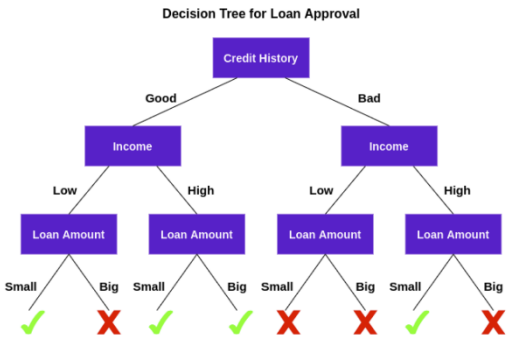
Siddharth Patel
CS 484 – 001
Date: 10/17/2021

My miner2 username is spatel53. My George Mason username is spatel53 as well. For part A, my best public score is 0.63% while part B's best public score is 0.69%. As of writing this report, my public score is 70 for part A and 13 for part B.

Overview

A decision tree can be defined as a supervised machine learning algorithm that can be used for classification problems. A decision tree is simply a series of sequential decisions made to reach a specific result. Importance of an attributes to be checked is decided based on criteria like Gini Impurity Index or Information Gain.



Sometimes, a single decision tree is not enough to produce effective results. Random Forest is a tree-based machine learning algorithm that leverages the power of multiple decision trees for making decisions. Each node in the decision tree works on a random subset of features to calculate the output. The random forest then combines the output of individual decision trees to generate the final output. Ensemble learning is a process of combining the output of multiple individual models.

Generally, a decision tree's accuracy is calculated based on the F1 score shown above.

Support Vector Machine, better known as SVM is a supervised and linear machine learning algorithm most commonly used for solving

Analysis

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Part A consists of two files, train and test data. Train data consists of attributes including the final output being either a 0 or 1. Although, the attributes does not have title defining each column, so it is read directly into the program without further preprocessing. Part B also consists of train and test data, but some of the data consists of strings which must be converted to a valid numerical value that can later be passed into the program for decision tree classification. My first thought, given the data, was that I need to figure out a quick yet efficient way to implement decision trees directly since I do not have any preprocessing to perform on the data for Part A. Part B, however, needed preprocessing since the data provided was in the form of strings. Unfortunately, I did notice a huge difference in the size of train and test data where the train data size consists of 5049 labels versus the test data being 2165 labels only.

Approaches

Part A

The first step to computing Part A is reading data from the train and test files. For this, I used the pandas library to import the contents within these files. I made sure to set header to none since Part A data did not have any headers in the excel sheet. I named the train and test files *train_file and test_file*. I then gave each files dummy headers, so I can access each column easily just as a future proofing measure.

	sex	age	age_cat	...	c_charge_degree	c_charge_desc	label
0	Male	40	25 - 45	...	M	arrest case no charge	0
1	Male	39	25 - 45	...	F	Poss Unlaw Issue Id	1
2	Female	22	Less than 25	...	F	arrest case no charge	0
3	Male	44	25 - 45	...	M	Reckless Driving	0
4	Male	32	25 - 45	...	F	arrest case no charge	0

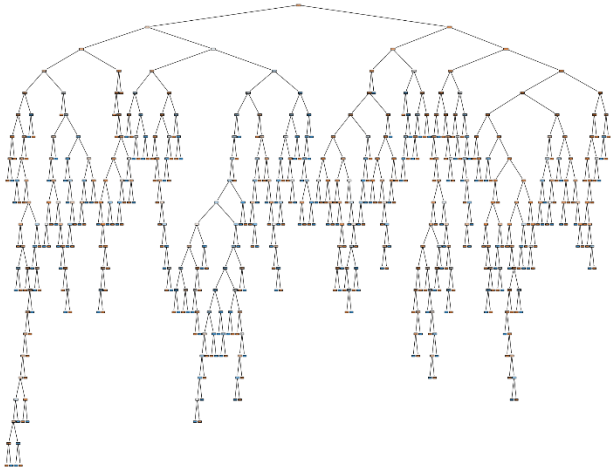
[5 rows x 11 columns]

	sex	age	age_cat	race	...	priors_count	c_charge_degree	c_charge_desc	label
0	0.0	40	1.0	0.0	...	2	0.0	0.0	0
1	0.0	39	1.0	1.0	...	0	1.0	0.0	1
2	1.0	22	0.0	0.0	...	2	1.0	0.0	0
3	0.0	44	1.0	1.0	...	1	0.0	0.0	0
4	0.0	32	1.0	0.0	...	2	1.0	0.0	0

[5 rows x 11 columns]

My initial approach involved using a decision tree which gave me an accuracy of approximately 0.54. I chose Gini Index to create split points and graphed my decision tree shown below.

```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(dt,
                   feature_names=X.columns,
                   filled=True)
fig.savefig("decision-tree.png")
```



After a little bit of research, I learnt about Random Forest classifier and how it outperforms Decision Trees in most cases. I learned that since Random Forest leverages the power of multiple decision trees, it does not rely on the feature importance given by a single Decision Tree. Since we had little to no data that relied on user interpretation (given that there are no headers to interpret each data) I chose to use Random Forest and ended up with improved score of 0.63.

Preprocessing

Part B

For part B, preprocessing certain columns in the data was an essential step prior to implementing any further machine learning classifiers. I first started converting each columns manually. For example, replacing “Male” with 0 and Female with “1”. I immediately realized this was not very effective in practice. I changed my previous idea with label encoding, a common technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

The problem with label encoding is your labels do not have an order or rank. Each label is assigned unique integer based on alphabetical ordering. Due to this, there is a high probability that my model would capture the relationship between labels. To tackle this issue, one-hot encoding emerges!

One-hot encoding simply generates additional features based on number of unique integers in the categorical feature. Every unique value in the category will be added as a feature. We solve the categorical problem by presenting our labels as features through a binary vector.

Hence, I used label encoding and one-hot encoding together to improve the accuracy of my algorithms making my preprocess virtually flawless. Below is a comparison of my data prior to preprocessing and after preprocessing.

Implementation

After preprocessing, I verified for Part B whether Random Forest will outperform Decision Tree again (similar to Part A). Lo and behold, Random Forest was the winner. Unfortunately, my implementation wasn't good enough as I was only able to hit 0.63 accuracy. I separated my train data into labels and outputs represented by either 0 or 1. I then used this data globally to train my models across the board.

Apparently, my algorithm was outperforming my cross validation when I submitted the format.txt file on miner resulting in an accuracy of 0.63 as I mentioned above. To improve my accuracy, I chose to implement support vector machine or SVM using sklearn library. To compare the two algorithms, Random Forest and SVM, I pitched them against each other by computing each algorithm's accuracy (show above). Based on the results, I was able to conclude that SVM performed better than Random Forest.

Here is why SVM performs better: SVM maximizes the margin and thus relies on the concept of distance between different points. Since I used one-hot encoding for categorical feature grinding it down to binary labels, SVM was able to take advantage of this data more effectively as compared to Random Forest.

To counteract the advantage SVM has with one-hot encoding, I removed the one-hot encoding implementation from my preprocessing and simply used the label encoding. To my surprise, my public score on miner went from 0.69 to 0.62. This difference was phenomenal as my score went below the Random Forest model too (0.63).

```
Accuracy: 0.5264900662251655

Accuracy Of SVM:  0.622442244
```

I replaced my preprocess with the combination of label encoding and one-hot encoding again. After running SVM model, I was able to achieve an accuracy of 0.69 on miner placing me on the 13th position on the leaderboard.

```
svc = SVC(kernel='rbf', random_state=1)
svc.fit(X_train, y_train)
y_pred_dummy = svc.predict(X_test)
cm = metrics.confusion_matrix(y_test,y_pred)
accuracy = float(cm.diagonal().sum())/len(y_test)
print("\nAccuracy Of SVM: ", accuracy)
```

Conclusion

To sum up this project, I would like to conclude that choosing the right model based on the given scenario can result in incredible accuracy as it happened to me. This project has made me realize the difference between each models along with it significance in different scenarios. However, let us not forget the importance of preprocessing. If one performs the preprocessing correctly, it can enhance your programs accuracy significantly and enhance the machine learning model as well. The efficiency and computation of each model increases significantly if I stray from a proper preprocessing.

Based on the accuracy I have achieved using SVM, I don't think this will be a useful aid to determine whether to let criminal defendants out of parole. Here is why:

- The data presented is based on a person's past credentials. It could be possible that the person may have changed over time.
- It does not predict based on the person's situation, rather, solely on criminal records. For example, a person who is homeless may have robbed multiple people/restaurants/ banks. Their situation has forced them into inflicting this situation upon them, but that does not mean the individual is necessarily guilty. If he had sufficient money to survive, he would never take such actions on the first place.
- Finally, given that there are multiple missing factors from the data presented to me, and a low accuracy of 0.69 which feels like almost a 50-50 chance, I do not think this is a great measure to predict whether to let criminal defendants out of parole.