CMPS 242 Homework 1

Arghyadeep Giri
argiri@ucsc.edu Student ID: 1566630

Language used:
Python (*https://github.com/Arghyadeep/Machine-learning-UCSC/tree/master/Homework1*)

Data:
The provided data were text files ("train.txt" and "test.txt"). The train data was read, and a Pandas data frame was created to associate the X and y values before shuffling and feeding data for K folds cross validation. The labels of the train data were "1" and "-1" and values in the features were typically "0" and "1". They were all converted to integer values. The train data matrix had a dimension of (123,2000) without the bias term where 2000 is the number of training examples and 123 is the number of training features. The test data for obvious reasons had the same dimensionality but had higher number of examples which was 2781.

Training:
The data was trained on the above-mentioned training data using ScikitLearn's LogisticRegression function. The parameters used were: penalty: "l2", solver: "liblinear", max_iters = "5000", fit_intercept = "True" (this took care of the bias term for which adding additional bias term wasn't required). I did try to vary the parameters to find better results, but these configurations provided the best results on the dataset. The entire dataset was used for training after performing cross validation to find best regularization parameter.

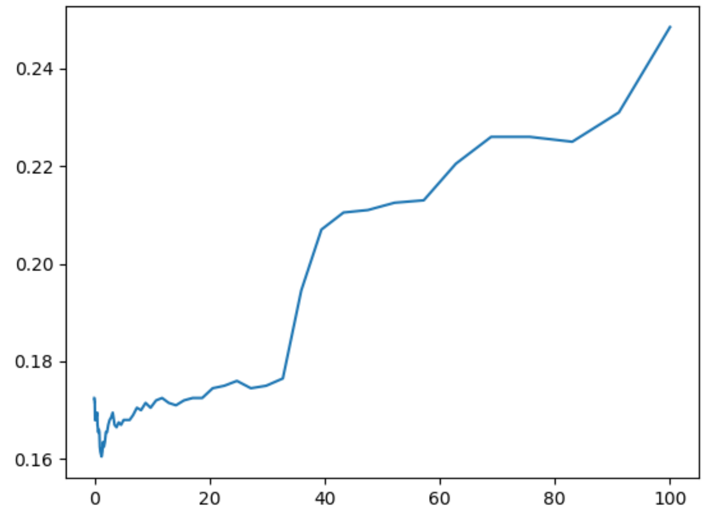
Hyperparameter Tuning:
The only hyperparameter that I had to deal with is the regularization parameter. I performed a k-fold cross validation (with k = 10) on the train data with 1000 lambda values in log space using np.logspace function ranging from -2 to 1 with a base of 10 (i.e. 10^-2 to 10). Initially I tried ranging the values from -5 to 2 with the same setting, but then by trial and error I found out the best results are produced within the lambda values ranging from the above mentioned. Every lambda value in the lambda values was fed in the model for every fold and the average performances on the holdout set was recorded with mean over the saved values. *Log space values were chosen for discretization of lambda values since we want a large variety of lambda values over a wide range to find out the optimal regularization parameter. An even spaced discretization would not have supported that for the coarse search before fine tuning the range*.

Testing:
The test data was read like the train data in similar fashion. The best lambda value was chosen from the hyperparameter tuning for which the mean train error was lowest on the validation sets. Similar settings were used for the final training on the entire dataset as it was for hyperparameter tuning to maintain consistency. The model was saved and was applied on the test data set which was saved in a Data frame of similar fashion to that of the train data. The Predict and score functions were used to find confusion matrix values and accuracy of the classifier respectively.

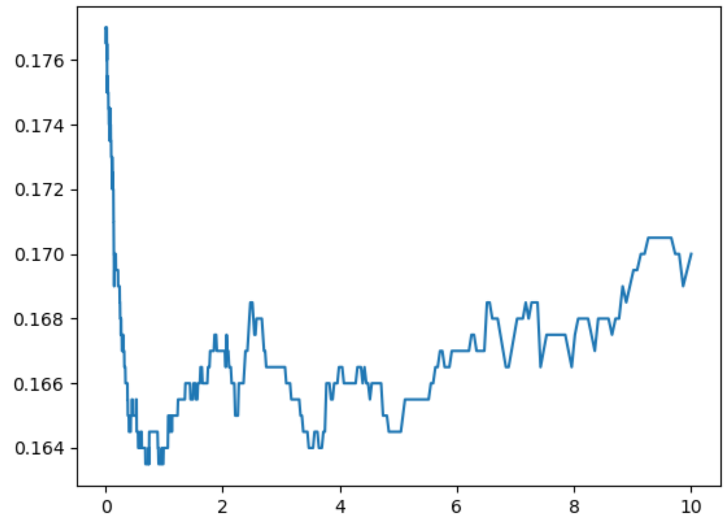
Results:

The Best value of lambda is **1.2618** on 10 folds cross validation.
Mean Accuracy on train set: **84.95** percent and mean error of **15.05** percent.
Confusion Matrix values on test set: TP = 404; TN = 1945; FP = 152; FN = 280
Mean Accuracy on test set: **84.46** percent and error of **15.54** percent.

Plot for values of lambda in range (10^-2 to 10^2) with mean validation error



Plot for values of lambda in range (10^-2 to 10^1) with mean validation error



Extra Credits:

- Bias Term Not Regularized:

This was performed by setting a very high number(10000) to a parameter in the logistic regression function named intercept_scaling. There was no drastic change in not regularizing the bias term. The accuracy and the error both on train and test data were almost the same.

Results:
Mean Accuracy on train set: **84.06** percent and mean error of **15.94** percent.
Confusion Matrix values on test set: TP = 371; TN = 1956; FP = 141; FN = 313
Mean Accuracy on test set: **83.67** percent and error of **16.33** percent.

Justification: Overfitting usually requires the output of the model to be sensitive to the small changes in the input data. The bias parameter does not contribute to the curvature of the model. Hence, there is a little point in regularizing them. So, regularization or no regularization on the bias term hardly makes any difference.

- Implement LOOCV:

Since LOOCV takes up a lot of time to implement the number of lambda values were reduced from 1000 to **100** over the same range.

Results:
Mean Accuracy on train set: **84.06** percent and mean error of **15.94** percent.
Confusion Matrix values on test set: TP = 371; TN = 1956; FP = 141; FN = 313
Mean Accuracy on test set: **83.67** percent and error of **16.33** percent.

Observed advantage:
The only advantage that I could figure out from this method is lack of variability/randomness in validation set result. The results were consistent over all the times that I run the code for the LOOCV setting.

- Results for different holdouts:

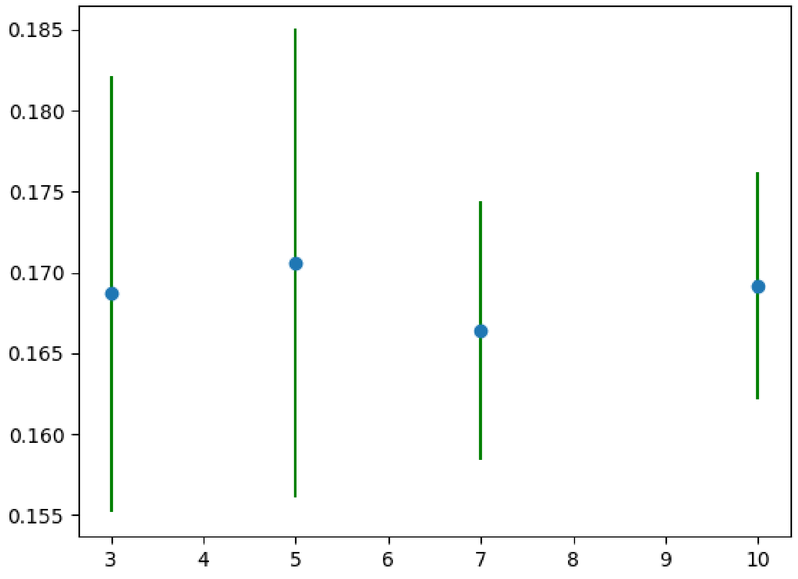
3 Folds:
Error on train set:
Min error: **0.164** Max error: **0.178** Mean error: **0.168**

5 Folds:
Error on train set:
Min error: **0.163** Max error: **0.177** Mean error: **0.17**

7 Folds:
Error on train set:
Min error: **0.163** Max error: **0.171** Mean error: **0.165**

10 Folds:
Error on train set:
Min error: **0.165** Max error: **0.172** Mean error: **0.169**

<u>Plot for validation error variances for different holdouts(k values are 3,5,7 and 10):</u>



**the blue points refer to the mean values where as the green lines represent the range of error values*

<u>Observation:</u> The mean error for all the holdouts did not vary much. The variances of the error values reduced with the number of folds.