Credit Card Fraud Detection

Eddie Tseng
A53245708
edtseng@eng.ucsd.edu

Ping-Chun Chiang
A53236468
p3chiang@eng.ucsd.edu

Abstract

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Our goal is to use machine learning skills solving this kind of problems. Though there have been much work in fraud detection, we would like to implement and evaluate the performance of several contemporary models such as Light GBM, XGBoost and CatBoost which are famous algorithms developed by different institutions such as Microsoft.

Our main idea is to compare these three well-known algorthms and find out the most efficient one which means having the less time for building a model and predicting data with highest accuracy and lowest time costing.

Keywords— Light GBM, XGBoost, CatBoost, Fraud, PCA, GridSearchCV

1. Introduction

Credit card fraud is a wide-ranging term for theft and fraud committed using or involving a payment card, such as a credit card or debit card, as a fraudulent source of funds in a transaction. The purpose may be to obtain goods without paying, or to obtain unauthorized funds from an account. Unfortunately, due to confidentiality issues, we cannot get the original features and more background information about the data. Furthermore, The problem of fraud detection, was defined as the **problem of classifying imbalanced data**[7] which makes the problem more difficult. In order to solve this issue, we will change our evaluation matrix in the following discussion.

To aid enhancing fraud detection accuracy, many scientists have used lots of machine learning skills, for example, random forest, SVM etc. However, more and more complex algorithms are developed nowadays, and we don't exactly know which one is better in this case. Therefore, in this paper, we will focus on three famous algorithms, **Light GBM, XGBoost and CatBoost**, to find out the most suitable classification algorithm for this dataset.

2. Experiments

2.1. Dataset

In this project, The models will be trained and evaluated on **Kaggle**. Transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.173% of all transactions as showm in fig.8. It contains only numerical input variables which are the result of a **PCA** transformation.

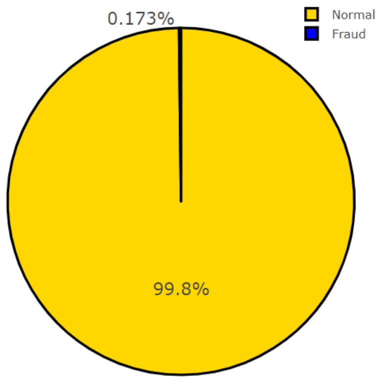
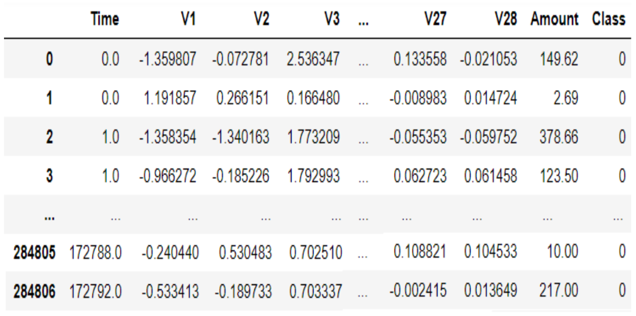


Figure 1: Distribution of class

In fig.2, features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are **'Time'** and **'Amount'**. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case



	Time	V1	V2	V3	...	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	...	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	...	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	...	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	...	0.062723	0.061458	123.50	0
...	...	...	...	...	...	...	...	...	...
284805	172788.0	-0.240440	0.530483	0.702510	...	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	...	-0.002415	0.013649	217.00	0

Figure 2: Dataset

of fraud and 0 otherwise.

2.1.1 Dataset exploration

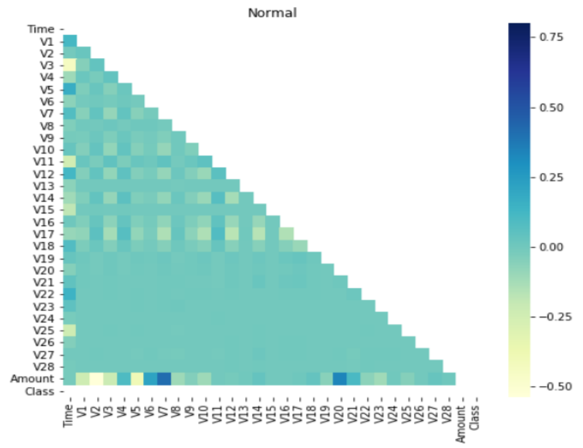


Figure 3: Correlation matrix: Normal

Because we have no idea what are the PCA variables imply in the data set, we have to look for the correlation between each feature based on their class. In fig.3, we first look at 'Normal' class and we can find out that 'Amount' has highly correlation with some features like 'V7' and 'V20'. Likewise, the features after 'V18' don't show too much correlation with other variables. It's could be the reason that those variables have less weight in PCA domain.

However, what we are more interested to is the relationship of variables in class 'Fraud'. Remember that this is a highly unbalanced dataset so we need to find out the pattern to help us do classify. In fig.4, it is not abnormal to see that the variables before 'V18' have extremely high or low correlation coefficient. We can use this result to train a better model by weighting variables.

Due to all variables come from a PCA decomposition ex-

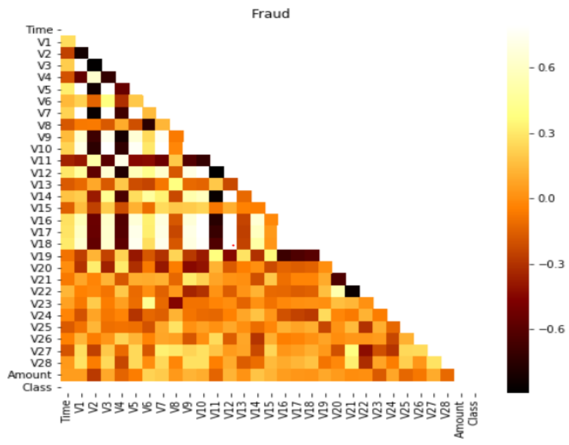


Figure 4: Correlation matrix: Fraud

cept 'Time' and 'Amount'. We normalize 'Amount' and use a variable named 'nAmount'. After that, we drop the original 'Time' because Time is not correlated with the target, and brings nothing to the model.After finalize the dataset, we also split the it into 80% training set and 20% testing set.

2.2. Models

When introducing models, first we liked to talk about **Gradient Boosted Decision Tree(GBDT)**[6]. GBDT builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

2.2.1 XGB

First of all, let's start from the earliest algorithm **XGBoost[3]: eXtreme Gradient Boosting**. XGBoost became famous in 2016 when it has been dominating applied machine learning and Kaggle competitions. It is an implementation of **gradient boosted decision trees**[6] which also called gradient boosting, stochastic gradient boosting or multiple additive regression tress.

2.2.2 Light GBM

LightGBM uses a novel technique of **Gradient-based One-Side Sampling (GOSS)** to filter out the data instances for finding a split value while XGBoost uses pre-sorted algorithm Histogram-based algorithm for computing the best split. Similar to CatBoost, LightGBM can also handle categorical features by taking the input of feature names. It does not convert to one-hot coding, and is much faster than one-hot coding. LGBM uses a special algorithm to find the split value of categorical features. LightGBM is designed to be distributed and efficient with the following advantages: Faster training speed and higher efficiency, Lower memory usage, Better accuracy, Support of parallel and GPU learning and Capable of handling large-scale data.

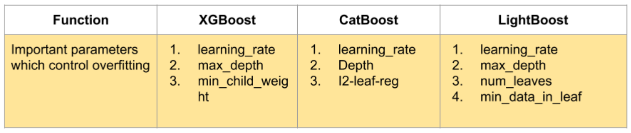
Algorithm 1 Gradient-based One-Side Sampling

```
Require: I: training data, d: iterations
Require: a: sampling ratio of large gradient data
Require: b: sampling ratio of small gradient data
Require: loss: loss function, L: weak learner
1: for $i = 1 \to$d do
2:     $preds \leftarrow models.predict(I)$
3:     $g \leftarrow loss(I, preds)$
4:     $sorted \leftarrow GetSortedIndices(abs(g))$
5:     $topSet \leftarrow sorted[1 : topN]$
6:     $randSet \leftarrow RandPick(sorted[topN : len(I)], randN)$
7:     $useSet \leftarrow topSet + randSet$
8:     $newModel \leftarrow L(I[useSet], g[useSet], w[useSet])$
9:     $models.append(newModel)$
10: end for
```

2.2.3 CatBoost

CatBoost is a machine learning method based on gradient boosting over decision trees.[5] It has the flexibility of giving indices of categorical columns so that it can be encoded as one-hot encoding. The main advantages of this model are: Superior quality when compared with other GBDT, Best in class inference speed, Support for both numerical and categorical features and Fast GPU and multi-GPU support for training

2.3. Hyperparameter Tuning



Function	XGBoost	CatBoost	LightBoost
Important parameters which control overfitting	1. learning_rate 2. max_depth 3. min_child_weig ht	1. learning_rate 2. Depth 3. l2-leaf-reg	1. learning_rate 2. max_depth 3. num_leaves 4. min_data_in_leaf

Figure 5: Important parameters for different functions

We use function GridSearchCV in Python scikit-learn which combine an estimator with a grid search preamble to tune hyper-parameters. The method picks the optimal parameter from the grid search and uses it with the estimator selected by the user. It is the hardest part we encountered when we building a model. Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations evaluate the performance of each model. We won't dig too much into how to decide the parameters and we only tuning the important ones for different models

In fig. 5, we pick the important parameters for different algorithms[1], and each parameter has the different meaning and default value and we pick the important ones to explain:

- learning_rate: shrinks the contribution of each tree
- max_depth: maximum depth of the individual regression estimators
- min_child_weight: minimum sum of instance weight (hessian) needed in a child
- num_leaves: number of leaves to be formed in a tree
- l2-leaf-reg: used for leaf value calculation.

3. Result

3.1. Evaluation Matrix

After training on original data, we got 99.9% training accuracy. However, the evaluation matrix is not quiet robust because accuracy is not meaningful for unbalanced classification.

Unbalanced data refers to a problem with classification problems where the classes are not represented equally.[2] There are many ways to solve the imbalanced data problem. For example, collecting more data, changing performance matric, resampling dataset, penalized models and even be creative.

Here we talk about **changing performance matric**: use **Area Under the Precision-Recall Curve (AUPRC)**[4] recommend by this competition. A AUPRC curve is plotting Precision against Recall. Precision is defined as:

$$Precision = \frac{N_{TP}}{N_{TP} + N_{FP}} \tag{1}$$

where $\mathbf{N_{TP}}$ is the number of true positive and $\mathbf{N_{FN}}$ is the number of false positive. Recall is define as:

$$Recall = \frac{N_{TP}}{N_{TP} + N_{FN}} \tag{2}$$

where $\mathbf{N_{TP}}$ is the number of true positive and $\mathbf{N_{FN}}$ is the number of false negative. The area under the curve is the integral of precision as a function of recall. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

3.2. Comparison

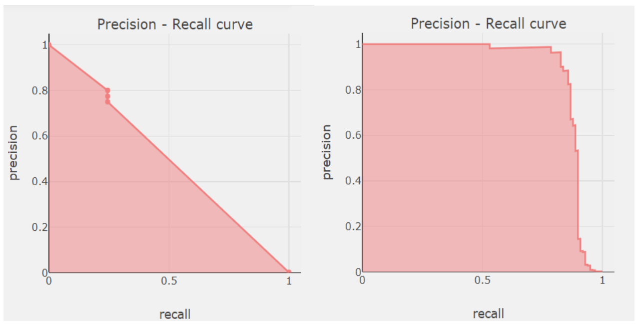


Figure 6: AUPRC for LGBM(Left: before GridSearchCV, Right: after GridSearchCV)

3.3. Comparison

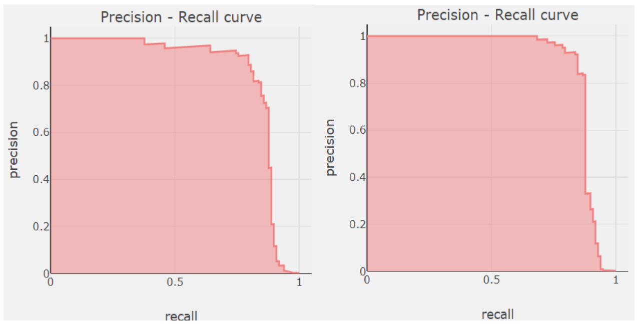


Figure 7: AUPRC for XGB(Left: before GridSearchCV, Right: after GridSearchCV)

3.4. Comparison

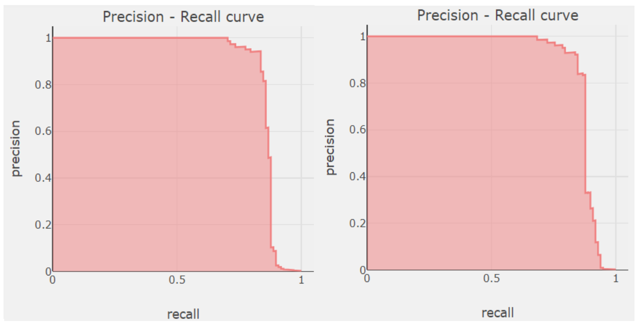
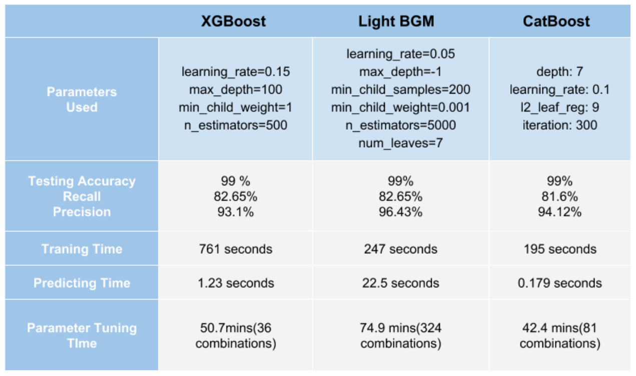


Figure 8: AUPRC for CatBoost(Left: before GridSearchCV, Right: after GridSearchCV)

4. Discussion

When choosing the best model, we need to consider the predicting speed and accuracy at the same time.

In our case, Light BGM comes out with best AUPRC but its predicting time takes a lot more than the other two. Because it cost us way more time on searching parameters for LBGM and the estimators number we used is 5000 which enhances the accuracy but adds the predicting time as well. Our winner belongs to CatBoost which give us the fastest predicting and fewest training time without losing accuracy.



	XGBoost	Light BGM	CatBoost

Parameters Used	learning_rate=0.15
	max_depth=100
	min_child_weight=1
	n_estimators=500
		learning_rate=0.05
		max_depth=-1
		min_child_samples=200
		min_child_weight=0.001
		n_estimators=5000
		num_leaves=7
			depth: 7
			learning_rate: 0.1
			l2_leaf_reg: 9
			iteration: 300

Testing Accuracy Recall Precision	99 % 82.65% 93.1%	99% 82.65% 96.43%	99% 81.6% 94.12%
Traning Time	761 seconds	247 seconds	195 seconds
Predicting Time	1.23 seconds	22.5 seconds	0.179 seconds
Parameter Tuning TIme	50.7mins(36 combinations)	74.9 mins(324 combinations)	42.4 mins(81 combinations)

Figure 9: Results

We implement these three latest algorithms on this is special case and finally decide which algorithm is fit with it. Unfortunately we didn't have enought time to optimize all the parameters. However, the results have no significant difference, so we just need to choose the most efficient one which is CatBoost. Thanks to CatBoost for successfully detecting fraud.

5. Future Work

The training time doesn't take too long, but if we want to get higher accuracy, we need to optimize our parameter. Due to the limitation of time and resource, we didn't spent much time on tuning parameters. Hence, we hope to complete the following works in the future:

- Study on the parameters influence on each models
- Achieve real-time fraud detection
- Implement more models
- Tune parameters to find the best performance
- Try more skills on unbalance data

References

[1] CatBoost vs. Light GBM vs. XGBoost. `https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db`.

[2] Combat Imbalanced Classes in Your Machine Learning Dataset. `https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes/in-your-machine-learning-dataset/`. Accessed: 2018-05-12.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[4] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[5] A. V. Dorogush, V. Ershov, and A. Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[7] Z. Sun, Q. Song, X. Zhu, H. Sun, B. Xu, and Y. Zhou. A novel ensemble method for classifying imbalanced data. *Pattern Recognition*, 48(5):1623 – 1637, 2015.