Travelers Claim Fraud Detection Project Report

Yu Yang

December 21, 2018

Abstract

In this paper, we apply the weighted combination of LightGBM and XGBoost to the problem of detecting Travelers claim fraud and employ the LIME tool to help explain the model prediction result. The method effectively combines the advantage of LightGBM and XGBoost and the explanatory capacity of LIME. We presents the whole modeling process with an emphasis on feature engineering, model selection, model interpretation and classification. The final result on leaderboard reveals the model to be a valuable contribution to effective fraud diagnosing support.

1 Introduction

The Coalition Against Insurance Fraud estimates that insurance fraud costs $80 billion a year across all lines of insurance and fraud comprises about 10 percent of property-casualty insurance losses each year[3]. And undoubtedly, insurance fraud has become a costly and urgent problem. One effective way to fight fraud is to identify fraudulent claims as early as possible. And this is the start point of our project. In this paper, we focus on the diagnosis of suspicious vehicle insurance claims. The operational problem is to fully make use of the claim-related information to classify the claims into valid and fraud categories.

The rest of this article is organized as follows. Section 2 describes the main steps and methods of the modeling procedure and classification. Section 3 shows the model result and explores the business insights. Section 4 discusses the gaining from this competition. Finally, conclusion are drawn in Section 5.

2 Modeling Method

The whole modeling process is seperated into several stages. In the first stage, we clean the data and then do exploratory data analysis to find the patterns of the data. In the second stage, we do feature engineering based on the EDA and our domain knowledge about insurance fraud. In this stage, we add new features from external resources and transform the existing features into new ones. In the third stage, we fit different models to the dataset and use model combination to combine the well-performed models. During this stage, we focus on parameter tuning and feature selection. In the forth stage, we use a tool to help explain the model. And in the last stage, we use the model prediction to do classification with cost ratio being 5:1. Throughout the whole modeling procedure, cross validation is employed to ensure the validity and avoid overfitting, and AUC is used as the metric of evaluation.

2.1 Data Cleaning and EDA

In this stage, we work on data cleaning and exploratory data analysis to prepare for the later modeling.

For data cleaning, we follow these steps: (1)cope with unrealistic data; (2)encode categorical variables; (3)impute missing values. In the first step, we drop the observations with fraud = -1, set annual income which is negative and age which is over 100 to be missing values. In the second step, we use binary encoding and one-hot encoding to transform categorical variables into numerical variables in order to fit them into the model. In the last step, we use mode to impute for categorical variables and mean for numerical variables.

Exploratory data analysis is vital to help identify the patterns of features, and thus help with feature engineering. For example, the curvature in scatterplots, the skewness in histograms, and the difference among types revealed by kernel density curves can tell us a lot about how we could

transform the features and which features might be helpful for building the model. In this stage, we plot scatterplots, histograms, kernel density curves, and correlation matrix, as shown in Appendix A Figure 1, 2, 3, 4, 5.

2.2 Feature Engineering

Feature engineering is the key to a good model. It requires us to apply domain knowledge to create new features that allow machine learning algorithms to work better. At the same time, it is difficult and time-consuming. We implement two strategies for feature engineering: (1)use domain knowledge to find external resources; (2)transform existing features to better capture the relationship between the explanatory features and the response. The followings are the details.

To our knowledge, when the economy began to decline, it is expected that insurance fraud would most likely increase. Considering the data accessibility and representativity, we choose three macroeconomic indices to reflect the economic dimension of claim fraud: interest rate, S&P 500[1], and state unemployment rate[2]. Also, to better make use of the zip_code information, we transform the zip code into state, latitude, longitude[3]. With these three new features, we have a more detailed version of geographical information. And as revealed by the kernel density curve shown in Appendix A Figure 4, most of the data are collected from approximately five regions.

Apart from that, during the EDA, we find the former part of safety rating shows relatively large variation. So we try do some transformation to lower down the variance and in that way, we might gain more explanatory capacity. And it turns out the transformation defined by

$$rating\ per\ claim = \frac{safety\ rating}{past\ number\ of\ claims + 1}$$

could greatly reduce the variance, as shown in Appendix A Figure 1. The reasons we choose to add 1 are: (1) to avoid problem when the past number of claims = 0 ;(2) adding 1 gives the largest performance boost according to the CV result. Additionally, the number of missing values also contains valuable information of features, so we include the count of missing values as a new feature.

According to the 5-fold CV AUC result shown in Appendix B Table 1, we could see that some of the new features indeed boost the performance.

Another thing that worth mentioning is that, when exploring plausible new features, we try to use grouped-by means as the new features. Namely, separate the variables into groups and then use the grouped mean values as the new features. Adding these grouped-by means does improve the performance on the local CV AUC. However, the result on the public leaderboard is not satisfying. The main reason might be that the grouped-by means are specific to the training dataset and that the grouped-by means are highly correlated with the existing features and therefore would lower down the entire explanatory capacity.

2.3 Models

The whole modeling procedure could be split into several stages: model fitting, model comparison, and model combination.

2.3.1 Model Fitting

Choosing the right model paves the way to accurate prediction. Gradient boosting methods, such as AdaBoost[4], CatBoost[2], XGBoost[1], and LightGBM[5], have been widely used and suggested in both classification and regression problems, especially in the Kaggle competitions. Compared to them, traditional methods such as logistic regression and random forest are not as popular due to the ineffectiveness. In our project, we consider both the gradient boosting methods, including AdaBoost, XGBoost and LightGBM, and the traditional logistic regression and random forest model.

One of the most important part for gradient boosting methods is parameter tuning. We utilize two tuning strategies: manual tuning and Bayesian tuning. We choose the tuning method based on its CV AUC performance. XGBoost and LightGBM are tuned using manual tuning, and AdaBoost uses Bayesian tuning. For manual tuning, we tuned the parameters step by step. In each step, we use

[1]Interest rate and S&P 500 Index are obtained from "https://finance.yahoo.com/quote/%5EGSPC/"

[2] Unemployment rate is obtained from "https://www.bls.gov/home.htm"

[3] Latitude and longitude are obtained from "https://www.zipcodedatabase.org/zip-code-database/" using the zip-code information

grid search to find the optimal parameter with the other parameters being controlled. The criterion to choose the optimal is CV AUC score. And to avoid overfitting, we tune the parameters towards the direction of increasing the penalty for overfitting.

After parameter tuning, we select features that are most conducive to modeling. Since there might be correlations among features, which would reduce the explanatory capacity, and some features, for example, the grouped-by means, would cause overfitting problem, it is not a good idea to utilize as many features as possible. Therefore, to achieve a better performance, we need to select features. The main strategy is to delete features based on the feature importance given by the model fitting result and then check whether or not there is a big change in the updated AUC. We also use manual selection to find the best combination of features. The features included in the final model are shown in Appendix B Table 2.

2.3.2 Model Comparison

After fitting the model, the next task is to select models based on their performances. We use the cross validation AUC score as the metric to evaluate these models. Among the five models, XGBoost and LightGBM outperform the others, which coincides with our findings that gradient boosting methods are usually the winners in Kaggle competitions. Therefore, we pick out XGBoost and LightGBM for model combination.

2.3.3 Model Combination

Model combination idea is applied to further boost the performance of the models. There are several ways to combine models: Bagging, Boosting, Stacking and averaging. We consider stacking and averaging strategies. The result given by stacking doesn't even beat the single LightGBM model, so we drop it. Instead, we resort to the simple averaging idea. Based on the CV result, we find that the best combining weight is 60% LightGBM + 40% XGBoost. The final performance rank using CV is: LightGBM + XGBoost > LightGBM > XGBoost > Adaboost > Random Forest > Logistic Regression, as shown in Appendix A Figure 6. The submissions to Kaggle also show the same result. Therefore, 60% LightGBM + 40% XGBoost is our ultimate model.

2.4 Model Explanation Tool

Another shining point of our work is that we employ the LIME[6] tool to help explain the model. We all know that tree-based models are mostly black boxes. We could get the prediction, but we don't know why the model gives us such a result. LIME is used to solve such a problem. With this tool, we could tell which features contribute most to the model decision. Figure 8 and Figure 9 display the explanations for the correctly and incorrectly classified cases respectively. With this tool, our model could better help the employees in Travelers check the claims and make decisions.

2.5 Classification

The basic idea to do classification is to use CV to choose the optimal threshold. Since the best model has already been set, the optimal parameters for XGBoost and LightGBM could be directly used here.

The algorithm for choosing the optimal threshold is shown in Algorithm 1. Firstly, set the range of threshold and the range of seed. And for each threshold, split the train data into the training part and the validation part(7:3) according to different seed. Next, train the model on the training set and calculate the cost on the validation set for each seed. Then, calculate the average of the cost under different seeds for every single threshold. Finally, choose the threshold that corresponds to the smallest average cost and apply this threshold to the test dataset to get the final prediction. The estimate of the total cost of the predictions is obtained by scaling according to the formula

$$cost\_test = cost\_validation * \frac{size\ of\ test\ data}{0.3 * size\ of\ train\ data}$$

Algorithm 1 Algorithm for Optimal threshold

for threshold ∈ threshold list **do**
  for seed ∈ seed list **do**
    Step 1: split train data into training part and validation part according to the seed.
    Step 2: train model on the training part and calculate the cost on the validation part.
  end for
  Step 3: calculate the average cost for this threshold.
end for
Final Step: Choose the threshold that corresponds to the smallest average cost.

3 Results and Insights

3.1 Results

Our final model is 60% LightGBM + 40% XGBoost, and it achieves 0.74961 AUC on Public Leaderboard and 0.75462 AUC on Private Leaderboard, which is relatively high. And the estimated classification cost on the test dataset is 6483.6218. The variables in the final model are shown in Appendix B Table 2. The heat map shown in Appendix A Figure 7 is obtained from the feature importance results given by LightGBM, XGBoost, and AdaBoost. We could check from the heat map that these variables are important in predicting claim fraud: claim_est_payout, liab_prct, age_of_driver, accident_site_parking_lot, age_of_vehicle, rating_per_claim. These features give us a basic idea about what information should be paid special attention to when investigating a suspicious claim.

3.2 Insights

Based on the results of the model, we could find that:

- The vehicle insurance claim fraud is tightly connected with the economic state. And among the three macroeconomic indices, interest rate has the highest explanatory capacity. For future modeling, Travelers might need to take more economic indices into account to better predict claim fraud.
- As our model reveals, state is not included in the final model but latitude and longitude could help with prediction. This suggests that localized geographic location may have better predicting ability. More accurate geographical information might further improve the prediction accuracy.
- Both liab_prct and rating_per_claim reflect the claimer's credit level and they are significantly important in the model as shown in Appendix A Figure 7. Therefore, we may conclude that credit level is associated with claim fraud and if possible, we should collect more credit-related information, such as credit score and mortgage.

4 Gain from the Competition

There are a lot worth learning in the presentations of other groups. The following is what we need to consider in future modeling.

- In actual business modeling, despite of the prediction accuracy, we also need to consider the time consumption. CatBoost is a faster choice than LightGBM and XGBoost, so we may need to consider this model as well.
- When doing stacking, the models in the same layer should have similar performances and the predictions ought to be somewhat different. If some badly performed models are included in the layer, the whole predicting level will be lowered down. And if the prediction results are very close to each other, it doesn't make much sense to combine them. To check that, we could calculate the correlation between the prediction results.
- Other methods could also be used to get feature importance: random forest, Chi-square, Gain ratio, and Kruskal test. And when coping with categorical variables, we might use count,

supervised ratio, and WOE as well. Also, despite of simply use mode and mean to impute, we could try random forest and kNN. Additionally, when the data is unbalanced, we could use oversampling and undersampling strategy to do data augmentation.

- Other possible features could be considered in future modeling: credit score, mortgage, criminal history, convenience to commit fraud claim.

And from the competition outcomes, we could find that sometimes the best performance on the public leaderboard doesn't necessarily ensure the best on the private leaderboard. The reason might be that the competitors tend to choose the model that performs best on the public leaderboard and this may lead to overfitting. Another point is that as long as we pay special attention to avoid overfitting during the modeling process, for example, tune the penalty parameters and use cross-validation, the difference between the public and private leaderboard would not be large.

5 Conclusion

In this paper, the modeling procedure is discussed in detail and the combination of XGBoost and LightGBM is used as the final model to predict the probability of fraudulent claim. The idea of cross validation is applied throughout the whole procedure to avoid overfitting and the LIME tool greatly help explain the model prediction. Specifically, the significant features help us better understand the claim fraud phenomenon. In sum, our model is generally applicable and offers relatively accurate prediction and interpretable decision support.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[2] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[3] Coalition Against Insurance Fraud. By the numbers: fraud statistics. `https://www.insurancefraud.org/statistics.htm#1`. Accessed: 2018-12-20.

[4] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[6] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

Appendices

A Figures

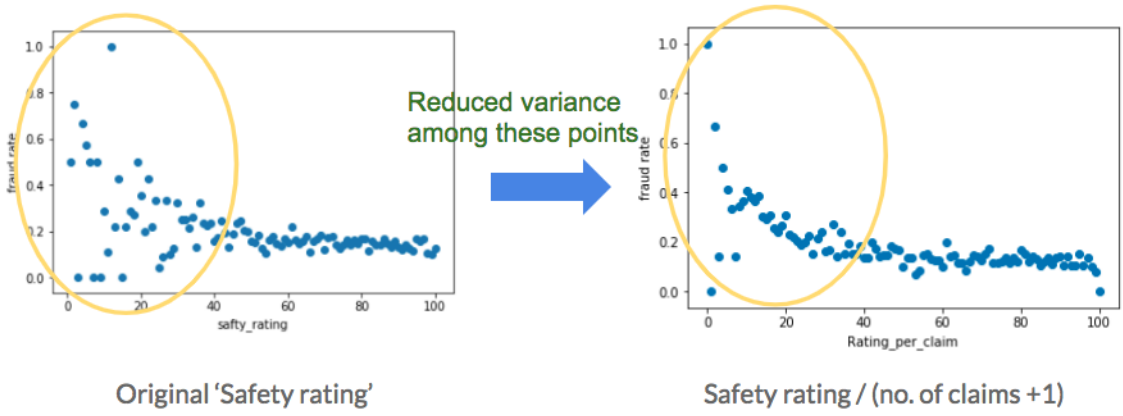
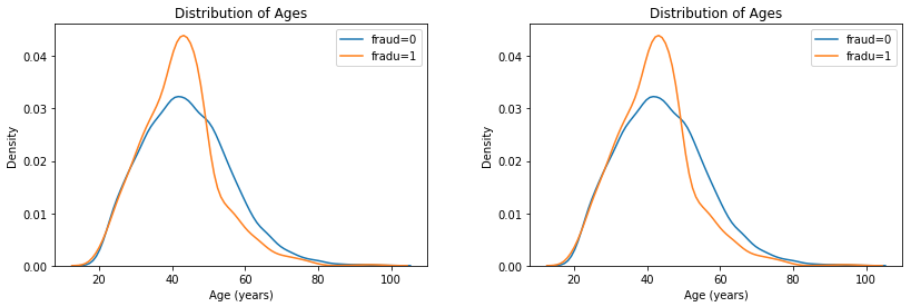
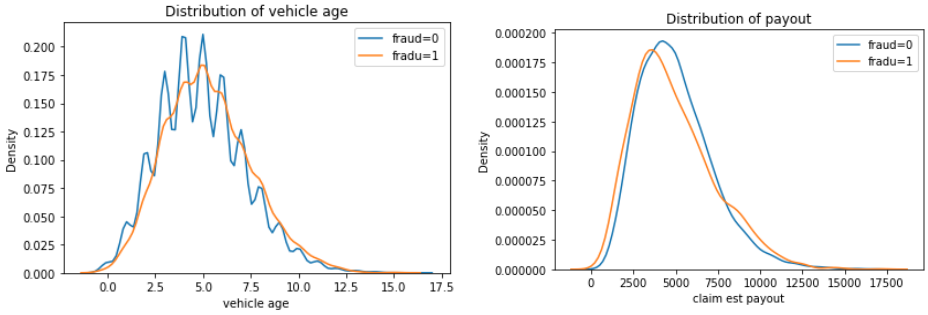


Figure 1: Rating Per Claim



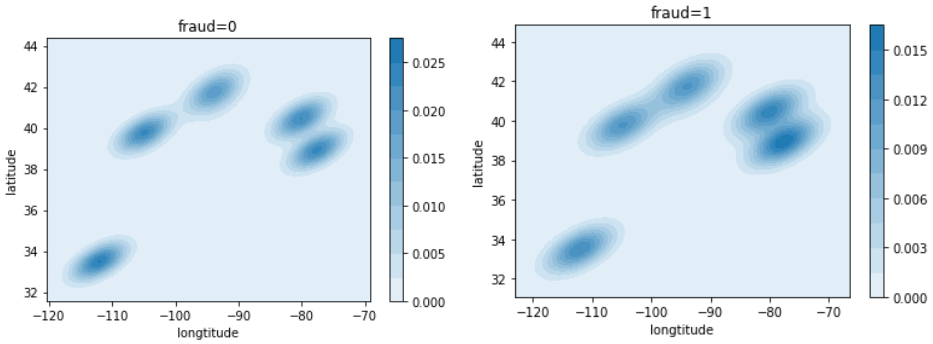
(a) Age of driver (b) Annual Income

Figure 2: Kernel density curves



(a) Vehicle Age (b) Claim estimated payout

Figure 3: Kernel density curves



(a) Fraud0: Lat/lon (b) Fraud1: Lat/lon

Figure 4: Kernel density curves

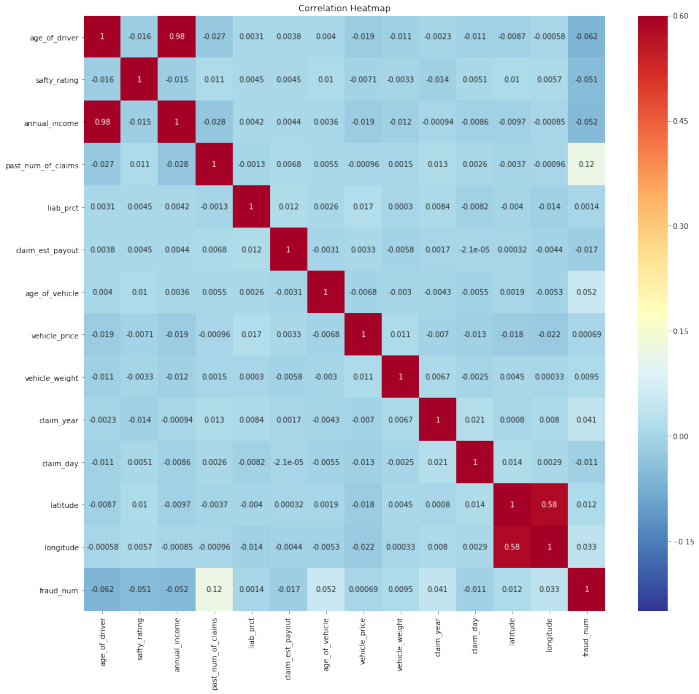


Figure 5: Correlation Heatmap

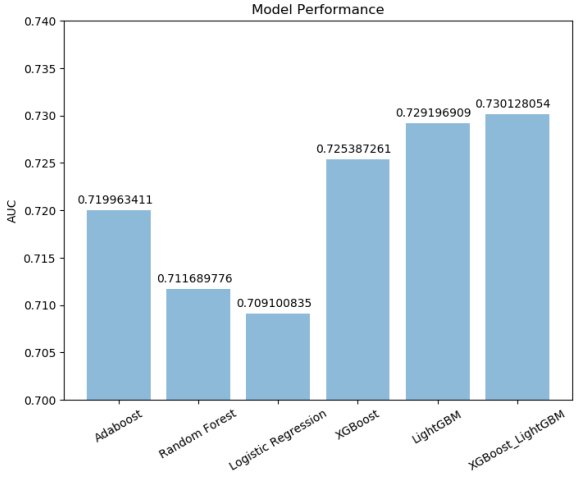


Figure 6: Model Performance

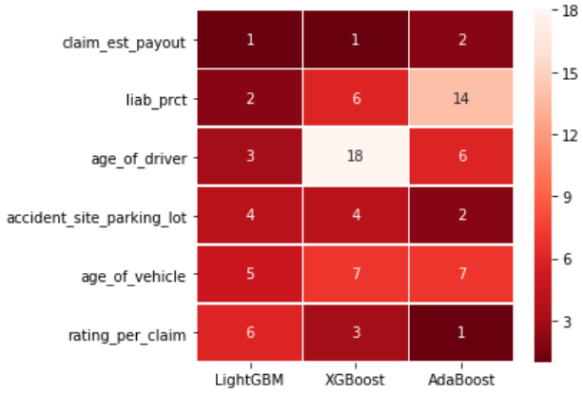


Figure 7: Feature Importance Heatmap

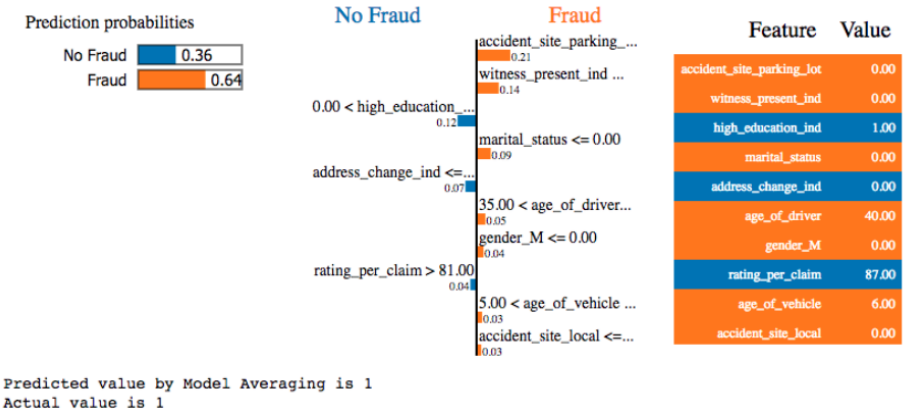


Figure 8: LIME Explanation: Correctly Classification Case

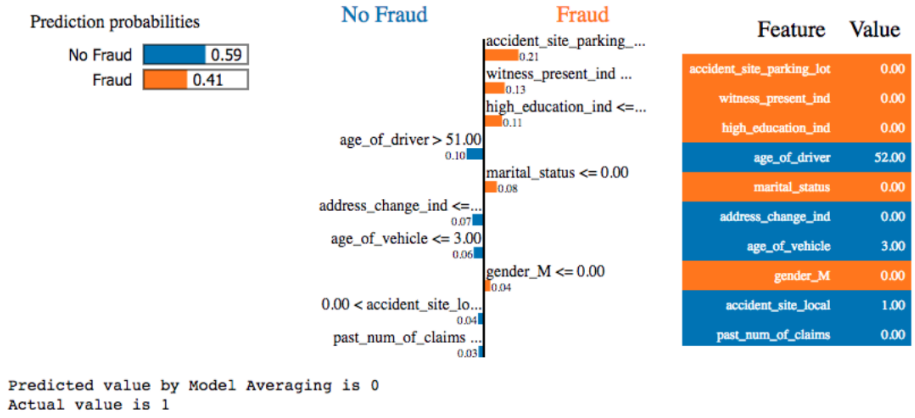


Figure 9: LIME Explanation: Incorrectly Classification Case

B Tables

Table 1: New Feature Performace Boost

Feature	**Performance Boost**
Latitude and longitude	0.001
Rating Per Claim	0.0017
Interest Rate	0.0007
Count of missing values	0.0004

Table 2: Features decision in the final model

Existing features deleted	claim_date, vehicle_color, zip_code
New features deleted	claim_day, claim_month, claim_year, weekday, SP_Index, unemployment rate, state
New features remained	latitude, longitude, interest_rate, rating_per_claim
Features in the final model	age_of_driver, gender, marital_status, safty_rating, annual_income, high_education_ind, address_change_ind, living_status, past_num_of_claims, witness_present_ind, liab_prct, policy_report_filed_ind, claim_est_payout, age_of_vehicle, vehicle_price, vehicle_weight, fraud, latitude, longitude, accident_site_local, accident_site_parking_lot, channel_online, channel_phone, vehicle_category_large, vehicle_category_medium, Interest_rate, rating_per_claim

C Python code

C.1 XGBoost.py

```
###### Import packages ######
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn import model_selection
from skopt import BayesSearchCV
from matplotlib import pyplot
from xgboost import plot_importance
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

###### Feature manipulation ######
train = pd.read_csv("train_data_clean_4_grouped.csv")
test = pd.read_csv("test_data_clean_4_grouped.csv")

# Encode gender and living status and state ######
train["living_status"] = pd.Categorical(train["living_status"])
train["gender"] = np.where(train["gender"].str.contains("M"), 1, 0)
train["living_status"] = np.where(train["living_status"].str.contains("Rent"), 1, 0)

test["living_status"] = pd.Categorical(test["living_status"])
test["gender"] = np.where(test["gender"].str.contains("M"), 1, 0)
test["living_status"] = np.where(test["living_status"].str.contains("Rent"), 1, 0)

# one-hot encoding for site of state
state_dummies = pd.get_dummies(test['state'],
                               prefix='state', drop_first=True)
test = pd.concat([test, state_dummies], axis=1)
test.drop(["state"], axis=1, inplace=True)

# one-hot encoding for site of state
state_dummies = pd.get_dummies(train['state'],
                               prefix='state', drop_first=True)
train = pd.concat([train, state_dummies], axis=1)
train.drop(["state"], axis=1, inplace=True)

# Drop month, day and year data, drop vehicle color, zipcode, claim_date, claim_number and SP_Index ######
train.drop(["claim_month_january", "claim_month_february", "claim_month_march", "claim_month_may",
            "claim_month_june", "claim_month_july", "claim_month_august", "claim_month_september",
            "claim_month_october", "claim_month_november", "claim_month_december",
            "claim_day_monday", "claim_day_tuesday", "claim_day_wednesday", "claim_day_thursday",
             "claim_day_saturday", "claim_day_sunday", "claim_year", "claim_day",
            "zip_code", "claim_date", "claim_number", 'SP_Index', "vehicle_color_blue",
             "vehicle_color_gray", "vehicle_color_other", "vehicle_color_red",
            "vehicle_color_silver", "vehicle_color_white"], axis =1, inplace=True)
```

```
test.drop(["claim_month_january", "claim_month_february", "claim_month_march", "claim_month_may",
              "claim_month_june", "claim_month_july", "claim_month_august", "claim_month_september",
              "claim_month_october", "claim_month_november", "claim_month_december",
              "claim_day_monday", "claim_day_tuesday", "claim_day_wednesday", "claim_day_thursday",
               "claim_day_saturday", "claim_day_sunday", "claim_year", "claim_day",
              "zip_code", "claim_date", "claim_number", 'SP_Index', "vehicle_color_blue",
               "vehicle_color_gray", "vehicle_color_other", "vehicle_color_red",
              "vehicle_color_silver", "vehicle_color_white"], axis =1, inplace=True)


# Add saftyrating/(number of past claim) feature ######
train['per_saftyrating'] = train['safty_rating']/(train['past_num_of_claims']+1)
test['per_saftyrating'] = test['safty_rating']/(test['past_num_of_claims']+1)


# Delete some fraud_mean variables ######
train.drop(["fraud_gender", "fraud_marital_status", "fraud_high_education_ind", "fraud_address_change_ind",
              "fraud_living_status", "fraud_zip_code", "fraud_claim_date", "fraud_witness_present_ind",
              "fraud_policy_report_filed_ind", "fraud_accident_site", "fraud_channel", "fraud_vehicle_category",
           "fraud_vehicle_color", "fraud_state","Unem_rate"],
              axis = 1, inplace = True)
test.drop(["fraud_gender", "fraud_marital_status", "fraud_high_education_ind", "fraud_address_change_ind",
              "fraud_living_status", "fraud_zip_code", "fraud_claim_date", "fraud_witness_present_ind",
              "fraud_policy_report_filed_ind", "fraud_accident_site", "fraud_channel", "fraud_vehicle_category",
          "fraud_vehicle_color", "fraud_state", "Unem_rate"],
              axis = 1, inplace = True)
train = train.filter(regex="^(?!state_).*$")
test = test.filter(regex="^(?!state_).*$")



###### Final Model######
y = train["fraud"]
X = train.drop("fraud", 1)
clf = xgb.XGBClassifier(max_depth=3,
            learning_rate=0.06,
            n_estimators=180,
            silent=True,
            objective='binary:logistic',
            gamma=0.35,
            min_child_weight=5,
            max_delta_step=0,
            subsample=0.8,
            colsample_bytree=0.785,
            colsample_bylevel=1,
            reg_alpha=0.01,
            reg_lambda=1,
            scale_pos_weight=1,
            seed=1440,
            missing=None)

## CV AUC score
scores = model_selection.cross_val_score(clf, X.values, y.values, cv = 6, scoring = 'roc_auc')
print(scores)
print("AUC: %0.4f (+/— %0.4f)" % (scores.mean(), scores.std()))

## Final prediction on the test data
clf.fit(X, y)
```

```
y_pred = clf.predict_proba(test)[:,1]
test = pd.read_csv("test_data_clean2.csv")
test_output = {'claim_number':test['claim_number'], 'fraud':y_pred}
test_output = pd.DataFrame(data = test_output)
test_output = test_output.set_index('claim_number')
test_output.to_csv("predictions/prediction_xgboost4.csv")

## Feature importance
plot_importance(clf)
pyplot.show()
```

C.2 LightGBM.py

```
##### Import packages ######
import numpy as np
import pandas as pd
from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, VotingClassifier)
from mlxtend.classifier import StackingCVClassifier
from lightgbm import LGBMClassifier
import lightgbm as lgb
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score


##### Data Manipulation ######
# read full training data set
df_train = pd.read_csv('../data/train_data_clean_5_grouped.csv')
gender_dummies = pd.get_dummies(df_train['gender'],
                                prefix = 'gender', drop_first = True)
df_train = pd.concat([df_train, gender_dummies], axis = 1)
df_train.drop(["gender"], axis = 1, inplace = True)

living_status_dummies = pd.get_dummies(df_train['living_status'],
                                prefix = 'living_status', drop_first = True)
df_train = pd.concat([df_train, living_status_dummies], axis = 1)
df_train.drop(["living_status"], axis = 1, inplace = True)

state_dummies = pd.get_dummies(df_train['state'],
                                 prefix = 'state', drop_first = True)
df_train = pd.concat([df_train, state_dummies], axis = 1)
df_train.drop(["state"], axis = 1, inplace = True)

df_train = df_train.sample(frac=1, random_state=5)
df_train['new_param'] = df_train.apply(lambda col: col['safty_rating']/(col['past_num_of_claims']+1), axis=1)
#df_train['prct_payout'] = df_train.apply(lambda col: col['claim_est_payout']/(col['annual_income']), axis=1)
#df_train['age_over_safety'] = df_train.apply(lambda col: col['age_of_driver']/(col['safty_rating']+1), axis=1)
df_train.set_index('claim_number', inplace=True)
df_train.sort_index(inplace=True)
df_train.drop(['claim_date','fraud_claim_date','fraud_zip_code',
        "fraud_gender", "fraud_marital_status", 'fraud_accident_site', 'fraud_high_education_ind',
         "fraud_address_change_ind", "fraud_living_status", "fraud_witness_present_ind",
         "fraud_policy_report_filed_ind", "fraud_channel", "fraud_vehicle_category",
```

```
        'fraud_vehicle_color', 'fraud_state', 'SP_Index', 'Unem_rate'], axis = 1, inplace = True)
df_train = df_train.filter(regex="^(?!state_).*$")
df_train = df_train.filter(regex="^(?!vehicle_color_).*$")
df_train = df_train.filter(regex="^(?!claim_day_).*$")
df_train = df_train.filter(regex="^(?!claim_month_).*$")

train_lgb = df_train.copy()


# read full testing data set
df_test = pd.read_csv('../data/test_data_clean_5_grouped.csv')
gender_dummies = pd.get_dummies(df_test['gender'],
                                prefix = 'gender', drop_first = True)
df_test = pd.concat([df_test, gender_dummies], axis = 1)
df_test.drop(["gender"], axis = 1, inplace = True)

living_status_dummies = pd.get_dummies(df_test['living_status'],
                                prefix = 'living_status', drop_first = True)
df_test = pd.concat([df_test, living_status_dummies], axis = 1)
df_test.drop(["living_status"], axis = 1, inplace = True)

state_dummies = pd.get_dummies(df_test['state'],
                                 prefix = 'state', drop_first = True)
df_test = pd.concat([df_test, state_dummies], axis = 1)
df_test.drop(["state"], axis = 1, inplace = True)

#df_test = df_test.sample(frac=1, random_state=5)
df_test['new_param'] = df_test.apply(lambda col: col['safty_rating']/(col['past_num_of_claims']+1), axis=1)
#df_test['prct_payout'] = df_test.apply(lambda col: col['claim_est_payout']/(col['annual_income']), axis=1)
#df_test['age_over_safety'] = df_test.apply(lambda col: col['age_of_driver']/(col['safty_rating']+1), axis=1)

df_test.set_index('claim_number', inplace=True)
df_test.sort_index(inplace=True)
df_test.drop(['claim_date','fraud_claim_date','fraud_zip_code',
        "fraud_gender", "fraud_marital_status", 'fraud_accident_site', 'fraud_high_education_ind',
         "fraud_address_change_ind", "fraud_living_status", "fraud_witness_present_ind",
         "fraud_policy_report_filed_ind", "fraud_channel", "fraud_vehicle_category",
         'fraud_vehicle_color', 'fraud_state', 'SP_Index', 'Unem_rate'], axis = 1, inplace = True)
df_test = df_test.filter(regex="^(?!state_).*$")
df_test = df_test.filter(regex="^(?!vehicle_color_).*$")
df_test = df_test.filter(regex="^(?!claim_day_).*$")
df_test = df_test.filter(regex="^(?!claim_month_).*$")

test_lgb = df_test.copy()


##### Final Model ######
lgbm_params = {'boosting_type':'gbdt',  'objective':'binary', 'num_boost_round':800,
               'feature_fraction': .321, 'bagging_fraction':0.50, 'min_child_samples':100,
               'min_child_weigh':35, 'max_depth':3, 'num_leaves':2, 'learing_rate':0.15,
               'reg_alpha':5, 'reg_lambda': 1.1, 'metric':'auc', 'max_bin': 52,
               'colsample_bytree': 0.9, 'subsample': 0.8, 'is_unbalance': 'true'
}

y_train = train_lgb["fraud"]
X_train = train_lgb.drop("fraud", 1)

lgbm = LGBMClassifier(**lgbm_params)
```

```
lgbm.fit(X_train.values, y_train.values)
y_preds = lgbm.predict_proba(test_lgb.values)[:,1]

test_lgb['fraud'] = y_preds
results = test_lgb.filter(['fraud'], axis=1)
results.to_csv('results_12_6—3.csv', header=True)
```

C.3 Combination.py

```
###### Import packages ######
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn import model_selection
from skopt import BayesSearchCV
from matplotlib import pyplot
from xgboost import plot_importance
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

###### Feature manipulation ######
train = pd.read_csv("train_data_clean_4_grouped.csv")
test = pd.read_csv("test_data_clean_4_grouped.csv")

# Encode gender and living status and state ######
train["living_status"] = pd.Categorical(train["living_status"])
train["gender"] = np.where(train["gender"].str.contains("M"), 1, 0)
train["living_status"] = np.where(train["living_status"].str.contains("Rent"), 1, 0)

test["living_status"] = pd.Categorical(test["living_status"])
test["gender"] = np.where(test["gender"].str.contains("M"), 1, 0)
test["living_status"] = np.where(test["living_status"].str.contains("Rent"), 1, 0)

# one-hot encoding for site of state
state_dummies = pd.get_dummies(test['state'],
                               prefix='state', drop_first=True)
test = pd.concat([test, state_dummies], axis=1)
test.drop(["state"], axis=1, inplace=True)

# one-hot encoding for site of state
state_dummies = pd.get_dummies(train['state'],
                               prefix='state', drop_first=True)
train = pd.concat([train, state_dummies], axis=1)
train.drop(["state"], axis=1, inplace=True)

# Drop month, day and year data, drop vehicle color, zipcode, claim_date, claim_number and SP_Index ######
train.drop(["claim_month_january", "claim_month_february", "claim_month_march", "claim_month_may",
            "claim_month_june", "claim_month_july", "claim_month_august", "claim_month_september",
            "claim_month_october", "claim_month_november", "claim_month_december",
```

```
              "claim_day_monday", "claim_day_tuesday", "claim_day_wednesday", "claim_day_thursday",
               "claim_day_saturday", "claim_day_sunday", "claim_year", "claim_day",
              "zip_code", "claim_date", "claim_number", 'SP_Index', "vehicle_color_blue",
               "vehicle_color_gray", "vehicle_color_other", "vehicle_color_red",
              "vehicle_color_silver", "vehicle_color_white"], axis =1, inplace=True)

test.drop(["claim_month_january", "claim_month_february", "claim_month_march", "claim_month_may",
              "claim_month_june", "claim_month_july", "claim_month_august", "claim_month_september",
              "claim_month_october", "claim_month_november", "claim_month_december",
              "claim_day_monday", "claim_day_tuesday", "claim_day_wednesday", "claim_day_thursday",
               "claim_day_saturday", "claim_day_sunday", "claim_year", "claim_day",
              "zip_code", "claim_date", "claim_number", 'SP_Index', "vehicle_color_blue",
               "vehicle_color_gray", "vehicle_color_other", "vehicle_color_red",
              "vehicle_color_silver", "vehicle_color_white"], axis =1, inplace=True)


# Add saftyrating/(number of past claim) feature ######
train['per_saftyrating'] = train['safty_rating']/(train['past_num_of_claims']+1)
test['per_saftyrating'] = test['safty_rating']/(test['past_num_of_claims']+1)


# Delete some fraud_mean variables ######
train.drop(["fraud_gender", "fraud_marital_status", "fraud_high_education_ind", "fraud_address_change_ind",
              "fraud_living_status", "fraud_zip_code", "fraud_claim_date", "fraud_witness_present_ind",
              "fraud_policy_report_filed_ind", "fraud_accident_site", "fraud_channel", "fraud_vehicle_category",
          "fraud_vehicle_color", "fraud_state","Unem_rate"],
              axis = 1, inplace = True)
test.drop(["fraud_gender", "fraud_marital_status", "fraud_high_education_ind", "fraud_address_change_ind",
              "fraud_living_status", "fraud_zip_code", "fraud_claim_date", "fraud_witness_present_ind",
              "fraud_policy_report_filed_ind", "fraud_accident_site", "fraud_channel", "fraud_vehicle_category",
         "fraud_vehicle_color", "fraud_state", "Unem_rate"],
              axis = 1, inplace = True)
train = train.filter(regex="^(?!state_).*$")
test = test.filter(regex="^(?!state_).*$")



###### Final Model######
y = train["fraud"]
X = train.drop("fraud", 1)
clf = xgb.XGBClassifier(max_depth=3,
            learning_rate=0.06,
            n_estimators=180,
            silent=True,
            objective='binary:logistic',
            gamma=0.35,
            min_child_weight=5,
            max_delta_step=0,
            subsample=0.8,
            colsample_bytree=0.785,
            colsample_bylevel=1,
            reg_alpha=0.01,
            reg_lambda=1,
            scale_pos_weight=1,
            seed=1440,
            missing=None)
```

##CV AUC score

```
scores = model_selection.cross_val_score(clf, X.values, y.values, cv = 6, scoring = 'roc_auc')
print(scores)
print("AUC: %0.4f (+/- %0.4f)" % (scores.mean(), scores.std()))

## Final prediction on the test data
clf.fit(X, y)
y_pred = clf.predict_proba(test)[:,1]
test = pd.read_csv("test_data_clean2.csv")
test_output = {'claim_number':test['claim_number'], 'fraud':y_pred}
test_output = pd.DataFrame(data = test_output)
test_output = test_output.set_index('claim_number')
test_output.to_csv("predictions/prediction_xgboost4.csv")

## Feature importance
plot_importance(clf)
pyplot.show()
```

C.4 Classification.py

```
###### Import packages ######
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn import model_selection
from skopt import BayesSearchCV
from matplotlib import pyplot
from xgboost import plot_importance
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc


from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, VotingClassifier)
from mlxtend.classifier import StackingCVClassifier
from lightgbm import LGBMClassifier
import lightgbm as lgb

from random import sample
import random
from statistics import mean


##### Transform dataset for XGBoost and LightGBM ######

### XGBoost
# Import Data
```

```
train = pd.read_csv("../data/train_data_clean_4_grouped.csv")
test = pd.read_csv("../data/test_data_clean_4_grouped.csv")

# Encode gender and living status and state
train["living_status"] = pd.Categorical(train["living_status"])
train["gender"] = np.where(train["gender"].str.contains("M"), 1, 0)
train["living_status"] = np.where(train["living_status"].str.contains("Rent"), 1, 0)

test["living_status"] = pd.Categorical(test["living_status"])
test["gender"] = np.where(test["gender"].str.contains("M"), 1, 0)
test["living_status"] = np.where(test["living_status"].str.contains("Rent"), 1, 0)

# one-hot encoding for site of state
state_dummies = pd.get_dummies(test['state'],
                               prefix='state', drop_first=True)
test = pd.concat([test, state_dummies], axis=1)
test.drop(["state"], axis=1, inplace=True)

# one-hot encoding for site of state
state_dummies = pd.get_dummies(train['state'],
                               prefix='state', drop_first=True)
train = pd.concat([train, state_dummies], axis=1)
train.drop(["state"], axis=1, inplace=True)


# Drop month, day and year data, drop vehicle color, zipcode, claim_date, claim_number and SP_Index #####
train.drop(["claim_month_january", "claim_month_february", "claim_month_march", "claim_month_may",
            "claim_month_june", "claim_month_july", "claim_month_august", "claim_month_september",
            "claim_month_october", "claim_month_november", "claim_month_december",
            "claim_day_monday", "claim_day_tuesday", "claim_day_wednesday", "claim_day_thursday",
             "claim_day_saturday", "claim_day_sunday", "claim_year", "claim_day",
            "zip_code", "claim_date", "claim_number", 'SP_Index', "vehicle_color_blue",
             "vehicle_color_gray", "vehicle_color_other", "vehicle_color_red",
            "vehicle_color_silver", "vehicle_color_white"], axis =1, inplace=True)

test.drop(["claim_month_january", "claim_month_february", "claim_month_march", "claim_month_may",
            "claim_month_june", "claim_month_july", "claim_month_august", "claim_month_september",
            "claim_month_october", "claim_month_november", "claim_month_december",
            "claim_day_monday", "claim_day_tuesday", "claim_day_wednesday", "claim_day_thursday",
             "claim_day_saturday", "claim_day_sunday", "claim_year", "claim_day",
            "zip_code", "claim_date", "claim_number", 'SP_Index', "vehicle_color_blue",
             "vehicle_color_gray", "vehicle_color_other", "vehicle_color_red",
            "vehicle_color_silver", "vehicle_color_white"], axis =1, inplace=True)


# Add saftyrating/(number of past claim) feature
train['per_saftyrating'] = train['safty_rating']/(train['past_num_of_claims']+1)
test['per_saftyrating'] = test['safty_rating']/(test['past_num_of_claims']+1)


# Delete some fraud_mean variables
train.drop(["fraud_gender", "fraud_marital_status", "fraud_high_education_ind", "fraud_address_change_ind",
            "fraud_living_status", "fraud_zip_code", "fraud_claim_date", "fraud_witness_present_ind",
            "fraud_policy_report_filed_ind", "fraud_accident_site", "fraud_channel", "fraud_vehicle_category",
          "fraud_vehicle_color", "fraud_state","Unem_rate"],
            axis = 1, inplace = True)
test.drop(["fraud_gender", "fraud_marital_status", "fraud_high_education_ind", "fraud_address_change_ind",
            "fraud_living_status", "fraud_zip_code", "fraud_claim_date", "fraud_witness_present_ind",
```

```
            "fraud_policy_report_filed_ind", "fraud_accident_site", "fraud_channel", "fraud_vehicle_category",
          "fraud_vehicle_color", "fraud_state", "Unem_rate"],
              axis = 1, inplace = True)
train = train.filter(regex="^(?!state_).*$")
test = test.filter(regex="^(?!state_).*$")

train_xgb = train.copy()
test_xgb = test.copy()
```

LightGBM

```
# read full training data set
df_train = pd.read_csv('../data/train_data_clean_5_grouped.csv')
gender_dummies = pd.get_dummies(df_train['gender'],
                            prefix = 'gender', drop_first = True)
df_train = pd.concat([df_train, gender_dummies], axis = 1)
df_train.drop(["gender"], axis = 1, inplace = True)

living_status_dummies = pd.get_dummies(df_train['living_status'],
                            prefix = 'living_status', drop_first = True)
df_train = pd.concat([df_train, living_status_dummies], axis = 1)
df_train.drop(["living_status"], axis = 1, inplace = True)

state_dummies = pd.get_dummies(df_train['state'],
                              prefix = 'state', drop_first = True)
df_train = pd.concat([df_train, state_dummies], axis = 1)
df_train.drop(["state"], axis = 1, inplace = True)

df_train = df_train.sample(frac=1, random_state=5)
df_train['new_param'] = df_train.apply(lambda col: col['safty_rating']/(col['past_num_of_claims']+1), axis=1)

df_train.set_index('claim_number', inplace=True)
df_train.sort_index(inplace=True)
df_train.drop(['claim_date','fraud_claim_date','fraud_zip_code',
        "fraud_gender", "fraud_marital_status", 'fraud_accident_site', 'fraud_high_education_ind',
         "fraud_address_change_ind", "fraud_living_status", "fraud_witness_present_ind",
         "fraud_policy_report_filed_ind", "fraud_channel", "fraud_vehicle_category",
         'fraud_vehicle_color', 'fraud_state', 'SP_Index', 'Unem_rate'], axis = 1, inplace = True)
df_train = df_train.filter(regex="^(?!state_).*$")
df_train = df_train.filter(regex="^(?!vehicle_color_).*$")
df_train = df_train.filter(regex="^(?!claim_day_).*$")
df_train = df_train.filter(regex="^(?!claim_month_).*$")

train_lgb = df_train.copy()
```

```
# read full testing data set
df_test = pd.read_csv('../data/test_data_clean_5_grouped.csv')
gender_dummies = pd.get_dummies(df_test['gender'],
                            prefix = 'gender', drop_first = True)
df_test = pd.concat([df_test, gender_dummies], axis = 1)
df_test.drop(["gender"], axis = 1, inplace = True)

living_status_dummies = pd.get_dummies(df_test['living_status'],
                            prefix = 'living_status', drop_first = True)
df_test = pd.concat([df_test, living_status_dummies], axis = 1)
df_test.drop(["living_status"], axis = 1, inplace = True)

state_dummies = pd.get_dummies(df_test['state'],
```

```
                                prefix = 'state', drop_first = True)
df_test = pd.concat([df_test, state_dummies], axis = 1)
df_test.drop(["state"], axis = 1, inplace = True)

df_test['new_param'] = df_test.apply(lambda col: col['safty_rating']/(col['past_num_of_claims']+1), axis=1)

df_test.set_index('claim_number', inplace=True)
df_test.sort_index(inplace=True)
df_test.drop(['claim_date','fraud_claim_date','fraud_zip_code',
        "fraud_gender", "fraud_marital_status", 'fraud_accident_site', 'fraud_high_education_ind',
         "fraud_address_change_ind", "fraud_living_status", "fraud_witness_present_ind",
         "fraud_policy_report_filed_ind", "fraud_channel", "fraud_vehicle_category",
         'fraud_vehicle_color', 'fraud_state', 'SP_Index', 'Unem_rate'], axis = 1, inplace = True)
df_test = df_test.filter(regex="^(?!state_).*$")
df_test = df_test.filter(regex="^(?!vehicle_color_).*$")
df_test = df_test.filter(regex="^(?!claim_day_).*$")
df_test = df_test.filter(regex="^(?!claim_month_).*$")

test_lgb = df_test.copy()


##### Use CV to get the result #####
# Set the cost for misclassification
cost_dict = {0: 0, 1: 1, -1: 5}

# Set the seed list for splitting dataset
seed_list = [100, 150, 200, 250, 300, 350]

# Set the parameters of XGBoost and LightGBM
clf = xgb.XGBClassifier(max_depth=3,
            learning_rate=0.06,
            n_estimators=180,
            silent=True,
            objective='binary:logistic',
            gamma=0.35,
            min_child_weight=5,
            max_delta_step=0,
            subsample=0.8,
            colsample_bytree=0.785,
            colsample_bylevel=1,
            reg_alpha=0.01,
            reg_lambda=1,
            scale_pos_weight=1,
            seed=1440,
            missing=None)

lgbm_params = {'boosting_type':'gbdt',  'objective':'binary', 'num_boost_round':800,
               'feature_fraction': .321, 'bagging_fraction':0.50, 'min_child_samples':100,
               'min_child_weigh':35, 'max_depth':3, 'num_leaves':2, 'learing_rate':0.15,
               'reg_alpha':5, 'reg_lambda': 1.1, 'metric':'auc', 'max_bin': 52,
               'colsample_bytree': 0.9, 'subsample': 0.8, 'is_unbalance': 'true'
}

cost_list = []
thre_list = [0.364]  # to try diffrent range, just modify this code
for threshold in thre_list:
    cost = []
    for seed in seed_list:
```

```
        # generate row indexes
        random.seed(seed)
        rindex =  np.array(sample(range(len(train_xgb)), round(0.7 * len(train_xgb))))

        # Split train dataset into training and validation parts
        # train_xgb and test_xgb are for XGBoost, train_lgb and test_lgb are for LightGBM

        training_xgb = train_xgb.iloc[rindex, :]
        validation_xgb = train_xgb.drop(train_xgb.index[rindex])

        training_lgb = train_lgb.iloc[rindex, :]
        validation_lgb = train_lgb.drop(train_lgb.index[rindex])

        # XGBoost
        y_training_xgb = training_xgb["fraud"]
        X_training_xgb = training_xgb.drop("fraud", 1)
        y_validation_xgb = validation_xgb["fraud"]
        X_validation_xgb = validation_xgb.drop("fraud", 1)

        clf.fit(X_training_xgb, y_training_xgb)
        y_validation_prob_xgb = clf.predict_proba(X_validation_xgb)[:,1]

        # LightGBM
        y_training_lgb = training_lgb["fraud"]
        X_training_lgb = training_lgb.drop("fraud", 1)
        y_validation_lgb = validation_lgb["fraud"]
        X_validation_lgb = validation_lgb.drop("fraud", 1)


        lgbm = LGBMClassifier(**lgbm_params)
        lgbm.fit(X_training_lgb.values, y_training_lgb.values)
        y_validation_prob_lgb = lgbm.predict_proba(X_validation_lgb.values)[:,1]

        # Combine the result of two models
        validation_prob = 0.4 * y_validation_prob_xgb + 0.6 * y_validation_prob_lgb

        # Calculate the cost
        validation_pred = (validation_prob > threshold)*1  # a trick to transform boolean into int type
        cost.append(sum([cost_dict[i] for i in (validation_pred - y_validation_xgb)]))

    cost_list.append(mean(cost))

min_index = cost_list.index(min(cost_list))
print(thre_list[min_index])
print(cost_list[min_index])


##### Fraud Classification ######
# Predict on the test dataset
cost_dict = {0: 0, 1: 1, -1: 5}
test_pred = pd.read_csv('../data/predictions/combined_predictions.csv')
test_pred['fraud'] = (test_pred['fraud'] > 0.364)*1
test_pred = test_pred.set_index('claim_number')
test_pred.to_csv('../data/predictions/fraud_classification.csv')

# Estimate the cost on the test dataset
cost_list[min_index] * len(test_xgb) / (0.3 * len(train_xgb))
```

C.5 Model_comparison.py

```
##### Import packages #####
import numpy as np
import pandas as pd
from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, VotingClassifier)
from mlxtend.classifier import StackingCVClassifier
from lightgbm import LGBMClassifier
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt; plt.rcdefaults()
import matplotlib.axes as ax


##### Data Manipulation #####
# read full training data set
df = pd.read_csv('data/train_data_clean_5_grouped.csv')

gender_dummies = pd.get_dummies(df['gender'],
                                prefix = 'gender', drop_first = True)
df = pd.concat([df, gender_dummies], axis = 1)
df.drop(["gender"], axis = 1, inplace = True)

living_status_dummies = pd.get_dummies(df['living_status'],
                                prefix = 'living_status', drop_first = True)
df = pd.concat([df, living_status_dummies], axis = 1)
df.drop(["living_status"], axis = 1, inplace = True)

state_dummies = pd.get_dummies(df['state'],
                                 prefix = 'state', drop_first = True)
df = pd.concat([df, state_dummies], axis = 1)
df.drop(["state"], axis = 1, inplace = True)

df = df.sample(frac=1, random_state=5)
df['new_param'] = df.apply(lambda col: col['safty_rating']/(col['past_num_of_claims']+1), axis=1)

df.drop(['claim_number', 'claim_date','fraud_claim_date','fraud_zip_code',
        "fraud_gender", "fraud_marital_status", 'fraud_accident_site', 'fraud_high_education_ind',
         "fraud_address_change_ind", "fraud_living_status", "fraud_witness_present_ind",
         "fraud_policy_report_filed_ind", "fraud_channel", "fraud_vehicle_category",
         'fraud_vehicle_color', 'fraud_state', 'SP_Index', 'Unem_rate'], axis = 1, inplace = True)
df = df.filter(regex="^(?!state_).*$")
df = df.filter(regex="^(?!vehicle_color_).*$")
df = df.filter(regex="^(?!claim_day_).*$")
df = df.filter(regex="^(?!claim_month_).*$")

X = df.drop(['fraud'], axis=1)
y = df['fraud']


##### Compare models #####
# AdaBoost parameters
```

```
ada_params = {
    'n_estimators': 116,
    'learning_rate': 0.1554
}

# Random Forest parameters
rf_params = {
    'n_estimators': 235,
    'max_depth': 84,
    'min_samples_leaf': 34,
    'max_features': 'sqrt'
}

# Logistic Regression parameters
lr_params = {
    'penalty': 'l1'
}

# XGBoost parameters
xgb_params = {
            "max_depth": 3, "learning_rate": 0.06, "n_estimators": 180, "silent": True, "objective": 'binary:logistic',
            "gamma": 0.35, "min_child_weight": 5, "max_delta_step": 0,  "subsample": 0.8, "colsample_bytree": 0.785,
            "colsample_bylevel": 1, "reg_alpha": 0.01, "reg_lambda": 1, "scale_pos_weight": 1, "seed": 1440, "missing": No
}

# LightGBM parameters
lgmb_params = {
    'boosting_type':'gbdt',  'objective':'binary', 'num_boost_round':800,
                'feature_fraction': .321, 'bagging_fraction':0.50, 'min_child_samples':100,
                'min_child_weigh':35, 'max_depth':3, 'num_leaves':2, 'learing_rate':0.15,
                'reg_alpha':5, 'reg_lambda': 1.1, 'metric':'auc', 'max_bin': 52,
                'colsample_bytree': 0.9, 'subsample': 0.8, 'is_unbalance': 'true'
}

ada = AdaBoostClassifier(**ada_params)
rf = RandomForestClassifier(**rf_params)
lr = LogisticRegression(**lr_params)
xgb = XGBClassifier(**xgb_params)
lgbm = LGBMClassifier(**lgmb_params)
xgb_lgbm = VotingClassifier(estimators=list(zip(['xgb', 'lgbm'], [xgb, lgbm])),
                            voting='soft', weights=[6, 4])
models = [ada, rf, lr, xgb, lgbm, xgb_lgbm]
long_labels = ['Adaboost', 'Random Forest', 'Logistic Regression', 'XGBoost', 'LightGBM', 'XGBoost_LightGBM']
print('5-fold cross validation:\n')
for clf, label in zip(models, long_labels):
    scores = model_selection.cross_val_score(clf, X.values, y.values, cv = 5, scoring = 'roc_auc')
    print("AUC: %0.9f (+/- %0.4f) [%s]" % (scores.mean(), scores.std(), label))


##### Performance Visualization ######
objects = ('Adaboost', 'Random Forest', 'Logistic Regression',
           'XGBoost', 'LightGBM', 'XGBoost_LightGBM')
y_pos = np.arange(len(objects))
performance = (0.719963411, 0.711689776, 0.709100835, 0.725387261,
               0.729196909, 0.730128054)

plt.figure(figsize=(8, 6))
plt.ylim(0.70, 0.74)
```

```
rects = plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.xticks(rotation=30)
plt.ylabel('AUC')
plt.title('Model Performance')


def autolabel(rects):
    for rect, perf in zip(rects, performance):
        height = rect.get_height()
        plt.text(rect.get_x() - rect.get_width()/6, 1.001*perf, '%s' % float(perf))
autolabel(rects)

plt.show()
```