in the classes. ie, For a randomly generated vector $\vec{m}$,

$$\begin{aligned} Deviation\ Vector &= \sqrt{diagonal(A)} \\ Mean\ Vector\ 1 &= \vec{m} + k * Deviation\ Vector \\ Mean\ Vector\ 2 &= \vec{m} - k * Deviation\ Vector \end{aligned} \tag{1.2}$$

k=0.65 has been used in the code.

2000 examples for each class is generated. 30% of each class (i.e., 600 data points per class) is randomly picked as test set, and remaining 70% of each class (i.e., 1400 data points per class) as training set and are stored as DS1-test.csv and DS1-train.csv. It is to be noted that the data

stored will have 22 dimensions, the last two dimensions corresponding to Y, which has been one hot encoded. This ensures that we can identify the classes even after mixing the 4000 samples together and shuffling it.

2 Linear Classification

The DS1-train.csv and DS1-test.csv files are read and the data seperated into X_{train}, Y_{train}, X_{test} and Y_{test}. A vector of ones is appended onto X_{train} and X_{test} for accounting for the bias term. And Y labels obtained are one-hot encoded. The coefficients and the perdiction are obtained using inbuilt functions in sklearn. Here, in the predicted labels, the class having highest score is chosen as the predicted class(column index) for each test case. Since it is a binary classifier, the comparison has been done with the second column of Y_{test}. Inbuilt functions are used to compute accuracy, precision, recall and F-measure.

2.1 Results Obtained

The results obtained are as follows :

1. Recall : 0.955
2. Accuracy : 0.9575
3. Precision : 0.939108040201
4. F-measure : 0.957393483709

The weights obtained are as follows :

[[0. 0.55842292 0.43702935 0.59744126 -0.97545007 -0.90787154 -0.2093079 0.06318583 -0.13522735 -0.55067144 0.70054208 0.14168781 0.12096063 0.46752148 -1.3612886 0.85535028 1.66280607 0.35454283 -1.17030443 -1.50442942 1.96294857] [0. -0.55842292 -0.43702935 -0.59744126 0.97545007 0.90787154 0.2093079 -0.06318583 0.13522735 0.55067144 -0.70054208 -0.14168781 -0.12096063 -0.46752148 1.3612886 -0.85535028 -1.66280607 -0.35454283 1.17030443 1.50442942 -1.96294857]]

3 k-NN classifier

The DS1-train.csv and DS1-test.csv files are read and the data seperated into X_{train}, Y_{train}, X_{test} and Y_{test}. A vector of ones is appended onto X_{train} and X_{test} for accounting for the bias term. And Y labels obtained are one-hot encoded. In-built functions in sklearn are utilised for obtaining the prediction.Here, in the predicted labels, the class having highest score is chosen as the predicted class(column index) for each test case. Since it is a binary classifier, the comparison has been done with the second column of Y_{test}. The algorithm is run for various values of 'k' (number of nearest neighbours using an iterative loop. The accuracy, precision, recall and F-measure corresponding to each value of 'k' are written onto "Results.txt" file.

3.1 Results Obtained

The best results obtained are as follows :

1. Best Accuracy = 0.784166666667 for k = 29
2. Best Precision = 0.721902800659 for k = 29
3. Best Recall = 0.808333333333 for k = 64
4. Best F-measure = 0.788273615635 for k = 72

3.2 Analysis

As we can see from the "Results.txt" file and the acquired best results, the k-NN classifier performs much worser compared to the linear classifier for the generated dataset. Although we can't generalise that Linear Regression perfroms much better than k-NN, the first being paramentric and the latter being non-parametric, and that as the dimensionality increases, the nearest neighbours might actually be far apart, in the particular dataset that we have generated, it so turns out that Linear Regression outperforms k-NN classifier as we can observe from the acquired results.

The k-NN classifier performance changes with different values of 'k'. The values of 'k' giving the best performance has been shown above. We can see this from the "Results.txt" file as well. For the generated dataset, we can approximately say that the performance increases to a maximum and then oscillates, though it's not a very correct statement to make and our results will depend on the dataset used.

4 Data Imputation

The Communities and Crime (CandC) Data Set from the UCI repository :
(http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime) is used for the regression. This is a real-life data set and it is made usable, by filling in all the missing values. The inbuilt function of sklearn is used to implement the imputation. Attribute mean has been considered during imputation. The first 5 attributes are non-predictive and they are ignored and eliminated. The modified dataset is stored in "CandC_modified.csv".

Note : The results obtained after shuffling the dataset proved to be worser than using the unshuffled dataset. Hence, the unshuffled dataset has been used and shuffling has been omitted in this case

We can also use mode or median for filling in missing data corresponding to each attribute. Mean consists of replacing the missing data for a given variable by the mean of all known values of that variable. I believe a better method would be such that, for each sample, select a set of k-nearest neighbors and then replace the missing data for a given variable by averaging (non-missing) values of its neighbors, since just mean does not make use of the underlying correlation structure of the data and thus might perform poorly. But the median would be a

more robust estimator for data with high magnitude variables which could dominate results (otherwise known as a "long tail")

5 Linear Regression

The "CandC_modified.csv" file is read and the data is split into 5 different 80-20 splits (319/80). Each split is seperated into X_{train}, Y_{train}, X_{test} and Y_{test}. A vector of ones is appended onto X_{train} and X_{test} for accounting for the bias term. The data is fit using Linear Regression and the RSS is estimated, averaged over the 5 different runs and the coefficients learned in each case are stored in seperate text files.

$$Average\ RSS = 2.60444840251 \tag{5.1}$$

6 Regularized Linear Regression

Regularised Linear regression or ridge regression helps in overcoming over fitting. The problem is seperated into two tasks and the run.py file executes two other python files in the same folder. The first file named ridge_lamda.py runs ridge regression on the read datasets for various values of the regularisation parameter λ. The RSS is estimated by averaging over the 5 different runs on the 5 different test data coresponding to each of the 5 training data and the coefficients learned in each case are stored in "Coefficients.txt". The result of the ridge regression are written onto "Ridge_lamda.txt" and the trial experiments have been stored in "ridge_lamda_expt.txt".

$$\begin{aligned} Optimal\ \lambda &= 5.0 \\ Calculated\ RSS &= 1.24972983522 \end{aligned} \tag{6.1}$$

The second python file called "feature_selection.py". It takes the optimal λ from "ridge_lamda.py". The code implements feauture selection using ridge regression for λ. The logic used for extracting the prominant columns is as follows :

$$|Coefficients - mean(Coefficeints)| > k * StandardDeviation(Coefficients) \tag{6.2}$$

The corresponding columns(of which the coefficeints satisfy the above condition) are used for training and fitting using λ. This is done for various values of "k" and the results are stored in "Feature_selection.txt". The other experiments conducted using the same have been stored in "Feature_selection_expt.txt".

$$\begin{aligned} Optimal\ k &= 0.04 \\ Calculated\ RSS &= 1.24915440536 \end{aligned} \tag{6.3}$$

6.1 ANALYSIS

As we can observe from the results, the error reduces slightly on regressing over a smaller number of features. We also see that the RSS for ridge regression is less than half of that obtained during normal Linear Regression. This could be mainly because normal Linear Regression is more prone to overfitting.