HW2: Credit Risk prediction

Name : Venkata Pratyush Kodavanti

Miner username: vkodavan

Accuracy: 0.65

I've included the below mentioned files, where I've implemented different functionalities. I've used these files just avoid commenting so much.

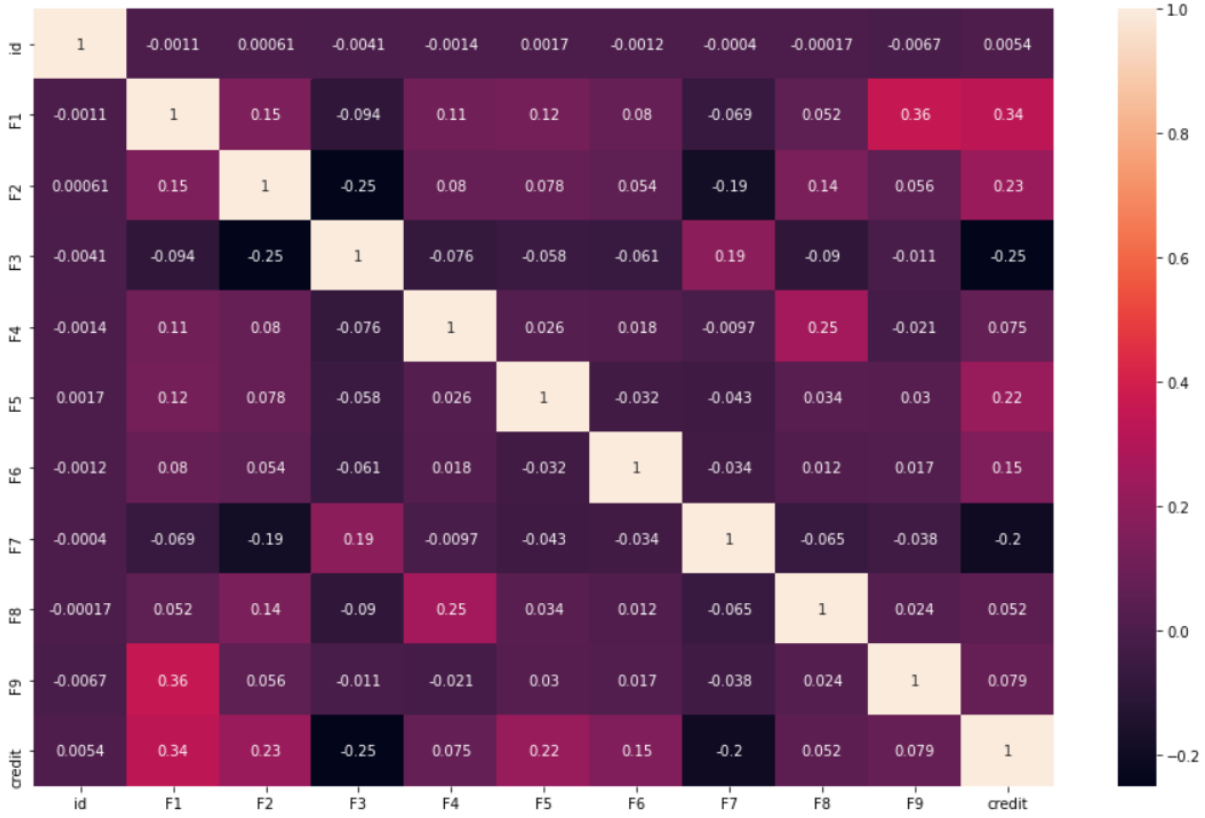
1) 584_Assignment2.py (Main submission file)
2) Correlation_cross_validation.py (Cross validation and F1-Score calculation)
3) Categorizing_continuous_variable.py
4) Post_pruning.py
5) Random_Forest.py

Preprocessing the data:

As mentioned in the question, the training data consists of categorical and continuous variables. First step is to check if the dataset contains any null values.

For that I've used standard **pandas** library to read the csv file and then stored it in a dataframe. After storing the training data in a dataframe, I've checked if the data contains any null values using the **isnull().sum()** function. There were no null values in the dataset.

After that I've checked the correlation between each of the features, in order to eliminate any feature. For this I've used the **corr()** function which calculates the correlation of each column to each column. This is implemented **in Correlation_cross validation.py file**.



To plot this heat map, I've used **seaborn heatmap** function. As the correlation of each feature to that of the credit is not significantly high or not significantly low, Feature selection was not helpful.

Categorizing the continuous variables: First I've scaled the continuous columns in such that they all fall in the interval [0-1] using the function **normalize().**

```
[ ]  def normalized(A):
         A = (A- np.min(A))/ (np.max(A) - np.min(A))
         return A
```

After normalizing the columns, I've divided the columns value into 4 intervals using the pandas cut() function, which cuts the continuous variables into bins of our given range, I have classified them into 4 classes (-0.1, 0.25], (025, 0.50], (0.50, 0.75], (0.75, 1.0] but this did not help in the

overall accuracy. This is implemented in **Categorizing_continuous_variable.py file**.

F1_(-0.1, 0.25]	F1_(0.25, 0.5]	F1_(0.5, 0.75]	F1_(0.75, 1.0]	F2_(-0.1, 0.25]	F2_(0.25, 0.5]	F2_(0.5, 0.75]	F2_(0.75, 1.0]	F5_(-0.1, 0.25]	F5_(0.25, 0.5]	F5_(0.5, 0.75]	F5_(0.75, 1.0]	F6_(-0.1, 0.25]	F6_(0.25, 0.5]	F6_(0.5, 0.75]	F6_(0.75, 1.0]
0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0
0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0
0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0

Further I've used pandas.get_dummies to eliminate dummies in the dataset.

The classifiers I've used are

1) Decision Tree
2) Random Forest

Decision Tree: For the decision tree classifier I've used Scikit Learn's tree decision tree classifier. Decision tree classifier uses fit(), which takes 2 parameters features of the dataset and the target variable. Then a decision tree is created using the data. Once the decision tree is created we can predict using the predict() method which takes the testing features.

Cross-Validation and F1-Score: For cross validation and calculating the F1- Score, I've used scikit learn's model_selection Kfold() which takes in a parameter of n_splits. I've made 15 splits of the data.

To calculate the F1 score, I've used scikit learns metrics f1_score. It takes 2 parameters y_predction and y_test.

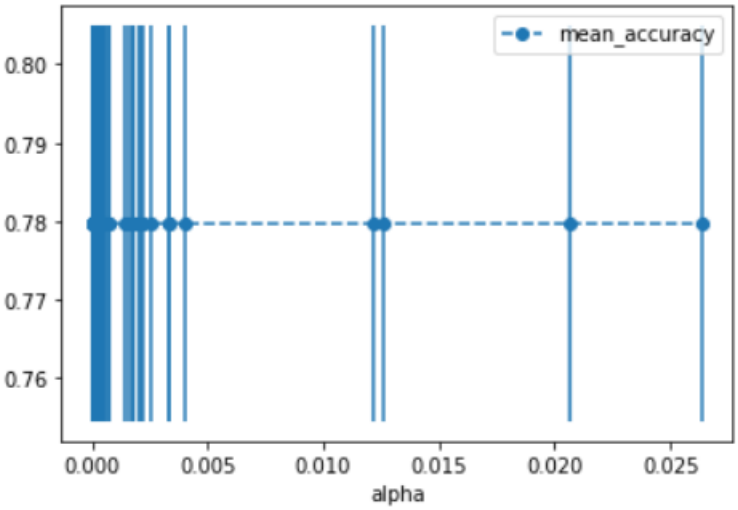
The F1 score for basic implementation was well over 82%. But the accuracy was 61%, This clearly is an indication that the model is overfitting.

To normalize the overfit model, I've used 4 different techniques

1) Over sampling
2) Under sampling
3) Pre-pruning the decision tree

4) Post-pruning the decision tree.

1) **Over Sampling:** For this I've tried to balance the data set by adding the data(replicas) where credit is 1 and then shuffled the data, after doing this the accuracy did increase by 2% i.e, 63%
2) **Under sampling:** For under sampling here I have deleted some of the training data values so that the classes of credit is balanced as there are fewer examples to train the accuracy did drop to 59%.
3) **Pre-Pruning the Decision Tree:** Rather than letting the tree grow to its maximum depth, I've set the maximum depth to 20 instead of normal (39). Further, I've also set the minimum samples required to split as 50. Meaning split if there are atleast 51 values in the node. By applying these 2 parameters the accuracy did increase to 64%
4) **Post – Pruning the Decision Tree:** Instead of stropping or regulating the tree from growing we let the tree grow normally and then prune the subtrees with the cost complexity greater than the ccp_alpha value. From the graph, it is clearly visible that

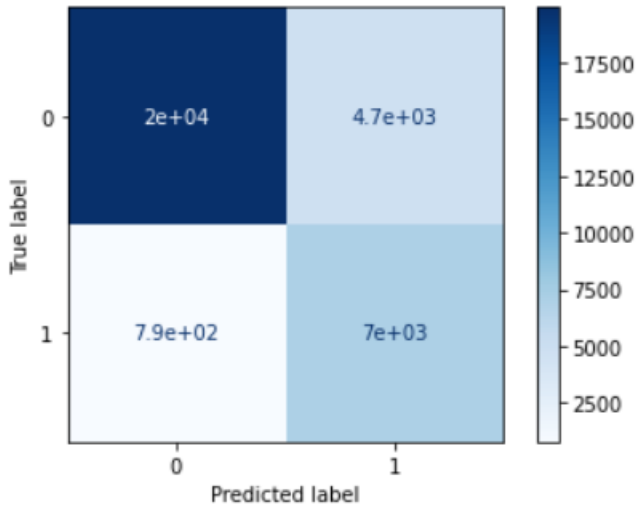


median CCP_alpha value is at 0.003

Balancing the weights of the imbalance class, As the dataset is imbalanced, we introduce the class weights in such as a way we balance the classes. Here I've used a weight of 0.2408 for class credit = 0 and a weight of 0.7491 for class credit =1 along with the pre pruning on the training data while fitting the model to get an accuracy of 65% on the training data.

```
Count: 0
 F1-Score 0.7174489795918367
20
==========================================================================================================================
Count: 1
 F1-Score 0.7177892214347626
20
==========================================================================================================================
Count: 2
 F1-Score 0.7178518556332635
20
==========================================================================================================================
Count: 3
 F1-Score 0.7173159290230472
20
==========================================================================================================================
Count: 4
 F1-Score 0.7176140708641346
20
==========================================================================================================================
Count: 5
 F1-Score 0.7175774877650897
20
==========================================================================================================================
Count: 6
 F1-Score 0.7174256718853588
20
==========================================================================================================================
Count: 7
 F1-Score 0.7179382495534574
20
==========================================================================================================================
Count: 8
 F1-Score 0.717446721729377
20
==========================================================================================================================
```

The final F1-scores.



Random Forest: In addition to decision tree, I've also used random forest classifier to get an accuracy of 64% on test data set. For this I've used scikit learn's RandomForestClassifer with a parameter of n_estimators as 500.

References:

1) Documentation of Scikit Learn's Decision Tree, Random Forest was helpful.
2) https://medium.com/@neevarp.v/decision-trees-cost-complexity-pruning-cross-validation-using-scikit-learn-b1a08f92303e