MGIS 489 Seminars in Business Intelligence – Summer 2022

Prof. Bui

Final Project # Group 5

Lin Jie 20711013

Su Jingyun 20711018

Zhou Xun 20711040

Niu Shuyuan 20711062

Su Fengrui 20711065

1. Objective

As labor costs gradually rise, some supermarkets are reducing the job of cashier. An established food retailer is promoting the project which has made a self-scanning system. The system allows customers to scan the selected items using a handheld mobile scanner while paying by themselves. This type of payment is efficient. But the retailers who use the system have to face the risk that a certain number of customers will not scanning all of the items, they may only scan some of their items. This behavior leading to retailers' profits decrease.

To minimize the losses, the food retailer requires a targeted follow-up checks to identify cases of fraud. Our goal is to identify as many false scans as possible to reduce retailer loss. On the basis, keep the number of checks as low as possible to reduce the extra added cost to the retailer as well as avoid putting off innocent customers due to false accusations.

The objective for us is to create a predictive model to classify the scans as fraudulent or non-fraudulent. The classification does not take into account whether the fraud was committed intentionally or inadvertently. The overall goal of this report is to analyze the data, build several machine learning models, compare the model effects, optimize the model super parameters, and accurately predict the fraud behavior of customers.

2. Data Preparation

2.1 data description

The given data is in form of a CSV file with a as field separator. We received two files:

mgis489_train.csv containing 316, 953 rows with 9 feature columns and one prediction column.

mgis489_test.csv containing 135, 836 rows with 9 feature columns and no prediction column.

The data set in the “mgis489_train.csv” file should be used for training your predictive models. The data sets in the “mgis489_test.csv” file should be used for the prediction. The result will be written into a new CSV file that has only one column, the prediction of fraud (1) or not fraud (0). This then has to be submitted, so the contest jury can evaluate the score of the candidate.

The columns in the data set are the following:

Table 1: Feature description table

Column name	Description	Value range
Trust Level	A customer's individual trust level. 6: Highest trustworthiness	{1,2,3,4,5,6}
Total Scan Time in Seconds	Total time in seconds between the first and last product scanned	Positive whole number
Grand Total	Grand total of products scanned	Positive decimal number with maximum two decimal places
Line-Item Voids	Number of voided scans	Positive whole number
Scans Without Registration	Number of attempts to activate the scanner without actually scanning anything	Positive whole number or 0
Quantity Modification	Number of modified quantities for one of the scanned products	Positive whole number or 0
Scanned Line Items Per Second	Average number of scanned products per second	Positive decimal number
Value Per Second	Average total value of scanned products per second	Positive decimal number
Line-Item Voids Per Position	Average number of item voids per total number of all scanned and not cancelled products	Positive decimal number
fraud	Classification as fraud (1) or not fraud (0)	{0,1}

2.2 Data cleaning and preprocessing

Both train.csv and test.csv contain outliers and missing values. The training set scannedLineItemsPerSecond contains 0.277% of missing values, the test set variable scannedLineItemsPerSecond contains 0.3224%, and valuePerSecond contains 0.9430% of missing values. Screen the data for outliers according to the description of the feature. For example, the data is described as an integer and the actual value is a decimal, and these outliers are regarded as missing values.

This report first deletes the data samples containing missing values and outliers in the training set, deletes the outliers in the test set, and replaces them with null values, then glues the processed training set and prediction set, and uses KNN to impute the test. Set missing values (including outliers that are considered missing). The number of processed training set samples is 316, 047.

2.3 Data exploration

Next, we want to explore the data. Of particular note is the distribution of fraud and non-fraud in the 316,953-line training data set. 63.11% of samples are non-fraud and 36.89% of samples are fraud, implying a slight imbalance in the data. Since the imbalance of the dataset is not very high and the sample size is huge, in the training process of this report, we adopt the method of balancing the sample weights to compensate for the imbalance of the training set's fraud distribution:

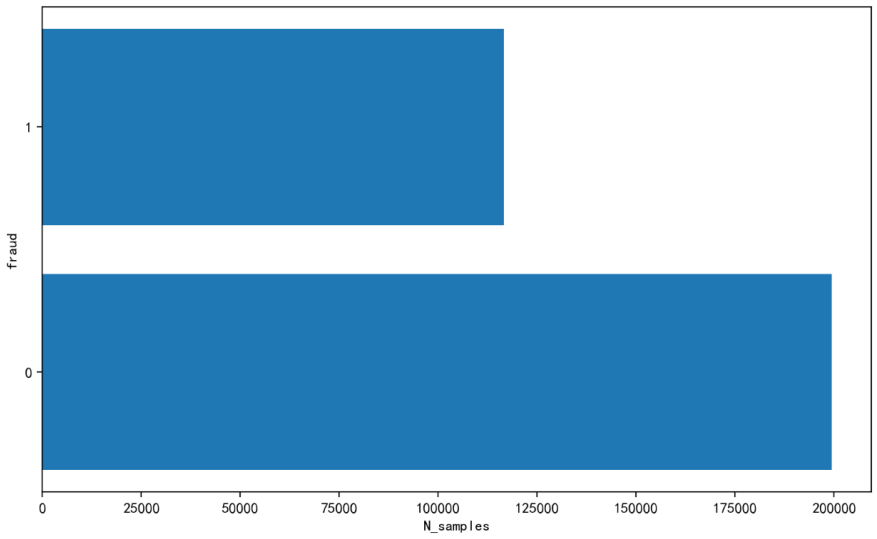

Figure 1: Fraud distribution

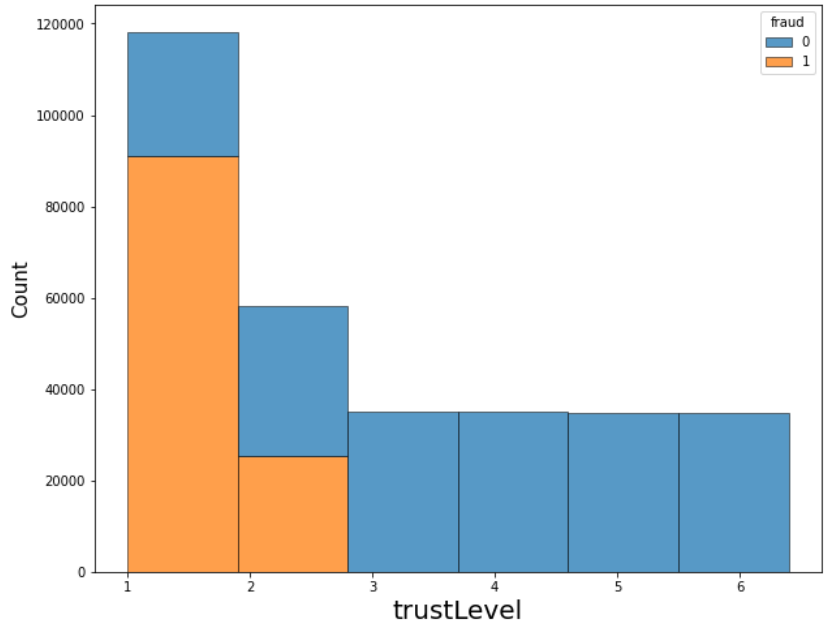

Figure 2: Relationship between trust level and fraud

From Figure 2, it can be seen that the level of customer trust has a great influence on whether fraud will occur. Customers with trust levels 3-6 are not fraudulent, customers with trust level 2 are more likely to commit fraud, and customers with trust level 1 are slightly less likely to commit fraud.

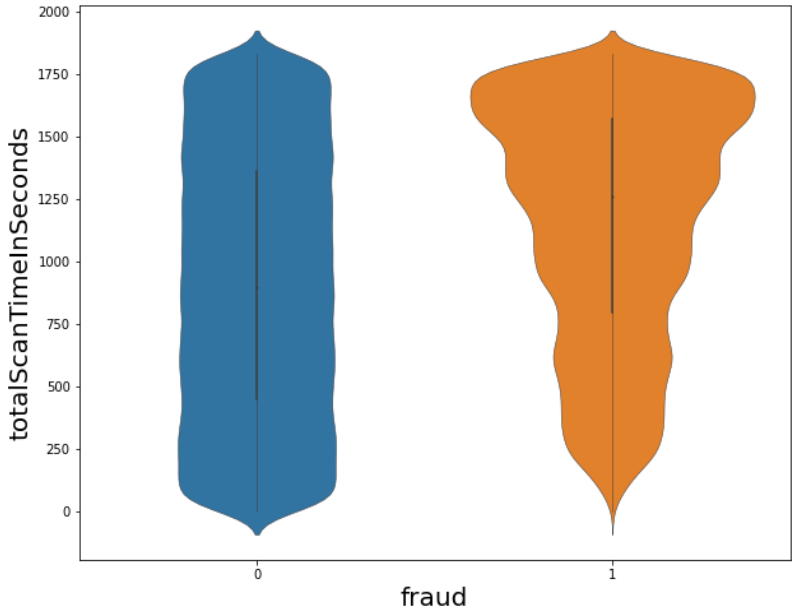

Figure 3: Relationship between Total Scan Time in Seconds and fraud

It can be seen from Figure 2 that the longer the customer scans the product, the more prone to have fraudulent behavior. The average and quartile of the purchase records with fraudulent behaviors are higher than the records without fraudulent behavior.

2.4 Model Preprocessing Feature Engineering

In order to fully discover the influence of features on the nonlinear relationship of labels, we use feature combination to encode nonlinear laws (Little). This report adopts the method of feature cross. A combination of features formed by multiplying the values of two features, so 9 variables are crossed to generate 36 new cross features, which together with the original 9 features form a total of 45 new features, which are used as model features.

[A × B]： A combination of features formed by multiplying the values of two features.

3.Data analysis

In this report, the tree integration algorithm XGBoost model, random forest, and multiple perceptron are selected. The three models are compared and the optimal model is selected. Finally, it is proved that XGBoost is the optimal algorithm. Bayesian optimization is used for the XGBoost model when the parameters are adjusted to get the final result's model.

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way (Liu).

3.1 Frame

To try out different pipelines, we decided to use scikit-learn as a framework as it comes with almost all the necessary tools to accomplish our task. The idea is to create a new Jupyter Notebook for each algorithm to try that heavily relies on helper functions from several Python scripts. Each notebook will be structured exactly the same and essentially execute the following steps:

1.Import packages

2.Load the part of fraud training data

3.Plot several diagrams about the training process:

- Confusion matrix
- Precision Recall
- Kaggle score

Find the best prediction to maximize our score. Test what score the model would get with the released test data. The goal was that each of those steps, except the model training, is a single function call. These utility functions have been implemented in python scripts that reside in the same folder.

3.2 Model comparison and selection

Random Forest, Multiple Perceptron and XGBoost results are as follows(cv=5):

Table 2: Model Results Comparison

	Random forest	Multiple Perceptron	XGBoost
Costs	567820	506920	386060
Accuracy	0.99646	0.97193	0.97863
Kaggle score	0.99676	0.97823	0.99830

According to this, XGBoost is used as the benchmark model, which lead to the highest Kaggle score without overfitting.

3.3 Parameter optimization

We use Bayesian optimization to find the best hyper-parameters for the model and the training task. Machine learning training a model is a time-consuming process, and as networks become more complex these days, the hyper-parameters become more complex. Therefore, in order to improve the efficiency, random search is also called Bayesian optimization (Shu). In the process of finding optimal hyperparameters, we additionally use Stratified and KFold to preserve the percentage of samples of each class, so we have a fair cross validation score. The scoring is our custom scoring implementation.

XGBoost parameters are as follows (Python):

1.n_estimator: Also known as num_boosting_rounds. This is the number of the largest tree generated and the largest number of iterations.

2.learning_rate: Sometimes called eta, the system default is 0.3, The step size of each iteration is very important. If it is too large, the running accuracy is not high, and if it is too small, the running speed is slow. We generally use a smaller value than the default value, about 0.1 is fine.

3.gamma: The system defaults to 0, and we also commonly use 0. When a node is split, the node will be split only if the value of the loss function drops after the split. gamma specifies the minimum drop in loss function required for node splitting. The larger the value of this parameter, the more conservative the algorithm will be. Because the larger the gamma value, the more the loss function drops before the nodes can be split. Therefore, it is not easy to split nodes when the tree is generated. Range: [0,∞].

4.subsample: The system defaults to 1. This parameter controls the proportion of random sampling for each tree. Decreasing the value of this parameter makes the algorithm more conservative and avoids overfitting. However, if this value is set too small, it can lead to underfitting. Typical value: 0.5-1, 0.5 represents average sampling to prevent overfitting. Range: (0,1], note that 0 is not acceptable.

5.colsample_bytree: The system default value is 1. We generally set it to about 0.8. Used to control the proportion of the number of columns randomly sampled for each tree (each column is a feature). Typical value: 0.5-1 Range: (0,1].

6.colsample_bylevel: The default is 1, and we also set it to 1. This is more detailed than the previous one. It refers to the ratio of column sampling when each node is split in each tree.

7.max_depth: The default value of the system is 6. We usually use numbers between 3-10. This value is the maximum depth of the tree. This value is used to control overfitting. The larger the max_depth, the more specific the model learns. Set to 0 for no limit, range: [0,∞].

8.max_delta_step: The default is 0, and we often use 0. This parameter limits the maximum step size of the weight change of each tree. If the value of this parameter is 0, it means that there is no constraint. If he is given a certain positive value, the algorithm is more conservative. Usually, we do not need to set this parameter, but when the samples of each class are extremely unbalanced, this parameter is very helpful for the logistic regression optimizer.

9.lambda: Also called reg_lambda, the default value is 0. L2 regularization term for the weights. (Similar to Ridge regression). This parameter is used to control the regularization part of XGBoost. This parameter is helpful in reducing overfitting.

10.alpha: Also known as reg_alpha which defaults to 0, the L1 regularization term for the weights. (Similar to Lasso regression). It can be applied to very high dimensions, making the algorithm faster.

11.scale_pos_weight: The default is 1. When the samples of each category are very unbalanced, setting this parameter to a positive value can make the algorithm converge faster. It can usually be set as the ratio of the number of negative samples to the number of positive samples.

We use Bayesian optimization to find suitable parameters within the following parameter ranges:

Table 3: Parameter optimization range and results

Parameter	Scope	Search results
estimators	50~300	201
lr	0.001~0.2	0.148
subsamples	0.1~1	0.432
gammas	0~10	1.6
max_depths	10~25	22
max_leaves	2~1000	302
colsample_bytree	0.1~1	0.668
colsample_byleves	0.1~1	0.952

4. Result

Indicators after model optimization.

4.1 Monetary value using cost matrix

Table 4: Cost matrix

	No fraud	Fraud

No fraud	0	-162875
Fraud	-1190	550125

4.2 Accuracy from the confusion matrix

Table 5: Confusion matrix

	No fraud	Fraud
No fraud	199269	6515
Fraud	238	110025

Accuracy=0.978632

4.3 Kaggle score

0.99830

The result of parameter optimization is not as good as the original default parameter XGBoost model

4.4 Feature importance

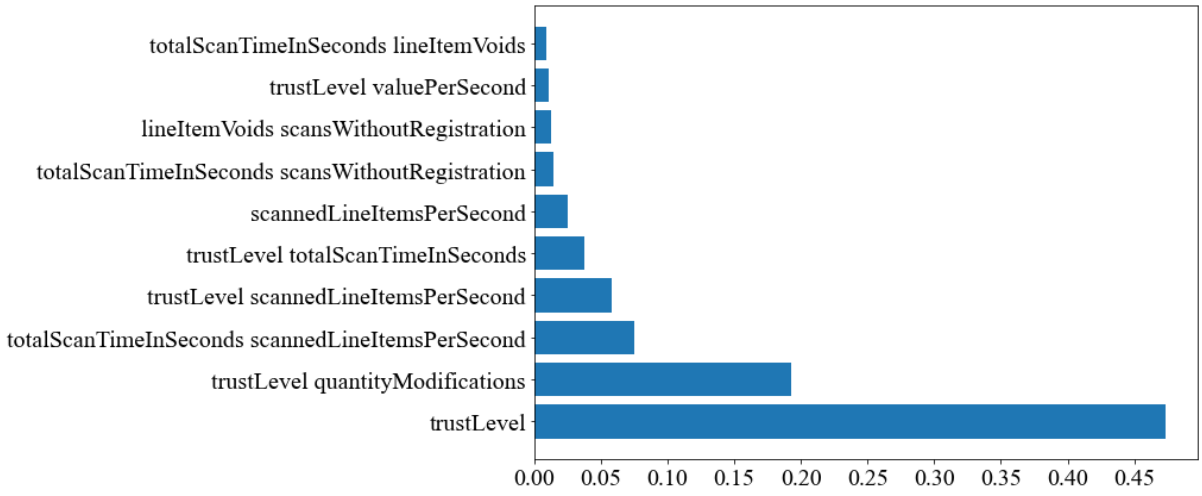

Figure 4: Feature weight map (top 10)

The feature weight map lists the top ten features with the greatest impact among the 45 features. It can be seen that the trust level has the greatest impact on the fraud variable, followed by the variable where the trust level is multiplied by the quantity Modification.

5. Conclusion

5.1 Model

The experiment results in the best model with high performance is XGBoost. XGBoost afford to get better performance without ranking attributes. The performance of research performed better than the previous research to predict fraud. Based on these results, XGBoost obtained Accuracy 0.998863, and other models only achieves. It happened because previous research used too many attributes to build the model, and the XGBoost classifier can Select useful features.

5.2 Suggestion

- It is recommended to pay more attention to customers with lower trust levels to be aware of their fraudulent behavior. If there are customers with low trust level, let the guide to help carry out commodity scanning.
- It is recommended that merchants monitor the length of time customers scan the code. If the customer cannot complete the scan of the product for a long time, let the shopping guide to ask if they need help.
- If the machine detects that the customer frequently changes the quantity of goods, the guide can help the customer count the goods.
- For customer with low trust level and repeatedly escaped from paying the order, building block list and forbidding them to use self-scanning system. They are only allowed to go for the chaser to buy things.

6. Reference

Little, Louluo. “Sklearn.preprocessing.PolynomialFeatures.” *Bowen*, 22 Sept. 2020, https://www.cnblogs.com/cgmcoding/p/13714024.html.

Liu, Qilin. “XGBoost Principle, Formula Derivation, Python Implementation and Application.” Zhihu, 23 Jan. 2021, https://zhuanlan.zhihu.com/p/162001079.

Shu, Liandan. “Knowledge and Line Bayesian Optimization Need Only Read This Article, the Algorithm to Python Implementation .” Zhihu, 23 July 2020, https://zhuanlan.zhihu.com/p/131216861.

Python, Amanda. “XGBoost Parameter Details and Code Examples.” CSDN, 2 Mar. 2021, https://blog.csdn.net/Amanda_python/article/details/114294984.

7. Appendix

Code for Random Forest:

```
rfc_prediction = cross_val_predict(RandomForestClassifier(n_jobs=-1), data_poly.iloc[:, :-1],
data_poly.iloc[:, -1], cv=5, verbose=True)
print(accuracy_score(rfc_prediction, data_poly.iloc[:, -1]))
rfc_cm = confusion_matrix(rfc_prediction, data_poly.iloc[:, -1])
rfc_cm
```

Code for Multi-layer Perceptron:

```
mlp_prediction = cross_val_predict(MLPClassifier(), data_poly.iloc[:, :-1], data_poly.iloc[:, -1],
cv=5,n_jobs=-1, verbose=True)
print(accuracy_score(mlp_prediction, data_poly.iloc[:, -1]))
mlp_cm = confusion_matrix(mlp_prediction, data_poly.iloc[:, -1])
mlp_cm
```

Code for XGBoost:

```
xgb_prediction = cross_val_predict(XGBClassifier(n_jobs=-1), data_poly.iloc[:, :-1],
data_poly.iloc[:, -1], cv=5, verbose=True)
print(accuracy_score(xgb_prediction, data_poly.iloc[:, -1]))
xgb_cm = confusion_matrix(xgb_prediction, data_poly.iloc[:, -1])
xgb_cm
```

Code for XGBoost with best parameter:

```
polyer = PolynomialFeatures(interaction_only=True, include_bias=False).fit(test_data)
test_data_poly = pd.DataFrame(polyer.transform(test_data),
columns=polyer.get_feature_names_out().tolist())[sample_data_poly.columns[:-1].values]
estimators=np.array([*range(50, 300)])
lr = np.linspace(0.001, 0.2, 20)
subsamples = np.linspace(0.1, 1, 20)
gammas = np.arange(0, 10, 0.1)
max_depths = np.arange(5, 25)
max_leaveses = np.arange(2, 1000, 5)
colsample_bytrees = np.linspace(0.1, 1, 20)
colsample_bylevels = np.linspace(0.1, 1, 20)

best_param = {'colsample_bylevel': colsample_bylevels[13],
        'colsample_bytree': colsample_bytrees[17],
        'gamma': gammas[21],
        'learning_rate': lr[11],
        'max_depth': max_depths[17],
        'max_leaves': max_leaveses[150],
        'n_estimators': estimators[195],
        'subsample': subsamples[8]}
best_param
best_xgb_prediction = cross_val_predict(XGBClassifier(n_jobs=-1,
**best_param),data_poly.iloc[:, :-1], data_poly.iloc[:, -1], cv=5, verbose=True)
print(accuracy_score(xgb_prediction, data_poly.iloc[:, -1]))
best_xgb_cm = confusion_matrix(xgb_prediction, data_poly.iloc[:, -1])
best_xgb_cm
```