Credit Risk prediction Report

Venkat Sai Charan Bandi
G01219773

Miner UserName: Cherry_SLYR

Rank	User	Submission Time	Public Score
252	Cherry_SLYR	Sept. 30, 2020, 8:44 p.m.	0.66

Instructions to the TA

I attached my PyCharm Project folder as src_code, the main code file is called the main.py to run the code

Project Walkthrough

During the span of this project, I had to make use of and try out multiple classification algorithms I could get my hands on. For me, i choose my final code i uploaded based on the F1 metric score only but i arrived at that score after by trying out and tuning the code by some various factors i choose discuss in this report.

The Approach

Using the F1 vector by importing into a pandas dataframe structure was the first step for all classifiers. This allowed me some flexibility on what metrics I could select, modify and arrange. To choose and select classification algorithms for this problem i considered: Logistic Regression, K-Nearest Neighbour, Decision Tree

Logistic Regression:
The issue i had with going forward with this approach was that it was very non-dynamical in approach and felt it needed a lot of code and tuning to generate sub optimal F1 Scoring metric, so i discontinued using this approach

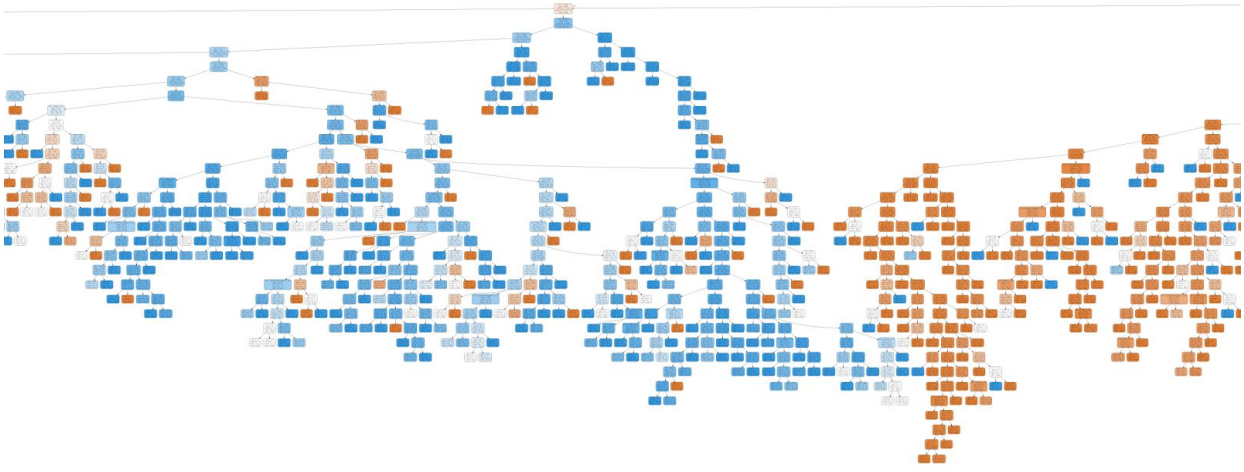
K-Nearest Neighbour:
This would have been my selected classifier, but it got me very bad results in the cross validation test. The biggest issue I had with this classifier is choosing to normalize the vectors and when I had to assign weights to each feature selection. Especially with features like:

● F1 - Continuous value describing number of years since last degree was completed
● F2 - Continuous value indicating hours worked per week
● F5 - continuous value denoting gains
● F6 - continuous value denoting loss
That has a continuous range of values, and hence writing similarity function was turning out to need more precision, and caused imbalances and it was inefficient as it needed to iterate over the entire dataset for each test case, hence i discontinued this classifier to explore other algorithms.

Decision Tree Classifier:
After trying out prebuilt libraries and gong through the lectures, the GINI index seemed like a good try since it conceptualizes the feature first selection, similar to the words Information gain or Entropy

Beginning with this approach, I used the scikit Decision Trees module (https://scikit-learn.org/stable/modules/tree.html). Using decision trees across various continuous values and binary values needed the concept of One Hot Encoding. It allowed me to normalize the values and enabled columns with easier binary read of data for decision trees. I used the pandas get_dummies function to convert my data frame into one hot data. With the scikit decision tree classifier I trained the test_data with the class difference of 'Credit' being 0(bad credit) or 1(Good Credit).

When i initially predicted this tree classifier into the cross validation module i got around 70% accuracy and around 0.61 on the Miner. I wanted to take a look at the visualization of the tree and what may be causing the results. I used pydotplus/ GraphViz to map and chart my data visually. The result was a very large image file generated by Graphviz Library, that wouldn't even render properly. Attached below and saved in the Source Code Folder:



Recognizing my mistake of selecting all the data and features I begin to reduce and analyse the dataset. Some of the main changes i made from here on out that helped me get my final score of 0.66 on the F1 Metric are:

- Looking at the graph from Dot Graph Data, the **Features that were selected important** were - Hours worked per week, years since last degree, type of occupation and so on. And the least important features boiled down to Race, Marital Status etc. Using this I tried constructing a data set using the important features and removed the least important ones. This did not result in any improvement score wise hence i sticked to using all of them at the end.
- The most trails i did during this project was to change and tune the **parameters of the decision tree classifier** by mainly modifying the max depth of the tree that would be considered, since the bottom of the tree results in less precision anyways (i tried from 0-30 and i achieved best results with 10-14 as the max depth) and the min sample split to be considered for a splitting the node, made a difference when it was set to more than 30 on an average.
- After I noticed the **imbalance in the class distributions** within the training set, there are 24720 (0) and 7841 (1). Luckily the scikit classifier came with an inbuilt class_weight parameter that would consider the number of each distribution and allowed me to normalize it by increasing the weights of the lesser occurring (1) and hence removing the overfitting that was occurring.To overcome this distribution i used: #class_weight = {0:0.24720 , 1:0.7841} This mainly pushed my score up by 4 points so i understand that imbalance in dataset plays a lot of value during training itself

Validation & Testing

To test This code before uploading on miner i performed a 70-30/75-25 cross validation and got around 70-75% accuracy but it didn't reflect the score on miner during the initial submissions so i didn't keep it for my final submission either.