ECO 395 Project: Transaction Fraud Detection

Patrick Chase

5/8/2021

Abstract

The problem I'd like to solve is the identification of credit card fraud for a leading payment service company, Vesta Corporation, who specializes in guaranteeing card-not-present (CNP) transactions. Specifically, I set out to create the most effective classification model to identify whether or not a given transaction is fraudulent. Additionally, I'd like to answer the question of which variables appear to be most connected to the incident of fraud. I choose to do so through a gradient boosted tree algorithm implemented through the use of xgboost R package. I chose this method of over gbm because of xgboost's ability to cope with over-fitting. In addition to that, it is also offers significantly more efficient processing and memory utilization through it's use of sparse matrices. While that presented its own challenges, the trade-off was still worth the investment given the size of this data set. I was able to achieve a classification accuracy of approximately 96% with relatively quick processing times.

Introduction

Fraud detection and prevention has been a persistent problem since the widespread use of non-cash payment systems became popular in the mid 1990s. Given the regulatory environment of the United States and the massive increase in the amount of transactions, corporations have a large incentive to prevent fraud in real time. These circumstances present a ripe environment for automation through the use of machine learning, which has been relatively common. Banks, payment processors, and tech companies such as Apple, Amazon, and Microsoft devote copious resources to the development of automated fraud detection systems.

Despite those efforts, billions of dollars are lost each year due to fraud in a diverse range of fields. Bad actors and corporations are in a constant arms race when it comes to fraudulent activity. Whether it's Facebook attempting to detect fraudulent ad buys on their site or a payment processor such as Vesta preventing fraud from occurring at the transaction level, being able to effectively detect and prevent fraud is in the interest of businesses and consumers. While potentially relatively simple, automated fraud detection through machine learning present an easy avenue into the use of algorithms for a typical business to solve everyday problems.

In the spirit of the problem, Vesta partnered with IEEE Computation Intelligence Society to host a competition on Kaggle with a real world data set. The goal was simple. Create the most accurate classification model. While the competition is closed, I think this particular problem and rich data set provide a real world opportunity to demonstrate the skills I've attained in this class.

Methods

Data

The *data set* used in this analysis was found through a Google search of "fraud detection data sets". It consisted pre-split files separated into four different comma-separated values (CSV) files, two CSV files for a training set and two for a test set. For each set, the transaction data is stored in one file and anonymized identity data is stored in the other. As this was posted as part of a competition, the posted test set does not include the outcome variable of interest, isFraud. Its purpose was to be used by participants submit their work to the competition. Because of that it was not suitable for testing the effectiveness of my work. As a result, I generated my own training and test sets from the training data provided by Vesta.

The data itself is a large collection of information for each transaction in a limited time period. As this is real world data, the time period is not specific when it was collected but instead is a delta in relation to the first transaction. There are also variables that are explicitly related to products purchased, addresses, and email domains. The rest of the data are features that have been engineered by Vesta. The actual meanings of those variables have been masked, however, for the purposes classification they were satisfactory.

Additionally, the technical constraints of my laptop prevented me from working with the full data in a timely manner. In short, my computer doesn't have enough memory to hold the full data set and the necessary objects to run a xgboost model. As a work around, I chose to import the data from the Kaggle, merge the relevant CSV files, and select a random sample that was the equivalent of 20% of the original data. This resulted in a data set that is 28,847 observations of 435 variables. The code for that work flow can be found in the Appendix of the RMarkdown file.

Data Cleaning and Preperation

My strategy of choice is to rely on gradient boosted trees through the use of the xgboost R package. Xgboost requires all variables be either numeric or factor before being fed into sparse matrices. As this is real world data, multiple variables had large majorities of their observations as NA. I one-hot-encoded my categorical variables and was able to output sparse matrices for my training and test data.

My first attempt at estimating my model ran without any issue. However, when I went to diagnose its effectiveness, my training data had one more column then the testing data. This was due to the high level of NAs in the data and the random sampling of the data when conducting my train/test split. The split was not capturing enough observations in the testing set in order to fully capture all the variables in the sparse matrix. This presented an interesting problem that took a nonnegligible amount of time to diagnose.[1]

For my purposes, I found it to be more expedient to drop those variables all together rather than attempting to account for all the potential omitted observations due the way sparse model matrix operates. If the isFraud variable hadn't been omitted from the posted test set, this wouldn't have been an issue.

Gradient Boosted Trees

My chosen method is gradient boosting through the use of xgboost. Boosting in a general sense, relies on decision trees and in each iteration sets a weight to those observations it mis-classified in earlier trees in order to get it right. Xgboost does applies this algorithm in a a highly efficient way due to its use of sparse matrices. This saves memory and simplifies computations.

[1] Interested parties can find more details of that work flow in the Appendix

Results

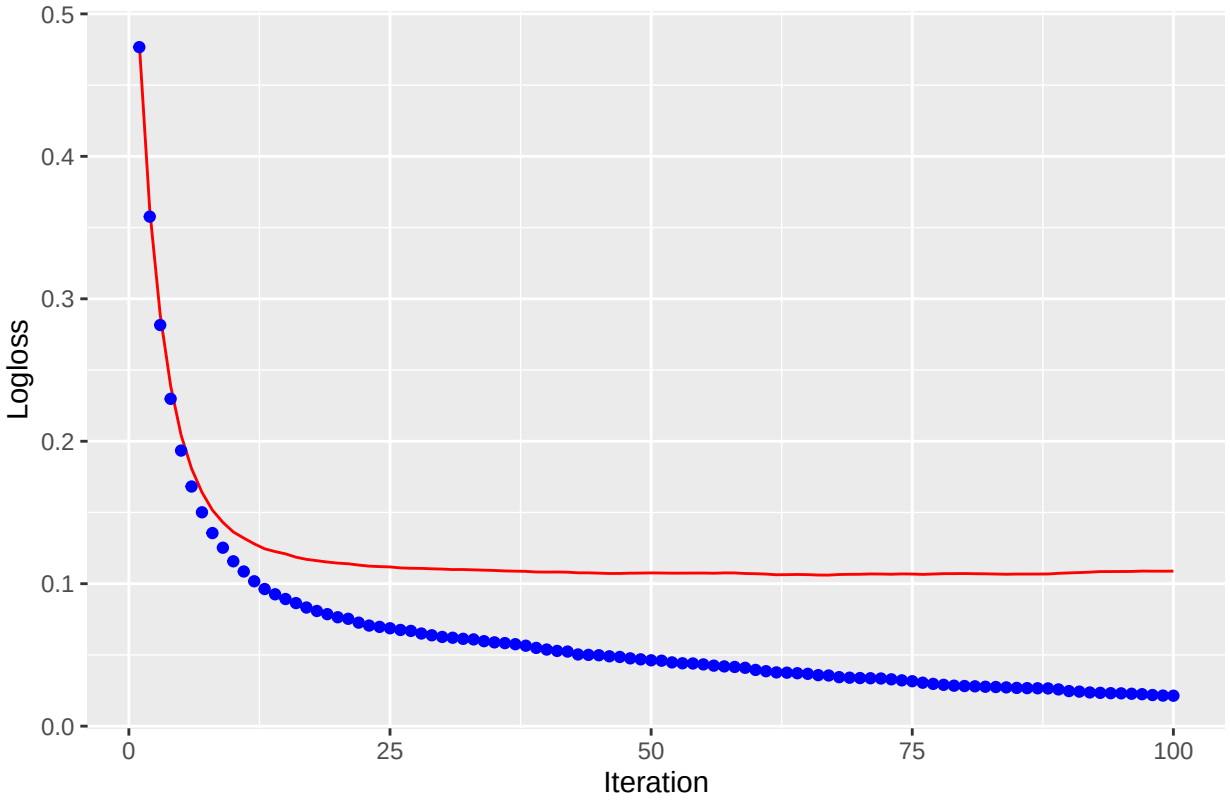
Table 1 shows the confusions matrix for my model's classification. Summing the diagonals and dividing by the total number of observations shows that the model had over 96% classification accuracy.

Table 1: Confusion Matrix

	0	1
0	7931	219
1	55	450

Classification.Accuracy
96.8342

Figure 1 Error Plot

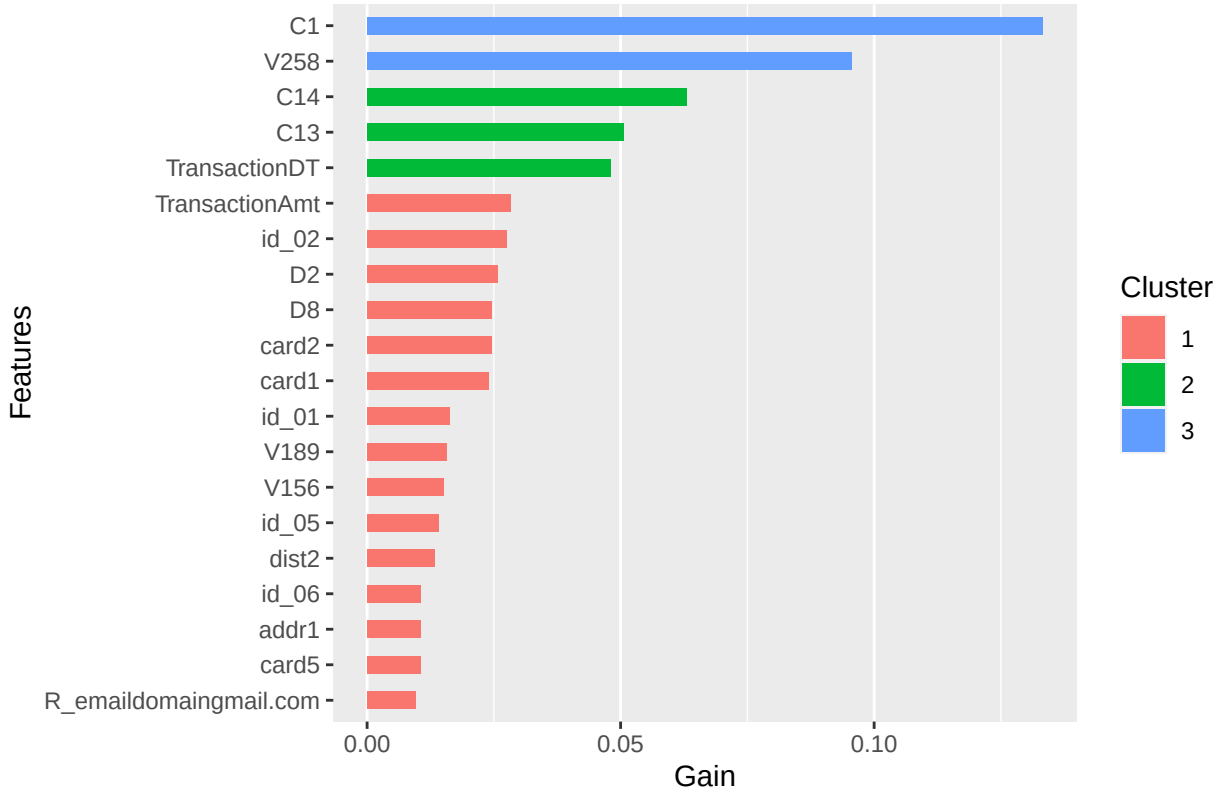


The red line in the error plot represents the log loss of the test data, while the blue dots represent the log loss of my training data. It above demonstrates that my model is unlikely to be over fitting. If over-fitting were occurring, the test log loss would initially fall but then quickly begin to climb. The fact that it follows closely is a good sign.

The table below shows the relative importance of each feature in the model, sorted by gain and focused on the top 20 variables. The variable importance plot focuses in on gain.

Feature	Gain	Cover	Frequency	Importance
C1	0.1332870	0.0722895	0.0209205	0.1332870
V258	0.0955857	0.0261719	0.0026898	0.0955857
C14	0.0631062	0.0622939	0.0089659	0.0631062
C13	0.0505060	0.0173649	0.0182307	0.0505060
TransactionDT	0.0481076	0.0541507	0.0621638	0.0481076
TransactionAmt	0.0283214	0.0369475	0.0484160	0.0283214
id_02	0.0274434	0.0276677	0.0615660	0.0274434
D2	0.0258083	0.0150631	0.0080693	0.0258083
D8	0.0245610	0.0232623	0.0394501	0.0245610
card2	0.0244878	0.0474061	0.0358637	0.0244878
card1	0.0239188	0.0241538	0.0385535	0.0239188
id_01	0.0162486	0.0061491	0.0179319	0.0162486
V189	0.0157097	0.0174106	0.0023909	0.0157097
V156	0.0150808	0.0070631	0.0026898	0.0150808
id_05	0.0141894	0.0163165	0.0268978	0.0141894
dist2	0.0133187	0.0151444	0.0239091	0.0133187
id_06	0.0106409	0.0111570	0.0197250	0.0106409
addr1	0.0105867	0.0163921	0.0233114	0.0105867
card5	0.0104953	0.0060878	0.0212194	0.0104953
R_emaildomaingmail.com	0.0095130	0.0110043	0.0071727	0.0095130

Figure 2 Feature Importance



Conclusion

My model performed very well after my data had been properly cleaned and formatted. This exercise demonstrates the relative ease with which gradient boosting can produce tangible and useful insights when it comes to classification.

Interesting to note is that a large proportion of the top features are ones that Vesta engineered and thus can't really be interpreted outside of their organization. The details they do provide indicate that variables beginning with "C" are counts. It could be count's of addresses, total transactions on this card, etc. Without knowing more behind it's meaning its difficult to say. Another interesting observation is that the time variable, TransactionDT, was a relatively important feature. Without knowing where the time measure begins, it's difficult to say what to make of that, although I suspect that it may correspond to working hours in countries where many organized credit card fraudsters base their operations. Regardless, if Vesta were inclined to do so they could have their system automatically flag transaction with abnormal counts of C1, C13, and C14. They could even add additional weights to those if they occur in the time periods I've suggested above. If that were they case they could then require multi-factor authentication in order to ensure that it's a true customer engaging in a transaction and not a bad actor.

Appendix

A point of interest I discovered in my analysis is just how unlikely it was for me to run into the difficulties I had with sparse.matrix.model command. If I had picked any random number other than "1234" to set my seed, I would have had my training and test matrices off by a dozen to hundreds of variables. Instead, it was only off by 1 column leading me to believe I had simply misplaced a rogue minus sign somewhere. I eventually ended up creating a data frame that matched all the coloumns of the two matrices, and created a short ifelse rule to identify where they weren't matching up.