Credit Card Transaction Fraud Detection

University of Southern California

DSO 562 Fraud Analytics

Spring 2020

Prepared By:

Aixuan Liu	aixuanli@usc.edu
Chao Wang	wang724@usc.edu
Cheng Shi	shi005@usc.edu
Chengjun Liu	cliu4428@usc.edu
Minglu Chi	mingluch@usc.edu
Shijie Xiang	shijiex@usc.edu

Table of Contents

1. Executive Summary

1.1. Business problem

Credit card fraud is a major type of financial fraud that incurs about $24.26 billion financial loss worldwide[1]. Among all kinds of credit card fraud, transaction fraud is one of the most common types of fraud that happens during the usage of credit cards. It usually occurs when the credit card got stolen or counterfeited, or the account of the card got hacked. Therefore, catching credit card transaction fraud in real-time is a major business problem for card issuers or organizations overseeing credit card transactions such as VISA and Mastercard to tackle, so that they can protect consumers, merchants, and banks from loss.

1.2. Proposed solution and result

In this project, we created a real-time supervised Neural Net machine learning model with 54.7% fraud detection rate at 3% in validation data. The data we used is the credit card transaction data of a Tennessee government agency in 2010, which contains a fraud label.

The project started with a data quality assessment, in which we identified outliers and columns with missing value. Next we conducted a data cleaning process to remove the outlier and fill in the missing value. Then we created 302 candidate expert variables to catch the common patterns of credit card transaction fraud. To ensure that our model can catch fraud transactions in real-time, we built candidate variables with only past information available to each record. After this step we splitted the data into two parts: one part for training and testing, including records before 2010-11-01, and one part for out-of-time validation, including the last two months of data. Then we performed feature selection on the training and testing data between 2010-01-15 and 2010-10-30 using a two-step approach: applying two univariate filters first, and performing stepwise forward selection, with which we selected 25 expert variables as our final features. Next we tried four machine learning algorithms for modeling, including Logistic Regression, Boosted Tree, Random Forest, and Neural Net, and conducted rigorous training and testing

[1]Data retrieved from: https://shiftprocessing.com/credit-card-fraud-statistics/

procedure to select the best algorithm based on Fraud Detection Rate at 3%. Finally, we selected Neural Net as the best-performing algorithm, produced the model performance tables on training and testing data, and used all training and testing data to train the model one more time and ran the model on the validation data.

2. Description of Data

Card Transaction Data documents the credit card transactions made on behalf of a government organization in Tennessee in the year of 2010. There are 10 fields, including a fraud label field, and 96,753 records in total. In total, there are 1059 fraudulent transactions in this dataset, and the overall fraud rate is 1.1%.

2.1. Field summary

Table 1 Numeric Field

	Number of populated records	% of populated records	# of unique values	# of records with value zero	Mean	STD	Min	Max
Amount	96753	100	34909	0	427.89	10006.14	0.01	3102045.53

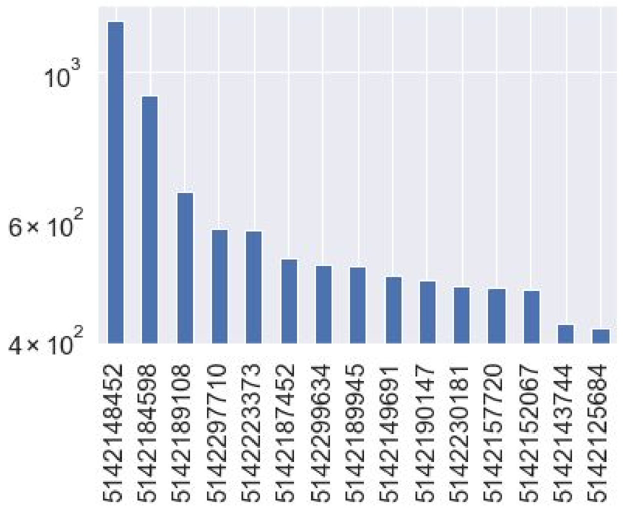
Table 2 Categorical Fields

	# of records that have a value	% populated	# of unique values	# of records with value zero	Most Common Field Value
Recordnum	96753	1	96753	0	NA
Cardnum	96753	1.00	1645	0	5142148452
Date	96753	1.00	365	0	2/28/10
Merchnum	93378	0.97	13092	0	930090121224
Merch description	96753	1.00	13126	0	GSA-FSS-ADV
Merch state	95558	0.99	228	0	TN
Merch zip	92097	0.95	4568	0	38118
Transtype	96753	1.00	4	0	P
Fraud	96753	1.00	2	95694	0

2.2. Field descriptions

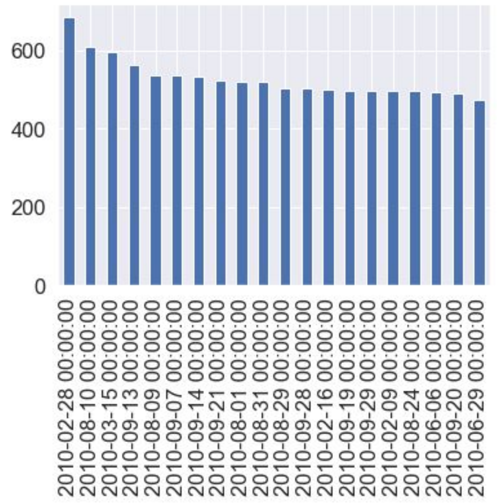
2.2.1. Cardnum

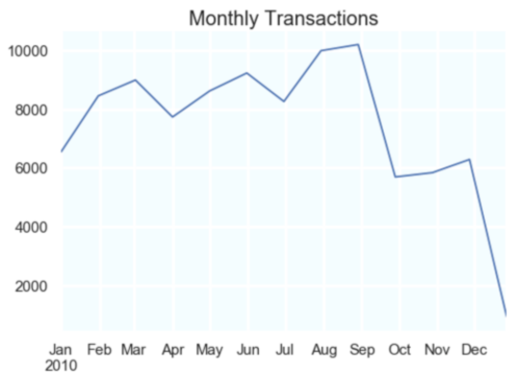
A categorical variable with 1645 unique values. It is the number of the credit card used in each transaction. The distribution of the top 15 card numbers with the most occurrences are shown below:

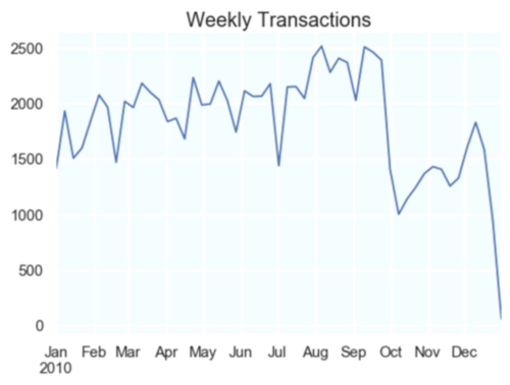


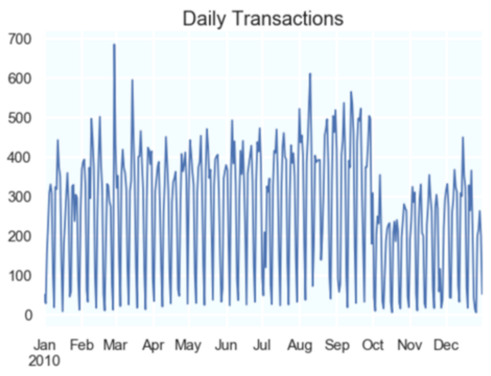
2.2.2. Date

This field is the dates of the transactions, ranging from 2010-1-1 to 2010-12-31. The number of occurrences of each date in the dataset is equal to the number of transactions made on that particular date. Therefore, we can plot the daily, weekly and monthly transaction to see the distribution of dates and trends in transactions. We can see that the credit card transaction has obvious weekly seasonality – activities on the weekend are usually close to zero. We can also see that September has the highest transaction volume, possibly due to the fact that the government tries to spend the remaining budget of the year before the fiscal year ends.



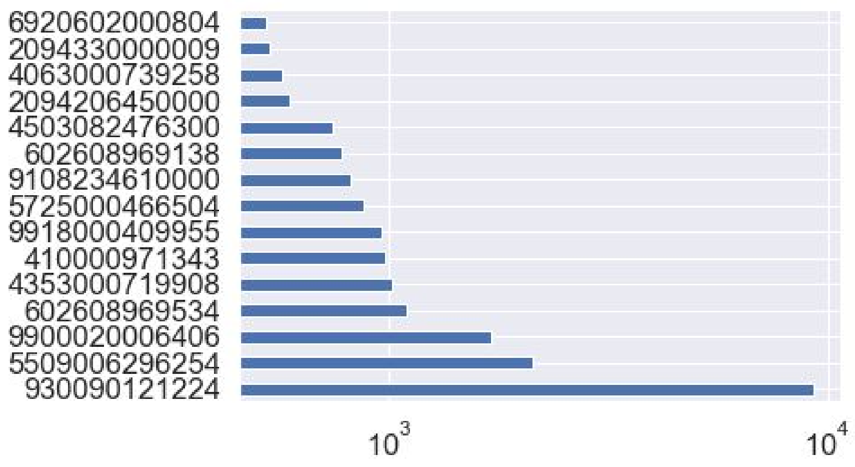






2.2.3. Merchantnum

A categorical variable with 13092 unique values. It is the unique identifier for each merchant. The top 15 merchant numbers with the most occurrences are shown below.



2.2.4. Merch description

A categorical variable with 13126 unique values. It contains detailed information about the merchant in each credit card transaction, such as the name of the merchant, etc. Top 10 values are shown in the chart.

GSA-FSS-ADV	1688
SIGMA-ALDRICH	1635
STAPLES #941	1174
FISHER SCI ATL	1093
MWI*MICRO WAREHOUSE	958

CDW*GOVERNMENT INC	872
DELL MARKETING L.P.	816
FISHER SCI CHI	783
AMAZON.COM *SUPERSTOR	750
OFFICE DEPOT #1082	748

2.2.5. Merchant state

A categorical variable with 228 unique values. It consists of not only the two letter abbreviation of American states, but also some digit combinations that indicate some specific geographical locations all around the globe. Top 10 values are shown in the chart.

TN	12035
VA	7872
CA	6817
IL	6508
MD	5398
GA	5025
PA	4899
NJ	3912
TX	3790
NC	3322

2.2.6. Merchant zip

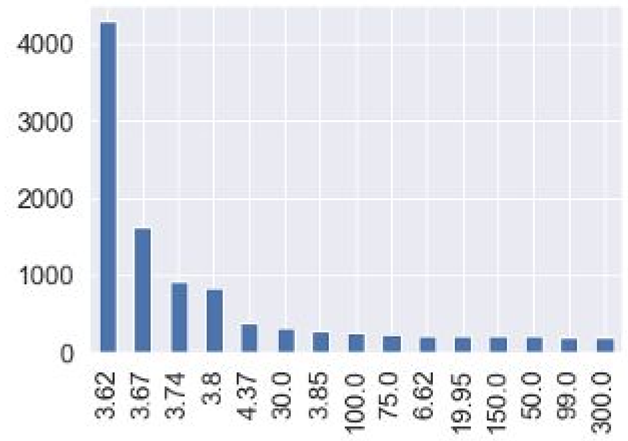
A categorical variable with 4,568 unique values. The top 10 zip codes with the most occurrences are shown below. A large number of records do not have the zip code field populated, and some values in this field are less than 5 digits.

38118	11868

63103	1650
8701	1267
22202	1250
60061	1221
98101	1197
17201	1180
30091	1092
60143	942
60069	826
78682	817

2.2.7. Amount

The only numeric variable in this dataset and it indicates the amount of each transaction. Below is the distribution of the top 15 amounts that have the most occurrences. Amounts with a large number of occurrences may be the result of regular recurring government expenses. In this dataset, we see that Fedex delivery fees are mostly a set amount with a lot of occurrences.



2.2.8. Fraud

A categorical variable with two unique values: 0 representing non-fraudulent records, and 1 representing fraudulent records. It is a label that is manually added to the dataset for educational purposes. With this column, we can build a supervised machine learning model.

Fraud	Count	Percentage
1	1059	1.1%
0	95338	98.9%

3. Data Cleaning

After exploratory data analysis, we can see that there is a significant number of records with missing values, and likely misentered values. Therefore we conducted data cleaning to prepare the data for analysis.

3.1. Remove irrelevant records and outliers

For this analysis, we focus on the dominant transaction type P. Therefore, other transaction types are removed from the dataset.

It can be seen from the data summary that the maximum value in the Amount column is a single extreme value much higher than all other amounts. After consulting with experts we believe that this large amount is likely a misentry. Therefore, we remove the record with the extreme amount from the dataset.

3.2. Filling missing values and cleaning misentered values

In exploratory data analysis, we found that Merchnum, Merch state, and Merch zip fields have missing or misentered values. Since merchants are a possible entity to commit fraud, and fraud behavior by a particular entity usually happens in a burst and can be caught by counting repetitions, we applied the following procedure to clean the data in order to restore matching information as much as possible, and at the same time to not create false repetition in data so we do not trigger false alarm.

3.2.1. Merchnum

For Merchnum field, the procedure is as follows:

1) Grouping by Merch description and Merch zip and filling the most frequent value
   Merch description is a 100% populated column that has direct information about each merchant. The Merch zip column further specifies merchants of different branches, though the column is not completely populated. Therefore, we first populated Merchnum by grouping by Merch description and Merch zip altogether. Yet Merchnum and the "Merch description + Merch zip" combination

still do not have the one-to-one relationship we were looking for, meaning if two records have the same Merch description and Merch zip, their Merchnum values might not necessarily be the same. Therefore, we decided to group records by Merch description and Merch zip, and filled the missing Merchnum field with the most frequent Merchnum value in the group.

2) Filling remaining records with Recordnum
   After the first step, we filled the rest of the missing Merchnum by record number. For records with the same Merch description and Merch zip, we filled the same record number in Merchnum by filling the smallest record number of all records with missing Merchnum value and the same Merch description and Merch zip values.

3.2.2. Merch state

For Merch state field, the procedure is as follows:

1) Grouping by Merch description and Merch zip and filling the most frequent value
   This step is similar to that of Merchnum.

2) Matching by Merchnum
   Since now Merchnum is all properly populated, we matched Merch state by Merchnum. If two records have the same Merchnum, their Merch state should be the same.

3) Matching by Merch zip
   Merch state and Merch zip have a one to many relationship. Records that have the same Merch zip should have the same Merch state. We first matched records that have missing Merch state values with records having the same Merch zip and populated Merch state value. For Merch zip values that do not have corresponding

Merch state information in the dataset, we used the python library uszipcode to identify their state affiliation.

4) Filling remaining records with Recordnum

   This step is similar to that of Merchnum.

3.2.3. Merch zip

For Merch zip field, the procedure is as follows:

1) Grouping by Merch description and Merch zip and filling the most frequent value

   This step is similar to that of Merchnum.

2) Filling remaining records with Recordnum

   This step is similar to that of Merchnum.

4. Creating Candidate Variables

We created candidate variables to catch the most common modes of card transaction fraud. Typical signals of fraud card transactions include several transactions with unusually high transaction amounts, previously unseen geography or merchants of the transaction, sudden burst in transaction frequency or infrequent small-amount recurring charges. In addition, fraud transactions usually have common points of compromise. A fraud merchant can make up transactions with several stolen credit card information, or fraudsters can make purchases with several stolen credit cards.

To catch these signals, we created six kinds of expert variables, and we only look back in time to calculate the value for each record:

- Amount: Several statistical summarization of transaction amount over a period of time
- Days Since: How many days since certain entity was last seen
- Frequency: The number of transactions with the same entity within a specific time period
- Velocity Change: The ratio of the number of transactions with the same entity in a short period of time versus a longer period of time
- Benford’s Law: Frequency distribution of the first non-zero digits of transaction amounts grouped by certain entities
- Day-of-Week Risk Table: Fraud risk on each day of week

4.1. Combining related fields into entities

Apart from Cardnum and Merchnum, we also concatenate Cardnum with Merchnum, Cardnum with Merch zip, and Cardnum with Merch state to create entities with linking information catching changes in card usage, merchants, and geography.

Entities:

“cardnum”, “merchnum”, “cardnum-merchnum”, “cardnum-merch zip”, “cardnum-merch state”

4.2. Creating candidate variables

This section covers the logic and implementation details of creating the variables in Python, specifically how we made sure that we only look back in time to calculate the value for each variable. Loop was used to automatically repeat the execution of variable creation within predetermined entities and timeframes. First, we created 2 duplicate data frames (df1, df2), which contain "record", "date", "amount" and all entities. A list of "lags" in days was also generated, which includes 0, 1, 3, 7, 14 and 30 days.

4.2.1. Amount variables

The first type of variable to create is amount variables, namely, six statistical summaries of the transaction amount of a particular card transaction record over a recent time period. This variable can identify records with unusual transaction amounts.

1) Create date_lags fields

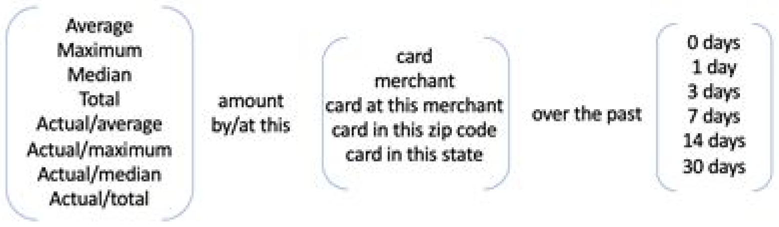
   For each number in the "lags" list, we concatenated it with "date" and generated a list of date_lags, such as "date_7". Then we employed the timedelta function from the datetime package, adding each lag number to each date.

2) Calculate six statistical measurements

   With each entity and combination group as a key, we performed an full outer merge of df1 and df2 and generated df3. For such fields as "record" and "date" that appear in both df1 and df2, df3 treats fields from df1 as "record_x", "date_x" and fields from df2 as "record_y" and "date_y".

   To ensure that only records before the time of a particular transaction are used, we set up filters: "record_x" >= "record_y" and "date_x" <= "date_y". Then the result was grouped by "record_x".

Finally, we calculated the average, maximum, median, total as well as the actual/average, actual/maximum, actual/median, actual/total amount with these five entities over the past 0, 1, 3, 7, 14 and days. In total we created 240 amount variables.



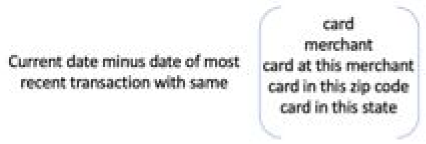
4.2.2. Days Since variables

The second type of variable to create is Days Since variables, namely the number of days elapsed since the transaction with the same entity was last seen. The days-since variable aims to identify records with abnormal transaction frequency.

With a similar method of creating time lags and calculating previous records as the Amount variables, the last date the entity was seen in the past was extracted and used to calculate the number of days lapsed.

However, the majority of records are not fraudulent. There are many empty values in this variable "#d_since", meaning that there is no transaction with the same entity in the past. Therefore, we replace the empty values with the number of days elapsed since the first day of year 2010.

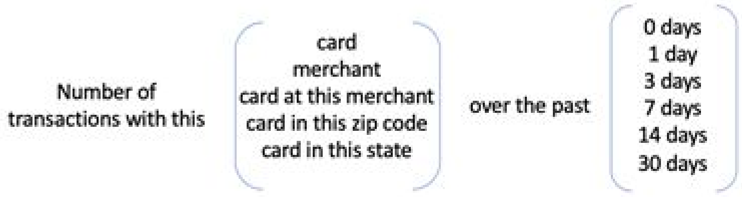
As a smaller number in this variable represents a higher possibility of fraud, replacing empty fields with a larger number will signal a low possibility of fraud. In total, we created 5 days-since variables.



4.2.3. Frequency variables

The third type of variable is frequency variables, which refers to the number of transactions with the same entity information within a specific time frame. If the same entity is frequently seen within a certain short time window, it indicates potential fraud transaction.

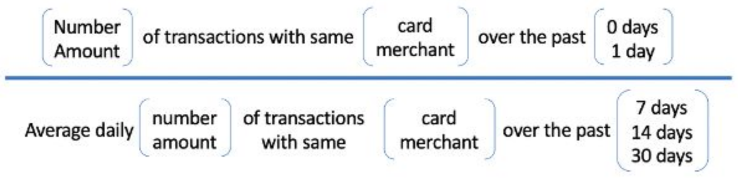
We counted the number of transactions before the time of a particular transaction but within a specific timeframe. For these variables, a similar implementation method was used as the Days Since variable, but we not only filter records by record number, but also date. In total, we built 30 frequency variables.



4.2.4. Velocity Change variables

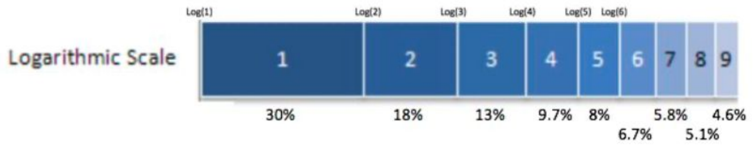
The fourth type of candidate variables is Velocity Change variables, which is the ratio of a short-term number of or amount of transactions versus a long-term average number of or average amount of transactions.

It has the same idea as the frequency variable, except that we also calculate the amount of transactions with the same cardnum or Merchnum. And we take those in a short period (0, 1 day) as numerator and those in a longer period (7, 14, 30 days) as denominators. The numerator is the total concept within a short period, whereas the denominator is the daily average concept within a long period. In this way we can identify the frequency of card transactions, especially if there is a burst of fraud events in a very short time.In total, we created 24 Velocity Change variables.



4.2.5. Benford’s Law variables

The fifth type is Benford’s Law variables. Benford’s law is an observation describing the frequency distribution of the leading non-zero digit of many real-life numerical cases. The distribution of the first digits do not follow a uniform distribution. Instead, the bigger the digit number, the smaller the frequency, and one, two, and three appear more than half of the time as the first digit. By investigating whether the first non-zero digits of amounts associated with Cardnum or Merchnum follow this distribution, we can identify whether transactions are made up by fraudsters, in this case possibly merchants. Before applying Benford’s Law to create variables, we first removed all the Fedex transactions. This government agency uses Fedex for shipping, and most of the shipping transactions are of the same amount. This is not unusual, but the repetition of a particular first-digit number may raise a red flag in the Benford’s Law variables.



1) Calculate n_{low} and n_{high}

   To quantify how different the distributions of non-zero first digits are from the Benford’s Law distribution, we first counted the number of records whose amounts start with either 1 or 2 (n_{low}), and the number of records of those with a higher number from 3 to 9 (n_{high}). If either number was zero, we replaced it with 1 for the convenience of the following step.

2) Calculate U

   Next we calculated an intermediate variable R according to the formula shown below on the left. 1.096 is the ratio of the number of higher leading digits (3-9) to that of lower leading digit (1-2) according to Benford’s Law (52.3%/47.7% ≈ 1.096). After that, we also calculated 1/R. For each record, the measure of unusualness U is the bigger number between R and 1/R. The bigger the U, the more unusual a record is.

$$R = \frac{1.096 \times n_{\text{low}}}{n_{\text{high}}} \qquad U = \max(R, 1/R)$$

3) Calculate U*

   Finally, we used a smoothing function to get the final measure of unusualness in case there is an insufficient number of records in either high-number or low-number group. Insufficient numbers of records will lead to bias in the measurement of unusualness. The formula of the smoothing function is shown below. We used c = 3 and n_{mid} = 15 according to expert suggestions.

$$U^* = 1 + \left(\frac{U-1}{1+\exp^{-t}}\right) \qquad t = (n - n_{mid})/c$$

In total, we created 2 Benford's law variables respectively for Cardnum and Merchnum.

4.2.6. Day-of-Week Risk variable

The last type of candidate variable is the Day-of-Week risk variable. We used the day of week of the transaction to group the data and calculated the probability of transactions happening on this day being fraud.

Since day of week is a categorical field, target encoding was employed to assign a value for each category in a supervised model. Target encoding will not result in dimensionality expansion. And each categorical value of the day of week will have a numerical value accordingly.

Similarly, a smoothing formula was used to prevent bias caused by insufficient numbers of records. We grouped the data by day in the week and calculated the average number of fraud as Y_{high} for each day of the week. Y_{low} is the average number of fraud in a day across all records, regardless of day of week. We used the same smoothing parameters (c = 3 and n_{mid} = 15) as in the Benford's Law variables.

$$\text{Value} = Y_{\text{low}} + \frac{Y_{\text{high}} - Y_{\text{low}}}{1 + e^{-(n-n_{mid})/c}}$$

In total, we created 1 Day-of-Week risk variable.

Overall, 302 candidate variables were built in the process.

4.3 Data normalization

After variable creation, we Z-scaled all variables to bring values to the same origin and the same scale, which can further distinguish anomaly from normality. The formula of z-scaling is displayed below:

$$z = \frac{x_i - \mu}{\sigma}$$

4.4 Splitting data

After normalizing all variables, we first removed the first 2 weeks of records from the dataset, since the Days Since variables have limited days to look back into in the first two weeks so normal records are likely to stand out as fraud with a small value in this field during that period of time. Next we splitted the data into two parts. Records with dates before 2010-11-01 were put in one file as training and testing data, and records with dates after 2010-10-31 were put in one file as the out-of-time validation data.

5. Feature Selection

With 302 variables created, we conducted feature selection with a two-step method. We first used univariate filters to select the most promising 80 variables, and then used a forward selection wrapper to reduce the dimensionality and keep the best number of variables.

5.1. Univariate filters

We used two measures of goodness as filters: “Kolmogorov-Smirnov” (KS) score and Fraud Detection Rate (FDR). The KS score is used to measure how separate is one distribution from the other distribution. The formulate of KS is displayed below:

$$KS = \max_{x} \sum_{x_{min}}^{x} [P_{\text{goods}} - P_{\text{bads}}]$$

By calculating KS, we can measure how well a particular variable can distinguish the distribution of the normal records and the fraud records, or the fraud label 0 and 1. The higher the KS score, the better performance of the variable in separating fraudulent records from normal records.

FDR is the percentage of correctly detected frauds among records at a certain percentile. We used FDR score to measure a variable’s ability to find true positives among the records with a top 3% highest or lowest value of this variable. The higher the FDR score at 3%, the better the variable is at fraud detecting.

Filter methods were implemented in the following steps:

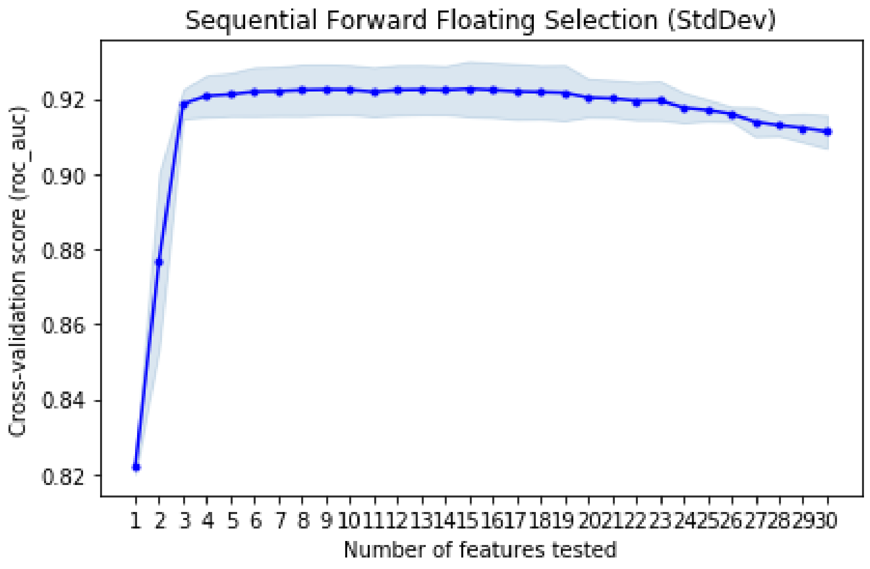
1) Create a new variable “random”, which is a sequence of random numbers generated, and perform filters on this “random” variable as well as the fraud label as a variable along with other 302 expert variables we created to check if the KS and FDR measurements work efficiently. Ideally, the fraud label should get perfect scores from the two filters and “random” should get the worst scores.

2) Calculate the KS scores and FDR at 3% for all 304 variables and rank the variables descendingly with the two scores and assign two rank orders to each variable. From our process we saw that the fraud label ranked at the top and the “random” variable ranked the last for both FDR and KS scores.
3) Remove the fraud label and the “random” variable, and take the average of two rank orders as the final rank, and select the top 80 variables to proceed to the wrapper selection.

5.2. Wrapper

We used Sequential Forward Selections (SFS) to reduce dimensionality based on cross validation scores. It uses the forward selection technique to select a subset of features that produces the best model results in a greedy fashion.

A Logistic Regression model was used as the estimator in SFS, and we dropped the number of features from 80 to 25 stepwise with five runs. Then we plotted the number of features versus cross validation scores from our final run in the picture below.



The optimal number of features suggested by the algorithm was 25 with a best performance score of around 0.91. Therefore, we picked the top 25 variables from SFS ranking. The top 25 variables are: 'card_ps_3d_total','card-state_ps_14d_total', 'card-state_ps_3d_maximum','card-state_ps_7d_maximum', 'card-state_ps_30d_total', 'card_ps_0d_total', 'card-state_ps_1d_maximum', 'card-merchant_ps_3d_total', 'card_ps_0d_maximum', 'card-state_ps_3d_average', 'card-state_ps_1d_average', 'card-state_ps_1d_median','card-zip_ps_30d_total', 'card-state_ps_7d_average', 'card-state_ps_3d_median', 'card-merchant_ps_30d_total', 'card-zip_ps_1d_total', 'card-merchant_ps_1d_total','card_ps_30d_average', 'card-state_ps_7d_median', 'merch#_ps_3d_total', 'card-state_ps_0d_maximum', 'card-state_ps_14d_median', 'card-state_ps_0d_total', 'card-state_ps_0d_average'.

6. Model Algorithms

Four candidate models were chosen for this project, namely Logistic Regression, Boosted Tree, Random Forest, and Neural Net. We ultimately selected Neural Net based on cross validated classification performance of training and testing data. FDR at 3% was selected as the model measure of goodness in line with industry standards.

The general procedure of model training is as follows:

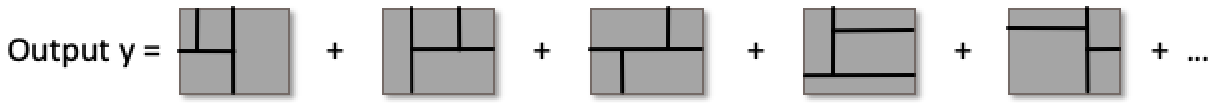
1) For different model types, we fit and test models with cross validation using a range of hyperparameters on the training and testing dataset (data from 2010-01-01 to 2010-10-31), and then use average FDR at 3% to choose our favorite model algorithm as well as its hyperparameter. To calculate FDR at 3%, we use the model to predict the fraud probability of each record as a fraud score. We sort all records by the predicted fraud scores, take the top 3% with the highest fraud score and count how many of these records are true positive, and calculate the percentage of these accurately detected fraud records among all fraud records in each dataset as the FDR rate at 3% for that dataset.
2) Split training and testing data into 80% training and 20% testing data, and build a new model with the favorite model algorithm and hyperparameters using only training data and then test the model on the testing data, and create a model performance table for the training and testing datasets respectively.
3) Rebuild the final model using all training and testing data with the frozen hyperparameters and test on out-of-time validation dataset, and create a model performance table for the out-of-time validation dataset.

6.1. Logistic Regression

Logistic Regression uses maximum likelihood method to fit the logistic function, and outputs the probability of fraud. Mathematically, a Logistic Regression model gives a linear fit to log-odds, and estimates the coefficients to yield a number close to 1 for possible fraud records, and a number close to 0 for normal records. We use the result from Logistic Regression as the baseline for model performance.

6.2. Boosted Tree

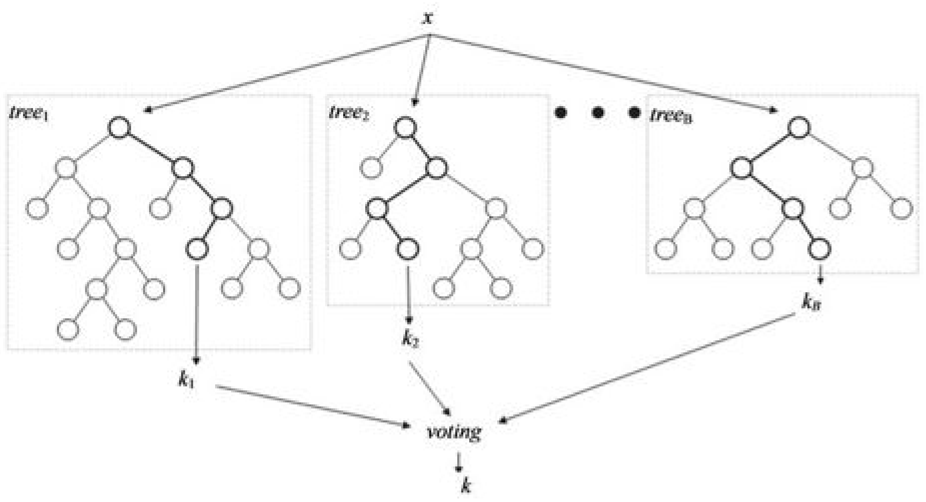
A Boosted Tree is a linear combination of weak learners or models to result in a strong learner. Each weak learner is trained to predict the residual error of the current sum. Specifically, the algorithm firstly builds a weak model to predict the output, and this first model is the closest approximation of the output. Based on the first model, the second weak model is built to predict the residual error of the first model. Then the third model is trained to predict the residual error of the previous two models summed up. The same logic applies to all the weak models. In the end, we get a strong model by summing up all the weak models.



For this project, we try out different numbers of trees or models. The “max_depth” parameter, which reflects the complexity of each weak tree, is set to be smaller than 5 as we prefer more weak and simple trees rather than a few complex trees. We also set the loss function for regularization to perform embedded feature selection to account for the complexity of the Boosted Tree algorithm.

6.3. Random Forest

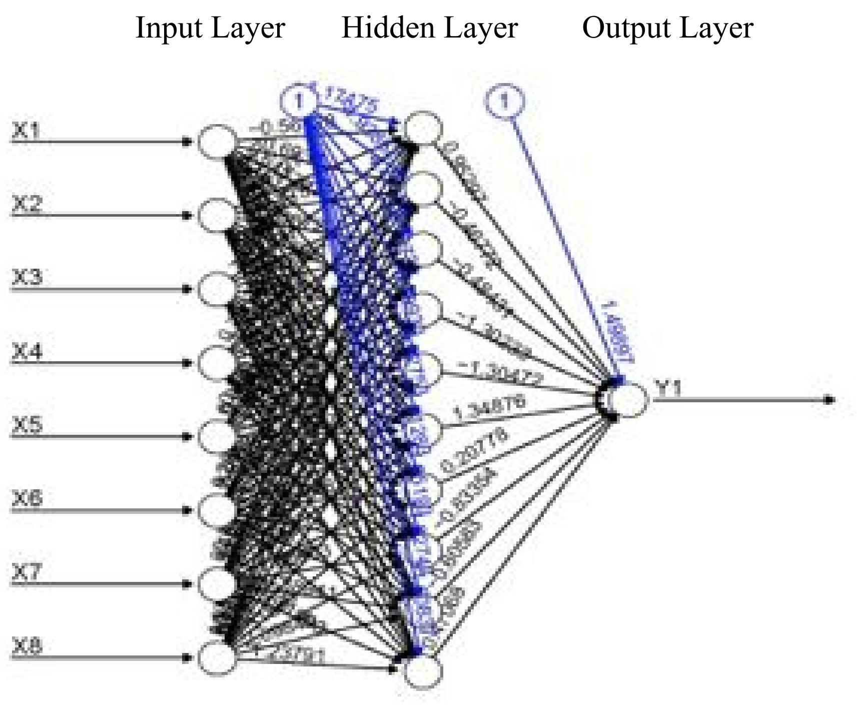
A Random Forest model is an ensemble model, which builds many complex independent decision trees. Each tree is a full model to predict the output and is a strong learner. The algorithm uses a different subset of variables when building each individual tree and/or during each split iteration. These randomness associated with the random forest algorithm makes all the trees relatively independent from each other. Voting and averaging are two of the common ways to combine all the strong trees, which cancel out the biases and therefore does not suffer from the overfitting problem. The Random Forest algorithm is usually robust, stable, and easily-generalizable.



For this project, we tried different versions of the model with different numbers of trees or models, with a range of “max_features”, and a range of “minimum samples leaf”, which is the minimum samples required to be at a leaf node for a split point.

6.4. Neural Net

A Neural Net model consists of an input layer, at least one hidden layer and an output layer. The input layer has many nodes, each node receiving information from one independent variable. The dependent variable y is the output layer. Typically, there is only one scalar output y. If there is a vector of Y, multiple classifications will be used for multiple outputs. Each node in a hidden layer receives weighted signals from all the nodes in the previous layer and does a transformation on this linear combination of signals. Usually, we use logistic regression as the transformation function.



The weights of signals are trained by back-propagating the error, record by record. At the beginning of training, the Neural Net model initializes a random combination of weights. When the first record is pushed through the neural net algorithm, the model transforms the linear combination of signals with the current weights using the specified transformation function, predicts the output and calculates the error. A typical error function is as follows:

$$E(y, \hat{y}) = |y - \hat{y}|^2$$

Then, according to the objective function, the Neural Net algorithm will take the derivative to calculate the gradient of the error with respect to the node weights, and then propagate them back to slightly adjust weights. With each data record pushed through the algorithm one by one, all the weights will be gradually adjusted. The entire data set will be passed through the algorithm many times as the weights settle into a local optimum. The speed of weight adjustment depends on the learning rate. The following formula shows how the weights are adjusted.

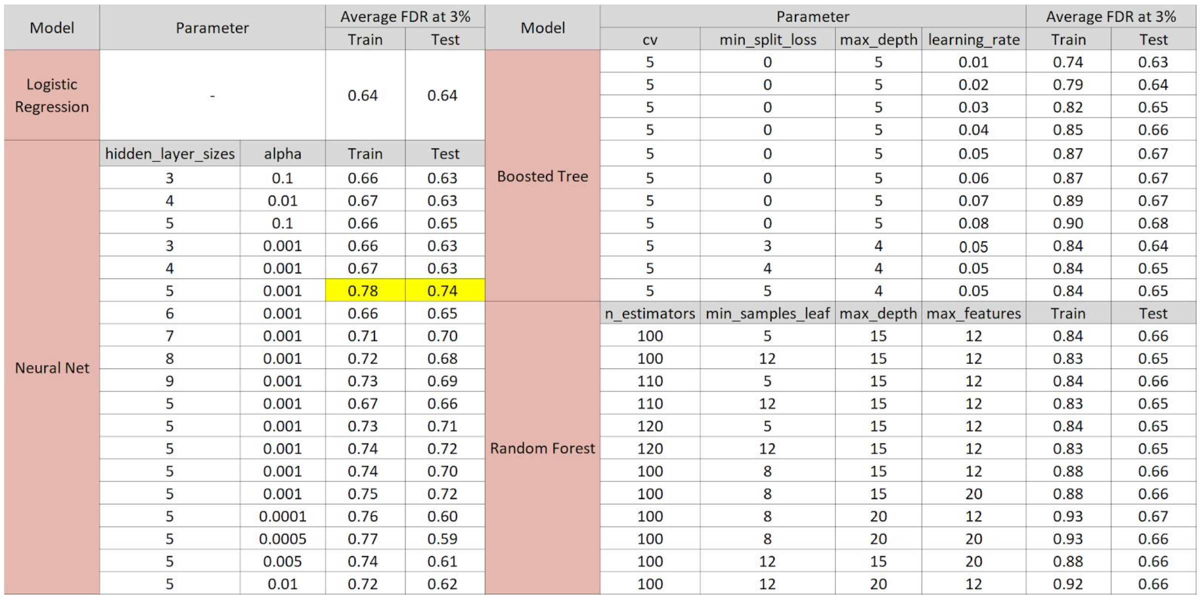
$$a_i^{n+1} = a_i^n + \underbrace{\eta}_{\text{Learning rate}} \Delta a_i, \quad \Delta a_i = -\frac{\partial E(\mathbf{a})}{\partial a_i}$$

For this project, we used only one hidden layer and tried out different numbers of nodes in this layer with different alpha levels and learning rates.

7. **Results**

7.1. Select favorite algorithm and hyperparameters

After building the above models and tuning the hyperparameters with cross validation, we compared the average performance of each model algorithm (shown below). We selected the Neural Net model with one layer of 5 nodes and alpha of 0.001 to build the final model, since it has the highest FDR at 3% (74%) in the testing set and a smaller difference of 4% between training and testing datasets.



Model	Parameter		Average FDR at 3%	
			Train	Test
Logistic Regression	-		0.64	0.64
Neural Net	hidden_layer_sizes	alpha	Train	Test
	3	0.1	0.66	0.63
	4	0.01	0.67	0.63
	5	0.1	0.66	0.65
	3	0.001	0.66	0.63
	4	0.001	0.67	0.63
	5	0.001	0.78	0.74
	6	0.001	0.66	0.65
	7	0.001	0.71	0.70
	8	0.001	0.72	0.68
	9	0.001	0.73	0.69
	5	0.001	0.67	0.66
	5	0.001	0.73	0.71
	5	0.001	0.74	0.72
	5	0.001	0.74	0.70
	5	0.001	0.75	0.72
	5	0.0001	0.76	0.60
	5	0.0005	0.77	0.59
	5	0.005	0.74	0.61
	5	0.01	0.72	0.62

Model	Parameter				Average FDR at 3%	
	cv	min_split_loss	max_depth	learning_rate	Train	Test
Boosted Tree	5	0	5	0.01	0.74	0.63
	5	0	5	0.02	0.79	0.64
	5	0	5	0.03	0.82	0.65
	5	0	5	0.04	0.85	0.66
	5	0	5	0.05	0.87	0.67
	5	0	5	0.06	0.87	0.67
	5	0	5	0.07	0.89	0.67
	5	0	5	0.08	0.90	0.68
	5	3	4	0.05	0.84	0.64
	5	4	4	0.05	0.84	0.65
	5	5	4	0.05	0.84	0.65
Random Forest	n_estimators	min_samples_leaf	max_depth	max_features	Train	Test
	100	5	15	12	0.84	0.66
	100	12	15	12	0.83	0.65
	110	5	15	12	0.84	0.66
	110	12	15	12	0.83	0.65
	120	5	15	12	0.84	0.65
	120	12	15	12	0.83	0.65
	100	8	15	12	0.88	0.66
	100	8	15	20	0.88	0.66
	100	8	20	12	0.93	0.67
	100	8	20	20	0.93	0.66
	100	12	15	20	0.88	0.66
	100	12	20	12	0.92	0.66

7.2. Performance of the final model

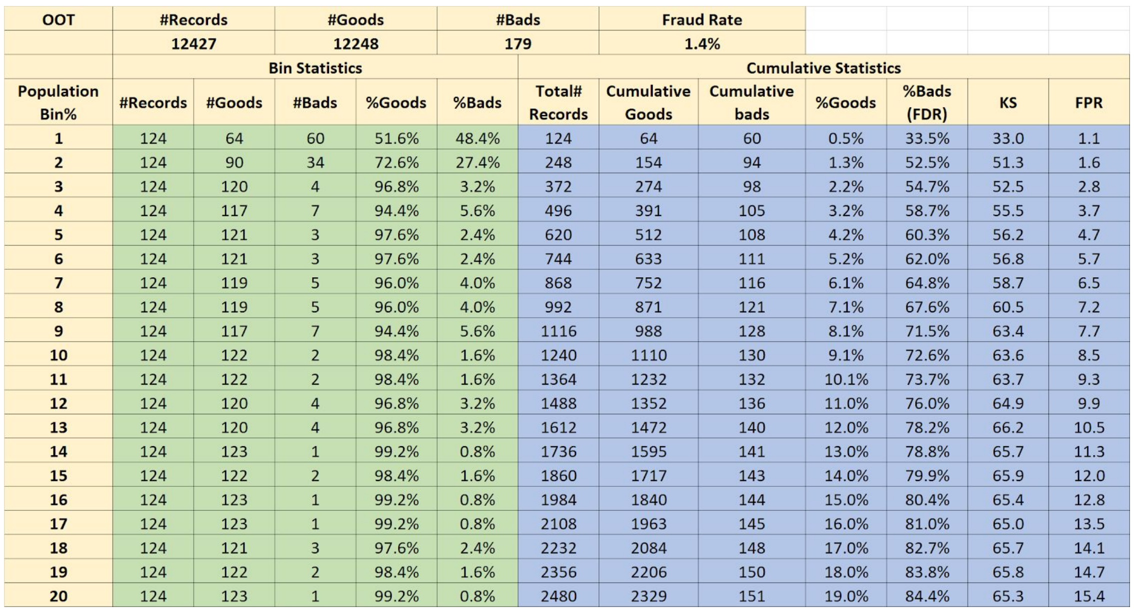
To see the performance of the final model, we first splitted the training and testing datasets into 80% training and 20% testing data, and ran the selected model with training and testing data separately. Then we sorted the data by predicted fraud probability and built model performance tables of the training and the testing dataset. Then we rebuilt the final model using the whole modeling data, and ran the model on the out-of-time data, and built the performance table for validation. All the performance tables are shown below. From the tables, we can see that 84.4% of fraud records in the validation data are caught in the top 20% bins. This proves the efficiency of our final Neural Net model.

Training	#Records		#Goods		#Bads		Fraud Rate					
	64505		63811		694		1.1%					
	Bin Statistics					Cumulative Statistics						
Population Bin%	#Records	#Goods	#Bads	%Goods	%Bads	Total# Records	Cumulative Goods	Cumulative bads	%Goods	%Bads (FDR)	KS	FPR
1	645	218	427	33.8%	66.2%	645	218	427	0.3%	61.5%	61.2	0.5
2	645	553	92	85.7%	14.3%	1290	771	519	1.2%	74.8%	73.6	1.5
3	645	619	26	96.0%	4.0%	1935	1390	545	2.2%	78.5%	76.4	2.6
4	645	630	15	97.7%	2.3%	2580	2020	560	3.2%	80.7%	77.5	3.6
5	645	633	12	98.1%	1.9%	3225	2653	572	4.2%	82.4%	78.3	4.6
6	645	638	7	98.9%	1.1%	3870	3291	579	5.2%	83.4%	78.3	5.7
7	645	637	8	98.8%	1.2%	4515	3928	587	6.2%	84.6%	78.4	6.7
8	645	637	8	98.8%	1.2%	5160	4565	595	7.2%	85.7%	78.6	7.7
9	645	637	8	98.8%	1.2%	5805	5202	603	8.2%	86.9%	78.7	8.6
10	645	637	8	98.8%	1.2%	6450	5839	611	9.2%	88.0%	78.9	9.6
11	645	642	3	99.5%	0.5%	7095	6481	614	10.2%	88.5%	78.3	10.6
12	645	644	1	99.8%	0.2%	7740	7125	615	11.2%	88.6%	77.5	11.6
13	645	643	2	99.7%	0.3%	8385	7768	617	12.2%	88.9%	76.7	12.6
14	645	642	3	99.5%	0.5%	9030	8410	620	13.2%	89.3%	76.2	13.6
15	645	641	4	99.4%	0.6%	9675	9051	624	14.2%	89.9%	75.7	14.5
16	645	645	0	100.0%	0.0%	10320	9696	624	15.2%	89.9%	74.7	15.5
17	645	641	4	99.4%	0.6%	10965	10337	628	16.2%	90.5%	74.3	16.5
18	645	641	4	99.4%	0.6%	11610	10978	632	17.2%	91.1%	73.9	17.4
19	645	642	3	99.5%	0.5%	12255	11620	635	18.2%	91.5%	73.3	18.3
20	645	644	1	99.8%	0.2%	12900	12264	636	19.2%	91.6%	72.4	19.3

Model Performance Table on Training Data

Testing	#Records		#Goods		#Bads		Fraud Rate					
	16127		15953		174		1.1%					
	Bin Statistics					Cumulative Statistics						
Population Bin%	#Records	#Goods	#Bads	%Goods	%Bads	Total# Records	Cumulative Goods	Cumulative bads	%Goods	%Bads (FDR)	KS	FPR
1	161	67	94	41.6%	58.4%	161	67	94	0.4%	54.0%	53.6	0.7
2	161	135	26	83.9%	16.1%	322	202	120	1.3%	69.0%	67.7	1.7
3	161	151	10	93.8%	6.2%	483	353	130	2.2%	74.7%	72.5	2.7
4	161	157	4	97.5%	2.5%	644	510	134	3.2%	77.0%	73.8	3.8
5	161	158	3	98.1%	1.9%	805	668	137	4.2%	78.7%	74.5	4.9
6	161	158	3	98.1%	1.9%	966	826	140	5.2%	80.5%	75.3	5.9
7	161	161	0	100.0%	0.0%	1127	987	140	6.2%	80.5%	74.3	7.1
8	161	161	0	100.0%	0.0%	1288	1148	140	7.2%	80.5%	73.3	8.2
9	161	160	1	99.4%	0.6%	1449	1308	141	8.2%	81.0%	72.8	9.3
10	161	160	1	99.4%	0.6%	1610	1468	142	9.2%	81.6%	72.4	10.3
11	161	160	1	99.4%	0.6%	1771	1628	143	10.2%	82.2%	72.0	11.4
12	161	161	0	100.0%	0.0%	1932	1789	143	11.2%	82.2%	71.0	12.5
13	161	160	1	99.4%	0.6%	2093	1949	144	12.2%	82.8%	70.5	13.5
14	161	159	2	98.8%	1.2%	2254	2108	146	13.2%	83.9%	70.7	14.4
15	161	161	0	100.0%	0.0%	2415	2269	146	14.2%	83.9%	69.7	15.5
16	161	161	0	100.0%	0.0%	2576	2430	146	15.2%	83.9%	68.7	16.6
17	161	161	0	100.0%	0.0%	2737	2591	146	16.2%	83.9%	67.7	17.7
18	161	159	2	98.8%	1.2%	2898	2750	148	17.2%	85.1%	67.8	18.6
19	161	160	1	99.4%	0.6%	3059	2910	149	18.2%	85.6%	67.4	19.5
20	161	159	2	98.8%	1.2%	3220	3069	151	19.2%	86.8%	67.5	20.3

Model Performance Table on Testing Data

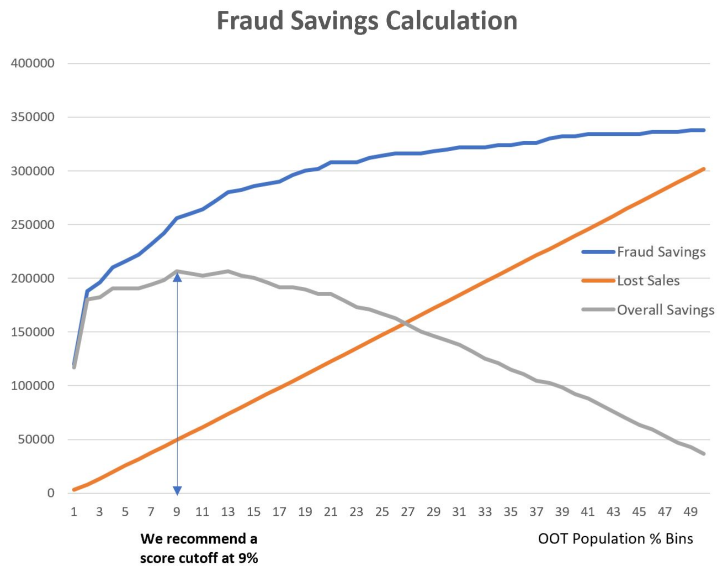


OOT	#Records		#Goods		#Bads		Fraud Rate					
	12427		12248		179		1.4%					
	Bin Statistics					Cumulative Statistics						
Population Bin%	#Records	#Goods	#Bads	%Goods	%Bads	Total# Records	Cumulative Goods	Cumulative bads	%Goods	%Bads (FDR)	KS	FPR
1	124	64	60	51.6%	48.4%	124	64	60	0.5%	33.5%	33.0	1.1
2	124	90	34	72.6%	27.4%	248	154	94	1.3%	52.5%	51.3	1.6
3	124	120	4	96.8%	3.2%	372	274	98	2.2%	54.7%	52.5	2.8
4	124	117	7	94.4%	5.6%	496	391	105	3.2%	58.7%	55.5	3.7
5	124	121	3	97.6%	2.4%	620	512	108	4.2%	60.3%	56.2	4.7
6	124	121	3	97.6%	2.4%	744	633	111	5.2%	62.0%	56.8	5.7
7	124	119	5	96.0%	4.0%	868	752	116	6.1%	64.8%	58.7	6.5
8	124	119	5	96.0%	4.0%	992	871	121	7.1%	67.6%	60.5	7.2
9	124	117	7	94.4%	5.6%	1116	988	128	8.1%	71.5%	63.4	7.7
10	124	122	2	98.4%	1.6%	1240	1110	130	9.1%	72.6%	63.6	8.5
11	124	122	2	98.4%	1.6%	1364	1232	132	10.1%	73.7%	63.7	9.3
12	124	120	4	96.8%	3.2%	1488	1352	136	11.0%	76.0%	64.9	9.9
13	124	120	4	96.8%	3.2%	1612	1472	140	12.0%	78.2%	66.2	10.5
14	124	123	1	99.2%	0.8%	1736	1595	141	13.0%	78.8%	65.7	11.3
15	124	122	2	98.4%	1.6%	1860	1717	143	14.0%	79.9%	65.9	12.0
16	124	123	1	99.2%	0.8%	1984	1840	144	15.0%	80.4%	65.4	12.8
17	124	123	1	99.2%	0.8%	2108	1963	145	16.0%	81.0%	65.0	13.5
18	124	121	3	97.6%	2.4%	2232	2084	148	17.0%	82.7%	65.7	14.1
19	124	122	2	98.4%	1.6%	2356	2206	150	18.0%	83.8%	65.8	14.7
20	124	123	1	99.2%	0.8%	2480	2329	151	19.0%	84.4%	65.3	15.4

Model Performance Table on Validation (OOT) Data

7.3. Fraud Savings suggested score cutoff

Based on the model performance on the out-of-time dataset, we suggest using the score cutoff point at 9% through the following fraud savings calculation.

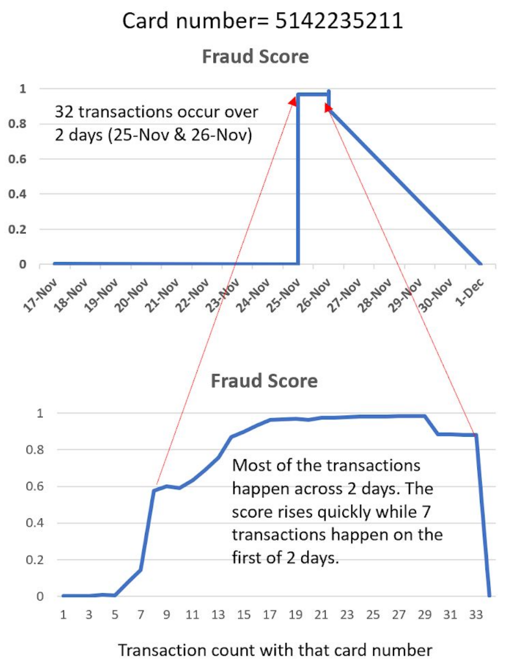


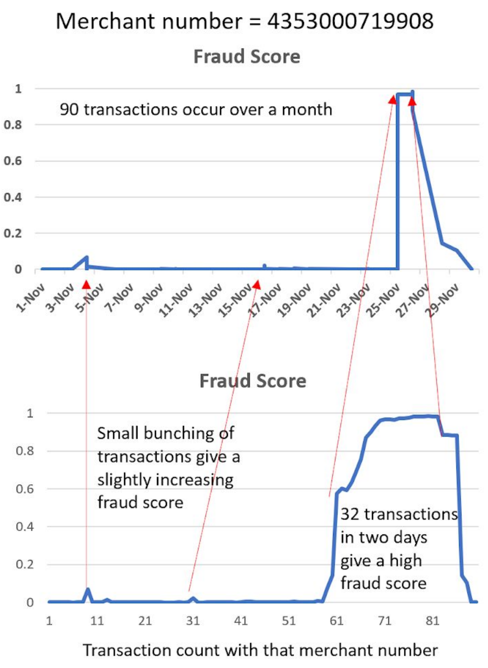
We assume a $2,000 savings in loss for each fraud that is caught by the model, and a $50 loss in sales for every good record that is wrongly caught as fraud. The blue curve above shows the accumulated fraud savings and the red line shows the accumulated loss from catching false positives. The overall savings is the difference between fraud savings and lost sales, plotted as the grey curve. We can see that the overall savings keep going up until we reach the score at 9% where the savings in catching real fraud is no more than the loss incurred by catching false positives. Therefore, we recommend using a score cutoff at 9% to get the maximum financial benefit from using the model.

7.4. Demonstration of fraud detection: fraud scores increase as activities increase

Though we built the model to catch fraud in real time, it is still hard for the model to catch the very first few instances of fraud, because fraud scores only increase when the model sees unusual activities, and the model needs to see enough records with the same entities to start labeling transactions as fraud. When the activities of an entity cross a threshold, the entity will be detected to be suspicious by the model.

Two examples of this pattern are shown below. For card number 5142235211, the graph on the left shows that the score rises quickly when many transactions happen across two days. For merchant number 4353000719908, there is a similar pattern: many transactions in a short time period yield high fraud scores. Also, small bunching of transactions will also give a slightly increased fraud score.





8. Conclusion

In this project, we built a supervised Neural Net machine learning model to detect credit card transaction fraud in real time in order to save financial loss for cardowners, merchants, and financial institutions.

We first did a data quality assessment to explore and understand the dataset. We identified a single outlier and missing values in several fields in this process. Then we cleaned the data by removing the outlier and filled in missing values carefully to restore linking information and avoid creating false repetitions that might be confused as fraud signals by the model.

Based on our analysis of credit card transaction fraud patterns, we built 302 candidate variables with the 10 original fields. After that, we Z-scaled all the candidate variables and performed feature selection. We used "Kolmogorov-Smirnov" (KS) score and Fraud Detection Rate (FDR) at 3% as the measure of goodness to filter out 80 candidate variables, and then used a wrapper to select the final 25 variables for model building.

Next we used the selected variables to build the baseline linear model (Logistic Regression) and three nonlinear models (Neural Net, Boosting Tree and Random Forest). After trying out different combinations of hyperparameters and comparing the results of each model, we selected the Neural Net algorithm with one layer, six nodes, and an alpha of 0.001 to build our final model. Finally, we ran the model on a training set and a testing set, and used all data other than the out-of-time validation data to retrain the model for best performance, and ran the final model on the validation data, which resulted in a 54.7% FDR at 3%.

If we were granted more time, we would like to continue our work on this project in the following aspects. First, we would like to consult with credit card transaction fraud experts to better understand the patterns of fraud behaviour, so that we can create even more powerful candidate variables to improve model performance. Second, we would further fine tune the hyperparameters of the four candidate models and try more nonlinear algorithms such as SVM for a wider selection for the final model.

Appendix

Data Quality Report on Credit Card Transaction Data

Card Transaction Data documents the credit card transactions made on behalf of a government organization in Tennessee in the year of 2010. There are 10 fields, including a fraud label field, and 96,753 records in total. In total, there are 1059 fraudulent transactions in this dataset, and the overall fraud rate is 1.1%.

1. Field Summary

Table 1 Numeric Fields

	Number of populated records	% of populated records	# of unique values	# of records with value zero	Mean	STD	Min	Max
Amount	96753	100	34909	0	427.89	10006.14	0.01	3102045.53

Table 2 Categorical Fields

	# of records that have a value	% populated	# of unique values	# of records with value zero	Most Common Field Value
Recordnum	96753	1	96753	0	NA
Cardnum	96753	1.00	1645	0	5142148452
Date	96753	1.00	365	0	2/28/10
Merchnum	93378	0.97	13092	0	930090121224
Merch description	96753	1.00	13126	0	GSA-FSS-ADV
Merch state	95558	0.99	228	0	TN
Merch zip	92097	0.95	4568	0	38118
Transtype	96753	1.00	4	0	P
Fraud	96753	1.00	2	95694	0

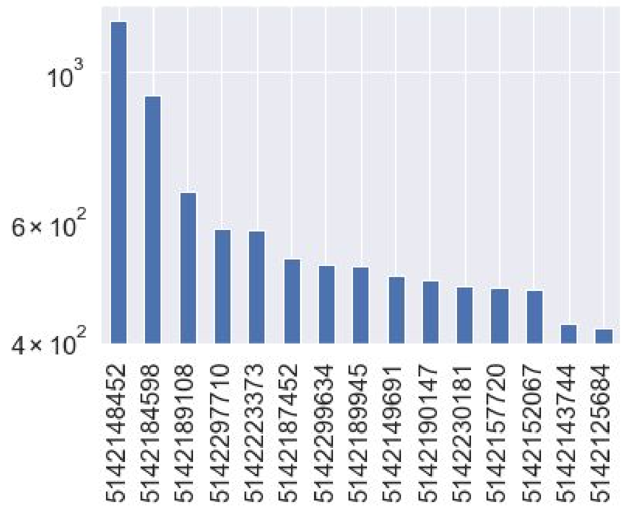
2. Field Descriptions

2.1. Recordnum

A categorical variable with 96753 unique values. It is the unique ordinal reference number for each transaction record.

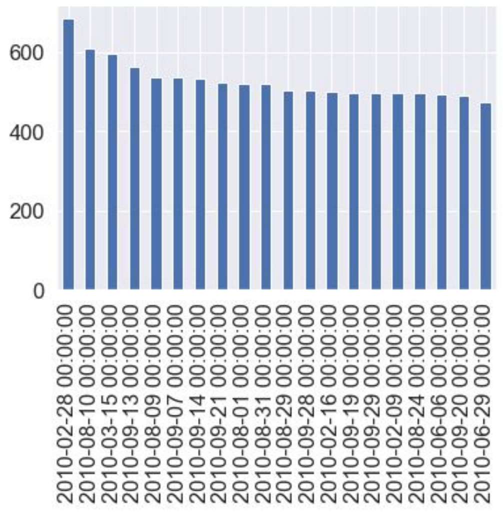
2.2. Cardnum

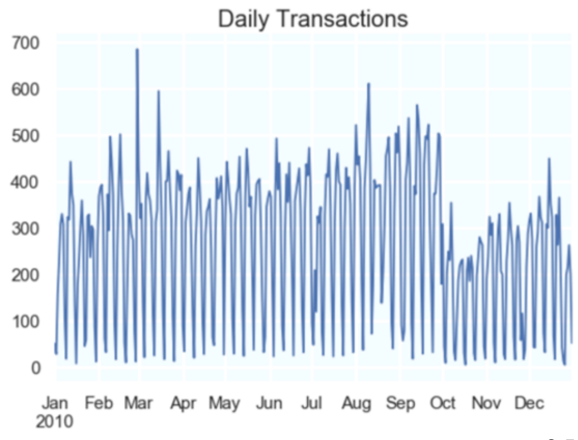
A categorical variable with 1645 unique values. It is the number of the credit card used in each transaction.

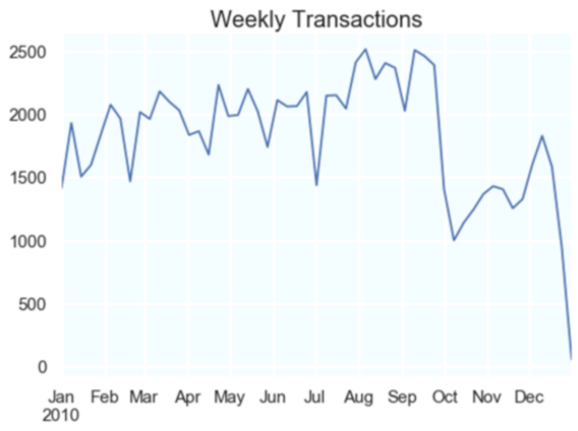


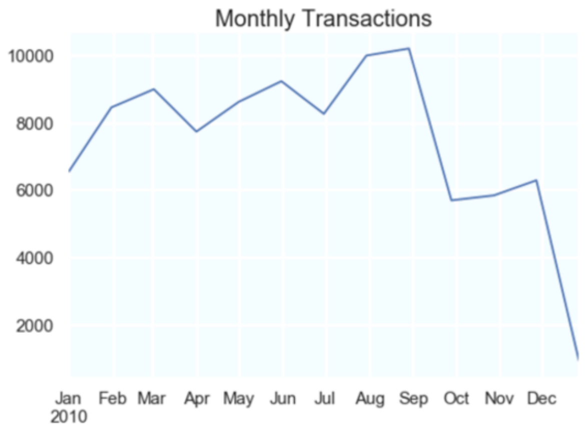
2.3. Date

This field is the date of the transactions, ranging from 2010-1-1 to 2010-12-31. The number of occurrences of each date in the dataset is equal to the number of transactions made on that particular date. Therefore, we can plot the daily, weekly and monthly transaction to see the distribution of dates and trends in transactions. We can see that the credit card transaction has obvious weekly seasonality – activities on the weekend are usually close to zero. We can also see that September has the highest transaction volume, possibly due to the fact that the government tries to spend the remaining budget of the year before the fiscal year ends.



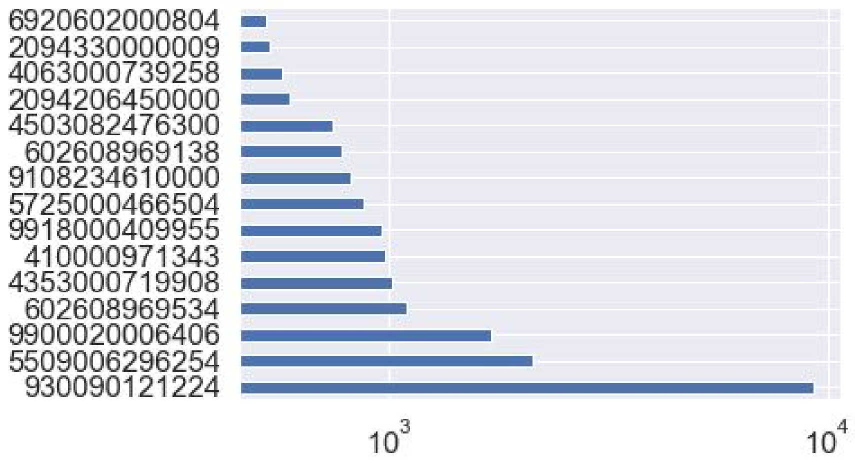






2.4. Merchantnum

A categorical variable with 13092 unique values. It is the unique identifier for each merchant.



2.5. Merch description

A categorical variable with 13126 unique values. It contains some detailed information about the merchant in each credit card transaction, such as the name of the merchant, etc. Top 10 variables show in the chart.

GSA-FSS-ADV	1688
SIGMA-ALDRICH	1635
STAPLES #941	1174
FISHER SCI ATL	1093

MWI*MICRO WAREHOUSE	958
CDW*GOVERNMENT INC	872
DELL MARKETING L.P.	816
FISHER SCI CHI	783
AMAZON.COM *SUPERSTOR	750
OFFICE DEPOT #1082	748

2.6. Merchant state

A categorical variable with 228 unique values. It consists of not only the two letter abbreviation of American states, but also some digit combinations that indicate some specific geographical locations all around the globe. Top 10 variables show in the chart.

TN	12035
VA	7872
CA	6817
IL	6508
MD	5398
GA	5025
PA	4899
NJ	3912
TX	3790
NC	3322

2.7. Merchant zip

A categorical variable with 4568 unique values. The top 10 zip codes with the most occurrences are shown below. A large number of records do not have the zip code field populated, and some values in this field are less than 5 digits.

38118	11868
63103	1650
8701	1267
22202	1250
60061	1221
98101	1197
17201	1180
30091	1092
60143	942
60069	826
78682	817

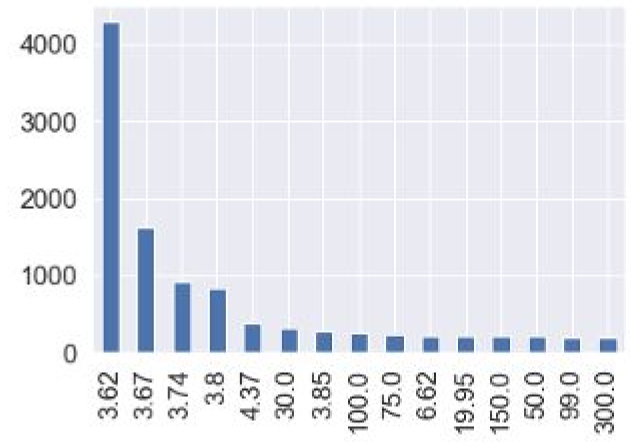
2.8. Transtype

A categorical variable with 4 unique values: P, A, D and Y, among which P takes up the majority.

P	96398
A	181
D	173
Y	1

2.9. Amount

The only numeric variable in this dataset and it indicates the amount of each transaction.



2.10. Fraud

A categorical variable with two unique values: 0 and 1, with 0 representing non-fraudulent records and 1 representing fraudulent record. It is a label that is manually added to the dataset for educational purposes.

Fraud	Count	Percentage
1	1059	1.1%
0	95338	98.9%