Principles of Machine Learning (CS4011)

Programming Assignment 1A

Ameet Deshpande - (CS15B001)

August 30, 2017

1 Question 1 - Synthetic Data Generation

- The question asks us to generate two classes of data points which share the same covariance matrix but have different data centers. The *sklearn* library was used to generate *Multivariate Gaussian* distributions. We need to ensure that there is some overlap between the classes, else the accuracy will be very close to 100%. The following were the steps taken to generate the dataset
  - The covariance matrix that was used was **diagonal**, but not spherical. This is ensured by sampling values of the diagonals from the set $[0, 40]$ and choosing only unique values. The justification for choosing a special case of a diagonal matrix is that, given any data set, if we perform ***Gram-Schmidt* orthogonalization**, we will end up with orthogonal feature vectors. The covariance matrix of these feature vectors will be diagonal as the random variables from which the data is bring drawn will be independent. Thus, any multinormal data can be modelled in such a way that after orthogonalization we end up with a Diagonal Covariance Matrix. We thus justified that our case is general. This also gives us a **clean interpretation** of the data as the features are now independent of each other and we need to find which carries more weight.
  - There is a need to create sufficient overlap between the two classes, else it will be very easy to learn a model on that data. To do this, we firstly use the mean of the first class as the $\overrightarrow{0}$. This is a general case as we can always translate our axes to make the mean of one of them as $\overrightarrow{0}$. To choose the other center, we do the following. We generate a vector called *overlap*, which is of length 20 and we consider the vector $\overrightarrow{D} = diag(covariance\_matrix)$. Thus, $\overrightarrow{D}$ contains the variance of all the random variables in that order. The elements of the *overlap* vector are random numbers sampled between 0.1 and 0.2. We calculate the new center as $center_2 = overlap^T \overrightarrow{D}$. **We do this because, variance determines the spread of the data**. More the variance, further the center can be and still have an overlap. Lesser the variance, more dense the data and thus the centers will have to be closer (in that dimension). It is thus natural to set the center distance along one of the orthogonal feature axis to be proportional to the variance of the random variable associated with that axis to make sure there is some overlap. This concludes the justification of the parameters used for generating the data.
  - The dataset that is generated is vertically stacked, shuffled and split to get the **train-test ratio** of $0.7 - 0.3$. This dataset is further used for questions 2 and 3.

- Since the variance of the data is in the range $[0, 40]$, there is no need for feature scaling, as the order is almost the same. In fact, on average, the difference will be close to 20 and thus feature scaling is not required.

2 Question 2 - Linear Classification

- The dataset generated is Multivariate Gaussian and is thus the best kind of clean data that we can expect a Linear Model can classify.
- Since we are regressing in this case and not classifying, we set the labels of the classes as 0 and 1 and use an indicator variable to denote which class the data sample belongs to. For this, we learn a weight vector w. For any data sample x, if $w^T x \geq threshold$, we classify it as class 1 and otherwise we classify it as class 0. Since the variables take only 0 and 1 as values, using the symmetry argument, we choose $threshold = 0.5$. Thus, the output of the regression is unbounded, but we apply the threshold and make sure that the output is either 0 or 1.
- Given the fact that the classes share the same covariance matrix, we would expect the boundary $w^T x$ to be perpendicular to the line joining the centers. Calculating the angle between the normal of the plane and the line joining the centers, we expect to see an acute angle. (The actual value that was obtained after taking dot product and dividing by norms was $cos^-1(0.8564)$, which is close to 30^o). This result is as expected.
- The coefficients generated give a good idea about the weights of the features. The coefficients are not the same and and lie in $[0.08, 0.16]$. There are thus features that are almost twice as important as others. But the important observation is that no coefficient was very less compared to the other and this is a testament of the fact that the features are orthogonal and driving one of the coefficients to 0 will only harm the accuracy.
- The model used for learning was $sklearn's$ Ordinary Least Squares Linear Regression. The scores were 93.5%, 92.23%, 95% and 93.6% for Accuracy, Precision, Recall and F-score respectively. Since the data classes are not skewed and there are an equal number of examples for both the classes, we can see that both the precision and recall metrics are equally high. This means that our classifier is performing well on both the datasets and it is not the case that it is performing extremely well on one and poorly on the other.

3 Question 3 - k-NN classifier

- The KNN classifier works on similarities of data samples with other data samples that are close to it. Here close is usually defined by a distance similarity like euclidean distance. The KNN classifier is trained on the labelled training data and it classifies the test data by taking a majority vote amongst the k nearest neighbours. Intuitively we expect the accuracy to increase as the value of k (which can be considered as a hyper parameter) increases, reach a maximum and then decrease further.
- The KNN algorithm was run on k values as small as 1 to as large as 2000, which is almost of the order of the total dataset size in this case. To answer the question in the problem statement, for low values of k ($[0, 20]$), the Linear Regression model performed better in terms of accuracy (and others because of the reason given above) score. As k increased, KNN did almost as good as the Linear Regression model, but it becomes computationally expensive.

The maximum accuracy score obtained was for the value of $k = 50$ with an accuracy score of 93.4% as opposed to the 93.5% of the Linear model. Thus, overall, considering the complexity and the accuracy score, we can see that on this dataset, the Linear Model does pretty well. The following table compares the values for few models.

Model	Accuracy	Precision	Recall	F-Score
Linear Model	93.5	92.23	95	93.6
(K=10)KNN	91.58	93.39	89.5	91.4
(K=50)KNN	93.4	92.9	94	93.4

- As explained before, we expect the accuracy values to increase as k increases, reach a maxima and then start decreasing. The value of k for which the best results were obtained was $k = 50$. The graph attached below (Figure 1) shows the trend of accuracy score vs k. This graph captures the behavior.

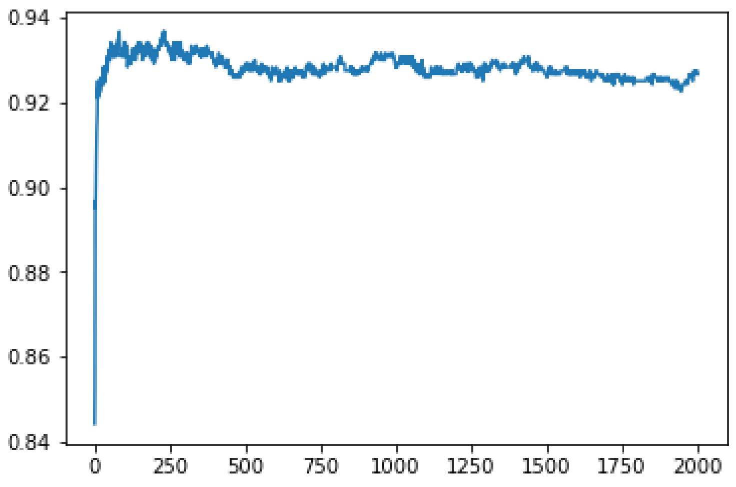

Figure 1: Accuracy v/s k

- The maximum accuracy scores are obtained for $k = 50$ and are 93.4%, 92.9%, 94% and 93.4% for Accuracy, Precision, Recall and F-Score. We can see that a simple Linear Model outperforms this model.

4 Question 4 - Data Imputation

- The dataset given is a real life dataset and thus has many missing values. Firstly, a few comments about the dataset are in order.
  - The dataset has a few nominal variables which cannot directly be used for classification. The first five columns are non-predictive. The missing values in those columns are filled

with 0 to indicate that the values were missing. They are columns like country-code etc, where filling the mean is not a logical choice.

  - A variable called *LemasGangUnitDeploy* is present, which is actually a nominal variable and takes only 0, 1, and 0.5 as its values. It thus doesn't make sense to take the mean and use that to fill the missing values. The mode would be a better choice for this attribute. That being said, since this is just one feature and is not that predictive, the mean is naively used.
  - The missing values all occur in a bunch. In other words, there are only certain columns which have missing values and for rows which have one of them missing, all the other rows in consideration are also missing. This is a clear indication of the fact that there is some kind of dataset merging that has happened and data for some of the columns were not found out.
  - The number of missing values for columns which do have missing values is about 1675 out of the total 1994 fields. This is about 84% of the data missing. This is a suggestion that it would probably not be a bad idea to exclude this feature and train the model. This is because, 84% of the data samples will have this feature's value set to the mean and thus the predictive power of this feature becomes very low. It would almost be as good as not having the feature.

- We still use the mean to fill in the missing data as we do not want to throw away those features. Though mean is not the best choice theoretically, practically it seems like a good choice to use. The following are other methods that could be considered.
  - It is quite possible that the missing values of some of the features are correlated with other features of the same data sample. To illustrate with an example, if there is a feature called $Area$ that has a missing value and we know the value of the $width$, using some correlation rules, we could set the $Area$ to be proportional to $width^2$. This method thus uses horizontal correlation. Note that the correlation could be non-linear and thus the feature vector could still define a new axis (not vanish after orthogonalization).
  - Another method that can be used is the $KNNImputer$. This method finds the most similar points based on feature columns that are not missing for an example and assigns the value of the chosen data point. This method should work better than taking Mean, but unfortunately the function is still under development in $scikit-learn$. It is however implemented in $MATLAB$ and is used to an extent.

5 Question 5 - Linear Regression

- The method used in this problem tries to reduce the randomness of the accuracy score by training on 5 different sets of the same data and then taking the average. The Best Fit Mean Squared error averaged over the 5 runs is 0.0195, and the Residual Sum of Squares error is 7.8164. The average over 5 runs is expected to be more stable than taking the accuracy score on just one run. Since the data is normalized to $[0, 1]$ already, an error of about 0.0195 per sample looks good. A linear model thus seems to be doing well in predicting the final answer.
- An interesting phenomena occurs when the train-test split is being performed. If sufficient runs of the programs are done, the error sometime shoots up to 1 from 0.019. This also happens only on one of the test-train splits. This probably occurs because all the outlier data happens to be in testset for one case. Thus, the predictions become very bad.

6 Ridge Regression

- Ridge regression involves using a λ parameter to control the bias-variance trade off. We initially plot the RSS errors v/s lambda values to check which lambda values are benefitting the model. The plot can be seen below.

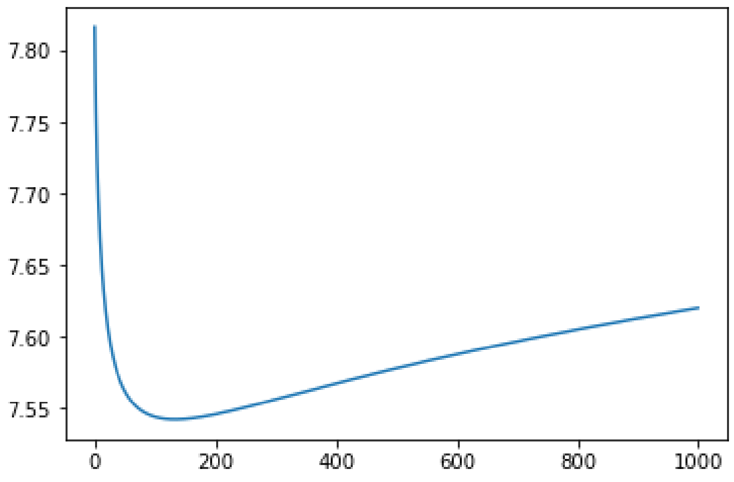

Figure 2: RSS vs Lambda*100

- After seeing this plot, to fine tune the model, lambda values between $[0, 2]$ are considered. The Lambda value of 1.35 gives the least RSS error of 7.54, which is lesser than the 7.8164 which a normal Linear Regression model gives.
- The RSS value is reported in the form of the graph and coefficients are submitted for lambda values in the directory. The behavior of the graph is as expected. The regularization parameter is supposed to rectify the bias-variance tradeoff to an extent, which it does initially. As the value of λ increases, the error decreases. Further increase is expected to cause increase in the error as the bias term takes off. When the regularization parameter is very high, many coefficients are close to 0 and thus the model under fits. The graph displayed explains these characteristics very well and gives the expected bowl shape of the graph.
- Regularization's main aim is to drive a few coefficients towards 0. Ridge regression does not drive the coefficients to 0 but it does reduce the coefficients. So we can use the coefficients generated to check which features are more important and which aren't. There are several ways to do this. Code can be found in $CS15B001\_PA1a\backslash Code\backslash q6\backslash improved\_features.py$.
  - We set a threshold and for coefficient values lesser than threshold, we remove those features. The method of using a common threshold across all features works because the data is normalized and centered around 0.5 and thus, without worrying about the magnitude of the data, we can say that a feature is important if it has more weight.
  - We can use Lasso Regression and check which coefficients are driven to zero. Lasso has the nice property of driving coefficients to 0 unlike Ridge, which drives it only near 0. Lasso will thus give a very clear idea of which features are important and which is not.

  - Another idea is to see how the coefficients compare to normal regression. If the decrease of the coefficient's value is large, there is a high chance that the feature is unimportant.

- After seeing the coefficients file corresponding to the best fit ($\lambda = 1.35$), we can see that most coefficients are in the range 0.2 and 0.01. There are however a few features that have weights lesser than 0.01, which seem unimportant. In general, this threshold can be decided by the maximum number of features that can be thrown away. We usually do not want to throw away more than say 10% of the features. We plot a graph to check which threshold performs well. Below is the graph.

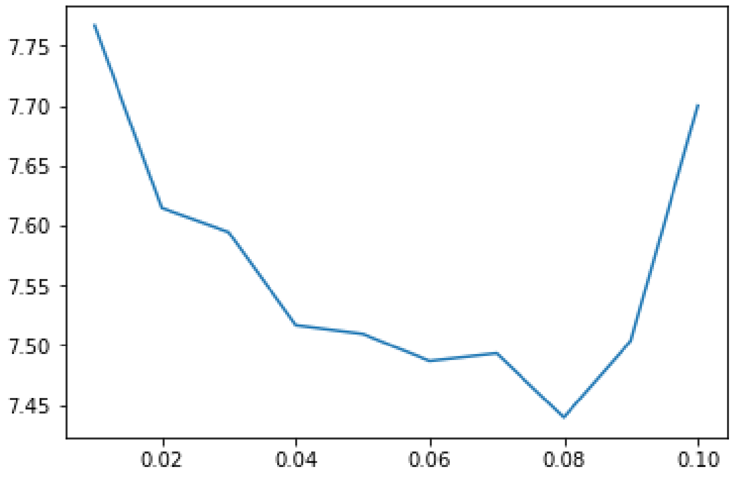

Figure 3: RSS vs Threshold

- As can be seen from the plot, the threshold of 0.08 gives the best results. In this case, a total of 19 features are thrown away. The RSS values were measured for a Linear Regression model without regularization. The RSS was 7.43, which was lesser than both the regularized Linear regression and Normal Regression. Below is a table to show the results.

Model	RSS
Linear Regression	7.8164
Regularized Regression	7.54
Linear Regression with feature removal	7.43

- The table's results are exactly as expected. After feature removal, since the unimportant features have been removed, the model performs the best. Regularized Regression naturally performs better than the Naive Liner model.