Final Project

Santander Customer Transaction Prediction

Chu Yun Hsiao, Yi Fang, Hao Chun Niu, Jiarui Hu, Shuyun Liu

MSBA 6420: Predictive Analysis

Professor Yichen Song

April 19, 2022

Table of Contents

1. Project Definition & Introduction

1.1 Background and Context

As a financial services company, Santander Group (Banco Santander) aims to help customers and businesses prosper in an inclusive and sustainable way. People at Santander are always dedicated to offering their clients financial health-related information and the best products and services recommendations that will help them achieve their monetary goals. Its data science team, in order to better understand its customers, is always testing and refining its machine learning algorithms, which will solve common binary classification challenges more efficiently and accurately, including problems such as customers' satisfaction, their propensity to buy a certain product, and their financial ability to pay a loan.

In this project, Santander Group offers a challenge to the data science community, asking for solutions to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted. Since the data provided has the same structure as the real data, the solutions might not only be helpful for this specific question in the real world case but also help with improving the binary classification machine learning algorithms Santander Group has in the future.

1.2 Problem Statement

The goal of this project is to build an ultimate binary classification model to identify which customers will make a specific transaction in the future, depending on several evaluation metrics. The first step is to come up with several models and tune them to their best performances. Next, we will have a comparison table, and decide on the final model that will support our conclusions.

1.3 Data Sources

All of the data used for this analysis was provided by Santander Group on Kaggle:https://www.kaggle.com/c/santander-customer-transaction-prediction/overview.

This is an anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column.

2. Analysis

2.1 Methodology

In this project, we tried three different classification models, Logistic Regression, XGBoost, and LightGBM. XGBoost and LightGBM are both tree-based boosting models and are especially good at tackling large amounts of data. In addition, based on our past experiences, XGBoost and LightGBM often deliver outstanding classification performance. On the other hand, Logistic Regression is a simple linear classification model, but it provides great interpretability.

All three models involve various hyperparameters. Hence, in order to find the best hyperparameters for the three models, we use Grid-Search Cross-Validation techniques on the

train data set to compare the model's performance with different hyperparameters. After addressing the best hyperparameters for each of these three models, we will test each model's performance based on the test data set and find the best-performing model.

As for our performance matrix, we will use accuracy as our main measurement. Eventually, we want a classification model with a high accuracy rate.

2.2 Data Preparation

In general, the data contains 2 important issues, outliers, and imbalanced data. Given that all the features are numeric and roughly normally distributed, we use the simple statistical approach to detect the outliers. If any of the features is apart from the mean by more than 3 standard deviations, that row of data will be defined as an outlier. With the simple statistical approach, we defined 5.52% of the data to be outliers.

```
#Remove those rows of outliers
train_no_outliers=train[(np.abs(stats.zscore(train.iloc[:,2:]))< 3).all(axis=1)]
print('After excluding the outliers, {}% of data is removed.'.format(round((1-(len(train_no_outliers)/len(train)))*100,
```

```
After excluding the outliers, 5.52% of data is removed.
```

Fig.1: Outlier Detection

As for the imbalanced data issue, we use SMOTE which is one of the most popular oversampling techniques to simulate minority data. To be more specific, to increase the amount of data belonging to the minority class, SMOTE will create new data points that have very similar characteristics to the original minority class data points. Originally, the positive cases (Class 1) only account for 9.89% of the entire data. Ideally, we expect the ratio between the number of minority cases and the number of majority cases should be close to 1. Eventually, after SMOTE, the positive cases now account for 50% of the entire data, meaning that the ratio between the number of minority cases and the number of majority cases is 1.

We notice that the data is extremely imbalanced. Therefore, we will **use both undersampling and oversampling** to fix the issue.

```
perc=y_train.value_counts()[1]/(y_train.value_counts()[1]+y_train.value_counts()[0])*100
print('The positive cases(Class 1) only take account for {}% of the entire train data.'.format(round(perc,2)))
```

```
The positive cases(Class 1) only take account for 9.89% of the entire train data.
```

```
y_train.value_counts()
```

```
0    136218
1     14957
Name: target, dtype: int64
```

```
#SMOTE on train data
smt=SMOTE()
x_train_smt, y_train_smt = smt.fit_resample(x_train,y_train)
```

```
perc=y_train_smt.value_counts()[1]/(y_train_smt.value_counts()[1]+y_train_smt.value_counts()[0])*100
print('After SMOTE, there are totally {} positive cases and {} negative cases.'.format(y_train_smt.value_counts()[1],y_
print('The positive cases(Class 1) only take account for {}% of the entire train data.'.format(round(perc,2)))
```

```
After SMOTE, there are totally 136218 positive cases and 136218 negative cases.
The positive cases(Class 1) only take account for 50.0% of the entire train data.
```

Fig. 2: Resampling the Imbalanced Dataset

2.3 Logistic Regression with gridsearchCV

Logistic Regression is a “Supervised machine learning” algorithm that can be used to model the probability of a certain class or event. It is used when the data is linearly separable and the outcome is binary or dichotomous. That means Logistic regression is usually used for Binary classification problems.

a. Strengths of Logistic Regression
   1. One of the simplest machine learning algorithms
   2. Easier to implement, interpret, and very efficient to train
   3. The predicted parameters give inference about the importance of each feature
   4. In low dimensional datasets, logistic regression is less prone to overfitting
   5. Very efficient if the features are linearly separable

b. Weaknesses of Logistic Regression
   1. May be over-fit on the training set when the datasets are on a high dimensional
   2. Sensitive to outliers
   3. Cannot solve nonlinear problems
   4. Difficult to capture complex relationships. Other more powerful models, such as Neural Network, can easily outperform logistic regression

c. Hyperparameter Tuning

- **Solver [default =lbfgs]**: This indicates which algorithm to use in the optimization problem. The default value is lbfgs. Other possible values are newton-cg, liblinear, sag, and saga.
- **Penalty [default = l2]**: This hyper-parameter is used to specify the type of normalization used. Few of the values for this hyper-parameter can be l1, l2, or none. The default value is l2.
- **C [default = 1.0]**: The C parameter controls the penalty strength, which can also be effective.

We used gridsearchCV to find the best hyperparameter. As the result shows, the best hyperparameter is:

- C: 0.001
- Penalty: l2
- Solver: lbfgs

With the best hyperparameters found, The accuracy of logistic regression is 91.29%, the weighted f-1 score is 0.89, and the ROC-AUC score is 0.85.

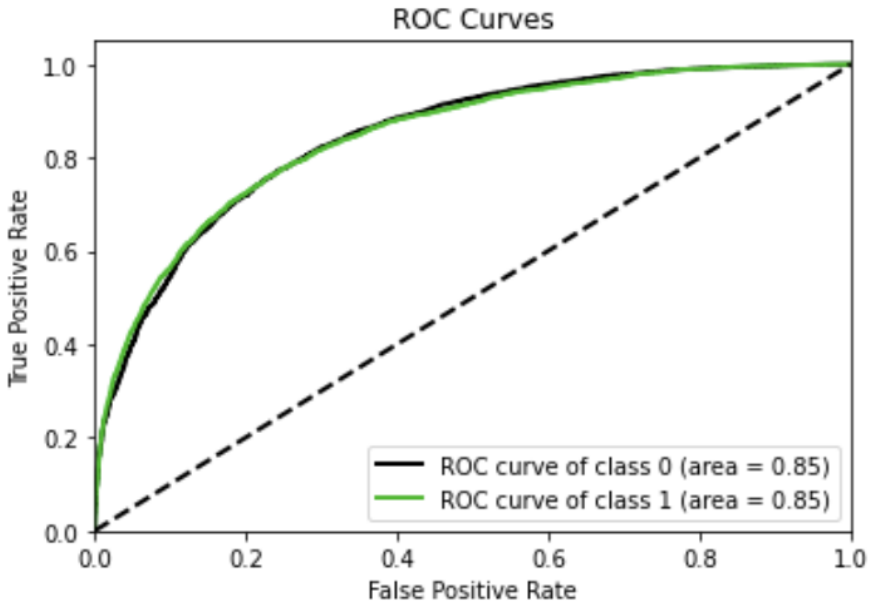


Fig. 3: ROC Curves for Logistic Regression

2.3.2 XGBoost with gridsearchCV

XGBoost stands for Extreme Gradient Boosting. Gradient Boosting is an ensemble technique where new models are created to predict the residuals or errors of prior models and then added together to make the final prediction. Models are added sequentially until no further improvements can be made. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. This approach supports both regression and classification predictive modeling problems.

a. Strengths of XGBoost
   1. Good Execution speed and model performance
   2. Less feature engineering is required
   3. Built-in feature importance score used for feature selection
   4. Less prone to overfitting because of built-in L1 and L2 regularization

b. Weaknesses of XGBoost
   1. XGB is sensitive to outliers.
   2. Unlike LightGBM, one has to encode categorical features before feeding them into the models manually.
   3. Training time is pretty high using a larger dataset than CatBoost/LightGBM.

c. Hyperparameter Tuning
   - **Objective [default=reg: linear]:** This defines the loss function to be minimized. Suppose we set the value to binary: logistic. In that case, it returns predicted probability using logistic regression for binary classification

- **Learning Rate [default = 0.3]:** This makes the model more robust by shrinking the weights on each step.
- **Max_depth[default=6]:** The maximum depth of a tree.
- **N_estimators[default = 100]**: The number of trees (or rounds) in an XGBoost model
- **Colsample_bytree[default = 1]**: XGBoost builds multiple trees to make predictions. This parameter defines what percentage of features (columns) will be used for building each tree.
- **Gamma[default = 0]:** It is the minimum loss reduction to create a new tree split. To make the algorithm more conservative, the high value of gamma is preferred.

We used gridsearchCV to find the best hyperparameter. As the result shows, the best hyperparameter is:

- C: 0.001
- colsample_bytree: 0.3
- gamma: 0.01
- learning_rate: 0.1
- max_depth: 3
- n_estimators: 200
- objective: binary:logistic

With the best hyperparameters found, The accuracy of XGBoost is 90.34%, the weighted f-1 score is 0.85, and the ROC-AUC score is 0.83.

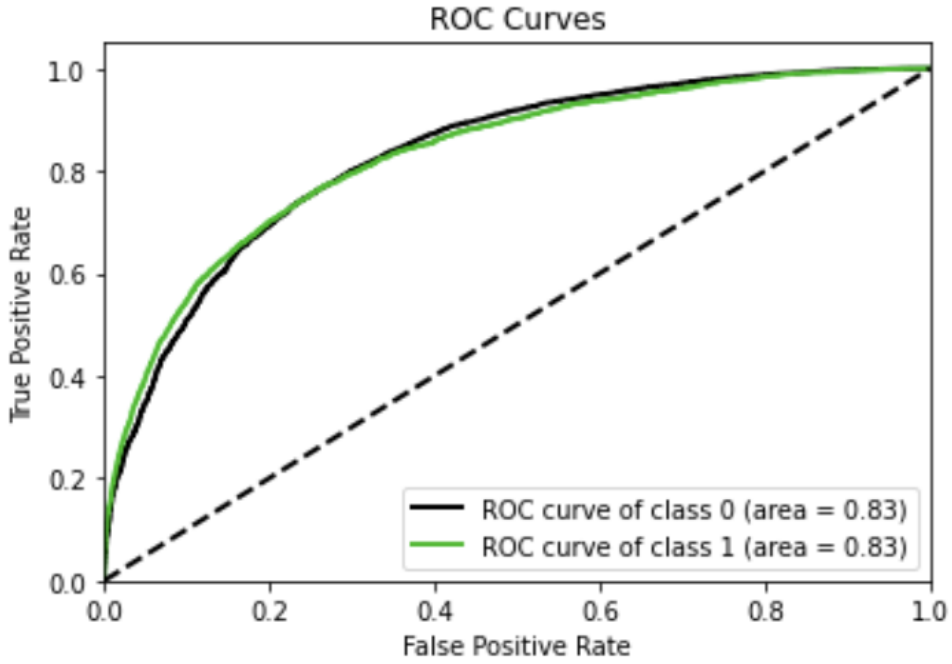


Fig. 4: ROC Curves for XGBoost

2.3.3 LightGBM with gridsearchCV

LightGBM stands for lightweight gradient boosting machines. LightGBM is an open-source implementation of a gradient boosting algorithm similar to XGBoost. The difference that makes LightGBM faster than XGBoost is that LightGBM applies leaf-wise tree growth, whereas XGBoost applies level-wise tree growth.

a. Strengths of LightGBM
   1. Faster training speed and higher efficiency
   2. Better accuracy than any other boosting algorithm
   3. More compatible with Large Datasets compared to XGBoost.

b. Weakness of LightGBM
   1. Prone to overfitting in a small dataset as it produces much more complex trees.

c. Hyperparameter Tuning
   - **Objective [default=reg: linear]**: This defines the loss function to be minimized. If we set the value to binary: logistic, it returns predicted probability using logistic regression for binary classification
   - **Learning Rate [default = 0.3]**: Makes the model more robust by shrinking the weights on each step.
   - **Max_depth[default=6]**: The maximum depth of a tree.
   - **N_estimators[default = 100]**: The number of trees (or rounds) in an XGBoost model
   - **Colsample_bytree[default = 1]**: XGBoost builds multiple trees to make predictions. This parameter defines what percentage of features (columns) will be used for building each tree.

We used gridsearchCV to find the best hyperparameter. As the result shows, the best hyperparameter is:

- colsample_bytree: 0.3
- learning_rate: 0.1
- max_depth: 3
- n_estimators: 200
- objective: binary

With the best hyperparameters found, The accuracy of LightGBM is 90.60%, the weighted f-1 score is 0.87, and the ROC-AUC score is 0.86.

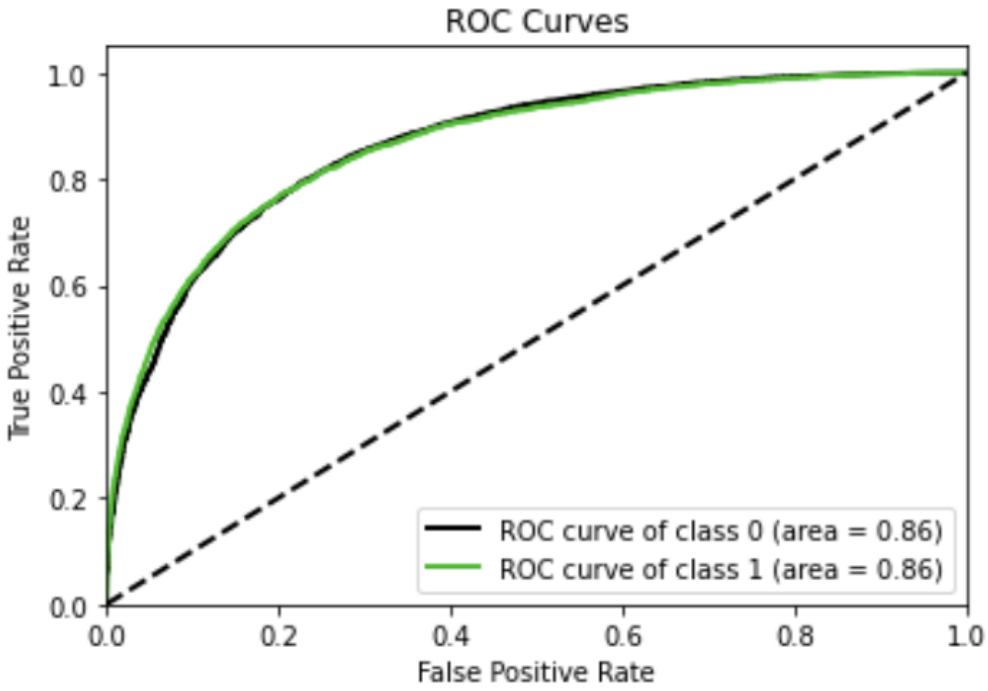


Fig. 5: ROC Curves for LightGBM

3. Results

3.1 Model Performance Comparison

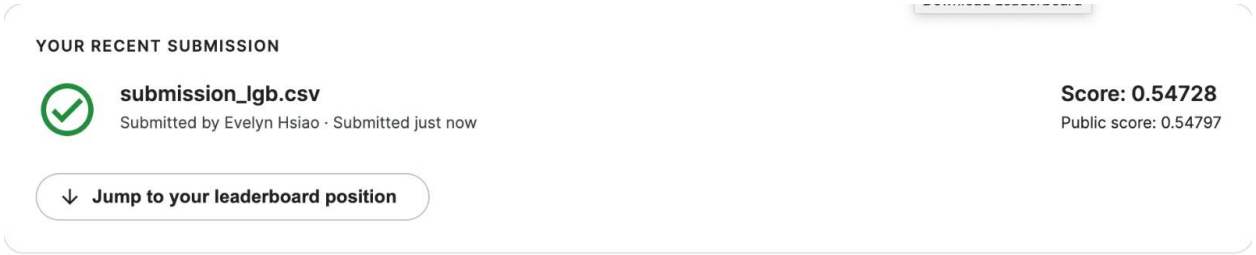
The following table is the summary scores of the above models we implemented:

Methods	Training	Testing	Precision	Recall	F1-score	Roc AUC
Logistic regression	91.29	91	0.89	0.91	0.89	0.85
XGBoost	90.34	89.94	0.90	0.90	0.85	0.83
LightGBM	91.80	90.60	0.90	0.91	0.87	0.86

Table 1: Summary Scores

As we can see, LightGBM has the best scores among 3 models. It can accurately predict 91.80% of the data, and its ROC-AUC score is 0.87, as we know that the AUC is the trade-off between precision and recall. Thus, we selected LightGBM as our final model for the future task. However, we noticed that the training accuracy is higher than the testing score, The reason can be that the dataset was synthetically oversampled by SMOTE, which led the model to the overfitting problem.

3.2 Kaggle Submission



3.3 Conclusion

Working on the imbalanced dataset and implementing 3 different classification algorithms on the Santander customer transaction prediction dataset provided by Kaggle, we learned that detecting anomalies and resampling the imbalanced dataset is the foundation before the actual model building stage, and selecting the metrics is critical for optimizing the result. Yet without the help of gridsearchCV, the progress of model optimization can be a prolonged journey. From this analysis, we also see that resampling the imbalanced data, in our case, the SMOTE method, may lead the model to the overfitting problem.

4. Future Work

Our future task is to implement LightGBM on other structured imbalanced datasets to determine scalability and universality. Secondly, we'll re-run our script on the latest machine to optimize the model and improve the model performance. Last but not least, we will dig into the resampling methods to figure out the relationship between resampling methods and the overfitting issue.

5. Appendix

5.1 Implementation of Logistic Regression

```
from sklearn.linear_model import ElasticNet
import sklearn
from sklearn import linear_model
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, StratifiedKFold, KFold
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
```

Packages used

```
c = list(range(-5, 10))
c_values = [10**i for i in c]

log_grid = {'C': c_values,
            'penalty': ['none', 'l2']}
log = LogisticRegression(random_state = 42)
clf_log = GridSearchCV(log, log_grid, cv = cv, scoring='f1_micro', error_score = 'raise')
clf_log.fit(x_train_smt, y_train_smt)
```

Parameters used

```
Training set score: 0.9128890358855631
Testing set score: 0.9099857120177806
```

	precision	recall	f1-score	support
0	0.92	0.99	0.95	33925
1	0.70	0.21	0.33	3869
accuracy			0.91	37794
macro avg	0.81	0.60	0.64	37794
weighted avg	0.89	0.91	0.89	37794

Detailed classification report

5.2 Implementation of XGBoost

```
!pip install scikit-plot
from xgboost import XGBClassifier
import scikitplot as skplt
from sklearn.metrics import mean_squared_error, roc_curve, auc, confusion_matrix, \
                            plot_confusion_matrix, classification_report, make_scorer, accuracy_score
```

Packages used

```
# XGB hyperparameter
xgb_paramGrid = {'estimator__objective':['binary:logistic'], 'estimator__learning_rate': [1e-1, 1e-2, 1e-3, 1e-4],
                 'estimator__max_depth': [3,5,7,9,11], 'estimator__gamma':[0.01, 0.1],
                 'estimator__n_estimators':[200, 300, 400, 500], 'estimator__colsample_bytree':[0.3, 0.7]}

# Find the best parameter
xgb = XGBClassifier(verbosity = 0, class_weight=None)
clf = GridSearchCV(estimator=xgb, param_grid=xgb_paramGrid, n_jobs=-1, cv=cv, scoring='f1_micro', error_score = 'raise')
grid_result = clf.fit(x_train, y_train)

print(f"Best score of gridsearch is {grid_result.best_score_}\n")
print(f"The best hyperparameter of XGBoost is  {grid_result.best_params_}\n")
```

Parameters used

```
Training set score: 0.9034099553497602
Testing set score: 0.8993755622585595

              precision    recall  f1-score   support

           0       0.90      1.00      0.95     33925
           1       0.93      0.02      0.04      3869

    accuracy                           0.90     37794
   macro avg       0.92      0.51      0.49     37794
weighted avg       0.90      0.90      0.85     37794
```

Detailed classification report

5.3 Implementation of LightGBM

```
from lightgbm import LGBMClassifier
```

Packages used

```
# LGBM hyperparameter
lgbm_paramGrid = {'estimator__objective':['binary'], 'estimator__learning_rate': [1e-1, 1e-2, 1e-3, 1e-4],
                  'estimator__max_depth': [3,5,7,9,11], 'estimator__n_estimators':[200, 300, 400, 500],
                  'estimator__colsample_bytree':[0.3,0.5, 0.7]}

# Find the best parameter of
lgbm = LGBMClassifier(verbosity = 0, class_weight=None)
clf = GridSearchCV(estimator=lgbm, param_grid=lgbm_paramGrid, n_jobs=-1, cv=cv, scoring='f1_micro', error_score = 'raise')
grid_result = clf.fit(x_train, y_train)


print(f"Best score of gridsearch is {grid_result.best_score_}\n")
print(f"The best hyperparameter of LightGBM is  {grid_result.best_params_}\n")
```

Parameters used

```
Training set score: 0.9179957003472796
Testing set score: 0.9059903688416151

              precision    recall  f1-score   support

           0       0.91      1.00      0.95     33925
           1       0.86      0.10      0.17      3869

    accuracy                           0.91     37794
   macro avg       0.88      0.55      0.56     37794
weighted avg       0.90      0.91      0.87     37794
```

Detailed classification report

6. References

Medium. 2019. *Santander Customer Transaction Prediction: A Simple Machine Learning Solution*. [online] Available at: <https://medium.com/@aganirbanghosh007/santander-customer-transaction-prediction-a-simple-machine-learning-solution-771613633843>.

Medium. 2019. *Santander Customer Transaction Prediction: An End to End Machine Learning project*. [online] Available at: <https://medium.com/analytics-vidhya/santander-customer-transaction-prediction-an-end-to-end-machine-learning-project-2cb763172f8a>.

OpenGenus IQ: Computing Expertise & Legacy. 2022. *Advantages and Disadvantages of Logistic Regression*. [online] Available at: <https://iq.opengenus.org/advantages-and-disadvantages-of-logistic-regression/> .

Sadaf Ilyas, Sultan Zia, Umair Muneer Butt, Sukumar Letchmunan and Zaib un Nisa, *Predicting the Future Transaction from Large and Imbalanced Banking Dataset*. International Journal of Advanced Computer Science and Applications(IJACSA), 11(1), 2020. http://dx.doi.org/10.14569/IJACSA.2020.0110134.