Risk & Fraud Analytics
Individual Assignment

Work by **Mohamed Khanafer**

Table of Content

1. Introduction

The following document is a summary and overview of the attached Jupyter notebook which includes the detailed code and explanation of the thinking followed throughout to get to the final model for this exercise. I would advise to start here to get the general idea of my thinking and also allocate some time for the code as I have tried to write it in a comprehensive way, building on tip of the starting code provided.

Throughout my exploration, I tried roughly to follow the following flow of ideas:

- Model Evaluation: KS and Gini
- Models Trials: 10 different methods
- Hyperparameter tuning
- Resampling the data
- Feature Selection
- Preprocessing the variables and feature creation
- Combinations of insights into models

Thus, from the baseline I have built I accumulate the insights into more performing models that have allowed me to finally reach a model that gets a KS2 of 0.422, a GINI of 0.539 and a GRADE of 10.0. I now go through these steps and explain what worked and did not in my search for the best model.

2. Building a Baseline

Metrics used

As provided in the starter code, the two main metrics used here to assess the performance of our models are the Gini and the KS scores. As mentioned by Idan Schatz in his article (see references section), the Gini or the Gini coefficient is one of the most used metrics when it comes to evaluating the performance of credit score models as well as other models in the Financial sector. It indicates the model's discriminatory power, how effective is the model when it comes to differentiate between the fraudulent and non-fraudulent transactions. It has been shown that there was a linear relationship between the AUC and the Gini coefficient, thus here we derive the Gini from the AUC score using the formula Gini = 2 * AUC-1.

And as said by Ratnakar Pandey, the KS (Kolmogorov-Smirnov) is similar to Gini, in that it also captures this discriminatory power of the model in its ability to separate the fraudulent from the non-fraudulent ones. It represents the highest separation between the Cumulative

Good Rate and Cumulative Bad Rate. The higher the KS, the better the model is, indicating a higher separation between those two classes.
These are the ones used to compare the performances of the models built along the way.

Assessing various algorithms

Before jumping into further processing, it is useful seeing which models would perform the best for the given problem. Because we do not have the target variable in the Out of Time sample, it is not possible to get the metrics on these values. I thus had to rely on the server to get them and here are the results of the trial:

	Gini Scores	KS scores	KS2 Score Ranking	Gini Score Ranking	Grade Score Ranking
CatBoost Classifier	1.000000	1.000000	0.363965933142	0.454567509866	8.625
Random Forrest	1.000000	1.000000	0.360761190029	0.46946020758	8.549
LGBM Classifier	1.000000	1.000000	0.313320825516	0.365285261416	7.425
XGB Classifier	0.979586	0.906865	0.300132163883	0.373023318145	7.112
Logistic Regression	0.387008	0.338449	0.171761060638	0.202321648074	4.07
K-nearest neighbors	0.916559	0.904639	0.165535263727	0.17026728528	3.923
Naive Bayes	0.130330	0.132849	0.110686096914	0.116892254087	2.623
Logistic Regression SMOTE	0.310069	0.257381	0.106024547362	0.124552907143	2.512
MLP Neural Network	0.175668	0.140933	0.0914079612566	0.086709442786	2.166
Support Vector Machine	-1.000000	1.000000	0.0236323810756	0.0372646697289	0.56

As can be seen from the table above, the best models for this problem seem to be the tree-based models. Even if it is clear that those models are highly overfitting here with perfect metrics in training but poor performances when it comes to predicting.

The models I thus decided to be focusing on in my approach to the problem are the Catboost Classifier, the Random Forrest, the LGBM Classifier, and the XGB Classifier.

3. Hyperparameter tuning

The different approaches to hyper parameter optimization could be summarized in the following categories:

1. Manual Search
2. Random Search
3. Grid Search
4. Automated Hyperparameter Tuning (Bayesian Optimization, Genetic Algorithms)

Because here the performance of these algorithms is already very high in training, it seems that hyper parameter optimization maybe could be useless. And this was proven when I tried Random Search, Grid Search, and Automated Hyperparameter Tuning.

The best hyperparameters thus were found updating them manually. I did this for the Catboost Classifier and the Random Forest. This did improve my scores to KS2 = 0.382706864204; GINI = 0.478597306814; GRADE = 9.069 for the Catboost Classifier and to KS2 = 0.373473876838; GINI = 0.515840026248; GRADE = 8.85 for the Random Forest.
This was I believe a very good advancement using all the variables.

The parameters I mainly played with are the:

- class_weight='balanced_subsample'
- n_estimators=1200
- criterion='gini'

Increasing the number of estimators (or the number of trees) does increase overfitting but this was needed for this problem as this has greater potential of finding the fraudulent cases.

4. Resampling

Given that the fraudulent cases are only 10% of the total data I thought that increasing the number of fraudulent cases would lead to a better performance by the models. Even if the dataset is not highly imbalanced as could be the case with other datasets related to fraud detection, I believed that this could play an important part for the construction of the model.
I first tried

Random Resampling

When resampling, I am actually creating a new transformed version of the training data where the studied variables have a different class distribution, solving the light imbalance problem we have for this dataset. And there are 2 ways of doing so:

- Random Oversampling: we randomly duplicate instances from the minority class (the fraudulent cases here);
- Random Undersampling: we randomly delete instances in the majority class.

Oversampling did improve the performance of the model to a 0.3817 KS2 as compared to 0.3734 in the previous section. Highlighting thus that this does help the model.

Undersampling, on the other hand, did not seem to be the proper way to deal with the class imbalance here with a decrease in performance to KS2 = 0.300

Manually Duplicating

Instead of randomly resampling the data, I could just duplicate the fraudulent cases. I here tried various approach that highlighted the increase benefits including these:

1. Tripling the cases of fraud + hyper parameter tuning for Catboost: it gave me a performance of KS2 = 0.388958077247; GINI = 0.477204040703; GRADE = 9.217.

2. Doubling the fraudulent cases + hyper parameter tuning for Random Forrest: it gave me a performance of KS2 = 0.393767502472; GINI = 0.525962578212; GRADE = 9.331.

I thus concluded that like the hyper parameter optimization, manually duplicating the fraudulent cases was the best approach to take here.

5. Feature Selection

Once I had a clear idea on the model I wanted to choose, the parameters that worked best, and the resampling strategy, the next step was to try filtering some of the variables from my data to push the performance higher.

Scikit Learn Feature Importance

To do so, I first took a look at Scikit-Learn's embedded feature importance function. So when training a tree it actually computes how much each feature contributes to decreasing the weighted impurity. Using the feature_importances_ in Scikit-Learn, we average the decrease in impurity over trees and select the most relevant features according to that.

I tried running some models setting some thresholds to remove the least relevant features, but this did not improve on my best performance reached so far. But nonetheless, what I found interesting is that with less features (51 out of 81 initially) the model achieved a performance of KS2 = 0.369567879555; GINI = 0.513550263866; GRADE = 8.758. This prove I believe that a good selection of features could significantly increase the score.

Feature Importance by permutation

I then tried to get the Feature Importance by permutation. This approach to feature selection measures directly the importance by assessing how random re-shuffling of each predictor in the data will influence the model performance. It first trains the model and saves the score; it then re-shuffles the values from one variable in the studied dataset, pass the dataset to the model again to get the predictions and compute the gini score for this modified dataset. The feature importance will then be the difference between the

benchmark score and the one from the modified dataset. And this will be repeated for all features.
Here, something even more interesting happened:
With only 29 columns selected, the performance is almost the same as the base model, proving once more that many variables are less significant for the model. Its performance after tweaking some hyperparameter with the 22 features was of KS2 = 0.363775312156; GINI = 0.506221175796; GRADE = 8.62.

Feature Selection based on the PSI

As mentioned in the *Statistics for Machine Learning* book, the Population Stability Index is a metric used to check that the drift/change in the current population on which the model is used will be the same as the population in the development time. And generally, from the value, we could infer that:

- A PSI lower than 0.1 indicates no change between the two;
- A PSI between 0.1 and 0.25 means that some change has taken place and should call for attention but could still be used in the model;
- A PSI larger than 0.25 means that there is an important shift in the score distribution of the current population as compared to the development time one.

I used the code provided in the hints to find this PSI and potentially use it for a model. The performance I got was poorer and I believed that maybe given that our dataset is very special, the consideration of the PSI may not be useful in this case with a significant drop in the scores.

Concluding on feature selection, even if I was not able to find the most appropriate features at that stage, I believe that playing around with these approaches showed that proper feature selection can increase the performance of the model. I found the proper variables in later stages as I highlight in the next sections.

Also, I tried implementing the Genetic Algorithm code but had some difficulties in its implementation.

6. Dealing with the NaNs

Up until here I had been replacing the nulls values in the Out of Time sample with zero, I tried changing the approach to see if I would get an improved performance. I tried the following approaches:

- Replacing NAs with the mean: KS2 = 0.349098651559; GINI = 0.496392064622; GRADE = 8.272

- Replacing NAs with the median: KS2 = 0.348392776273; GINI = 0.495559108679; GRADE = 8.256
- Replacing NAs using the KNN Imputer: KS2 = 0.349512241333; GINI = 0.496300797604; GRADE = 8.282
- Replacing NAs manually with a probability: I tried replacing the predicted probabilities by 0.5, 1 and 0 and it did not improve the performance.

Concluding on the approach for the null values, I believe that replacing the null values with 0 was the best approach to follow as all other ways did not improve the score of the model.

7. Some Preprocessing

Scaling and Skewness

I then tried various preprocessing ideas to see if they will affect the performance of the model. I first thought of addressing scaling and the skewness of the data.
For fixing the scale of the data, I decided to use the MinMaxScaler from scikit-learn as can be seen in my code. As for fixing the skewness, I used an already built function and realized that many of the numerical variables were actually highly skewed. After fixing the scaling and the skewness, I ran a model but its performance was not satisfactory (KS2 = 0.341442619618; GINI = 0.487809499164; GRADE = 8.091).

Other processing ideas

I then tried binning the continuous variables into equal width bins after scaling, but this did not improve the model performance, leading to the abandonment of the idea.
I have tried as well dummy encoding the categorical variables with no success either and thus did not pursue the idea.

Feature Creation

I then generating some new features with the polynomial features function. This did not improve the performance which was: KS2 = 0.365498987976; GINI = 0.510779905544; GRADE = 8.661.

8. From insights to Models

From the insights gathered all throughout the flow, I combine them into various models that I tested before choosing the best model. I tried around 400 combinations but summarize some of them into the following table:

	HP KS2	HP Grade	HP + OV KS2	HP + OV Grade	HP + DUP + FS KS2	HP + DUP + FS Grade
XGBC Model 1	0.3340	7.916	-	-	-	-
XGBC Model 2	-	-	0.3316	7.858	-	-
XGBC Model 3	-	-	-	-	0.3518	8.337
CATB Model 1	0.3566	8.451	-	-	-	-
CATB Model 2	-	-	0.3778	8.953	-	-
CATB Model 3	-	-	-	-	0.3824	9.062
RF Model 1	0.3755	8.9	-	-	-	-
RF Model 2	-	-	0.3821	9.056	-	-
RF Model 3	-	-	-	-	0.4219	10.0
Ensemble: RF and CATB	-	-	-	-	0.5352	9.63
Ensemble: 3 RF	-	-	-	-	0.5381	9.981

What I actually did is the following: I tried 3 approaches for each classifier:

- Hyperparameter Optimization alone (HP in the table)
- Hyperparameter Optimization + Oversampling (HP + OV in the table)
- Hyperparameter Optimization + Duplicated fraud + Feature Selection (HP + OV + FS in the table)

For each of these approaches and model I have given here the KS and grade from the ranking.

9. Conclusion

As can be seen in the above table, the best model I got was a Random Forrest following the 3rd approach which is the Hyperparameter Optimization + Duplicated fraud + Feature Selection.
And I would like to comment on an interesting aspect of this problem. All of these steps were performed manually. Indeed, as mentioned above, the hyper parameters that gave me the best results were those that I manually set. For the resampling approach, the manually duplicated features gave me the best results as well. And finally, for the feature selection, I manually selected the features based on the insights I got running the feature importance functions as well as noticing the high skewness in the float variables. Indeed, it is by excluding those from the model that I got the best results for my model. Given that the variables are not known and that this dataset is somehow a special one, the process I followed including a lot of manual decisions is for sure not the most appropriate one when it comes to Machine Learning per se but I believe that it shows how for this problem, combining the insights generated from proper machine learning and with some creativity a data scientist can achieve great things.

For further details on the specificities of the best model as well as the detailed approach, they are included in the Jupyter Notebook submitted along the report.

References

- *Using the Gini coefficient to evaluate the performance of credit score models*. Can be found here: https://towardsdatascience.com/using-the-gini-coefficient-to-evaluate-the-performance-of-credit-score-models-59fe13ef420

- *Which one is better to evaluate a logistic regression: Gini, KS or ROC?*. Can be found here: https://www.quora.com/Which-one-is-better-to-evaluate-a-logistic-regression-Gini-KS-or-ROC

- *Learn classification algorithms using Python and scikit-learn*. Can be found here: https://developer.ibm.com/tutorials/learn-classification-algorithms-using-python-and-scikit-learn/#set-up

- *Python package: CatBoostClassifier*. Can be found here: https://catboost.ai/docs/concepts/python-reference_catboostclassifier.html

- *Random Oversampling and Undersampling for Imbalanced Classification*. Can be found here: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

- *Explaining Feature Importance by example of a Random Forest*. Can be found here: https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e

- *Statistics for Machine Learning*. Can be found here: https://books.google.com.lb/books?id=C-dDDwAAQBAJ&pg=PA94&lpg=PA94&dq=population+stability+index+(PSI):+this+is+the+metric+used+to+check+that+drift&source=bl&ots=j1br-tuY-q&sig=ACfU3U3lxkf0DwE9WK_8IefldeOjRf3m_Q&hl=fr&sa=X&ved=2ahUKEwictYbC8srpAhV-BGMBHfh2B2gQ6AEwB3oECAoQAQ#v=onepage&q=population%20stability%20index%20(PSI)%3A%20this%20is%20the%20metric%20used%20to%20check%20that%20drift&f=false

- *Imputing missing values before building an estimator*. Can be found here: https://scikit-learn.org/0.18/auto_examples/missing_values.html