FRAUD ANALYTICS PROJECT 3

Card Transaction data

ABSTRACT

Design a supervised fraud model on the card transaction data

Team 9

Anni Cai,
Suraj Patel,
Yuyao Shen,
Nanchun Shi,
Bingru Xue

Contents

Executive Summary

The purpose of this project is to investigate the “Card Transactions” file and build a supervised model to effectively identify fraudulent transactions. Potentially there are two types of fraudulent transactions, one associated with lost and stolen cards, and the other associated with merchants. For this project, we identify and address both of them.

The raw dataset “Card Transactions” contains transactions made by a Tennessee government agency from January 1st, 2010 to December 31st, 2010. The dataset contains relevant transaction information including transaction date, transaction type, merchant number, merchant description, merchant locations and card number. Dataset also has a binary label indicating fraudulent transactions.

The report begins with a detailed description of the dataset. (The full data quality report is included as an appendix for reference.) The dataset has 96,753 records and 10 fields, among which nine are categorical fields and one is numerical field. Among these categorical fields, three fields are not fully populated, which are merchant number, merchant state and merchant zip. For the numerical field, amount, we detected a potential outlier as we saw an exceptionally high standard deviation.

After exploring the dataset, the report explains our data cleaning process that deals with missing values and the outlier mentioned above, and some other inconsistencies in the data. We first removed the outlier and selected only transactions with type of interest (“P”). Then we addressed the missing values for the three categorical fields on a case-by-case basis. The basic logic for filling in these missing values is to group the records from a low granularity level to a high granularity level and replace the missing values with the most frequent number in the associated group. In addition, we also did some work to restore the authenticity of the dataset. For example, we utilized the “uszipcode” library in Python to look for corresponding merchant zip code and state, we used Merchant number to refer merchant locations and we carefully identified merchants located outside the U.S. For the remaining records with missing fields, we treated them as frivolous records and replaced them with negative record numbers. The negative record numbers are unique and unlinked so that they will not trigger false alarms in the later model building process.

After data cleaning, we built 430 candidate variables using the initial 10 fields. First, we created five entity groups in addition to Cardnum, Merchnum to capture the dynamic nature of data across different fraud signals (e.g. Cardnum - Merchnum). Then for each of the 7 entity groups, we created amount variables, day-since variables, frequency and relative frequency variables. We also created a risk table variable, which is the probability of a fraud transaction on a particular day of the week, and two Benford’s Law variables, which measure the unusualness of a record based on its deviation from Benford's Law. (Benford’s Law is the non-intuitive fact that the first digit of many measurements is not uniformly distributed, and the first digit “1” appears about 30% of the time.)

Since there are a limited number of records but a large number of dependent variables, we performed feature selection for dimensionality reduction. The feature selection process mainly consists of two steps: filter and wrapper. To filter variables, we calculate their univariate Kolmogorov–Smirnov (KS) and Fraud detection rate (FDR), where KS looks at distribution of good records versus bad records, and FDR is the percentage of frauds caught by looking at a certain

percentage of overall records. We ranked variables based on their respective KS and FDR scores along with the average of the two and selected 80 variables with top performance. We further applied sequential forward selection as our wrapper method and chose 22 final variables for model building.

To build fraud detection models, we divided the whole dataset by subsetting the last 2 months' as validation/Out of Time data (oot) and the rest as modeling (training and test) data. Then, we developed a Logistic regression model using training data as our baseline model and calculated the FDR at a 3% penetration level on training, test and oot validation data. We further explored some non-linear models with parameter tuning, including K-Nearest Neighbors, Neural Networks, Boosted Trees and Random Forest, and measured the accuracy by calculating the FDR at 3% cutoff population. Random Forest resulted in the highest fraud detection rate of 59.89% at a 3% penetration rate on the out-of-time validation dataset.

We conclude our report with some specific case studies to better illustrate how our fraud model works and benefits our customers. We deep dived into one fraudulent card number and one fraudulent merchant number as examples to show the interaction between fraud scores and fraudulent activities. We also showed the business implementation of our model: based on given business assumptions, we recommend a cutoff at 7% penetration level, using the results of our model which will approximately generate an annual profit of 1.3 million dollars for our customers.

Description of Data

This is a credit card transaction dataset from a Tennessee government agency. The dataset contains information about the transaction, the merchant involved in the transaction and a fraud label for each application. The time period of this dataset is from January 1, 2010 to December 31, 2010. It has 10 fields and 96753 records. The 10 fields are -

1. Recnum - a unique index number for each record
2. Cardnum - card number of the card used for transaction
3. Date - date of transaction
4. Merchnum - merchant number of the merchant involved in the transaction
5. Merch description - brief description of the merchant
6. Merch state - abbreviation of the state where the merchant is located
7. Merch zip - 5-digit zip code pertaining to the location of the merchant
8. Transtype - indicates the transaction type (P, A, D, Y)
9. Amount - the dollar amount of the transaction
10. fraud_label - Label indicating whether the application is fraud. (0: Not Fraud, 1: Fraud)

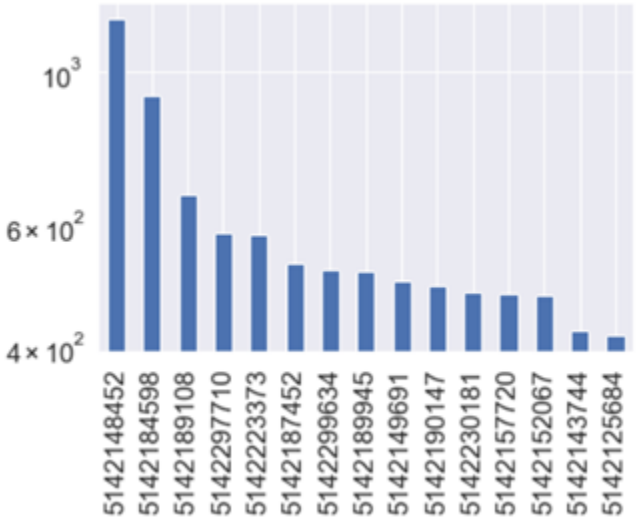
Summary table

Field name	Field type	# of non-NA values	% populated	# unique values	Most Common Field
Recnum	Categorical	96,753	100	96,753	N/A
Cardnum	Categorical	96,753	100	1645	5142148452
Date	Categorical	96,753	100	365	2010-02-28
Merchnum	Categorical	93,378	96.51	13091	930090121224
Merch description	Categorical	96,753	100	13126	GSA-FSS-ADV
Merch state	Categorical	95,558	98.76	227	TN
Merch zip	Categorical	92,097	95.19	4568	38118
Transtype	Categorical	96,753	100	4	P
Fraud	Categorical	96,753	100	2	0

Field name	Field type	# of non-NA values	% populated	# unique values	Most Common Field
Amount	Numerical	96,753	100	28,244	3.62

Fields Distribution

Cardnum:



Date:

Date	Count (top 10)
2010-02-28	684
2010-08-10	610
2010-3-15	594
2010-09-13	564
2010-08-09	536
2010-09-07	536
2010-09-14	533
2010-09-21	522
2010-08-01	521
2010-08-31	518

Merchnum:

Merchnum	Count (top 10)
930090121224	9310
5509006296254	2131
9900020006406	1714
602608969534	1092
4353000719908	1020
410000971343	982
9918000409955	956
5725000466504	872
9108234610000	817
602608969138	783

Merch description:

Merch description	Count (top 10)
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

Merch state:

Merch state	Count (top 10)
TN	12035
VA	7872
CA	6817
IL	6508
MD	5398
GA	5025
PA	4899
NJ	3912
TX	3790
NC	3322

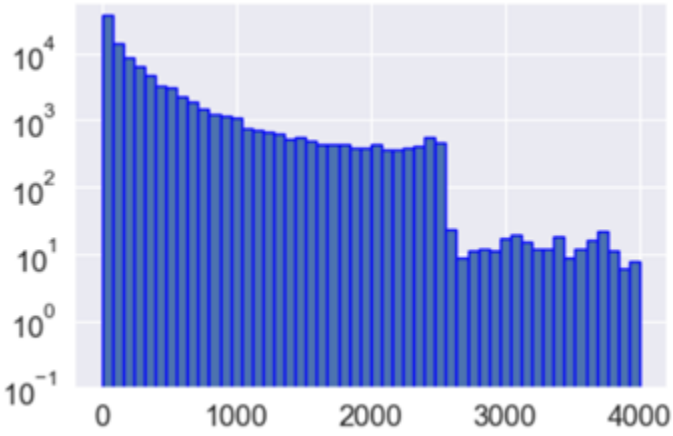
Merch zip:

Zip5	Count (top10)
38118	11868
63103	1650
8701	1267
22202	1250
60061	1221
98101	1197
17201	1180
30091	1092
60143	942
60069	826

Transtype:

Transtype	Count
P	96398
A	181
D	173
Y	1

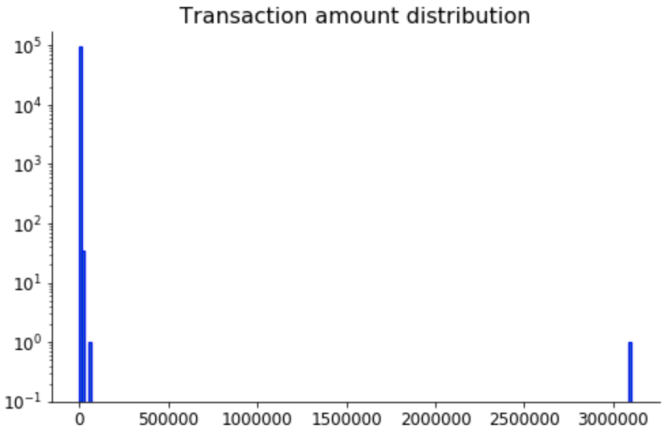
Amount:



Data Cleaning

Exclusions

We started data cleaning by excluding records based on our business knowledge and statistical analysis. We excluded a transaction amount which had a value greater than 3 million dollars. This was classified as an outlier as you can see from the distribution below. Due to this record the standard deviation for amount variable was as high as 10,000 dollars, while the mean was just 429 dollars. Thus, we excluded this record from the dataset.



Also, according to our client's needs, we will only focus on the transaction type "P". Therefore, we removed 355 records associated with other transactions types.

	Transtype
P	96398
A	181
D	173
Y	1

After making these exclusions, there were 96397 records left.

Treating missing values

Fields	# of null	# of 0s	Total
Recnum	0	0	0
Cardnum	0	0	0
Date	0	0	0
Merchnum	3198	53	3251
Merch description	0	0	0
Merch state	1020	0	1020
Merch zip	4300	0	4300
Transtype	0	0	0
Amount	0	0	0

We treat zero values as missing values, and found that for Merchnum, Merch state, and Merch zip, there are positive numbers of missing values.

1. Merchnum

We start with Merchnum's with low granularity levels of mechant description and merchant state. we group these merchnum by merchant description and merchant zip codes and replace the missing values with the most frequent merchant number in each group. If there is no information for certain records, we move up to a higher granularity levels of merchant description and merchant state and replace them with the most frequent. If there is still no information, we group by only merchant description. After this process, we still have 2,094 missing values for Merchnum. These records are treated as frivolous records. This means, we believe some values were missing for these records during the collection of data and don't want to "penalize" these records by making them more possible of being frauds, or trigger false alarms. We then replace the missing merchant numbers with the corresponding record number but take the negative value of that. This is because when we build the candidate variables later, we will build linkages between records using different entities. A frequent appearance of a certain entity will indicate a possibility of being frauds. So, we don't want to use the same value to impute them as this will increase the frequency dramatically. Therefore, record number is a good choice since it is unique for every record. Taking the negative is to further avoid the cases where some record numbers could be the same as the merchant number of other records.

2. Merch state

Similarly, we first group by different granularity groups, merchant description & merchant number, merchant description & merchant zip codes, and then merchant description only. After this process we are left with 363 missing values. Then, we find that the records that miss merchant state, but not merchant zip codes have some strange numbers in merchant zip codes:

1400, 65132, 86899, 23080, 60528, 6, 8, 9, 50823, 2, 48700, 1, 3, 801

We manually looked them up online and found that some of them are postal codes in other countries. Also, if we input these values to the python library we are going to use, uszipcode, the

library will return None. Therefore, we decided to replace the missing values in the merchant state field for these records that are associated with these numbers with their own zip code values. Doing this is to keep the differentiation of these records and also reserve the potential that these values were manually inputted by fraudsters. Then for the other records, we use the library to retrieve the state information by inputting corresponding zip codes. After this, we have 256 missing values left.

For the remaining missing values, we group the records by the card number and replace with the most frequent values across the card groups. The motivation of this is that for a certain card, most of the merchants where it was used to make purchases are likely to be in the same place. Then, for missing values left after this, we treat the associated records as frivolous and replace them with the negative record number, using the same reasoning as before.

3. Merch zip

We follow similar steps for imputing merchant zip code as we do in the previous imputation. First, we group by merchant description and merchant number. If there is no information, we move up to merchant description and merchant state, and then merchant description only. We are left with 2043 missing values after this process. We then conducted another case study to We find that the records that miss merchant zip codes values but not merchant state values, some of these merchant states are actually canadian province codes:

BC, QC, ON, MB, PQ, NS, AB

There will be an issue if we simply use the library to retrieve zip code values. This is because the library implements fuzzy matching functionality. For example, if we input BC into the library searcher, and retrieve state value, we will get AL, which is Alabama in the United States. This is not what we want. Therefore, we decide to replace the missing merchant zip code values of the records associated with these Canadian provinces with their own province codes. And for those records not associated with Canadian provinces, we use the library to retrieve the zip code information. After this, there are 454 missing values left.

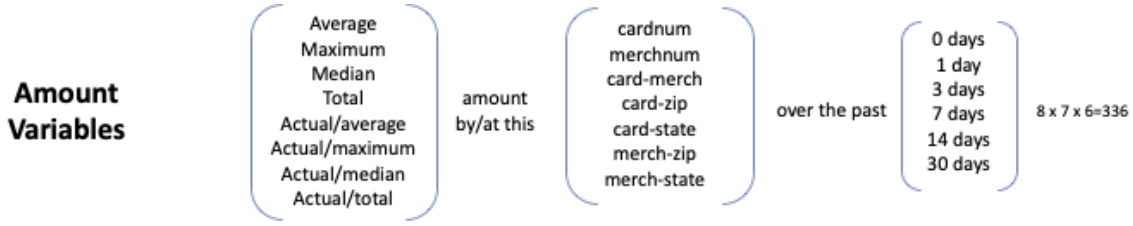
Then, we group by card number and pick the most frequent value in each group. For the remaining records, we replace them with the negative record number.

Variable Creation

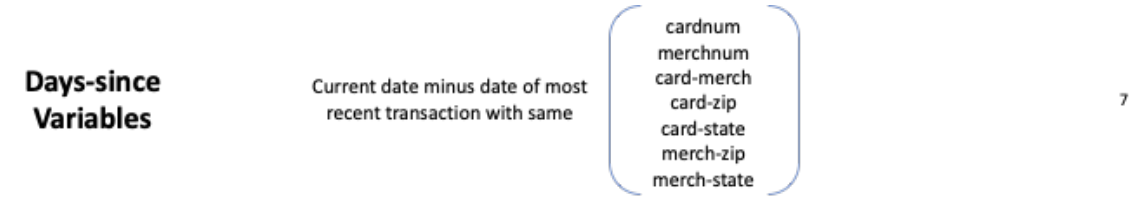
Data has 10 fields and 96753 records. Most of these variables are dynamic and change values for different entities like Cardnum and Merchnum across dates. To capture the dynamic nature of the data across different signals, we started by creating five entity groups. We created 5 new entities in addition to Cardnum, Merchnum. They are 'Cardnum-Merch zip', 'Cardnum-Merch state', 'Cardnum-Merchnum', 'Merchnum-Merch zip' and 'Merchnum-Merch state'. We have created these entities to capture frauds due to lost/stolen cards and frauds committed by the merchants themselves.

For each entity group we created the following variables:

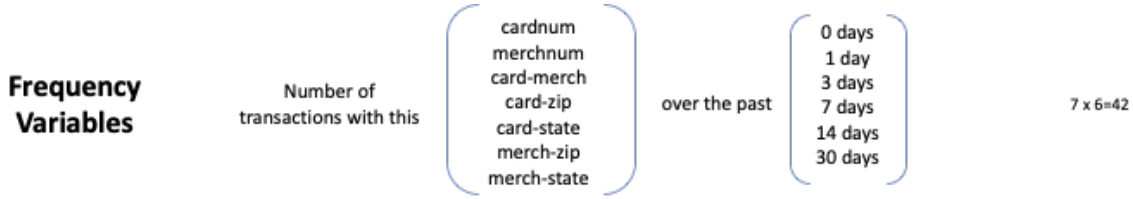
1. Amount: Amount variables represent summary (average, maximum, median, total, actual/average,actual/maximum,actual/median,actual/total) of amount spent by/at different entity groups over the past time window. We set different time windows, including 0,1,3,7,14, 30 days to look back in the past. We believe that a sharp increase in these variables for different entities could indicate a potential fraud. For example, if actual/average for card-merch is 10, it indicates that the transaction was 10 times greater than usual for this entity combination. Using this logic, we have created a total of 336 amount variables.



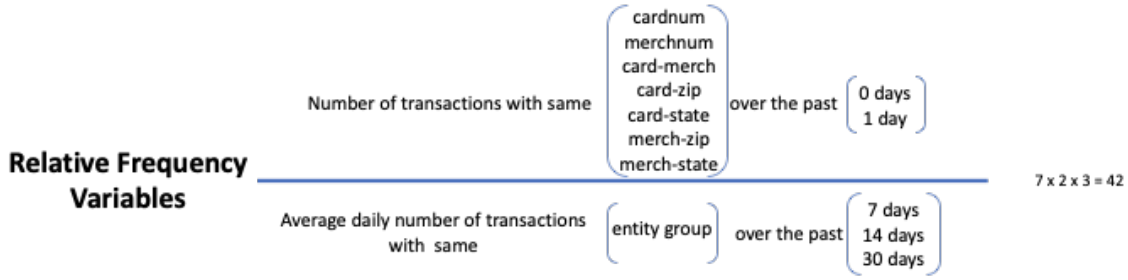
2. Day-since: Day since variables represent the count of days since we last saw a particular value of the entity. We believe a small-time gap between appearances of the same entity could indicate a high possibility of fraud. For example, if cardnum_days_since is zero (the # of days between the latest transaction and the previous to the current transaction), it means that this card number was used for some other transaction today itself. We have created a total of 7 day-since variables.



3.Frequency: Frequency variables represent how many times we have seen a particular entity in a time window looking in the past. We set different time windows, including 0,1, 3, 7, 14, 30 days in the past. We believe high frequency for entities could indicate a high possibility of being fraudulent. For example, if card_merch_frequency_3d is equal to 20, it indicates that this is the twentieth time this card has been used at this merch over the last 3 days. We have created a total of 42 frequency variables.



4. Relative Frequency: Relative frequency variables represent how many times we have seen a particular entity in a recent time window over the average number of times we have seen that particular entity in a bigger time window. For example, if cardnum_rel_frequency_1d_7d is equal to 10, it indicates that the number of transactions of this card number in the past 1 day is 10 times higher than its daily average over the past 7 days. We have created a total of 42 relative frequency variables.



5. Risk table (denoted as dow_risk) : We created a risk table variable "day of week" that represents the percentage of frauds for each day of week. We want to introduce our out-of-time set (oot) here. We decided to keep all records happening in and after November 2016 as out-of-time set, which we won't use any information to train on. The purpose of this is to hold out data from machine learning models training. By doing so, the prediction accuracies on this set would be a good measure of model performance on data that it has never seen before. To calculate risk table variables, we only applied aggregation on records that are not in the out-of-time set. In other words, we calculated percentages of frauds for each day of the week across all records except out of time records. For example, if dow_risk is 0.02, it means 2% of the transactions on the data are fraud transactions.

We used the following smooth transition function:

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

(c=3, nmid= 15)

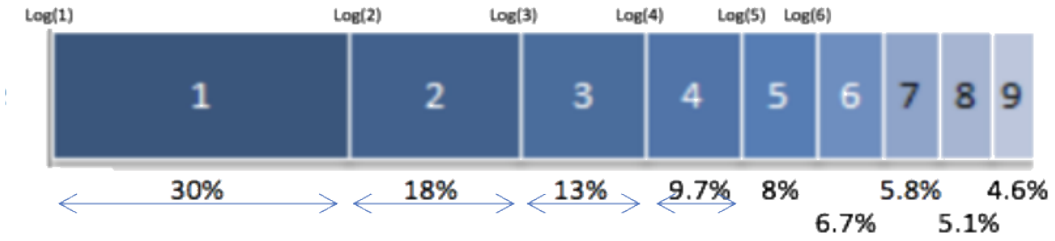
where Y-high is the fraud rate over all records (except oot), Y-low is the fraud rate for that day of the week, n is the total number of records in each group, e.g. the number of records happened on Fridays, nmid and c are parameters to adjust the shape of the function. The rationale for smoothing is when a group has not enough records, the aggregation values might become misrepresentative. For example, if we only have 5 records happening on Friday, it would be

dangerous if we simply calculated the fraud rate for Friday on those 5 records. Instead, we would like to use the overall fraud rate for future use. So basically, the function would return the overall rate when there are too few records in a group.

The risk table we got is as follows:

	Dow	Risk
0	Friday	0.022419
1	Monday	0.008749
2	Saturday	0.010810
3	Sunday	0.010444
4	Thursday	0.018797
5	Tuesday	0.007964
6	Wednesday	0.009530

6. Benford: Benford's Law is the non-intuitive fact that the first digit of many measurements is not uniformly distributed, and the first digit "1" appears about 30% of the time. In fact for each digit there is a percentage assigned (as shown in the graph). So, if we create a low bucket for digit 1,2 and a high bucket for digit 3 to 9, the ratio of high bucket to low bucket should be around 1.096. Before we built these variables, we removed all the transactions from Fedex, as this dataset contains an unusually high amount of Fedex transactions that violates benford's law. After removing these records, we created Benford variables for both Cardnum and Merchnum. We believe if this ratio for cardnum or merchnum is very different from 1.096, it could be a high possibility of fraud. So the deviation from Benford distribution can be an indication of fraud. We also applied a smooth function similar to the one used for the risk table.



After the entire variable creation process, we had 430 candidate variables. But this is a very high dimension data and there is a need to reduce the dimensionality. So, we carry feature selection after this.

Feature Selection

Pre-processing

Before the feature selection process, we carried out 2 pre-processing steps:

1. Remove first 2 weeks of data: we did this to avoid false tagging of variables with limited look back information
2. Z-scaling: we rescaled all variables to have properties of standard normal distribution with mean = 0 and standard deviation = 1

Filter

Univariate Kolmogorov Smirnov (KS)

Kolmogorov Smirnov (KS) looks at distributions of good (non-fraudulent) records and bad (fraudulent) records. More specifically, KS plots goods and bads over the current variable value and sees how well the two distributions separated. The KS score ranges from 0 to 1. If the score is close to 1, it means the distributions are well separated and this indicates the current feature is potentially good for predicting the label. The formula for KS is as follows:

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

In short, KS returns the maximum difference between the cumulative distribution functions (CDF) for the two distributions.

We then rank the candidate features by KS scores, and here are the top 10:

Sort by KS rank					
Field	**KS**	**FDR**	**KS rank**	**FDR rank**	**Average rank**
Fraud	1	1	430	430	430
card_zip_total_7	0.686346785	0.639400922	429	428	428.5
card_merch_total_7	0.683970825	0.634792627	428	427	427.5
card_merch_total_14	0.678766647	0.633640553	427	426	426.5
card_zip_total_3	0.677901595	0.641705069	426	429	427.5
card_merch_total_3	0.677411902	0.631336406	425	425	425
card_state_total_3	0.674160259	0.630184332	424	424	424
card_zip_total_14	0.673518215	0.627880184	423	423	423
card_state_total_7	0.669160082	0.59562212	422	419	420.5
card_state_total_14	0.668412543	0.521889401	421	407	414

Fraud Detection Rate (FDR) @3%

Fraud Detection Rate (FDR) is the percentage of actual frauds caught by examining a certain percentage of the population by using only one feature. The higher the percentage, the higher the predicting capability of its label.

More specifically, we ranked actual fraud labels by the value of a feature and calculated how many fraud labels were caught by this ranking among the x% of the population. As suggested by the field expert, we used x = 3 in this project. We divided the number of frauds caught in the 3% of the population by the total number of fraud records to get the FDR. Here, since we did not know if a feature is positively or negatively related to the fraud label, we compared the FDR among the top 3% and the FDR among the bottom 3% and picked the higher value. FDR is a value from 0 to 1. If the FDR at 3% of a feature is higher compared to the others, it would mean the feature is potentially better for predicting the label.

We then ranked the candidate features by FDR at 3%, and here are the top 10:

Sort by FDR rank					
Field	**KS**	**FDR**	**KS rank**	**FDR rank**	**Average rank**
Fraud	1	1	430	430	430
card_zip_total_3	0.6779016	0.6417051	426	429	427.5
card_zip_total_7	0.6863468	0.6394009	429	428	428.5
card_merch_total_7	0.6839708	0.6347926	428	427	427.5
card_merch_total_14	0.6787666	0.6336406	427	426	426.5
card_merch_total_3	0.6774119	0.6313364	425	425	425
card_state_total_3	0.6741603	0.6301843	424	424	424
card_zip_total_14	0.6735182	0.6278802	423	423	423
card_state_total_1	0.6591508	0.6013825	417	422	419.5
card_merch_total_1	0.6602793	0.6002304	418	421	419.5

We take the average of these two rankings and keep the first 80 variables as the result of filters. Here are the top 10:

Sort by Average rank					
Field	**KS**	**FDR**	**KS rank**	**FDR rank**	**Average rank**
Fraud	1	1	430	430	430
card_zip_total_7	0.686346785	0.6394009	429	428	428.5
card_merch_total_7	0.683970825	0.6347926	428	427	427.5
card_zip_total_3	0.677901595	0.6417051	426	429	427.5
card_merch_total_14	0.678766647	0.6336406	427	426	426.5
card_merch_total_3	0.677411902	0.6313364	425	425	425
card_state_total_3	0.674160259	0.6301843	424	424	424
card_zip_total_14	0.673518215	0.6278802	423	423	423
card_state_total_7	0.669160082	0.5956221	422	419	420.5
card_zip_total_1	0.66079335	0.5979263	419	420	419.5

Wrapper

While filter functions are univariate, which means they only take care of relationships between a single variable and the fraud labels, we would like to select a group of features that could work well collaboratively on predicting the labels. To achieve this, we used wrappers. Wrappers use machine learning models to measure the performance of a combination of features. In other words, given a specified estimator model, wrappers could compare the performance of different combinations of features on predicting labels using different evaluation criterions. We used two kinds of wrappers in this project: **Stepwise selection** and **Forward selection**.

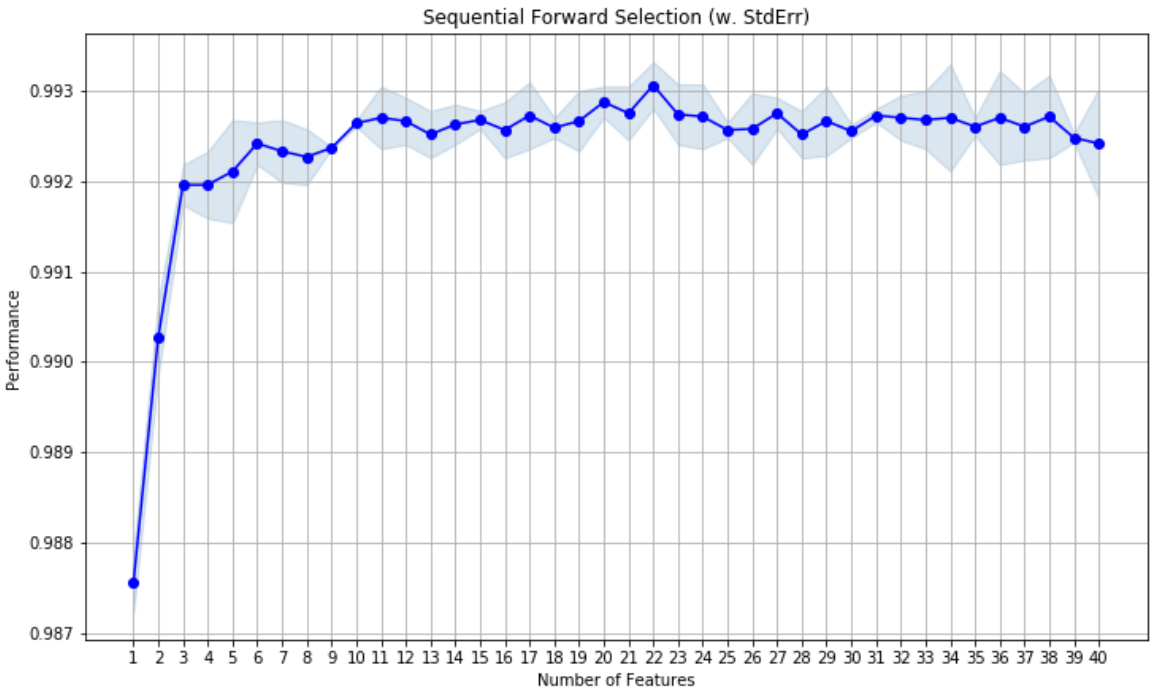
Stepwise selection:

The top 80 ranked variables from filters is subset and used in the wrapper process. We started by applying a stepwise logistic wrapper, where it used a logistic regression model as the estimator and t-value as evaluation criterion to choose the best variable at each step. More specifically, the wrapper starts by building single variable logistic regression of available features, so in our case, there are 80 of them, and then it picks the feature with the best prediction capability which is also represented by the t-value for the variable . Then it builds two-variable logistic regression using the one it just picked with all the other features and picked the best combination and so on. While adding a new feature, it will check the significance of already added features and if any of the variables become insignificant in terms of the prediction accuracy, the wrapper will remove it. In short, stepwise wrapper will not keep features that are highly correlated to the others.

The method we used is RFECV from scikit learn. For each iteration, the function would calculate the cross-validation score, a measure of how well the model performance evaluated by prediction accuracy.

Sequential Forward Selection (SFS):

A drawback of RFECV is we could not know the order of importance of each selected feature. Therefore, we could not determine the final list of features to use since the wrapper would always returns a number of features more than needed. Knowing the optimal number of features to keep, we then used a forward selection function from the SequentialFeatureSelector of mlxtend library, as it allows us to specify the number of features to keep. The logic behind forward selection is similar to stepwise selection. It will start by building simple models and keep adding significant features. As results, it will return an optimal list containing a specified number of features.



We first used logistic regression again as the estimator. But, the results were not very encouraging. So, we used random forest as the estimator. Based on the graph above, we specified the wrapper to keep 22 features as it has the highest prediction performance.

The list of 22 selected features after wrapper are:

card_zip_total_7	card_state_total_3
card_zip_total_30	card_zip_max_7
card_merch_max_30	card_zip_max_3
card_merch_max_3	card_merch_max_7
Cardnum_total_3	Cardnum_total_7
merch_zip_total_1	card_merch_max_1
card_state_max_1	Merchnum_total_3
Cardnum_total_1	merch_state_total_7
Cardnum_max_1	Cardnum_total_14
Cardnum_max_14	card_state_avg_7
card_state_avg_3	card_zip_total_14

Model Algorithms

Logistic Regression

The first algorithm we applied is the logistic regression with L1 and L2 (Lasso and Ridge) regularization feature selection.

Ridge Regression

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2+\lambda\sum_{j=0}^{p}w_j^2$$

Ridge regression puts constraint on the coefficients (w). The penalty term (lambda) regularizes the coefficients such that if the coefficients take values the optimization function is penalized. So, ridge function shrinks the coefficients and it helps to reduce the model complexity and multicollinearity.

Lasso Regression

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2+\lambda\sum_{j=0}^{p}|w_j|$$

Just like Ridge regression, Lasso Regression also shrinks the coefficient. The only difference is that instead of taking the square of the coefficients, magnitudes are taken into account. This type of regularization can lead to zero coefficients. So, Lasso regression not only helps in reducing overfitting, but it can help us in feature selection.

To choose from L1 and L2, we first tried the two types of regularization at the default inverse of regularization strength (C), which equals to 1. It turns out that L1, or Lasso Regression, works better on detecting fraud at 3% level, therefore we decided to go for this methodology.

Then we used SelectFromModel() function with Lasso Regression as the estimator to perform feature selection. We trained the model using 21, 13 and 9 variables. We transformed the training sample and used the transformed one to train a regular logistic regression model to predict the probability of fraud. We sorted the train, test and out of time samples by the probability of fraud in a descending manner and calculated the FDR rate at 3% level. It turns out that 9 variables will result in the highest FDR at 3% level for the test data set which is 69%.

Model	Parameters	Average FDR (%)		
Logistic	# Features	Train FDR3	Test FDR3	OOT FDR3
1	21	66.34%	67.53%	37.43%
2	13	65.35%	66.79%	37.99%
3	9	66.50%	69.00%	37.99%

K-Nearest Neighbors

We also tried K-Nearest Neighbors which is an algorithm based on feature similarity. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.

Model	Parameters	Average FDR (%)		
KNN	# of Neighbors	Train FDR3	Test FDR3	OOT FDR3
1	5	50%	39.95%	18.66%
2	10	48.03%	41.15%	24.19%
3	15	45.98%	41.06%	25.65%
4	20	45.37%	41.20%	26.20%
5	25	44.44%	41.81%	25.92%
6	30	44.24%	41.28%	28.04%
7	40	43.52%	41.10%	27.99%
8	50	43.30%	40.68%	27.49%

Boosted Tree

Then we applied Gradient Boosting tree, which is one of the leading ensemble algorithms which trains a series of weak learners (tree) to result in a strong learner. Gradient boosting algorithm uses gradient descent method to optimize the loss function, for our case, logistic regression for classification. What we do here is fitting an additive model (ensemble) $\sum_t p_t h_t(x)$ in a forward stage-wise manner to the original model. In each stage, we introduced a weak learner to compensate the shortcomings (residual error) of existing weak learners. There are three primary parameters we have used for tuning: number of trees, learning rate and max depth. The number of sequential trees is usually high to make the model robust but as we can see here, when the number of trees greater than 800, the FDR decreases. Learning rate determines the impact of each tree on the final outcome. Lower values are usually better for generalization but require a higher number of trees. Max depth indicates the maximum depth of a tree. Higher depth might lead to overfitting. The best

boosted tree model we have is the model with 600 trees, 5 depths and learning rate of 0.01, which gave us FDR at 3% of 81.66% on the test set.

Model	Parameters			Average FDR (%)		
Boosted Tree	# of Trees	Max Depth	Learning Rate	Train FDR3	Test FDR3	OOT FDR3
1	100	2	0.001	56.12%	54.68%	29.38%
2	100	4	0.01	72.53%	71.35%	49.31%
3	200	5	0.0001	67.47%	65.85%	37.41%
4	200	5	0.01	81.42%	76.31%	50.40%
5	400	5	0.0001	67.31%	66.43%	39.78%
6	400	4	0.001	70.39%	67.27%	41.43%
7	600	5	0.001	73.00%	71.57%	42.46%
8	600	5	0.01	83.33%	81.66%	51.22%
9	600	3	0.01	86.33%	80.39%	48.60%
10	800	5	0.01	81.36%	79.68%	51.38%
11	800	7	0.0001	72.65%	71.24%	46.37%
12	1000	5	0.0001	71.25%	69.93%	41.90%

Random Forest

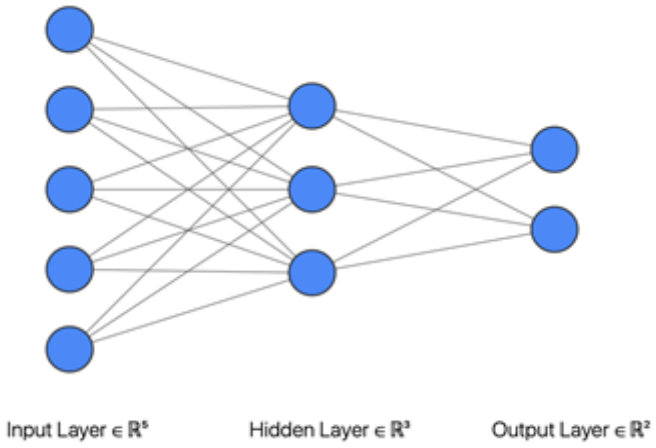
Then we built a random forest model, which is another ensemble model using trees but unlike boosted trees random forest is an ensemble of strong learners. Random forest can be used for classification as well as regression. The random forest is a combination of hundreds of multiple trees, where each tree is trained on a slightly different set of observations and the splitting nodes in each tree consider a random list of limited features. The final prediction of the tree is made by averaging the prediction of each individual tree.

We have tried multiple combinations of different parameters to tune the model like number of trees which represents the total number of trees in the random forest, max features which represents the maximum number of features to consider while looking for best split and min node size which represents the minimum number of samples required to be at a node. The best random forest model we have has 150 trees, max depth of 30 and max features of 15. We were able to achieve 83.36% FDR on the test set and its FDR on out of time dataset is higher than other models.

Model	Parameter			Average FDR (%)		
Random Forest	Max Depth	Max Features	# of Trees	Train FDR3	Test FDR3	OOT FDR3
1	15	7	120	88.53%	82.36%	60.67%
2	15	7	150	88.32%	81.49%	60.34%
3	15	7	200	88.39%	81.41%	60.17%
4	15	12	120	88.24%	82.59%	60.56%
5	15	12	150	88.03%	81.32%	60.67%
6	15	12	200	87.68%	81.47%	60.33%
7	15	15	120	87.68%	80.96%	59.89%
8	15	15	150	87.73%	82.06%	60.56%
9	15	15	200	88.02%	82.34%	59.89%
10	30	7	120	90.06%	82.59%	60.89%
11	30	7	150	90.86%	82.76%	60.22%
12	30	12	120	90.23%	82.15%	60.67%
13	30	12	150	90.58%	82.46%	60.22%
14	30	12	200	90.55%	81.40%	60.22%
15	30	15	120	90.37%	82.31%	60.33%
16	30	15	150	90.09%	83.36%	60.22%
17	30	15	200	90.32%	82.97%	59.27%

Neural Networks

We then built simple neural networks with 1 or 2 hidden layers. The mechanism of neural networks is matrix multiplication. The model takes out data as input, applies linear transformation using weights, catches nonlinear patterns by applying activation functions, and finally outputs the results. A simple example of neural networks could be depicted as below:



Each edge has a weight, and values in the current layer come from the matrix transformation from previous layers. In the project, we used two nodes in the output layer since we were conducting a binary classification, where the predicted probabilities for each class would be a node. To optimize the weights to get the best performance on predicting labels, the model used the idea of gradient descent. This concept is commonly used for finding global minima of a loss function. The loss function measures how far are the model's predictions away from real labels, and therefore the lower the value the better the performance. The model iterated many times through the training set and learned the direction of gradient of the loss function each time. It has been mathematically proven that the direction of the gradient of a function at a point is the direction with the largest change. Therefore, the model calculates the gradient at each iteration and updates all weights by moving the previous weights to the opposite of the gradient (since we want to find the minima).

A complex architecture of networks could result in overfitting issues. Since our input dimension is relatively small, we focused on the number of nodes in (3,10) and didn't go beyond 2 layers. Here is the result table of our neural network models:

Model	Parameter			Average FDR (%)		
Neural Network	# Hidden Layers	Nodes	Alpha	Train FDR3	Test FDR3	OOT FDR3
1	1	3	0.1	73.02%	74.89%	45.92%
2	1	3	0.01	74.18%	75.30%	53.35%
3	1	3	0.001	84.18%	80.07%	50.84%
4	1	4	0.1	75.31%	77.31%	47.93%
5	1	4	0.01	77.12%	78.36%	54.47%
6	1	4	0.001	74.66%	76.23%	49.78%
7	1	5	0.01	77.68%	77.20%	53.86%
8	1	5	0.001	78.32%	78.36%	52.51%
9	1	6	0.01	79.00%	78.02%	51.79%
10	1	6	0.001	80.46%	80.04%	53.63%
11	1	7	0.001	82.16%	79.85%	51.18%
12	1	8	0.001	80.64%	79.52%	55.36%
13	1	8	0.005	82.16%	78.62%	50.22%
14	1	9	0.001	82.11%	79.22%	55.14%
15	2	3,3	0.001	73.74%	76.04%	45.92%
16	2	4,4	0.001	77.06%	77.32%	48.49%
17	2	5,5	0.001	80.41%	77.39%	49.94%

Results

After comparing all models, we selected the Random Forest model which has FDR of 59.78% at 3% with 150 trees, 30 max depth on out-of-time dataset. Thus, we composed this final performance table for train, test, and out-of-time populations separately:

Training set:

Train	# Records	# Goods	# Bads	Fraud Rate
	58779	58135	664	0.011

	Bin Statistics					Cumulative Statistics						
Population_Bin%	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bad (FDR)	KS	FPR
1	587	197	390	33.56%	66.44%	587	197	390	0.34%	60.56%	0.6022	0.5051
2	588	463	125	78.74%	21.26%	1175	660	515	1.14%	79.97%	0.7883	1.2816
3	588	532	56	90.48%	9.52%	1763	1192	571	2.05%	88.66%	0.8661	2.0876
4	588	546	42	92.86%	7.14%	2351	1738	613	2.99%	95.19%	0.922	2.8352
5	587	561	26	95.57%	4.43%	2938	2299	639	3.95%	99.22%	0.9527	3.5978
6	588	585	3	99.49%	0.51%	3526	2884	642	4.96%	99.69%	0.9473	4.4922
7	588	587	1	99.83%	0.17%	4114	3471	643	5.97%	99.84%	0.9387	5.3981
8	588	587	1	99.83%	0.17%	4702	4058	644	6.98%	100%	0.9302	6.3012
9	588	588	0	100%	0%	5290	4646	644	7.99%	100%	0.9201	7.2143
10	587	587	0	100%	0%	5877	5233	644	9.00%	100%	0.91	8.1258
11	588	588	0	100%	0%	6465	5821	644	10.01%	100%	0.8999	9.0388
12	588	588	0	100%	0%	7053	6409	644	11.02%	100%	0.8898	9.9519
13	588	588	0	100%	0%	7641	6997	644	12.04%	100%	0.8796	10.8649
14	588	588	0	100%	0%	8229	7585	644	13.05%	100%	0.8695	11.7780
15	587	587	0	100%	0%	8816	8172	644	14.06%	100%	0.8594	12.6894
16	588	588	0	100%	0%	9404	8760	644	15.07%	100%	0.8493	13.6025
17	588	588	0	100%	0%	9992	9348	644	16.08%	100%	0.8392	14.5155
18	588	588	0	100%	0%	10580	9936	644	17.09%	100%	0.8291	15.4286
19	588	588	0	100%	0%	11168	10524	644	18.10%	100%	0.819	16.3416
20	587	587	0	100%	0%	11755	11111	644	19.11%	100%	0.8089	17.2531

Testing set:

Test	# Records	# Goods	# Bads	Fraud Rate
	25191	24955	236	0.009

	Bin Statistics					Cumulative Statistics						
Population_Bin%	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bad (FDR)	KS	FPR
1	251	117	134	46.61%	53.39%	251	117	134	0.47%	56.78%	0.5631	0.8731
2	252	204	48	80.95%	19.05%	503	321	182	1.29%	77.12%	0.7583	1.7637
3	252	236	16	93.65%	6.35%	755	557	198	2.23%	83.90%	0.8167	2.8131
4	252	239	13	94.84%	5.16%	1007	796	211	3.19%	89.41%	0.8622	3.7725
5	252	250	2	99.21%	0.79%	1259	1046	213	4.19%	90.25%	0.8606	4.9108
6	252	249	3	98.81%	1.19%	1511	1295	216	5.19%	91.53%	0.8634	5.9954
7	252	249	3	98.81%	1.19%	1763	1544	219	6.19%	92.80%	0.8661	7.0502
8	252	250	2	99.21%	0.79%	2015	1794	221	7.19%	93.64%	0.8645	8.1176
9	252	252	0	100%	0%	2267	2046	221	8.20%	93.64%	0.8544	9.2579
10	252	251	1	100%	0.40%	2519	2297	222	9.20%	94.07%	0.8487	10.3468
11	252	251	1	100%	0.40%	2771	2548	223	10.21%	94.49%	0.8428	11.4260
12	251	246	5	98.01%	1.99%	3022	2794	228	11.20%	96.61%	0.8541	12.2544
13	252	251	1	100%	0.40%	3274	3045	229	12.20%	97.03%	0.8483	13.2969
14	252	252	0	100%	0%	3526	3297	229	13.21%	97.03%	0.8382	14.3974
15	252	251	1	100%	0.40%	3778	3548	230	14.22%	97.46%	0.8324	15.4261
16	252	252	0	100%	0%	4030	3800	230	15.23%	97.46%	0.8223	16.5217
17	252	251	1	100%	0.40%	4282	4051	231	16.23%	97.88%	0.8165	17.5368
18	252	251	1	100%	0.40%	4534	4302	232	17.24%	98.31%	0.8107	18.5431
19	252	252	0	100%	0%	4786	4554	232	18.25%	98.31%	0.8006	19.6293
20	252	252	0	100%	0%	5038	4806	232	19.26%	98.31%	0.7905	20.7155

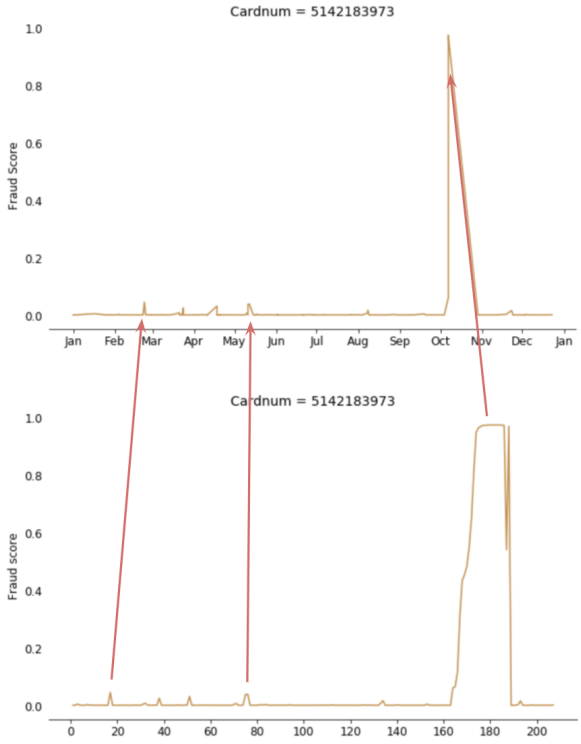
Out of time validation set:

OOT	# Records	# Goods	# Bads	Fraud Rate						
	12427	12248	179	0.014						
	Bin Statistics					Cumulative Statistics						
Population_Bin%	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bad (FDR)	KS	FPR
1	124	67	57	54.03%	45.97%	124	67	57	0.55%	31.84%	0.3129	1.1754
2	125	89	36	71.20%	28.80%	249	156	93	1.27%	51.96%	0.5069	1.6774
3	124	110	14	88.71%	11.29%	373	266	107	2.17%	59.78%	0.5761	2.4860
4	124	120	4	96.77%	3.23%	497	386	111	3.15%	62.01%	0.5886	3.4775
5	124	117	7	94.35%	5.65%	621	503	118	4.11%	65.92%	0.6181	4.2627
6	125	117	8	93.60%	6.40%	746	620	126	5.06%	70.39%	0.6533	4.9206
7	124	121	3	97.58%	2.42%	870	741	129	6.05%	72.07%	0.6602	5.7442
8	124	123	1	99.19%	0.81%	994	864	130	7.05%	72.63%	0.6558	6.6462
9	124	120	4	96.77%	3.23%	1118	984	134	8.03%	74.86%	0.6683	7.3433
10	125	123	2	98.40%	1.60%	1243	1107	136	9.04%	75.98%	0.6694	8.1397
11	124	123	1	99.19%	0.81%	1367	1230	137	10.04%	76.54%	0.665	8.9781
12	124	122	2	98.39%	1.61%	1491	1352	139	11.04%	77.65%	0.6661	9.7266
13	125	124	1	99.20%	0.80%	1616	1476	140	12.05%	78.21%	0.6616	10.5429
14	124	123	1	99.19%	0.81%	1740	1599	141	13.06%	78.77%	0.6571	11.3404
15	124	122	2	98.39%	1.61%	1864	1721	143	14.05%	79.89%	0.6584	12.0350
16	124	124	0	100%	0.00%	1988	1845	143	15.06%	79.89%	0.6483	12.9021
17	125	125	0	100%	0%	2113	1970	143	16.08%	79.89%	0.6381	13.7762
18	124	120	4	96.77%	3.23%	2237	2090	147	17.06%	82.12%	0.6506	14.2177
19	124	124	0	100%	0%	2361	2214	147	18.08%	82.12%	0.6404	15.0612
20	124	123	1	99.19%	0.81%	2485	2337	148	19.08%	83%	0.636	15.7905

Case Study

To better illustrate how the model really works to detect fraud, it's better to look at a specific fraud case and show it dynamically. We choose respectively one fraudulent card number and one fraudulent merchant number as examples to show that fraud score increases with activity.

(1) Cardnum = 5142183973

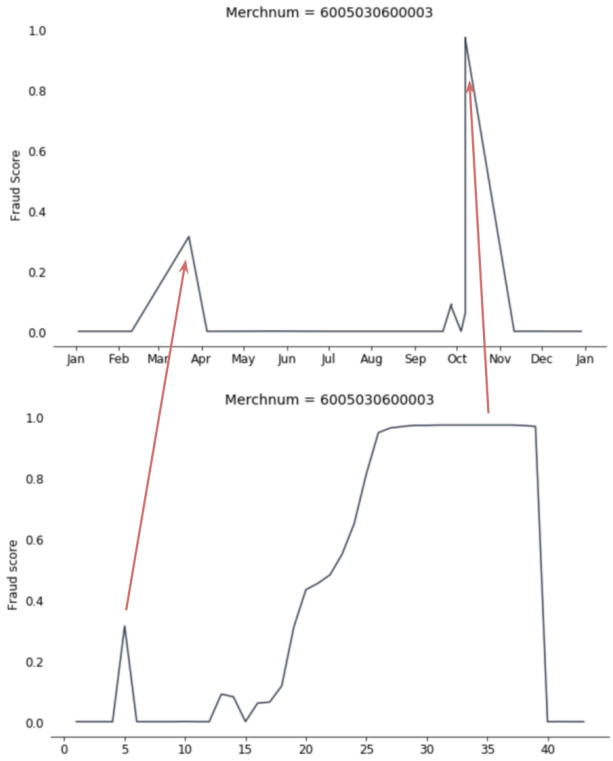


First, we use records from card number ending with 3973 to illustrate the dynamics of fraud detection. These two graphs above are respectively the change of fraud score of this card number against the time across the year and against the number of times we see it.

The graph at the top shows that this card had fraudulent transactions spreading over a variety of months. While the first few times, there were only some small bunching of transactions, so the fraud score only went up a little bit. And it wasn't until the mid of October, where there were 30 transactions happening in a very short time period where we see the fraud score really went up dramatically.

This dynamic represents a fundamental concept of our real-time fraud detection algorithm. The first few transactions will look normal. However, as the same card number appears multiple times in a short time window, it becomes increasingly suspicious. As the count variables increases, so as the fraud scores given by our model. The more times we've seen that card number in a short period of time, the more likely it's a stolen card.

(2) Merchnum = 6005030600003



Next, we use records from merchant number ending with 0003 to illustrate the same dynamics. When there was a small increase in transactions during March, the score went up only by a little bit. But when the large burst of transactions with this merchant number happened within a few days in October, the fraud score went to a peak point.

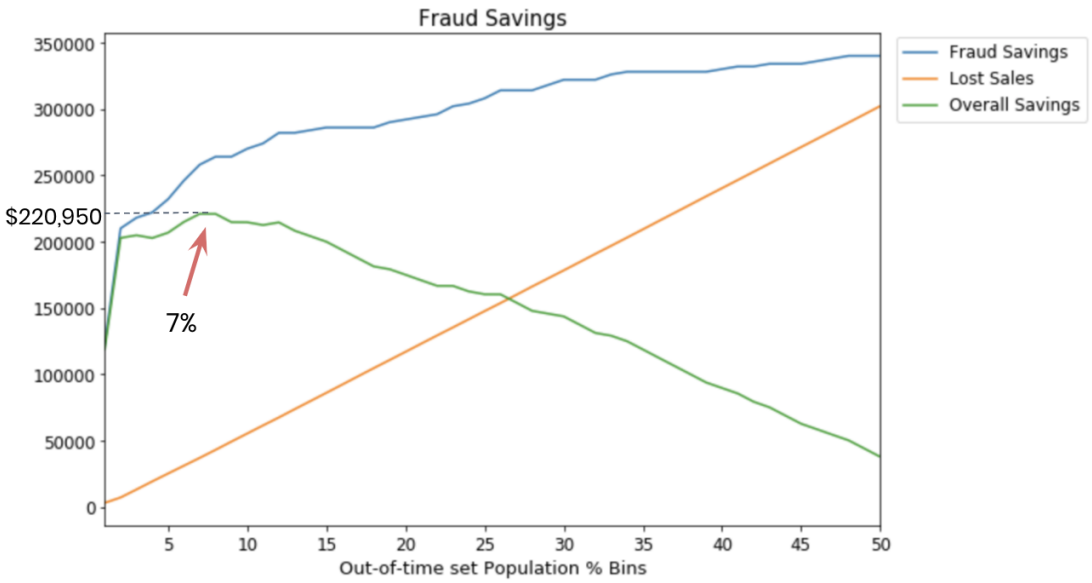
Therefore, based on this mechanism, it's reasonable that our model would not detect the first few fraudulent transactions from a given card or a merchant. However, with the increase of fraud activity, our model would respond by assigning a high fraud score.

Business Implementation

When applying the model into business, we need to set a fraud score cutoff as criteria to determine our business decision. To maximize profit, we make a recommendation for the cutoff level based on the model performance on the out-of-time dataset.

There are two business assumptions we obtained from the business team:
1) Each time we catch a fraud successfully, we will gain 2,000 dollars.
2) Each time we accidently label a good transaction to be a fraud will cause a loss of 50 dollars.

With these 2 assumptions, we use our result on the out-of-time records, calculate the profit by gradually going deeper in penetration of population bins. The graph below shows the relationship between penetration level and fraud savings.



The blue line represents the monetary savings from fraud detected. There is a burst of fraud savings at the left side where high fraud score transactions are located. Then the degree of increase slows down as we move further right. The orange line represents how much money we lose due to mistakes. And the green line combines two lines, representing the total savings from implementing our model. As shown in the graph, the total saving goes up first and then gradually goes down. Therefore, we recommend setting a cutoff at 7% penetration level, which is the elbow part of this green curve. The corresponding score cutoff is 0.017492. We believe it will maximize our profit. Based on our result, it will approximately generate an annual saving of 1.3 million dollars.

Conclusions

We have built a model with a 59.78% fraud detection rate at a 3% penetration rate on out of time data to identify fraud transactions in the card transaction dataset. This means that we can catch 59.78% of fraud transactions by flagging 3% of the overall dataset.

We first built a data quality report for the raw dataset of card transactions that contained 10 variables and 96,753 records. In the data quality report, we defined the variables and looked at the properties and distributions of all the variables. We removed an outlier in the amount field and kept purchase records only, which brought the number of records to 96397.

We also found out that there are three variables that are not fully populated: merchant number, merchant state and merchant zip code. To impute merchant numbers, we started with filling in with the most frequent value in their corresponding entity groups at different granularity levels including merchant description, merchant state etc. If nothing has been matched, we treat them as frivolous records and replace them with their negative record number to avoid triggering false alarm. For merchant state and merchant zip, because there are some invalid instances, for example, Canadian states and non-US zip code, we treat them separately.

After the data cleaning process, we created 430 expert variables by using the initial 10 variables of the dataset. First, we created various combinations of identity groups, such as entities for linking, and then we created amount variables, frequency variables, day since variables, and relative frequency variables based on these groups. We also created a risk table variable for the day of the week, that is the probability of a fraud application on a particular day of the week as well as two variables based on Benford's Law.

As we now have 430 variables and just 96,397 records, there is a problem of high dimensionality and there is a need to reduce the dimensionality of the dataset. Hence, we built a series of feature selection methods to reduce dimensionality. We calculated the KS and FDR score and ranked the variables based on their combined ranking and subset the dataset with the highest 80 variables. Then we further reduced the dimensionality by using the sequential forward selection method. Then we reduced to 22 variables by using a forward selection random forest model. The forward selection runs iterations of models starting from the most informative variable and adds the next best variable until the predictability of the model increases with the addition of the variable.

Then we subset the whole dataset by reserving the last 2 months' records as validation/Out of Time data (oot) and the rest as modeling (training and test) data. we first built a linear model i.e. logistic regression on the training data and predicted the training, test and out of time sample data. This linear model acts as a base for our prediction. We then built several non-linear models like KNN, Decision Tree, Boosted Tree, Random Forest and Neural Network on the training data and predicted the FDR at a 3% cutoff population for training, test and out of time sample data. Based on these results the Random Forest has the highest FDR of 59.78% at a 3% on the out of time data set. Hence, the business can potentially reduce 59.78% of fraud transactions by rejecting 3% of transactions. Given the assumptions that the profit gained from each fraud caught is 2000 dollars

and the loss caused by each mistake is 50 dollars, we would recommend setting the cutoff point at 7%. By implementing this strategy, we estimate the annual savings would be 1.3 million. For the further scope of the project, we could have built an ensemble model by combining the results of all the models to predict the fraud transactions.

Appendix: Data Quality Report

PART I. High-Level Description of Data

This is a credit card transaction dataset. The dataset contains information about the transaction, the merchant involved in the transaction and a fraud label for each application. The time period this dataset covers is from January 1, 2010 to December 31, 2010. The dataset has 10 fields and 96,753 records of product applications.

PART II. Summary Tables

Following are two summary tables for 10 fields in the dataset.

(1) Numeric Field:

Field Name	# of Records	% Populated	# Unique Values	# of Records with Value Zero	Mean	Standard Deviation	Min	Max
Amount	96,753	100%	34,909	0	427.89	10,006.14	0.01	3,102,046

(2) Categorical Fields:

Field Name	# of Records	% Populated	# Unique Values	Most Common Value
Recnum	96,753	100%	96,753	-
Cardnum	96,753	100%	1,645	5142148452
Date	96,753	100%	365	2010-02-28
Merchnum	93,378	96.51%	13,092	930090121224
Merch description	96,753	100%	13,126	GSA-FSS-ADV
Merch state	95,558	98.76%	228	TN
Merch zip	92,097	95.19%	4,568	38118
Transtype	96,753	100%	4	P
Fraud	96,753	100%	2	0

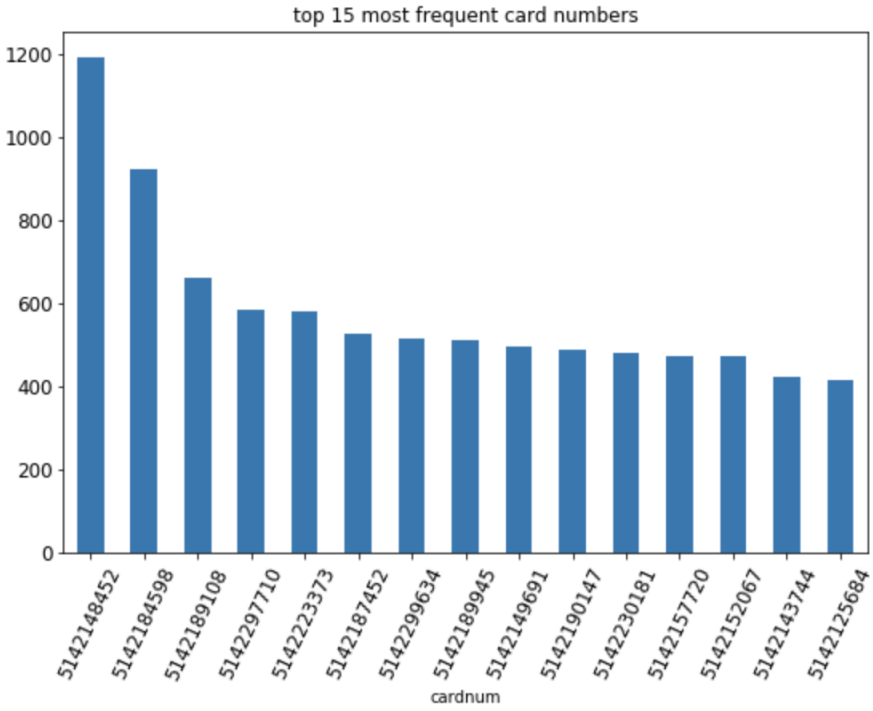
PART III. Fields Description

Field 1: Recnum

Description: A unique index number for each record.

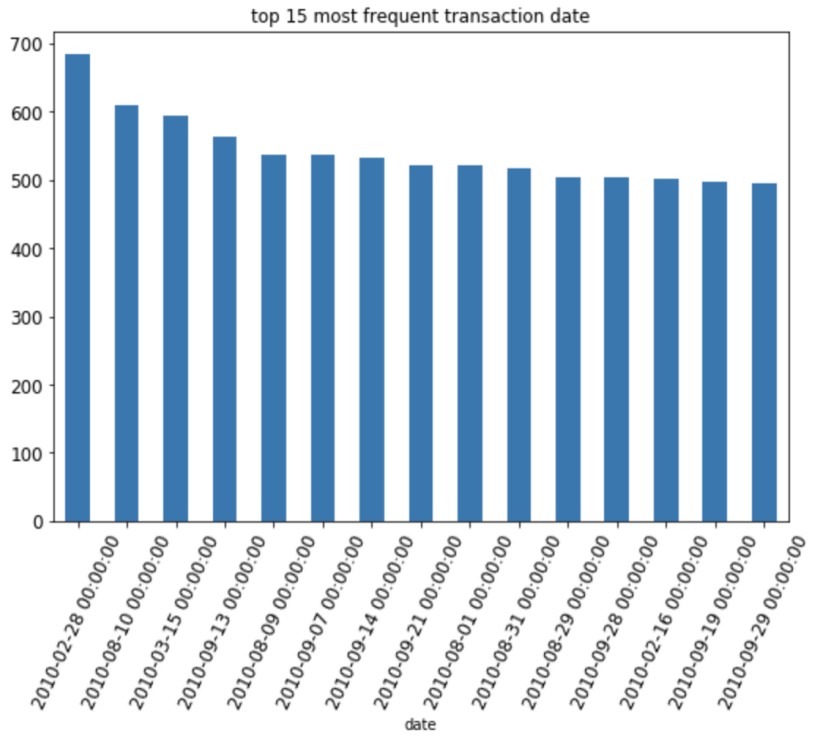
Field 2: Cardnum

Description: The card number of the credit card used in the transaction.



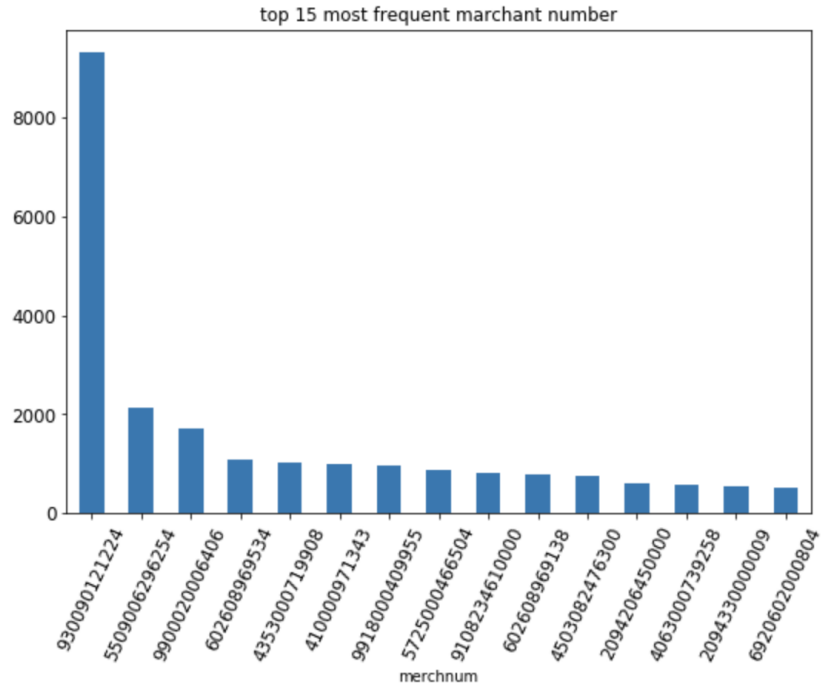
Field 3: Date

Description: The date when the transaction happened.



Field 4: Merchnum

Description: A unique identifier for each merchant who received the credit card payment.



Field 5: Merch description

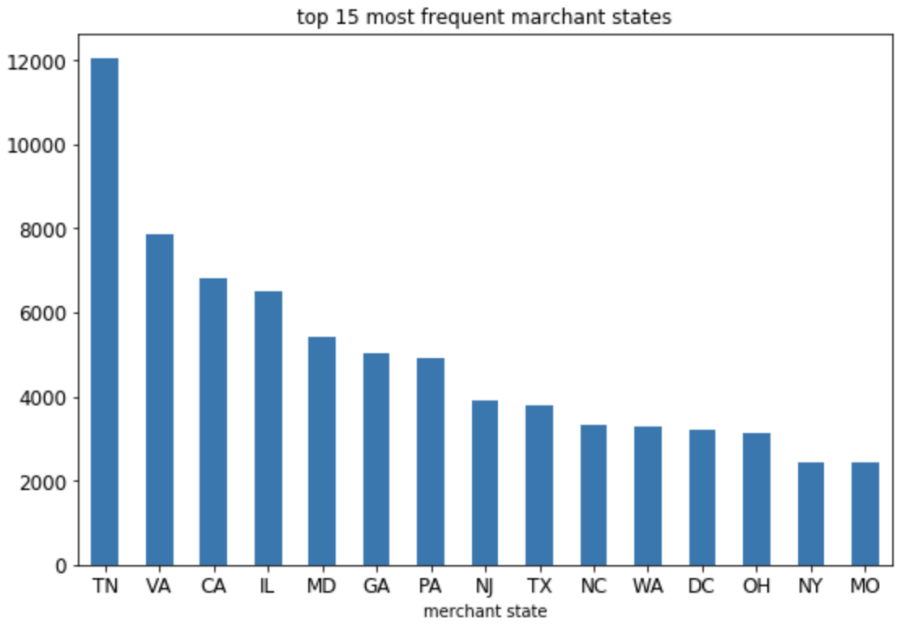
Description: A short description of the merchant who received the credit card payment.

Graph: Top 15 most frequent merchant descriptions

	count
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
VWR SCIENTIFIC PROD VCTS	688
PC *PC CONNECTION	570
C & C PRODUCT SERVICES	558
BUY.COM	481
FISHER SCI HUS	442

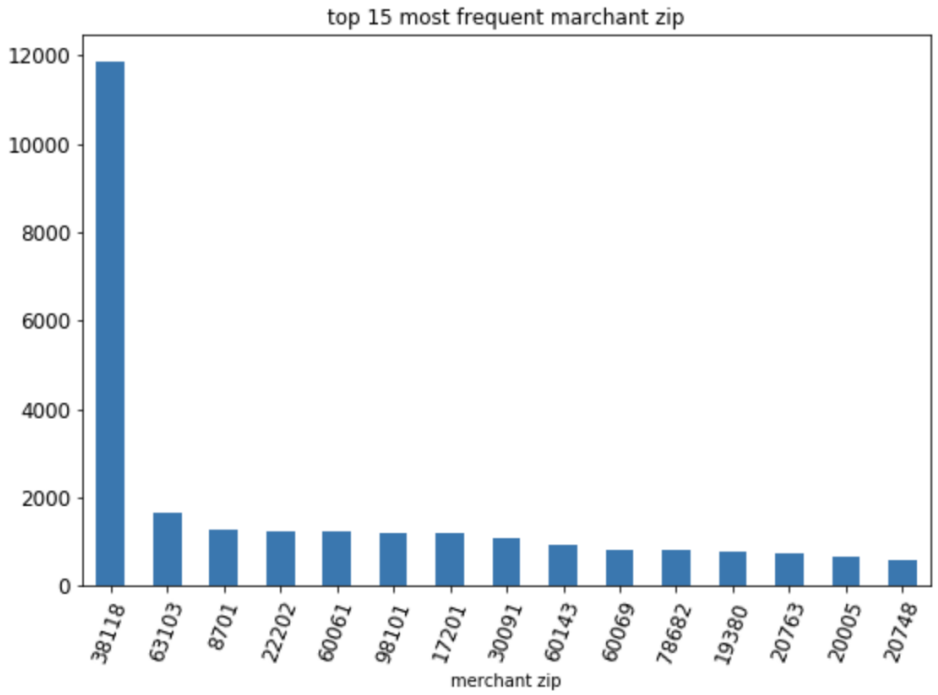
Field 6: Merch state

Description: The state code of the state where the merchant who received the credit card payment is located in.



Field 7: Merch zip

Description: The zip code (5 digits) of the merchant who received the credit card payment.



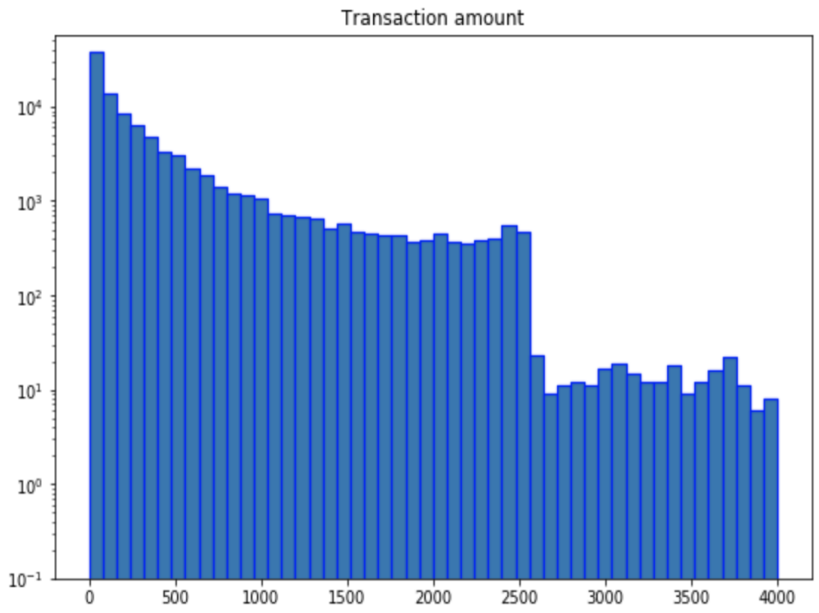
Field 8: Transtype

Description: Type of the transaction. (4 different types)

	count
P	96398
A	181
D	173
Y	1

Field 9: Amount

Description: The dollar amount of the transaction.



Field 10: **Fraud**

Description: A label indicating whether the transaction is fraud. (0 - Not Fraud; 1 - Fraud).

	count
0	95694
1	1059