Assignment 1 - A LightGBM-Based Fraud Detection Model

Luka Beverin[1], **Sanya Anand**[1], **Prerna**[1], **and Dishani** [1]

[1]KU Leuven, Master of Statistics and Data Science, Advanced Analytics in Business

Introduction

> Our ability to manufacture fraud now exceeds our ability to detect it.
>
> *Al Pacino as Viktor Taransky*

The goal of this task is to develop a predictive model which can identify fraudulent claims, based on claims which were previously identified as being fraudulent or suspicious. Multiple reasons can make a claim suspicious or fraudulent, e.g. exaggeration of damages, staging an accident, trying to submit the same claim twice or at different insurance companies, etc. In the past many machine learning models have been proposed to tackle the problem of fraud prediction, however, most of these models were built on relatively small data sets with a large target class imbalance. In this assignment, a LightGBM-based method for fraud detection is proposed. The data set used for this model consists of car insurance claims of a Belgian insurer for retail policy-holders (i.e. not companies but individuals). The data was obtained in the years 2017 (train set) and 2018 (test set). Various performance metrics have shown that the LightGBM-based method outperforms most classical and well known methods such as Random Forest, Support Vector Machines and Logistic Regression. Furthermore, resampling techniques and bayesian optimization for fine-tuning of model hyperparameters is also explored.

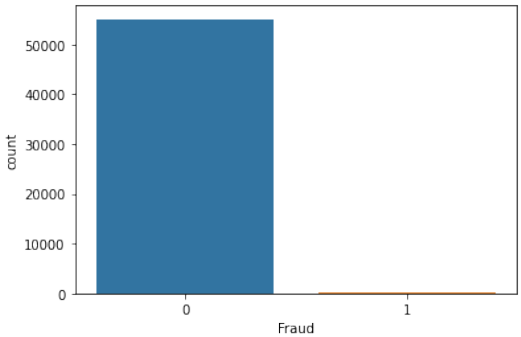

Fig. 1. Count of Fraud

The data is made up of a train set (2017) and a test set (2018), which does not contain the target variable. In the training set there are 308 instances of detected fraud and 55044 instances of non-fraud, resulting in a large class imbalance. This class imbalance is visible in the plot above. Fraud is our target variable and in the initial data set provided it was encoded as either "Y" or "N", representing fraud and non-fraud respectively. We have encoded the target variable to 1-0, where 1 represents the presence of fraud and 0 the complement of the event.

When the model is evaluated, the claim amount for the true positive cases are aggregated. This implies that models which are able to identify fraudulent high-value claims will be ranked higher than models identifying low-value fraudulent claims. The claim amount is the amount paid out by the insurance company and ultimately used to evaluate our proposed model. The claim amounts are only available in train set, as this amount is not known at claim registration. The average claim amount amongst fraudsters is 6 556.037, whereas the average is much lower for non-fraudsters with a value of 2 084.35.

Data Preprocessing

One of the first and foremost steps of any machine learning task is data preprocessing and cleaning. This involves detecting outliers, treating missing values, removing duplicates and addressing other data errors.

Dataset	No. of rows	Cases of Fraud	No. of Features
Train set	55 463	308	77
Test set	29 955	Unknown	76

Table 1. Summary of the data

Treatment of Missing Values. Missing data is a common problem in practical data analysis and the aim of this section to outline the missing value imputation that was performed on our train and test sets. The simplest approach to a missing data problem is dropping the rows with missing entries. This is usually an acceptable solution if the number of data points we have access to is sufficiently high such that dropping some of them will not cause us to lose generalizability in the models we consider. Dropping missing data that is not at random should be done with caution. In this assignment we drop columns with 50% or more missing data. Dropping these features simplifies our model, but gives us fewer features to work with.

The most common imputation method is replacing missing values with the mean, median or mode values of the data set. Features relating to the claim vehicle such as the load and power were replaced with the mean type. If the feature contained outliers then the missing values were imputed with

the median value instead of the mean.

There also exists other sophisticated missing value approaches such as Maximum likelihood imputation and Nearest neighbors imputation that should be investigated.

Feature Engineering. Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models. The features in our data directly influence the predictive models and the results achieved. The introduction of good features will allow use to use less complex models that are faster to run, easier to understand and easier to maintain.

A new feature was introduced which represents the difference between when the claim was registered and date when the accident happened. On average, fraudsters delay submitting claims by 17 days, compared to an average of 7 days amongst non-fraudsters.

Using a reference year (2017 for train set and 2018 for test set), we were also able to create other new features such as the age of the vehicle, the likely age of the policy holder and the age of the driver.

The number of years that the policy is valid was also calculated by making use of the features describing policy start and end dates.

Label Encoding. The train and test sets contain multiple categorical features of which they themselves contain multiple labels. In order for our machine learning models to better understand the data, the labels are converted into numeric form. Label enncoding is made possible with the sklearn.preprocessing package called LabelEncoder.

Model Evaluation Using Performance Metrics

Detecting fraudulent claims is considered as a classification problem which involves training a model on a data set and then using the trained model to make predictions on unseen data. Evaluation metrics are then used to quantify the performance of a predictive model.

The most commonly used threshold metric is classification accuracy:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

In our scenario we have imbalanced targets and most models will achieve a high accuracy by only predicting the majority class. Non-fraudulent claims are referred to as the majority class in this classification problem and fraudulent claims are the minority class. For this reason, a confusion matrix may provide more insight into the performance of our model. For imbalanced classification tasks, sensitivity-specificity and precision-recall metrics are the most widely used. The metrics can be calculated from the confusion matrix. Specificity

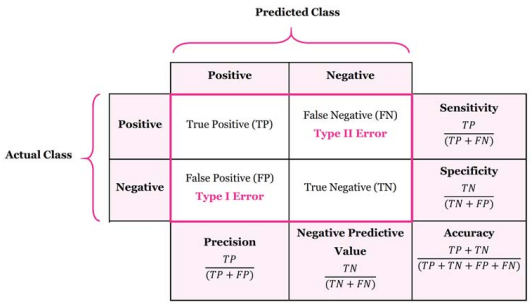

Fig. 2. Confusion Matrix

summarises how well the model predicts the negative classs (non-fraudulent cases). Sensitivity summarises how well the model predicts the positive classes (fraudulent cases).

Precision summarizes the proportion of all predicted fraudulent claims that were actually correct. Recall, or negative predicted value, is the proportion of actual fraudulent claims that were predicted correctly by the model. The F-measure then combines precision and recall and returns a harmonic mean that is popular for imbalanced classification tasks.

$$\text{F-Measure} = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision} + \text{Recall}}$$

The leader board shows the AUC, which stands for "Area under the receiver operating characteristic curve (ROC)" and is displayed for informational purposes. The ROC curve plots *True Positive Rate vs False Positive Rate* at different classification thresholds.

$$\text{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

The Precision-Recall curve reveals a more complete picture of the accuracy achieved since it focuses on the performance of the classifier on the minority class and is robust even in imbalanced data sets. Nevertheless, we consider the AUC score as a general performance measure which is used to compare models.

Methods

Support Vector Machines. Support Vector Machines (SVM) is a class of supervised machine learning methods commonly applied to data that has exactly two classes. An SVM classifies data by finding the best hyperplane that separates all data points of one class (fraud) from those of the other class (non-fraud). With help from the *sklearn* package in Python we are able to instantiate the model. All parameters of the model are set to default and the linear kernel is the kernel type used in the algorithm.

Random Forest. A Random Forest (RF) algorithm is also a supervised learning algorithm that builds multiple decision trees and aggregates them together in order to get a more stable prediction. An added feature of RFs is that we are able to measure the importance of a feature on the predictions. We only tune the number of estimators (set to 200) in our RF model, which is the number of trees the algorithm builds before taking the averages of predictions.

Logistic Regression. The Logistic Regression Model uses a logistic function and is most often used to model a binary dependent variable. That is, an outcome that can have only two possible values, either "0" or "1". The model is simpler to interpret than the aforementioned and has fewer parameters.

LightGBM. LightGBM, which stands for Light Gradient Boosting Machine, is a gradient boosting framework that uses tree based learning algorithms. It was designed to have fast training speed, higher efficiency, lower memory usage, and other impressive advantages. For this reason, the model is very popular in online Kaggle competitions.

Experiments

The train-test split technique in combination with the k-fold cross-validation procedure is used for evaluating the performance of our different machine learning models. The idea behind splitting data into a train-test (or validation set) is to use the first subset to fit the model and the second subset to evaluate the model fit. The test set provided in this assignment has unseen targets. We are therefore required to create training and validation sets from our preprocessed train set in order to evaluate and possibly fine tune our machine learning models. Below is a visual description of the k-fold cross-validation procedure. Cross-validation

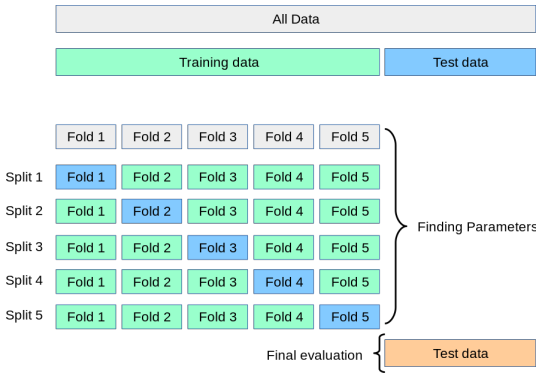

Fig. 3. Cross-valdiation

is often used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited. In our case, we use cross-validation to achieve more accurate estimates in the presence of imbalanced data. The number of folds is fixed to five for this experiment. For each fold, the AUC was calculated as well as the overall AUC average across all folds. LightGBM and Random Forest were the two best performing models.

	AUC					
	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4	Average
Model						
SVM	0.76	0.59	0.63	0.61	0.68	0.66
Random Forest	0.86	0.85	0.79	0.78	0.50	**0.75**
Logistic Regression	0.55	0.60	0.63	0.61	0.64	0.61
LightGBM	0.80	0.84	0.76	0.70	0.66	**0.75**

Table 2. Performance of different models.

A basic Neural-Network (NN) model was also explored, however, due to its poor results and lack of interpretability it was disregarded. XGBoost is another model that competes strongly with LightGBM in the data science competition world but LightGBM was preferred in this task due to its efficiency and training speed. After analysing the preliminary results (including F-Measure), LightGBM was chosen as the model to explore further in hopes of building an accurate and efficient fraud detection model with desirable results.

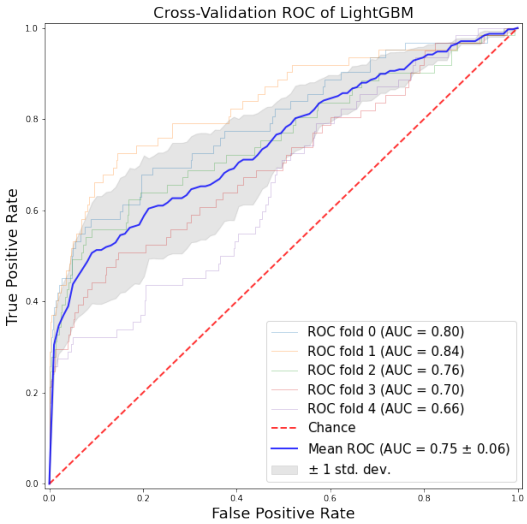

Fig. 4. ROC K-fold cross-validation

LightGBM-Based Fraud Detection Model

In this section we introduce the LightGBM model along side Bayesian fine tuning procedures, interpretability, and techniques for addressing the problem of class imbalance.

The preprocessing stage was revisited to see whether or not the AUC score could be improved. The following steps resulted in an increase of the AUC with the same parameters as before:

1. Only remove features with 50% missing values
2. Using KNN algorithm to impute remaining missing values

By default, the LightGBM will ignore missing values during a split, then allocate them to whichever side reduces the most loss. However, re-sampling algorithms cannot handle missing data, and for this reason missing values were imputed.

Within the validation set there were 92 cases of fraud, of which our first LightGBM model detected 21 correctly. Evidently there are still many cases (71) of fraud that our model did not pick up. The following sections are aimed towards improving the LightGBM classification results.

Addressing Class Imbalance. The issue with imbalanced target classes is that there are too few examples of the minority class for a model to effectively learn the decision boundary. The most common way to solve this problem is to make use of SMOTE: Synthetic Minority Over-sampling Technique. In its simplest form, SMOTE is an oversampling technique wherein synthetic samples are generated for the minority class. In our case, SMOTE would create synthetic samples of fraudulent claims.

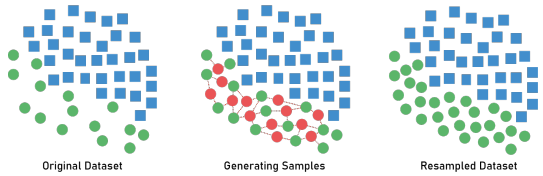

Fig. 5. SMOTE visualized

The LightGBM model (which is trained on the re-sampled data set) is then used to make predictions on the original validation set. The model that is trained on the re-sampled data

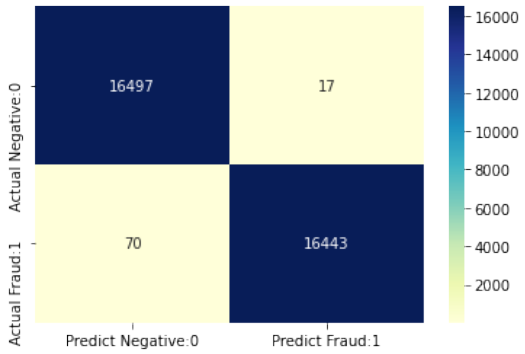

Fig. 6. SMOTE confusion matrix

seems to perform much better. The number of actual fraudulent claims that were predicted correctly increased by 48 and the miss-classifications decreased from 13 to 6.

Bayesian Optimization. The LightGBM model has certain parameters that which are not part of the training procedure. One can evaluate possible parameters using a validation set, but there's still the issue of deciding which values to try. A common approach to parameter selection is brute force search (e.g. grid search, random search), unfortunately it is very resource expensive and time consuming trying out different hyperparameter configurations. Bayesian optimization in turn brings down the time spent on searching for the best LightGBM parameters. It is used to tune key parameters such as learning rate, max tree depth, and the number of leaves.

The Bayesian optimization algorithm is applied to the

Parameter	Description	Range	Value
num_leaves	Number of leaves	[24,45]	40
feature_fraction	Fraction of features to consider	[0.1,0.9]	1
bagging_fraction	Control size of sample drawn	[0.8,1]	0.9
max_depth	Depth of the tree	[5,9]	8
lambda_l1	Regularization	[0,5]	0
lambda_l2	Regularization	[0,3]	0

Table 3. Bayesian optimization parameters

SMOTE data set in the aforementioned section. Unfortunately, the tuned parameters did not make a significant difference in our performance metrics.

Results. To investigate the effectiveness of our proposed approach for fraud detection, the 5-fold cross-validation procedure is implemented. The LightGBM model is trained with default parameters besides the number of leaves being set to 44. The model was trained on the resampled SMOTE train set and achieved an average AUC score of 0.8612 after 5-fold cross-validation. Our proposed model also did fairly well

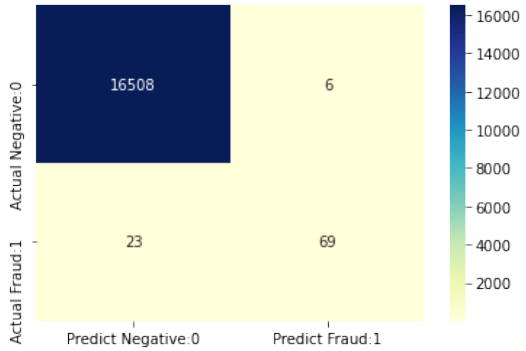

Fig. 7. SMOTE Confusion Matrix

with respect to precision and recall. The F-1 score is 0.826 and the area under the precision-recall curve 0.840. The F-1 score is an important measure, especially if the number of fraudulent and legitimate claims is not balanced.

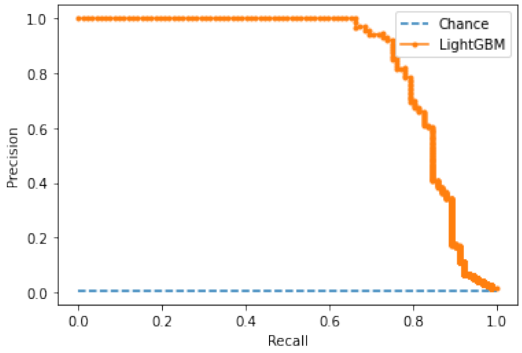

Fig. 8. Precision-Recall

Discussion

In this section we will conclude our final thoughts on the task and reflect on where we could improve, other models to consider, lessons learnt and more. Below we summarise methods and techniques we would like to explore in the future to possibly improve our results

- Improve our data preprocessing by investigating any correlated features. This also includes searching for features that are normally distributed and identifying noisy features
- Exploring dimension reduction techniques such as Principal component analysis (PCA)
- Attempt different methods for missing value imputation
- Consider other techniques for addressing the class imbalance, such as under-sampling and Methods for resampling
- More in depth work with other models such as XGBoost and Catboost
- Explore ensemble machine learning algorithms such as stacking and voting.
- Attempt different versions of encoding (hot one encoding, target encoding)
- Try different ways to find optimal parameters, such as grid search, or let our bayesian opimizer run for longer.
- Look into interpretability options
- More/better exploratory data analysis to help us get useful insights
- Focus on high claim amounts
- Optimize the model based on various metrics
- Use a different loss function when training the model
- Improve our cross-validation methods
- Explore Recursive feature elimination and other feature selection techniques
- Improve feature engineering ten-fold