Detection and Analysis of Credit Card Application Fraud Using Machine Learning Algorithms

Yaodong Han¹,*, Shun Yao², *, Tie Wen³, Zhenyu Tian⁴, Changyu Wang⁵, Zheyuan Gu⁶

¹Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, SAR, China
²College of computer and information science, Southwest University, Chongqing, 400715, China
³School of computer science and information engineering, Harbin Normal University, Harbin, 150025, China
⁴Industrial technology, XianJiaotong-Liverpool University, Suzhou, 215123, China
⁵Beijing Royal School, Beijing, 102207, China
⁶Worcester academy, Worcester, Massachusetts, 01604, USA

*Corresponding author’s e-mail: xxys123xx@163.com
yaoshun666@email.swu.edu.cn

Abstract. Fraud is a widespread problem in the financial industry with devastating effects. It is essential to prevent and reduce fraud effectively. Traditional approaches, such as expert system, suffers from the incapability to handle complex problems and tremendous amount of data, while the recent development of various machine learning techniques brings new solutions. With many research works focusing on tackle frauds of credit card transaction or insurance, only few mentioned the identity fraud of credit card application. This article presents a few machine learning models to detect such fraud. We firstly explore and clean up the data. Then 331 expert variables are created with professional consult and selected to 30 to reduce dimensionality of our data. Multiple models, such as logistic regression and decision trees, are built and fit on the training set. Finally, we found that the random forest model performs the best in terms of fraud detection rate, achieving 54% in out-of-time test. The obtained model can be applied in anti-fraud monitoring systems, or a similar model development process can be performed in related business areas to detect fraud and reduce the occurrence of such behaviors.

1. Introduction

Financial fraud is a criminal act, where fraudsters profit illegally by deceiving or stealing personal information. Fraud may affect the development of industry, undermine economic stability, and reduce people's wealth [1]. For a long time, it has been a huge concern for banks, companies, and individuals. About 10% of insurance payouts are caused by fraudulent claims [2]. According the Federal Trade Commission of the USA, in 2019 alone, 1.7 million frauds were reported with a staggering loss of $1.9 billion [3]. Over the years, with the change of fraud detection methods, the fraudulent party has also continuously changed the fraud methods in order to avoid the tracing[4]. With the development of the...
internet and online services, the number of frauds has recently increased [1]. Credit card fraud and identity theft account for 20% of the total fraud and is becoming a big issue [3]. Fraudsters use wrongly or illegally obtained identities to conduct credit card transactions or apply for new cards to avoid paying bills. Therefore, a reliable method of analyzing and detecting fraud is essential to the safety and prosperity of online financial activities.

The traditional approach to identify frauds is expert system [5], which is a set of rules made by experts, and will determine whether a transaction is fraudulent or not. However, as financial systems get more and more complicated, the number and complexity of rules grow to a point where no one could construct and maintain such complex system [6]. As a result, more and more attention has been focused on machine learning and data mining [2]. Instead of writing rules by hand, computers can learn the patterns and signals of fraudulent activities and identify potential frauds based on some relatively simple algorithms. And with the development of more powerful and tailor-made hardware for training such models, handling huge data sets containing billions of records became plausible [7]. Researchers have been building such models with a wide range of algorithms and achieved excellent accuracy. Most research, though, focused on credit card payment or transactions [8], but the area of identity theft, especially for credit card application, remains open. In this article, we investigate the performance and possibility of machine learning algorithms to detect fraudulent credit card applications.

In this article, a dataset with identified fraud cases is provided. It is a supervised classification problem since fraudulent records are labeled. Section 2 and 3 includes the description and the cleaning of the dataset. All frivolous values are located in this step and replaced with the record number of the respective data point. Once the dataset is cleaned, candidate variables are created in Section 4 using different combination methods. Three variable groups are generated: the velocity variables, the day-since variables, and the relative velocity variables. In Section 5, filter and wrapper selection methods are employed to reduce the dimensionality. The particular filter methods used are Kolmogorov-Smirnov (KS) and Fraud Detection Rate (FDR) at 3% to eliminate variables [9]. As for the wrapper method, backward selection is used with logistic regression to evaluate performance for each feature. Section 6 listed machine learning models we built and optimized on the dataset, and the comparison of results are discussed in Section 8. After extensive parameter tuning, we find that the random forest algorithm achieves the best fraud detection rate. Section 9 concludes this article with model results and possible future work.

2. Data Description
The data set on which this project is conducted was built for organizations conducting academic research in the field of application fraud. It does not come from any real credit card applications due to privacy protection, instead, it is synthetically generated based on a statistical analysis of one billion real US applications collected over 10 years. Contents of all fields are randomly generated while mimicking the format of real data, for example, names are strings with more than nine letters and zip codes are 5-digit numbers. More importantly, the distributions of each field and correlations between fields are retained as well. All fields have the same univariate distribution as real data. Important joint distributions, such as the number of addresses corresponding to a certain name, resembles real data as well. Also, the signatures of fraud records look similar to real frauds, in terms of filed distribution and frequency. Hence, this synthetic dataset is considered sufficient for building machine learning models and reflecting their performance on real-world data.

The data in this project consists of one million credit card applications with 14393 frauds, and there are 10 fields in each entry, including one record field and one fraud label field. To better understand the data we are working with, statistics of all fields are listed in the Table 1. Since the data are purely generated, all fields are 100% populated and no value is missing. All fields except records are categorical, and no linear relationship exists between any two values in the entire dataset. Therefore, the transformation of fields is needed for machine learning model building.

Detailed description of all fields is shown below:
- Record: A distinct number for each data entry, which also preserves time order, with smaller records happening first.
- Date: The date when an application took place, in the format of year, month and date.
- SSN: Social security numbers of applicants, which are nine-digit numbers and used as identification of applicants.
- Fristname: First names of applicants generated randomly in English letters substituting real names.
- Lastname: Last names of applicants generated randomly in English letters substituting real names.
- Address: Street addresses of applicants generated randomly in the format of address number, street name and street type.
- Zip5: Five-digit numbers representing USA zip codes of corresponding addresses.
- Dob: The date of birth of an applicant in the format of year, month and date.
- Homephone: A nine- or ten-digit random phone numbers of applicants.
- Fraud Label: An identification variable with value 0 or 1, labeling the application as fraud or non-fraud.

Table 1. Statistics of each field in the dataset.

| Field Name | Type    | # Non-empty records | Population Percentage | Common value   | # Unique Values |
|------------|---------|---------------------|-----------------------|----------------|-----------------|
| record     | index   | 1,000,000           | 100%                  | 42             | 1,000,000       |
| date       | date    | 1,000,000           | 100%                  | 1/1/2016       | 365             |
| ssn        | integer | 1,000,000           | 100%                  | 938972725      | 835,819         |
| firstname  | string  | 1,000,000           | 100%                  | EAMSTRMT       | 78,136          |
| lastname   | string  | 1,000,000           | 100%                  | ERJSAXA        | 177,001         |
| address    | string  | 1,000,000           | 100%                  | 123 MAIN ST    | 828,774         |
| zip5       | integer | 1,000,000           | 100%                  | 68138          | 26,370          |
| dob        | date    | 1,000,000           | 100%                  | 19640318       | 42,673          |
| homephone  | integer | 1,000,000           | 100%                  | 6384782007     | 28,244          |
| fraud_label| integer | 1,000,000           | 100%                  | 0 or 1         | 2               |

The daily distribution of applications is then examined, shown in the Figure 1. The sudden drop in Figure 1 (a) is because no data was generated on 2016-02-29 and 2016 is a leap year. After excluding February 29, the distribution (Figure 1 (b)) shows that all applications are spread out throughout the whole year. The missing data on this specific day will not affect further processes.
Figure 1. Daily distribution of records in the dataset. a) daily distribution with Feb 29th. b) daily distribution without Feb 29th.

Histograms of 15 values with the highest frequency of each field are compiled in Figure 2. We can see that for SSN, homephone, address and dob (Figure 2 (a, d, e, f)), there is a large number of records with frivolous values, which are values that are meaningless, and only used to fill in an empty value. When an application form contains illegal inputs or missing inputs, the application system often fills such spots with a default value, such as ‘123 MAIN ST’ for address, or ‘99999999’ for SSN. Other than frivolous values, the distribution of all fields looks normal, with only a few values having high supports, and almost all other values having similar low frequency.

3. Data Cleaning
Since all fields are 100% populated, there is no need for filling missing values. However, frivolous values shown in Figure 2 needs to be cleaned. It’s obvious that these frivolous data are meaningless and most of them are system defaults, so they may potentially introduce unwanted correlation between records and redundancy, which will negatively affect classification accuracy of machine learning models. In order to minimize the effect, we assume that all frivolous values are different from each other and thus substitute them with their corresponding record numbers. Also, we converted all values in number to be strings for easier handling, since all fields are categorical. And leading zeros are added to zip5 field for some records to match the format of 5-digit number.
4. Feature Creation
Creating features, or expert variables, from original data is a crucial step in most machine learning projects. The ultimate performance of models largely depends on how much useful information is extracted from the original data into new features. We consulted with experts in the field of fraud detection to aid us to understand possible signatures of credit card application fraud.

The major type of frauds involved here is identity theft, where fraudsters use leaked or stolen identity information of other people with their own contact information to try to apply for credit cards [10]. A fraudster may possess compromised information of a number of individuals, and the same identity information could be used by several fraudsters. Frauds are usually conducted in batches, so frequency is a key signal of potential frauds. Frequent application with the same address or phone number but different names of SSN might imply frauds from one fraudster, or alternatively, applications with the same name or SSN but different contact information may suggest that the core identity information of this individual is disclosed to different fraudsters. Based on these analyses, variables related to frequency are created.

The first step of our feature creation is to build identity entities. We noticed that one single field from the original data is not sufficient to identify a person; for instance, many people may have the same first name. As a result, fields are combined by concatenating to serve as identifiers. Core identity fields and contact fields are then linked to capture the abnormal frequency distribution described above. All combined entities generated are listed in Table 2.
Table 2. All created field by combining two or more other fields. Name is created by concatenating first name and last name. Fulladdress is created by concatenating address and zip5.

| name | fulladdress_homephone | ssn_zip5 |
|------|------------------------|----------|
| fulladdress | dob_homephone | ssn_dob |
| name_dob | homephone_name_dob | ssn_homephone |
| name_fulladdress | ssn_firstname | ssn_name |
| name_homephone | ssn_lastname | ssn_fulladdress |
| fulladdress_dob | ssn_address | ssn_name_dob |

All combined fields together with SSN, address, dob and homephone, are then used to generate frequency variables. There are three categories of variables created for each field:

- **Day since variables.** This variable measures how much time has passed since a certain individual last appeared in the dataset. It is unlike for someone to apply for credit card many times frequently, so one with a low day since values may imply fraud. We found that most records (around 85%) only appeared once in our dataset, which is consistent with the fact that most people only apply for credit card once in a long while but making calculating day since variables impossible. All such fields are given a value of 365 days, which is the maximum possible value of this variable.

- **Velocity variables.** Velocity variables reflects how often an individual apply for credit card over a certain time span. The number of applications is counted for each individual during the last 0, 1, 3, 7, 14 or 30 days.

- **Relative velocity variables.** Though velocity variables can measure the frequency of someone’s application, there’s a chance that some shared identities are used among a group of people, for example, people working in the same office building may have the same mailing address, making that particular address a hot spot in terms of velocity. Thus, relative velocity variables are created to normalize this effect, and finds spikes in frequency, which can better reflect the situation where fraudsters use stolen identities to make a bunch of applications at a time.

Besides the three major categories, a day of week risk is also introduced as a measurement of the probability of finding a fraud on a certain day of a weak, hoping to catch any underlying logic behind frauds [11]. The risks are statistically smoothed by sigmoid with the overall average to rule out outliers.

After feature creation, we ended up with 331 variables using 22 original or combined fields, including 1 day of week risk, 22 day since, 132 velocity and 176 relative velocity variables. The exhaustive list of features created can be found in Appendix A.

### 5. Feature Selection

The procedurally generated expert variables contain valuable information extracted from the original data; however, they cannot be fed into a machine learning model directly, because the dimensionality is too high. Training a model with such high dimensionality can take extremely long time, rendering most algorithms computationally intractable. Besides, some models may grow unnecessarily complicated and suffer from overfitting greatly, if we do manage to train them. Thus, feature selection is vital to the success of building machine learning models.

Since most generated features are highly correlated, we can choose a subset of them to reduce redundancy. Generally, there are three major categories of feature selection methods: filter, wrapper and embedded. Filter methods calculate statistics or correlation to rank all features and determine which ones are to be excluded. Wrapper methods, like the name implies, ‘wrap’ around some machine learning models and use them as predictors to evaluate the importance of each feature. And embedded methods incorporate feature selection within model training process, intended to reduce the intense computation needed for wrapper methods while retaining decent selection. Here, we use filter and wrapper methods to select our features.
5.1. Filter

Filters are first applied to reduce the number of features to 80, since it runs much faster than wrappers. The filter method we implemented uses two ranking metrics, Kolmogorov-Smirnov (KS) and fraud detection rate (FDR) [8]. Kolmogorov-Smirnov test, or KS, measures how separated two distributions are by quantifying the maximum distance between their cumulative distribution curves. For continuous distribution, we have:

\[
KS = \max_x \int_{x_{\min}}^{x} (P_1 - P_2) \, dx
\]

And for discrete distribution, KS test can be written as:

\[
KS = \max_x \sum_{x_{\min}} (P_1 - P_2)
\]

For each candidate feature, two distributions of fraud and non-fraud records are built respectively, and then KS score is calculated based on the formula above. KS test can examine how different fraud records distribute compared to non-fraud records and tell how well each feature by itself can potentially separate fraud from non-fraud.

Fraud detection rate (FDR) is also used as a filter metrics. Fraud detection rate describes how many frauds we can catch within certain population, and it is often used as a measurement for model performance, which we are using in this project. In a generic FDR, a machine learning model gives ranking to all records based on the probability of being fraud, and the records in the given population bin, usually top 3%-5%, is considered as predicted fraud. The fraud detection rate is calculated to be the number of true frauds in the bin, which are caught by the model, divided by the total number of true frauds exists in the entire dataset. FDR reflects how many frauds can be caught by a model, with a fixed number of predicted positives.

FDR cannot be used as a filter method directly, because it requires a classification model to give a predicted probability list, but a univariate FDR can be used by taking the values of each record as its probability. The univariate FDR measures whether frauds tend to cluster at one end of the distribution for one particular feature, and generally performs well in selecting useful features in fraud detection. We apply FDR at 3% cutoff as a filter metrics.

In filter selection, each feature is scored and ranked by both KS and FDR, and the average ranking is used as final ranking, with each metrics contributing 50%. The top 80 features are selected for further selection by wrapper methods.

5.2. Wrapper

After obtaining the best 80 features from filter selection, we then put them into a wrapper for more fine-tuned selection. A wrapper usually selects features in a stepwise fashion, where in each step, a machine learning model evaluates performance of all candidate features exhaustively [12]. Arbitrary number of top-ranking features can then be selected, and the rest are put into another round of selection, until the desired number of features are reached. Compared to filters, wrappers are much more computationally intensive, since a machine learning model is trained for all possible combinations of features at each step, which is why filters are applied before wrapper, to quickly reduce the dimension of the problem.

Forward and backward selections are two simplest yet most commonly used selection sequences for wrappers [12]. In forward selection, we start with zero features and add one or several best features each time, whereas in backward selection, one or several worst features are removed from the entire feature set.

Here, we use backward selection with a logistic regression model [13]. Each score is calculated with two-fold cross validation to reduce bias. Thirty best features are selected to build our models. The entire workflow of our feature selection process is shown in Figure 3.
6. Model Algorithms
Since each record is labeled with either fraud or non-fraud, our fraud detection problem is a special case of supervised binary classification. Thus, we explore several widely used classification algorithms with diverse complexities. In each section, the principles of each algorithm are described and tuned hyperparameters are reported.

We split our dataset into three parts, training, testing and out-of-time (OOT) dataset. The first 85% of data, containing 850,000 records, are used for model building and validation. Typically, 10-fold cross validation is applied to obtain relatively unbiased scores, where the dataset is divided randomly into 10 complementary subsets with equal size. In each run, 9 folds are used for training the model leaving 1 for validation, and the process is repeated for 10 times, with each of the 10 subsets chosen exactly once as validation test set. Ten results are then averaged to give the final validation score. The last 15% of data, containing 150,000 records, are reserved to be OOT and put aside for testing purposes only. This can reflect model performance on unseen data and mimic situation in production to some extent.

The metrics we used to measure the performance of a model is FDR at 3%. As mentioned in section 5.1, FDR emphasizes more on the true positives in a certain population bin, which is frauds that we can catch, than that of non-frauds.

6.1. Logistic Regression
After selected 30 variables which have relatively useful information, we choose to use logistic regression model as our baseline model [13]. Due to its simplicity and high efficiency, logistic regression algorithm is widely used in most classification problems. Logistic regression and linear regression actually have many similarities, the biggest difference lies in their dependent variables are different, the two kinds of regression can be attributed to the same family, that is, the generalized linear model [13]. It is necessary to find a function whose range is in the [0,1] which can fit the binary classification problems. Hence, we combine sigmoid function and linear regression function; and set the output of linear regression model as the input of sigmoid function. Sigmoid function describes the relationship between the outcome dependent binary variable and the independent variables (features). Based on this problem, we use this function to deal with it, with the equation below,

\[ p(X) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}. \]  

Firstly, we used 30 variables that mentioned above to build a logistic regression model. Then we used GridSearchCV in scikit-learn package to adjust parameters of model (GridSearchCV is used to systematically traverse various parameter combinations and determine the best effect parameters through cross-validation), the results are shown in Table 3.

From the Table 3 we can identify that the best parameters are C: 0.0001, class_weight: balanced with a score of 0.534166 which is a favorable number for a baseline model.
Table 3. Hyper-parameters and corresponding results of logistic regression models.

| Trails | C    | class_weight | FDR     |
|--------|------|--------------|---------|
| 1      | 0.001| balanced     | 0.533271|
| 2      | 0.001| None         | 0.526118|
| 3      | 0.01 | balanced     | 0.532946|
| 4      | 0.1  | balanced     | 0.532404|
| 5      | 10   | balanced     | 0.532187|

And we also tried to adjust several hyper-parameters manually in the model like set parameter “solver” to saga and then “penalty” to 1 to see if there were any other results. However, the FDR is just 0.50051 which is not changed much and takes much more time to calculate.

6.2. Decision Tree

A decision tree algorithm, or Classification and Regression Tree (CART), is one of the most used and intuitive machine learning algorithms. As its name suggests, a decision tree consists of a tree of decisions to partition the data space into smaller subspaces, where each subspace is given a label or a probability [14]. During training, the tree is built by splitting one node at a time, and the algorithm scans all possible splits along all axes to make sure that the split is made optimal. There are several metrics to measure how good a split is, such as entropy, information gain and Gini, which all measures the impurity of the resulting two partitions, and the best split point is found to result least impurity among partitions. After the tree is built, we then can traverse the tree according to the rules and get the predicted label and its probability for new records.

A decision tree is commonly used due to its fast training, reliability and robustness to dirty data; however, it is prone to overfit, where the algorithm can generate over-complex trees that fit training data so perfectly that they do not generalize to unseen data [14]. There are some methods to overcome the overfitting problem. One is to control the grow of the tree during training, so that it won’t grow too deep or split a leaf node already too small. Another way is to prune the tree after it is built. Some subtrees are selected using a method called minimal cost-complex pruning and removed to help avoid overfitting. There are 4 related hyper-parameters tuned for our decision tree algorithm:

- max_depth: Maximum depth of the tree.
- min_samples_leaf: minimal number of samples in each leaf node. If the number is lower, two leaf nodes will be combined.
- criterion: The criterion to measure impurity.
- ccp_alpha: Cost coefficient for cost-complex pruning. The high the number is, the more node

After tuning, the final setting for parameters are shown in Table 4.

Table 4. Hyper-parameters and corresponding results of decision tree models.

| Trails | max_depth | min_samples_leaf | ccp_alpha | criterion | FDR     |
|--------|-----------|------------------|-----------|-----------|---------|
| 1      | 20        | 64               | 1e-6      | gini      | 0.555979|
| 2      | 80        | 64               | 1e-6      | gini      | 0.555979|
| 3      | 20        | 32               | 1e-6      | gini      | 0.551097|
| 4      | 20        | 128              | 1e-6      | gini      | 0.550120|
| 5      | 20        | 64               | 0         | gini      | 0.542961|
| 6      | 20        | 64               | 1e-6      | entropy   | 0.540900|
6.3. Naïve Bayes

Naïve Bayesian (Naïve Bayesian) is a classification method based on Bayesian theorem and independent assumption of characteristic conditions. It calculates the probability of classification through characteristics and selects cases with large probability. In some explanations Bayesian theorem can tell us how to use new evidence to modify the existing views. Normally the probability of occurrence of event A under event B is different from event B under event A. However, there is a definite relationship between the two, which is Bayesian theorem:

\[
p(A/B) = P(B/A) \times P(A) / P(B)
\]  

From the formula we should know 3 things. The first is \(P(A)\), which is also called Prior probability. That is, we make a subjective judgment on the occurrence probability of the A event without knowing the occurrence of the B event. The second is Likelihood(\(P(B|A)/P(B)\)) which is an adjustment factor that help \(P(A)\) be closer to reality. At last is the result \(P(A|B)\), its meaning is after event B occurred, re-evaluate the probability of occurrence of event A.

It is a supervised learning method based on probability theory. Naïve Bayesian classification is the simplest one in Bayesian classification. The use of naive Bayesian by importing scikit-learn library is relatively simple [15]. Compared to other algorithms like decision trees and KNN, Naïve Bayesian just needs to pay attention to fewer parameters. There are 3 Naïve Bayesian classification algorithms in scikit-learn library. Based on this data set we choose to apply GaussianNB which is used when the distribution of sample features is mostly continuous values.

After we built the model with default parameters (as the GaussianNB model is very simple there is no need to change any parameters), we get the FDR of 0.41013.

6.4. Random Forest

A random forest is an ensemble algorithm based on decision trees. As we mentioned, decision trees have a severe deficiency of overfitting, as a result, the depth and complexity of a decision tree are always controlled, so that it can generalize well on unseen data, which greatly limit its application on sophisticated data. A random forest algorithm combines the idea of ensemble with decision trees to mitigate this drawback. Instead of one decision tree, a random forest builds a cluster of trees on different sub-datasets drawn by random subspace method. Similar to bootstrap aggregation, or bagging, random subspace method sub-samples features randomly with replacement from all features of training sets, from which each tree is built in an attempt to simplify the tree and avoid overfitting. Results of all classifiers are then combined by averaging to give a final prediction.

A random forest algorithm can generally perform better than a single decision tree. It is more robust to noisy data and less prone to overfitting, and it can handle high dimensionality quite well. Since each tree in random forest is built independently, the training and predicting process can be highly parallelized to speed up the computationally intensive model training [16].

We have tuned our random forest model based on the Scikit-Learn package. There are 5 parameters that we find are important:

- **n_estimators**: number of decision trees in a random forest.
- **max_features**: maximum number of features to be drawn to build each tree.
- **max_depth**: maximum depth of each decision tree.
- **ccp_alpha**: cost coefficient for cost-complex pruning.
- **min_samples_leaf**: minimal number of samples in each leaf node.

6.5. Boosted Tree

A boosted tree, or gradient boosting tree, is another type of ensemble classifier based on decision trees. Similar to random forest, it also consists of a collection of trees, but they are constructed differently. Boosted trees add up a series of “weak learners”, such as shallow decision trees, to build a strong estimator, with each contributing small amount of correction to the prediction result. Boosted trees are built incrementally; in each stage, one weak learner is added to try to minimize error between the current
predictions and the real labels represented by predefined loss functions [17]. Each weak learner is fit on the negative gradient of the loss function to try to minimize the loss, resembling gradient descent.

We built our boosted tree model using Python Scikit-Learn package, and explored the following hyper-parameters:

- **n_estimators**: number of weak learners in a boosted tree model.
- **max_depth**: maximum depth of each weak decision tree.
- **learning_rate**: the rate of each learner learning the loss.
- **subsample**: the fraction of samples to be used for training each individual learner.

The learning rate is determined by observing learning curves Figure 4 and other parameters are optimized via grid search. Some trials in tuning and the final set of hyper parameters are shown in Table 5.

![Figure 4. Learning curve of boosted trees.](image)

| Trails | n_estimators | learning_rate | max_depth | subsample | FDR       |
|--------|--------------|---------------|-----------|-----------|-----------|
| 1      | 500          | 0.05          | 5         | 0.5       | 0.553928  |
| 2      | 500          | 0.05          | 5         | 0.75      | 0.553928  |
| 3      | 500          | 0.05          | 5         | 1         | 0.553385  |
| 4      | 750          | 0.05          | 5         | 0.5       | 0.553494  |
| 5      | 400          | 0.05          | 5         | 0.5       | 0.554036  |
| 6      | 200          | 0.05          | 5         | 0.5       | 0.552517  |

6.6. **AdaBoost**

An AdaBoost classifier, short for adaptive boosting, is another ensemble algorithm [18]. It builds a complex model by combining results from much simpler models. At each time during training, a copy of the simple classifier is fit on the whole dataset where weight of each record is adjusted according to the current AdaBoost model. Wrongly classified records will receive higher weights, such that subsequent classifiers can focus more on those challenging cases. Here, shallow decision trees are used as the simple base model, because of their short training time on large dataset and decent accuracy. There are three hyper-parameters we found interesting and important to an AdaBoost model:

- **n_estimators**: number of weak learners in an AdaBoost model.
- **max_depth**: maximum depth of each weak decision tree.
- **learning_rate**: the rate of each learner learning the loss.

The learning rate is tuned first via learning curve plots in FIGURE.
6.7. Neural Network

The neural network model is a kind of imitation of the information input and output of the neural network composed of multiple neurons in the human brain. This model is often used in supervised learning to train the model to recognize and distinguish one thing from another through large data sets. In the model of identifying credit card fraud, we found that the neural network model with fewer layers and fewer nodes performed better than the complex neural network model. We tried to go from one hidden layer to ten hidden layers. The best performer is the two hidden layers. After that, the model performed worse and worse in the out of date (OOT) dataset [19]. In addition, as the number of model layers reached six layers, it's getting harder to converge. After a lot of testing, we found that the best performance was at 30 nodes per layer. The following is a selection of seven representative groups of experimental results that can be compared with each other.

We used the package sklearn.neural_network and the parameters are as follows:
- hidden_layer_sizes: the ith element represents the number of neurons in the ith hidden layer
- activation: activation function for the hidden layer
- solver: the solver for weight optimization
- alpha: L2 penalty (regularization term) parameter
- max_iter: maximum number of iterations

6.8. Support Vector Machine

Support vector machine (SVM) model is a supervised learning model. After the marked data set is given, the model will search for the optimal hyperplane in the high dimensional space as a classifier. In the case of credit card fraud, the performance of the model is very poor. The dimension is too high, so it is difficult to converge. Coupled with the data imbalance, the accuracy of the prediction is very low, so we abandon the model.

6.9. K-Nearest Neighbor

KNN is also known as K nearest neighbor, which literally means that the category of the test sample is determined according to the category of the K training samples whose test sample is relatively near (attribute is relatively near). KNN is mainly dealing with classification problems [20]. It is a relatively simple machine learning algorithm, there are only hyper-parameters in this model rather than model parameters, therefore we need to choose the most appropriate hyper-parameters to best fit the data set:

- Algorithm: which means what algorithm is used to search the nearest k points when building the KNN model, for example brute, KD_tree, ball_tree, and auto
- n_neighbors: That is, to select the nearest points, the default value is 5
- Weights: It can be set to uniform(default) or distance. The sample weight can link to the distance, the near point weight is significant while the far point weight is small

Figure 5. Schematic diagram of K-nearest neighbor Algorithm.
Since this data set is extremely large, the algorithm we chose here is KD_tree which is used when the number of samples is greater than 2 to the number of variables. Next, we set the n_neighbors to 3 to get a better performance as 3 is the most efficient options after many attempts. In order to save compute time, we set the parameters to {'n_neighbors': [3], 'weights': ['uniform'], 'Algorithm': [KD_tree]} to build our model, but there are no results after a long time. So we changed few parameters to fit the data however it still failed.

The biggest problem we are facing is that the data is too sparse and therefore the model is unable to find the nearest sample around it or if the model algorithm has its own problems. To figure this out, we then reduce the size of training dataset to 2000, it turns out the results have come up and we also embed KNN models to bagging classifier; nevertheless, FDR is still lower than our baseline. Based on this, we find that KNN model algorithm is not suitable for this data set and the problem is not the data set itself. Therefore, we discarded this model.

6.10. Random Under Sampling and SMOTE
Frauds are rare by nature, which makes capturing them extremely hard, especially with machine learning algorithms. Our dataset contains only 1.4% fraud records. Training a model with such imbalanced input data can affect its performance sometimes, so we tried a few approaches to address this issue [21].

We can already see that for logistic regression model shown in Table 4, setting the weights to be ‘balanced’ boosts weights of fraud records and can indeed improve FDR by roughly 1%. Boosting weights is equivalent to over sample all frauds evenly, but it can introduce more computation, which is not preferred for complex models. Random under sampling is also another common way to tackle the imbalanced data problem. Records from the major category are randomly drawn without replacement to certain size less than its original one, making the skewness of the dataset less severe. Since less data are fed into machine learning algorithms, training time can generally be shorter. We tried random under sampling with the random forest model we tuned, several models are built on differently resampled dataset to minimize random error.

Resampling only generates new dataset using existing data, however, the dataset sometimes may be too small to make resampling strategies work properly, so generating new data based on distribution of existing data can be important. Synthetic minority oversampling technique, or SMOTE, was invented to tackle this exact problem. A new sample is generated by linear combination of two similar samples. SMOTE is found to have more profound impact on distance-based models such as KNN or SVM. However, both KNN and SVM algorithms cannot converge since the size and dimensionality of our data are too large. Instead, a decision tree model on dataset generated by SMOTE is tested. The dataset is first randomly under sampled, and SMOTE is run subsequently, which lowers the number of new samples generated and shortens runtime of SMOTE. Several models are trained on differently resampled dataset to minimize random errors with hyper-parameter.

7. Results
Having tuned best hyper-parameters for 9 models we tested, all models are built and tested with 10-fold cross validation, and the FDR at 3% scores, including training, validation test and out-of-time test, are shown in a boxplot (Figure 6) and Table 6 below. We find that more complex non-linear models, including random forests, boosted trees, neural networks and AdaBoost all performs better than simple ones, like logistic regression. The random forest algorithm outperforms others only marginally, in terms of both testing and OOT scores, but it also has the least standard deviation in test scores among all models, which means that random forests are more stable and robust. Therefore, we choose random forest as our best model for further study.
Table 6. Average and standard deviation of FDR (3%) scores of training, testing and OOT set for all models tested. All scores are obtained by 10-fold cross validation.

| Model | Training Score | Testing Score | OOT Score |
|-------|----------------|---------------|-----------|
|       | Mean  | STD   | Mean  | STD   | Mean  | STD   |
| LR    | 0.5298 | 0.0019 | 0.5286 | 0.0104 | 0.5185 | 0.0009 |
| NB    | 0.5294 | 0.0021 | 0.5291 | 0.0192 | 0.5169 | 0.0000 |
| DT    | 0.5458 | 0.0018 | 0.5440 | 0.0116 | 0.5353 | 0.0006 |
| RF    | 0.5543 | 0.0012 | 0.5507 | 0.0096 | 0.5401 | 0.0004 |
| BT    | 0.5512 | 0.0016 | 0.5494 | 0.0140 | 0.5386 | 0.0008 |
| ADA   | 0.5498 | 0.0012 | 0.5491 | 0.0105 | 0.5391 | 0.0003 |
| NN    | 0.5429 | 0.0015 | 0.5418 | 0.0119 | 0.5317 | 0.0014 |
| RF_US | 0.5664 | 0.0014 | 0.5493 | 0.0100 | 0.5384 | 0.0009 |
| SMOTE | 0.5563 | 0.0011 | 0.5450 | 0.0135 | 0.5335 | 0.0009 |

It is also noticed that the tests scores have much larger deviations than that of train or OOT, which could be caused by the way train and test data are split. In the cross-validation implementation we used, data are randomly split without retaining the ratio of frauds and non-frauds, as a result, records that are hard to classify for machine learning algorithms can sometimes cluster in one group of test data and possibly affect the FDR of that particular model. On the other hand, OOT data are split before cross-validation and should always be the same, and thus might make OOT scores more stable.

Figure 6. The boxplot of FDR (3%) scores of training, testing and OOT for all models. The green bar indicates mean, and the orange bar indicates median.

As for models on resampling dataset and SMOTE, they have much higher training score than the ones fit on non-balanced dataset, indicating non-linear models can indeed be trained better on more balanced datasets. However, their performance on test and OOT set is much worse than others. This could be because the resampled dataset or SMOTE generated dataset are still biased towards some
records and hindered the generalization ability. Also, though our dataset is highly skewed, there are still large number of records even for the minority class. Additionally, almost all features are created with discrete values with frauds often being extreme values, which might make decision tree models and other models based on it perform better, because it would be easier to make a “clean cut” in the data space, regardless of the skewness of data.

With our best performing model random forest, we then built three tables using training, testing and OOT set respectively, each showing FDR and other statistics for each population bin. Records are ranked in descent order based on their probability of being fraud given by the trained random forest model. From the tables, we can see that more than 50% of frauds are caught within the very first percentile bin. Fraud detection rate at 3%, or the cumulative percentage of bad, are 55.67%, 54.08% and 53.52% for training, testing and OOT respectively, which are consistent with our cross-validation results.

8. Conclusion
In this article, we build machine learning models on a synthetically generated dataset about credit card applications and evaluated their performance. We first explore the statistics of the data consisting 1 million records and 10 fields. Frivolous values are then cleaned by substituting. After the data is cleaned, expert variables are created mainly based on application frequency. Identity entities, such as full name and full address, are used to create variables for three categories, day since, velocity and relative velocity, yielding 331 features in total. Since the dimensionality of the data is too high for any machine learning models to be practical, we apply filter and wrapper for feature selection and dimensionality reduction. KS and FDR are used as filter metrics to reduce the number of features to 80, and then we run a wrapper for more fine-tuned selection. The best 30 features are eventually selected to build machine learning models.

Logistic regression is first built as a baseline model due to its simplicity, which results 52.86% FDR at 3% cutoff. We subsequently explored naïve Bayes, decision trees, random forests, AdaBoost, gradient boosted tree, neural network, support vector machine and K-nearest neighbor, among which the random forest model has the best performance of 55.07% FDR at 3% in testing. We did not manage to make KNN and SVM models to converge, probably because the dataset we are using is too large for such computationally intensive models. The test using out-of-time dataset are applied and the OOT scores are consistent with training and testing results. Tables of the predicted percentile bins using the leading model random forest are built, and we can see that over 50% of the frauds can be captured within the very first percentile.

There are also aspects where our models can be improved. Given additional time, we may do more extensive searching in the hyper-parameter space of each machine learning model to improve accuracy and reduce complexity. More research on related literature can also listed to give a more comprehensive comparison between models and fraud types, which may be beyond the scope of this study. As for building more models, we can also try ensemble model, such as voting or stacking classifiers, to further improve performance.

Reference
[1] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, “Data mining for credit card fraud: A comparative study,” Decision Support Systems, vol. 50, no. 3, pp. 602–613, 2011.
[2] J. West and M. Bhattacharya, “Intelligent financial fraud detection: A comprehensive review,” Computers & Security, vol. 57, pp. 47–66, 2016.
[3] Federal Trade Commission, 2020, CONSUMER SENTINEL NETWORK DATA BOOK 2019
[4] R. J. Bolton, D. J. Hand, Unsupervised profiling methods for fraud detection, Conference on Credit Scoring and Credit Control, Edinburgh, 2001.
[5] Leonard Kevin J.. Detecting credit card fraud using expert systems[J]. Pergamon,1993,25(1-4).
[6] Dragomir Dimitrijevic, Vesna Milovanovic, Vladimir Stancic. The Role of A Company’s Internal Control System in Fraud Prevention. 2015, 11(3):34-44.
[7] M. Albashrawi. “Detecting Financial Fraud Using Data Mining Techniques: A Decade Review from 2004 to 2015”, Journal of Data Science. 14. 553-570, 2016.
[8] Cristina Soviany. The benefits of using artificial intelligence in payment fraud detection: A case study. 2018, 12(2):102-110.
[9] Dan Frechtling. Recognising and thwarting transaction and payment laundering. 2017, 11(2):110-120.
[10] Anonymous. Identity theft rockets as criminals buy more personal info online[J]. Computer Act!ve,2015(451).
[11] Business Week Group. Swiping Back at Credit-Card Fraud[J]. Bloomberg Business Week,2005,(3942):72-0.
[12] Hancer, Emrah. Differential evolution for feature selection: a fuzzy wrapper-filter approach[J]. Soft computing: A fusion of foundations, methodologies and applications,2019,23(13):5233-5248.
[13] Worster Andrew,Fan Jerome, Ismaila Afisi. Understanding linear and logistic regression analyses.[J]. CJEM,2007,9(2).
[14] Sikdar Md. S Askari, Md. Anwar Hussain. I FDTC4.5: Intuitionistic fuzzy logic based decision tree for E-transactional fraud detection[J]. Journal of Information Security and Applications,2020,52.
[15] Farhad Alam, Sanjay Pachauri. Comparative Study of J48, Naive Bayes and One-R Classification Technique for Credit Card Fraud Detection using WEKA[J]. Advances in computational sciences and technology,2017,10(62):1731-1743.
[16] Lakshmi, S. V. S. S., Selvani Deepthi Kavila. Machine Learning For Credit Card Fraud Detection System[J]. International Journal of Applied Engineering Research,2018,13(24 Pt.1):16819-16824.
[17] James Zhang, Robert Gardner, Ilija Vukotic. Anomaly detection in wide area network meshes using two machine learning algorithms[J]. Future Generation Computer Systems,2018,93.
[18] Anusree B, B. Ramesh Kumar. Credit Card Fraud Detection Using Adaboost And Majority Voting[J]. International Organization of Scientific Research,2018,8(11):48-53.
[19] Masoumeh Zareapoor, Pourya Shamsolmoali. Application of Credit Card Fraud Detection: Based on Bagging Ensemble Classifier[J]. Procedia Computer Science, 2015,48.
[20] Alin Dobra, Amit Dhurandhar. Probabilistic characterization of nearest neighbor classifier[J]. International journal of machine learning and cybernetics, 2013,4(4):259-272.
[21] Blagus, R., Lusa, L. SMOTE for high-dimensional class-imbalanced data. BMC Bioinformatics 14, 106 (2013).