Mid-term Kaggle Report

MIS 385N - Business Data Science

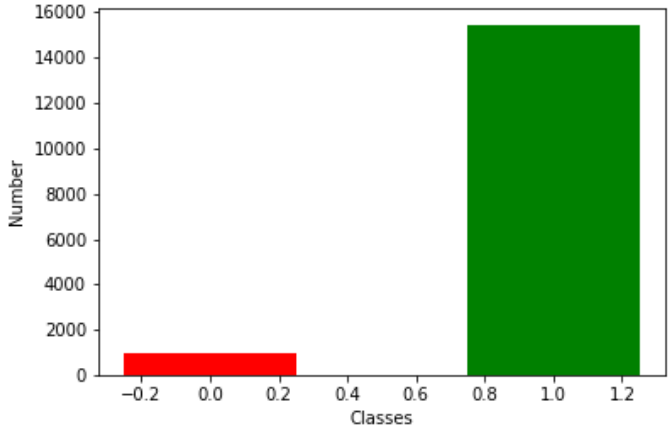
Subhayu Chakravarty

This is a brief report of the steps I followed to predict the label of a dataset in a binary classification problem. The feature names, and thus meaning, were hidden and so it was purely based on optimizing the results considering just the data.

A. Initial Data Exploration and Analysis

First, I began by looking at the dataset's class distribution. It turned out to be quite an imbalanced dataset where the ratio of positive to negative labels were 94:6.

```
Y=1:  15435  - Percent:  0.94
Y=0:  948  - Percent:  0.06
```



Other observations:
24 features, 16383 samples in train, 16385 samples in test set.

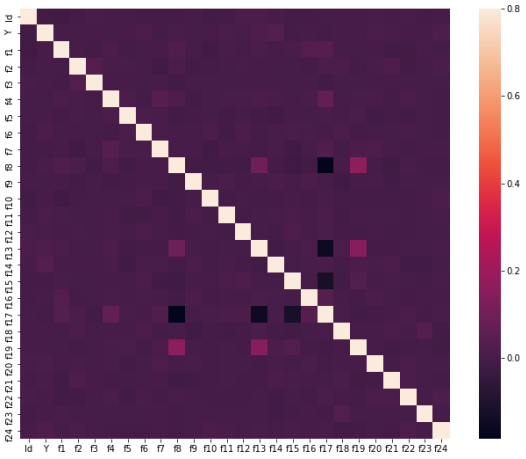
The next steps involved the following standard exploration techniques:

1. Checking for null values in features. None found.

2. Describing the dataset to get all relevant statistics – means, standard deviations and quartiles/percentiles.

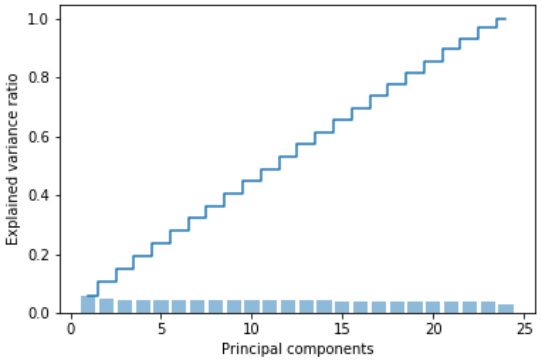
3. Finding the unique values of each feature. This can help in determining which features to use for one-hot encoding – the ones with less unique values, indicating categorical features.

4. Finding correlation matrix and getting the features highly correlated with the label Y.

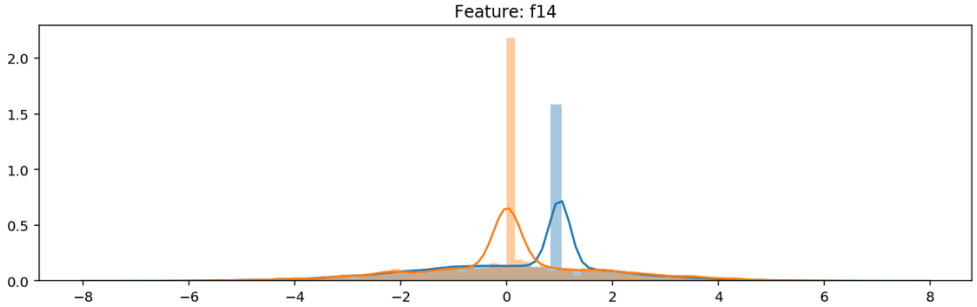


Candidates for interaction terms : (f8,f19), (f8,f13), (f8,f17), (f13,f19), (f13,f17), (f15,f17)

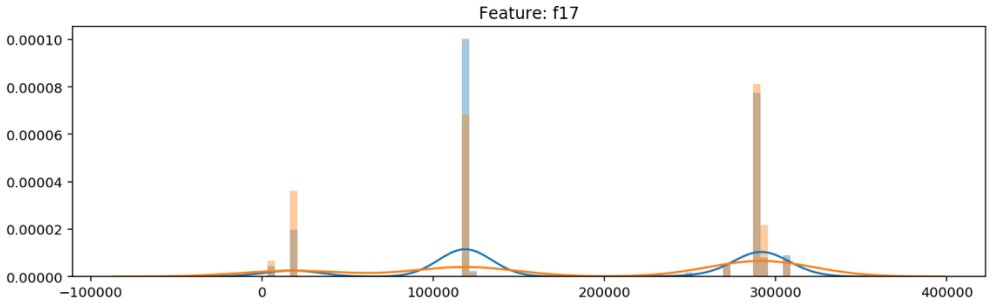
5. Finding Principal Components and the explained variance of each feature to learn which features are responsible for most of the sample variance. Turns out it is quite evenly distributed among the 24 features.



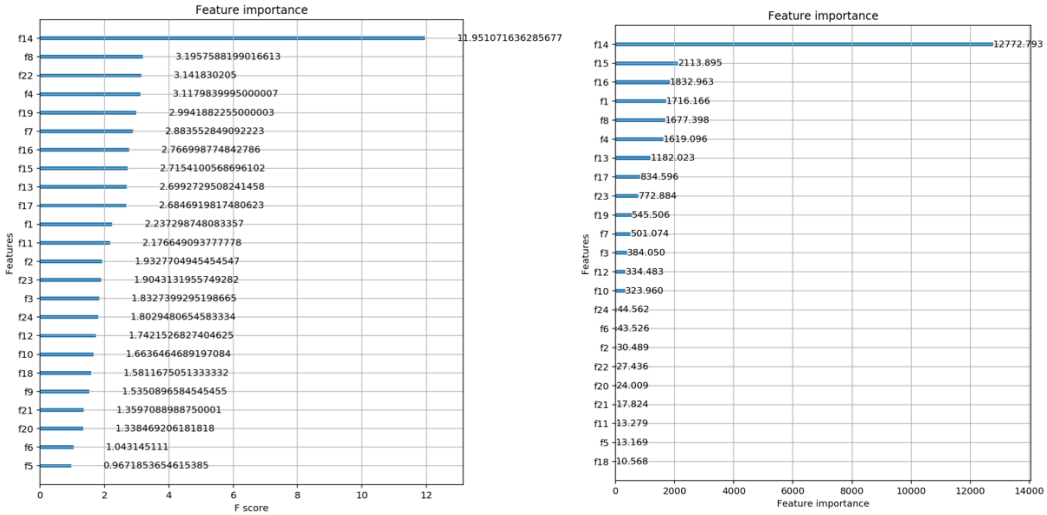
6. Finding the variation/distribution of features within each class. Most of the features did not vary too much between the labels, but f1, f6 and f14 seemed to show different trends for Y=1 and Y=0.



This is the kind of plot that is best for a model to distinguish between the classes. But unfortunately, most of the features show a uniform distribution of features among the classes like below.



7. Finding the gain-based plot of feature importance after applying an XGBoost and LGBMClassifier models to determine the relative importance of features. This will help eliminate certain noisy features when we apply XGBoost and LightGBM.



XGBoost and LGBM Feature importances

B. Approaches to Modeling

1. Plain models:
Initially I just applied a variety of models to test which gave the best AUC score without any parameter tuning and on the original dataset.
In their plain forms, LightGBM, BalancedRandomForest, GradientBoosting, AdaBoost and XGBoost came out on top and in that order.
The highest AUC at this stage: 0.878

2. Resampling approaches:
I used the following resampling methods to balance the classes in the dataset-
a. IMBLearn's RandomUnderSampler – This performed worse than the plain models and was expected as the lesser the samples to train with, the lower will be the accuracy. Randomly cutting off the dataset was bound to lower the accuracy.

b. SMOTE Oversampling – This technique extrapolates the minority class and generates more of such samples in the vicinity of the minority samples. This also gave a worse accuracy and AUC score than the plain model. This is probably because the overlapping of features between classes. As highlighted in point 6 in Part A(Initial Exploration) the features do not show different trends between the classes. I believe this is why oversampling did not work as the more samples got generated, there was more uncertainty in predicting the class label – more **entropy**.

c. Tomek Links Undersampling – This is an intelligent undersampler because it removes pairs of samples, one from each class that are close to the decision boundary. In the 'auto' mode, it only removes the majority class samples, thus making the boundary more prominent. This definitely helps the model prediction much easier. As expected this gave a better accuracy and AUC score than the plain models in every classifier except BalancedRandomForestClassifier because it has its own balancing mechanism, which I believe might have interfered with the TomekLinks approach.

The highest AUC at this stage: 0.8956 (LGBMClassifier and TomekLinks combination)

3. Feature Reduction:
Based on the feature importance plots, I chose to take the top 20, 18, 15 and 10 features and trained the best models on these subsets of columns.
In comparison with the plain model, the results were as follows:

- LGBM gave the best results for 18 features selected.
- XGBoost gave the better than plain model's results for 10 and 15 features, so I tried with 9,11,12 features as well. 11 Features gave the best AUC score for XGBoost.

The highest AUC at this stage: 0.8803 from LGBM and 18 features.

When combining Tomek Links and taking 18 features for LGBM and 11 for XGBoost, the AUC dropped.

The decrease in AUC and accuracy can be explained by the reduction in both sample size and feature count. It is generally better to have more relevant features and more samples, so a reduction in both performed worse than each individual approach.

4. Hyperparameter Tuning:

The next step I took was to arrive at the best hyperparameters for the two top models giving highest accuracies and two more in case I find a better version of it – XGBoost, LightGBM, GradientBoosting and BalancedRandomForest.
Using GridSearchCV I followed a systematic approach to tuning the various tree-based algorithms:

a. N_estimators and learning_rate:
Taking the default learning rate I plotted the AUC scores for a range of n_estimators, or number of trees. In the pattern I could see that the scores were highest for XGBoost with 600 trees, LGBM with 400 trees. Next, I fine-tuned the learning_rate for these n_estimators values.

b. Max_depth and Base_score:
Next I fine-tuned the max_depth values for a fixed combination of n_estimators and learning_rate. Also for the imbalanced data, 0.8 was the best base_score for XGBoost.

c. Gamma and others:
Opting for a higher gamma value reduces over-fitting and, in this case, the default value of 0 was optimal because the model was not overfitting the data as of now, in my opinion. For the original dataset I tuned colsample_by_tree and subsample which also gave the default as their optimal values.

The highest AUC for XGboost at this stage: 0.8933
XGBClassifier(learning_rate=.15,n_estimators=500,max_depth=5,base_score=.7, colsample_bytree=.98, gamma=0.16)

The highest AUC for LGBM at this stage: 0.8837
LGBMClassifier(learning_rate=0.1,n_estimators=400, max_depth=5)

Applying the above tuned models with Tomek Links Undersampling gave the best AUC Score of the lot.
AUC=0.90 approx.

5. Outlier Removal:

I removed samples on the basis of distance of its features from the mean. If a record had more than 5 features that were two standard deviations away from the mean of that column, I removed that row. This search led to only one row getting removed and did not have any impact on the AUC score or accuracy.

6. Stacking and Ensembling:

I have not attempted stacking or ensembling as I did not see good results when I attempted to use the output of LGBM as a feature and classifying using XGBoost.

7. Combination of Feature Reduction and Tuning:
Taking optimized XGBoost and tuning it further using a plot and simply looking at highest AUC scores allowed me to choose the best XGBoost and the optimal number of features. Of course, there is still scope of improving the score but that would come at the cost of overfiiting the data.

Optimal Classifier for 10 features:
XGBClassifier(random_state=7,learning_rate=.15,n_estimators=310,max_depth=5,base_score=0.8)
AUC: 0.900910

Final Thoughts

It was a brilliant assignment to play with data when completely blinded from the meaning behind features. This was purely an exercise in applying models, finding the best way to engineer the features simply based on their statistics/distribution and fine-tuning the models to give best scores.

I tried a combination of these methods to better the AUC score:

1. Combination of one-hot encoded data with feature reduction, then applying models
This might not have worked because I was manually choosing the number of features to reduce. I had plotted the feature importances after one-hot encoding and found 36 out of the 90 features to use for modeling. However this did not give a better AUC score, because I think the features were less information-rich after reduction and encoding.

2. One-hot encoding and TomekLinks undersampling:
This approach was taking too much time. This might not have worked because with the huge number of binary features, it was computationally more complex to figure out the samples to remove. I tried this on Colab's GPU and it was still running for more time than the session duration.

3. Tuning parameters after above two:
Since the approaches above did not perform better individually, they also did not perform better on tuning.

4. Interaction Terms:
Since this dataset's features were hidden(meaning of features) I did not try to find interaction terms simply based on meaning. Although I did try taking two interaction terms of features of (f8,f19) and (f13,f17). This did not work in giving better AUC or accuracy as we're reducing features that were quite low in the Gain or their correlation with the label.

In conclusion, I believe this exercise was a great learning opportunity as I tried a range of techniques to get a better AUC score. Researching the techniques, and going deep inside the documentation to optimize these algorithms showed great insights into these algorithms and data science in general. The Kaggle leaderboard proved to be a great motivator as well. This made all of us familiar with Kaggle's rules and approach to scoring Private and Public. I will continue participating in such competitions as this was truly an extremely educational assignment.