The University of Oklahoma

Intelligent Data Analytics

(DSA/ISE-5103)

Detecting Fraud on Plastic Card Transactions

Final Project Report

December 16, 2017

Daniel Casares and Felipe Olivera

Executive Summary

In this work, the problem of identifying whether a credit or debit card account on the transactional level has been subject to fraudulent activity is addressed. The goal is to construct supervised learning models for binary classification that can detect fraud on new (previously unseen) plastic card transactions and then to compare them with each other.

The dataset was obtained by "Universidad Católica del Uruguay" in a collaboration project with a Latin American card processing company and is not publicly available. The original dataset had been under-sampled due to the low proportion of the target class (i.e. frauds) and since enough computing power was not available to process the original dataset with 41,091,288 individual transactions. Feature engineering is performed since a single transaction leaves behind the customer spending behavior. Thus, transaction aggregation is done in order to create aggregated features.

Three basic tried and tested classification algorithms for plastic card fraud detection are implemented. This is logistic regression, decision tree and random forest that all can be used for binary classification. Apart from these three models another tree model using *Extreme Gradient Boosting (XGBoost)* that is often a winning model for data science competitions on *Kaggle* is implemented. All four models are implemented in R using the *caret* package for training and validation with 10-fold cross validation and 5 repeats and for tuning of the hyper-parameters using Area Under the ROC Curve (AUC) as performance measure. The models are finally compared on how they perform on the test set using a custom metric *savings* in USD.

All four binary classification models have satisfying results since they generate positive savings that range from USD 412,802 for Logistic Regression model to USD 471,249 for Random Forest. The best-performant model, Random Forest, reaches 84% of the maximum possible savings of USD 561,712.

Problem description and background

The use of plastic cards (i.e. credit and debit cards) as a payment method has grown significantly over past years, unfortunately so has fraud (Bahnson 134). Plastic card fraud is defined as an unauthorized account activity committed by means of the debit and credit facilities of a legitimate account. Some successful fraud tactics observed in the industry are lost and stolen card fraud, counterfeit card fraud, card not present fraud, mail non-receipt card fraud, account takeover fraud and application fraud (Krivko 6070). Based on the latest figures gathered in 2015, card fraud accumulated USD 21.84 Billion worldwide in losses (The Nilson Report 6). When banks lose money due to credit card fraud, the losses are partially passed to customers through higher interest rates, higher membership fees and reduced benefits. Hence, it is both the banks' and cardholders' interest to reduce illegitimate use of credit cards (Maes 2).

In this work, we consider the problem of identifying whether a credit or debit card account on the transactional level has been subject to fraudulent activity, using real-life transaction data from a Latin American credit card processing company. The goal is to construct supervised learning models that can detect fraud on new (previously unseen) plastic card transactions and then to compare them with each other. Fraud detection is, given a set of credit card transactions, the process of identifying those transactions that are fraudulent. Thus, the transactions are classified as genuine or as fraudulent transactions (Maes 2). Different detection systems that are based on machine learning techniques have been successfully used for this problem, in particular: neural networks, bayesian learning, artificial immune systems, association rules, hybrid models, support vector machines, peer group analysis, decision tree techniques such as ID3, C4.5, and random forest, discriminant analysis, social network analysis and logistic regression (Bahnson 135, Mahmoudi 2510).

Exploratory data analysis

For this project we used one dataset provided by a Latin American card processing company. Before explaining the details of the dataset is important to state that it isn't available in public sources (Kaggle, UCI Machine Learning Repository, etc.), it was obtained by "Universidad Católica del Uruguay" in a collaboration project with a Latin American card processing company. The dataset consists of fraudulent and legitimate transactions made with debit and credit cards between July 2014 and June 2015. The total dataset contains 41,091,288 individual transactions with no missing values, each one with 13 attributes, including a fraud label indicating whenever a transaction is identified as fraud (Table 1). This label was created internally in the card processing company, and can be regarded as highly accurate. In the dataset only 12,632 transactions were labeled as fraud, leading to a fraud ratio of 0.031%.

Attribute name	Description
amount	Amount of the transaction in USD
id_issuer	Unique identifier of the bank issuer of the card
id_merchant	Unique identifier of the merchant
datetime	Date and time of the transaction
country_code	Numeric code that identifies the country of the transaction
tokenized_pan	Unique identifier of the credit card (Primary Account Number (PAN))
pos_entry_mode	Numeric code that identifies the transaction entry mode (e.g. Chip and PIN, magnetic strip, etc.)
id_mcc	Identification of the Merchant Category Code (ISO 18245)
is_upscale	Indicates if the card holder is an upscale customer
mcc_group	Merchant Category Code grouped by major type of business
type	„C“ for credit cards, „D“ for debit cards
is_fraud	1 if the transaction was fraudulent, 0 otherwise

Table 1: All 13 attributes of the dataset

Due to the low proportion of the target class (i.e. frauds) in the given dataset, the class imbalance problem arises. Classification of imbalanced data is difficult because standard classifiers are driven by accuracy, thus the minority class may simply be ignored (Visa 67). Generally all classifiers present some performance loss when the data is unbalanced (Prati 253). Additionally, many imbalanced datasets experience problems related to its intrinsic characteristics, such as lack of density and information. To illustrate these issues, a dataset containing of 5 : 95 minority-majority examples and a dataset of 50 : 950 are compared. Though the imbalance factor is the same as in both datasets in the first case the minority class is poorly represented and suffers more from the lack of information factor than in the second case.

Another difficulty associated with dataset on hand is the computing power required to preprocess the data and train the different predictive models, due to large number of observations contained in it. Since we haven't enough computing power, we decided to under-sample the dataset. Our approach was to include all the frauds of the original dataset. To do that, we loaded the 41 million records in R, and selected the list of "tokenized_pan" for all the fraudulent transactions ("is_fraud" equals to 1), resulting in 3,841 unique customers. Then, from the whole dataset, 8,000 customers were randomly selected and added to the previously mentioned list. After executing the *unique* function again over the list, it contains 11,296 unique tokenized PANs. By selecting all the transactions associated with those card numbers, the *new* dataset contains 523,049 transactions and a fraud ratio of 2.33%. This process not only resulted in a smaller dataset, decreasing the computing power needed to work with it, but also decreased the imbalance problem (increasing the minority class proportion more than 75 times). In this *new* dataset, the total financial losses due to fraud are USD 1,876,697. From plotting the amount of fraudulent and total transactions over time we can see that the proportion of fraudulent transactions varies over time (Figure 1). In Dec 2014 the fraud rate was the highest. However, we have no explanation for this behavior.

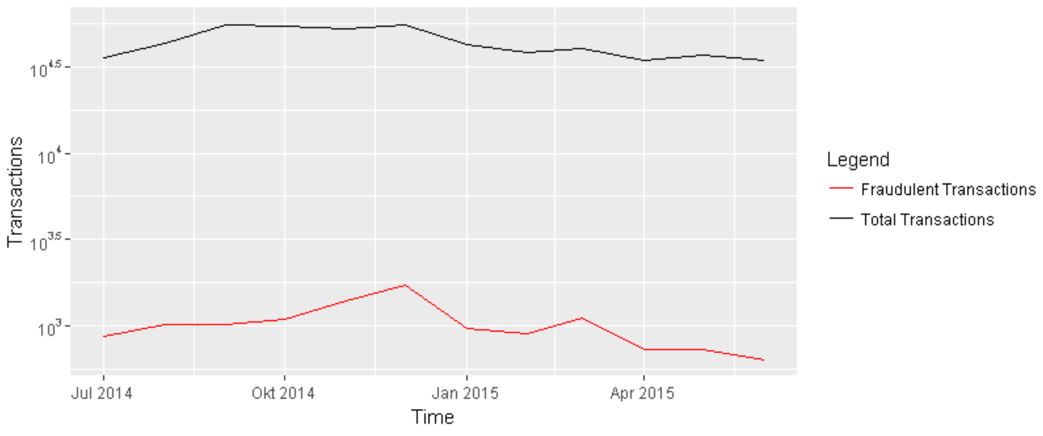


Figure 1: Amount of fraudulent and total transactions over time

The boxplot of the log amount per transaction in USD versus non fraudulent and fraudulent transactions shows that the amount of fraudulent transaction is a bit higher and with less extreme values (Figure 2). So, they were neither as extremely low nor as extremely high like the non-fraudulent transactions. The natural logarithm of amount was taken because amount has a lot of high outlying values making a meaningful interpretation of the boxplot impossible. In addition it is remarkable that some transactions have USD 0 as amount. This could happen if a merchant wants to verify the cardholder's card information upon acceptance when there is a delay between collecting the card data and actually charging the card. For this reason we do not consider those transactions as outliers and retain all 523,049 transactions of the data set.

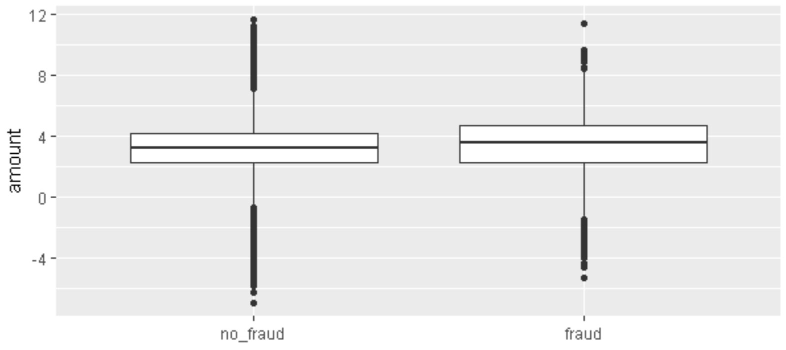


Figure 2: Log amount per transaction in USD versus non fraudulent and fraudulent transactions

Analysis plan

Explanation of modeling choice

In scientific literature three basic tried and tested classification algorithms for plastic card fraud detection are discussed. These are logistic regression, decision tree and random forest that all can be used for binary classification (Whitrow 31-51, Bahnson 134). Logistic regression is known for its simplicity, but does require the user to identify effective representation of the predictor data that yield the best performance (Kuhn 286). The decision tree is a basic regression tree that parts the data into smaller groups that are more homogenous with respect to the response. An advantage of tree based models is that, when the tree is not large, the model is simple and interpretable. However, a disadvantage of single regression trees is that they are more likely to have sub-optimal predictive performance compared to other modelling approaches (Kuhn 175-181). Random forests combine the technique of bagging trees to reduce variance of the prediction with generating bootstrap samples to introduce a random component into the tree building process. An advantage of random forests is that they are protected from overfitting. One disadvantage of random forests is that they are more computational intense that basic models. (Kuhn 198-200). Apart from these three models we implement another tree model using *Extreme Gradient Boosting (XGBoost)* that is often a winning model for data science competitions on *Kaggle* (Gordon). Two of the advantages of XGBoost are that it is regularized, a technique that helps to prevent over-fitting, and that it allows parallel processing what makes training of the model faster. However, implementing a model using XGBoost is easy but improving the model using XGBoost challenging (Jain).

For modeling purposes the *new* dataset with its aggregated features (see Feature Engineering) is split randomly in 70% training and 30% test taking into account the same proportions of class labels in both data sets. The models are implemented in R and trained and validated with the *caret* package using repeated k-fold cross validation (5 repeats of 10-fold CV). An advantage of this resampling technique is that all observations are used for both training and validation. K = 10 folds is often used but there is no formal rule. K-fold cross validation has low bias but generally has high variance compared to other methods. Repeating k fold cross validation can be used to efficiently increase the precision of the estimates while still maintaining a small bias (Kuhn 70).

The *caret* package allows tuning of the hyperparameters of the models. To evaluate the models in training and testing a suitable performance metric needs to be defined. Mostly predictive accuracy is used but might not be appropriate when the data is imbalanced because a simple default strategy of guessing the majority class would give a high predictive accuracy without considering the minority class. For imbalanced datasets the binary classification measures Area Under the ROC Curve (AUC) and Cohen's Kappa are recommended. AUC summarizes the plot of true positive rate against false positive rate (the ROC curve) in a single value (Chawla 855). Kappa is the percentage of correctly classified instances out of all instances normalized at the baseline of random chance on the dataset (Brownlee). All models were trained and validated twice, once with AUC and once with Kappa as performance measure. The decision tree trained with AUC scored about 1% higher on the test set on absolute savings (see Validation savings). The other three models had no difference in performance if trained with Kappa or AUC.[1] Thus, AUC was selected for all models.

Feature Engineering

The raw data contains typical raw credit card fraud detection features for each transaction such as amount, date and time, merchant type (e.g. gas station), entry mode, among others (Table 1). Just with those attributes, fraud may be identified at the transactional level. However, a single transaction is not enough to detect a fraudulent transaction since it leaves behind the customer spending behavior. To address this issue, Whitrow et al. propose to perform transaction aggregation in order to create aggregated features (31-51).

Creating the aggregated features consists in grouping the transactions made during the last given number of hours by card number (*tokenized_pan*), followed by calculating the number of transactions (*cnt_* feature) and the total amount spent on those transactions ("sum_" feature) for every transaction within the time window. We processed those new attributes for time windows of 1 day, 2 days, 1 week and 30 days, respectively, resulting in 8 new features for the model. When selecting the transactions to calculate the new features, we took two assumptions. First the own transaction is not considered as past behavior is modeled. Second the transactions must be non-fraudulent as normal customer behavior is modeled. Let us exemplify these new features for 1-day and 7-day time windows. Table 2 summarizes four transactions associated with two different card numbers (identified as 132 and 49). The first two columns (*tokenized_pan* and *datetime*) are used to compute the number of transactions within one day (*cnt_1d*) and within seven days (*cnt_7d*) time windows. The *amount* combined with *cnt_1d* and *cnt_7d* is used to compute the *sum_1d* and *sum_7d* attributes. In this example the first row has a value of 0 in every aggregated feature, because it is the first transaction and consequently there're no other transactions in its time window. The same is with the transaction of card number 49 since it is the only one for that card in this example. However, on the second and third transaction of card number 132 it can be seen how the quantity of

[1] Since that seemed strange to us, we asked the TA four days before final submission about this issue. Unfortunately, we got a reply only on the day of the final submission and were not able to correct our error. The error was that we used predicted probabilities (*classProbs = TRUE*) instead of predicted values (0 and 1).

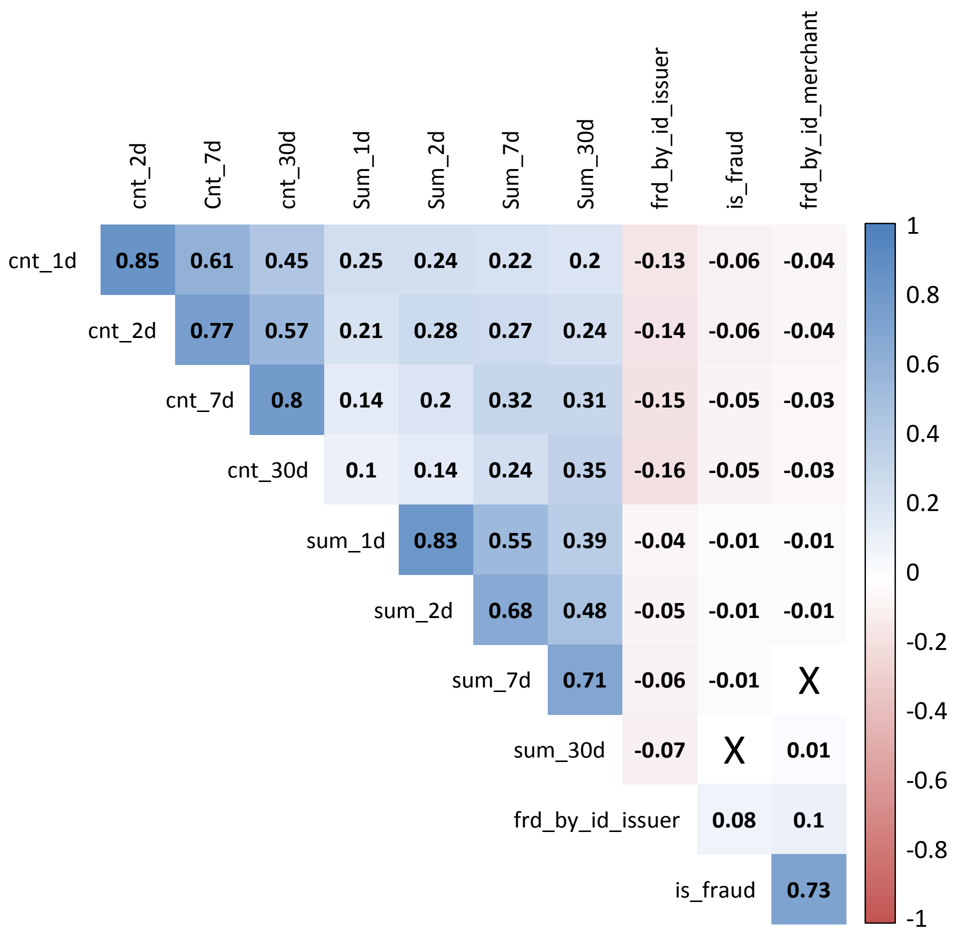
transactions for the same card within the defined number of days (*cnt_1d* and *cnt_7d*) and the sum of the amount of these transactions (*sum_1d* and *sum_7d*) are calculated.

tokenized_pan	**datetime**	**amount**	**cnt_1d**	**sum_1d**	**cnt_7d**	**sum_7d**
132	2015-05-21 23:55	141,99	-	-	-	-
132	2015-05-24 09:41	6,00	-	-	1	141,99
49	2015-05-24 09:57	38,20	-	-	-	-
132	2015-05-25 08:23	6,00	1	6,00	2	147,99

Table 2: Illustrative calculation of aggegated features cnt_1d, sum_1d, cnt_7d and sum_7d

In addition to the previously mentioned features, we added two more features, indicating fraud indexes by issuer bank (*id_issuer*) and by merchant (*id_merchant*), respectively. The *frd_by_Id_issuer* feature is the ratio of the number of frauds for each bank and overall frauds and *frd_by_id_merchant* is the ratio of the number of frauds for each merchant and overall frauds. The correlation matrix of all engineered features and *is_fraud* reveals a high positive correlation (0.73) between *is_fraud* and *frd_by_id_merchant* (Figure 3). This valuable feature indicates that certain merchants are associated with fraudulent activity.

It is worth noting that the feature engineering on the *new* dataset should be done before the dataset is split into 70% training and 30% test. This ensures that all available information is used to model past behavior of the customers. If we did the aggregates features after the split we would have two incomplete descriptions of past customer behavior.



	cnt_2d	Cnt_7d	cnt_30d	Sum_1d	Sum_2d	Sum_7d	Sum_30d	frd_by_id_issuer	is_fraud	frd_by_id_merchant
cnt_1d	0.85	0.61	0.45	0.25	0.24	0.22	0.2	-0.13	-0.06	-0.04
cnt_2d		0.77	0.57	0.21	0.28	0.27	0.24	-0.14	-0.06	-0.04
cnt_7d			0.8	0.14	0.2	0.32	0.31	-0.15	-0.05	-0.03
cnt_30d				0.1	0.14	0.24	0.35	-0.16	-0.05	-0.03
sum_1d					0.83	0.55	0.39	-0.04	-0.01	-0.01
sum_2d						0.68	0.48	-0.05	-0.01	-0.01
sum_7d							0.71	-0.06	-0.01	X
sum_30d								-0.07	X	0.01
frd_by_id_issuer									0.08	0.1
is_fraud										0.73

Figure 3: Correlation matrix of all engineered features

Validation

The models performance is evaluated using the standard binary classification measures Area Under the Curve (AUC) and Kappa. In addition, a custom cost-based metric (savings) is used for evaluation. AUC and Kappa may not be the right evaluation criteria when evaluating fraud detection models because they implicitly assume that misclassification errors carry the same cost as the correct classified transactions. In practice, wrongly predicting a fraudulent transaction as legitimate usually carries a considerably higher financial cost than the opposite case (Bahnsen 136-137). The goal of companies, when it comes to fraud detection, is to take a decision to minimize the losses. Using a cost matrix that defines the cost for both types of misclassification error, a savings metric can be computed as the difference between the cost of using no algorithm (sum of the amounts of fraudulent transactions) and the associated cost of the predictions (Figure 4). In this experiment the highest savings of USD 561712.9 are achieved if all frauds are detected on the test set while having zero false positives. The lowest savings is USD 0. This is when no model is used at all.

	Predicted Negative	Predicted Positive
Actual Negative	*USD 0*	*USD 20*
Actual Positive	*Total transaction amount*	*USD 20*

Figure 4: Cost matrix

Results and validation of analysis

All four models were implemented in R using the *caret* package for training and validation with 10-fold cross validation and 5 repeats and for tuning of the hyperparameters. Logistic Regression does not require any hyperparameters. When building the model the predictor issuer bank, *id_issuer*, was removed since it was highly correlated with the newly created feature fraud ratio by issuer bank, *frd_by_Id_issuer*, and did not improve the model. It is the worst performing model out of the four models with an AUC of 0.78, Kappa of 0.66 and total savings of USD 412,802 (Table 3). The Decision Tree can be tuned on a complexity parameter (cp) that defines how much of the tree is retained to prevent overfitting (Thernau 12, Kuhn 177). The optimal value of *cp* is 0.0002 (Figure 5). The Decision Tree has an AUC of 0.82, Kappa of 0.71 and total savings of USD 447,750. The Random Forest also has one hyperparameter, the number of variables randomly sampled as candidates at each split called *mtry* (Liaw). Here *mtry* of 13 was selected as the optimal value (Figure 6). The Random Forest scored highest on all performance metrics with an AUC of 0.85, a Kappa of 0.79 and savings in USD 471,249. The XGBoost model has a total of seven hyperparameters that can be tuned with the *caret* package. This are the maximum number of iterations (nrounds), the maximum depth of a tree (max_depth), the step size shrinkage used in update to prevents overfitting (eta), the minimum loss reduction required to make a further partition on a leaf node of the tree (gamma), the subsample ratio of columns when constructing each tree (colsample_bytree), the minimum sum of instance weight (hessian) needed in a child (min_child_weight) and the subsample ratio of the training instance (subsample)(Chen). The values selected are 100 for *nrounds*, 20 for *max_depth*, 0.3 for eta, 0 for *gamma*, 0.5 for *colsample_bytree*, 1 for *min_child_weight* and 1 for *subsample* (Figure 7). The tuned XGBoost has an AUC of 0.82, Kappa of 78 and savings in USD of 452741.

Method	Package	Hyperparameter	Selection	AUC	Kappa	Savings in USD
Logistic Regression	stats	-	-	0.7812106	0.665	412,802.6
Decision Tree	rpart	cp	0.0002	0.8223049	0.711	447,750.5
Random Forest	random-Forest	mtry	13	0.8573076	0.798	471,249.5
XGBoost	xgboost	nrounds max_depth eta gamma colsample_bytree min_child_weight subsample	100 20 0.3 0 0.5 1 1	0.8283171	0.783	452,741.0

Note: cross validation was repeated k-fold (5 repeats of 10-fold CV for all models) using the caret package

Table 3: Hyperparameters and validation results of models

Conclusion

All the four binary classification models have satisfying results since they generate positive savings that range from USD 412,802 for Logistic Regression model to USD 471,249 for Random Forest. The best-performant model, Random Forest, reaches 84% of the maximum possible savings of USD 561,712.

However, fraud detection wasn't an easy problem to tackle. One issue in this project was the strong imbalance between fraudulent and legitimate transactions on the original dataset with a fraud rate of 0.031%. Although we under-sampled the dataset we had a fraud ratio of only 2.33%, what is still a strong imbalance between the classes. We mitigated this issue, in the training and validation stages, by using performance measures that are well-known to perform properly for imbalanced classification (such as ROC and Kappa). Another issue was the large size of the training and validation dataset: 366,134 observations with 12 raw features and others 10 aggregated features. This made the modelling computational intense. Thus, we could not try the whole range of values for the hyper-parameter tuning and tried only some of them. Lastly, we had some issues with the gradient boosted trees' model (provided by the XGBoost library). We had expected it to be the best-performant but the results were relatively moderate. XGBoost performed really well on training and validation but was poor in testing, what generally indicates overfitting. One reason could be that we tried the wrong values of hyper-paramters. However, by only choosing two different values for every of the seven hyper-parameters we get 128 different combinations that need to be tested. Next, XGBoost has a parameter to control the balance of positive and negative weights (*scale_pos_weight*) that is useful for unbalanced classes. Although setting the parameter as recommended the model still performed poor.

Although we got satisfying results in this experiment we are aware that additional steps are needed to implement this solution in practice at the company's production environment. This includes the development of some interfaces to ease the data handling, so the data can be transferred from the company's databases or data-warehouse to R. In addition to this, since credit and debit cards are potentially used in a 24/7 basis, the aggregated features need to be computed on the fly for every new transaction and a new prediction needs to be performed on this transaction to classify it as fraudulent or legitimate.

Finally, we identified some improvements that could be done in order to make the most of the models. The models could be trained with a custom performance metric: ultimately, we want to save as much money as possible with our model and thus the cost metric *savings* used to evaluate the models could also be used for training the models. The *caret* package allows training the models with a custom function over the argument *summaryFunction* that specifies a function for computing performance. Here the user can define its own performance metric. Also, more aggregated features could be computed in order to increment the information available for the model at a transactional level. Correa Bahnsen et al proposed a time confidence interval for the usual spending time of each customer represented by a dummy variable.

References

Bahnsen, Alejandro Correa, et al. “Feature engineering strategies for credit card fraud detection.” *Expert Systems with Applications*, vol. 51, 2016, pp. 134–142.

Brownlee, Jason. “Machine Learning Evaluation Metrics in R.” *machinelearningmastery,* 29 Feb. 2016. *machinelearningmastery.com/machine-learning-evaluation-metrics-in-r/.* Accessed 9 Dec. 2017.

Chen, Tianqi et al. “xgboost: Extreme Gradient Boosting. R package version” *cran.r-project,* 2017. *CRAN.R-project.org/package=xgboost.* Accessed 11 Dec. 2017.

Chawla, Nitesh V. “Data Mining for Imbalanced Datasets: An Overview.” *Maimon O., Rokach L. (eds) Data Mining and Knowledge Discovery Handbook.* Springer, Boston, MA, 2005.

Gorman, Ben. “A Kaggle Master Explains Gradient Boosting”. *blog.kaggle,* No Free Hunch, 23 Jan. 2017.*http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/.* Accessed 9 Dec. 2017.

Jain, Aarshay. “Complete Guide to Parameter Tuning in XGBoost (with codes in Python)”. *Analytics Vidhya,* 1 Mar. 2016, *https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.* Accessed 14 Dec. 2017.

Krivko ,M. “A hybrid model for plastic card fraud detection systems.” *Expert Systems with Applications*, vol. 37, 2010, pp. 6070–6076.

Kuhn, Max, and Johnson, Kjell. “Applied Predictive Modeling”. Springer, New York, 2013

Liaw A et al.” Classification and Regression by randomForest.” *R News* vol.2/3,2002, pp.18-22.

Maes, Sam et al. “Credit Card Fraud Detection Using Bayesian and Neural Networks”. *Proceedings of NF*, 2002.

Mahmoudi, Nader, et al. “Detecting credit card fraud by Modified Fisher Discriminant Analysis.” *Expert Systems with Applications,* vol. 42, 2015, pp. 2510–2516.

Prati, R.C. et al. “Class imbalance revisited: a new experimental setup to assess the performance of treatments methods.” *Knowledge and Information Systems*, vol. 45, 2015, pp. 247–270.

“The Nilson Report.” David Robertson, 17 Oct. 2016, www.nilsonreport.com/upload/content_promo/The_Nilson_Report_10-17-2016.pdf. Accessed 02 Dec. 2017.

Therneau, Therry M et al. “An Introduction to Recursive Partitioning Using the RPART Routines” *cran.r-project,* 12 March 2017. *cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf.* Accessed 11 Dec. 2017.

Visa, Sofia. “Issues in mining imbalanced data sets – a review paper.” Proc. 16th Midwest Artificial Intelligence and Cognitive Science Conference, 2005, pp. 67–73.

Withrow, C. “Transaction aggregation as a strategy for credit card fraud detection”. Data Mining and Knowledge Discovery, vol. 18, 2009, pp. 30-55.

Appendix

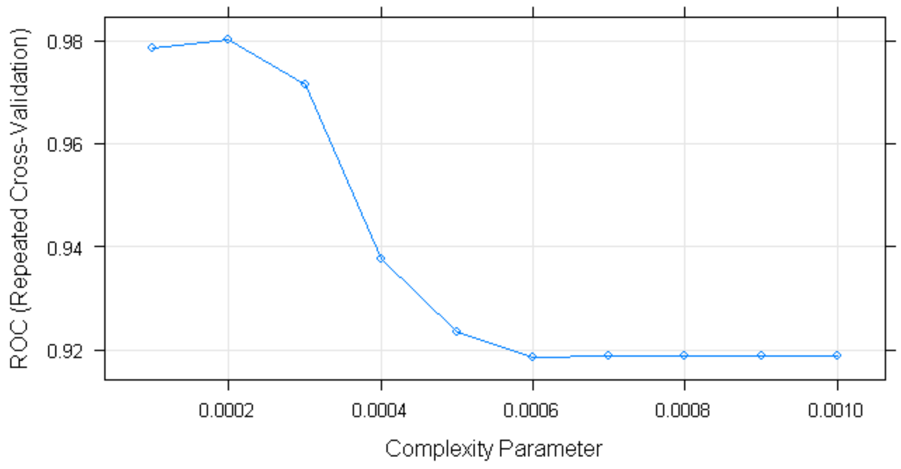


Figure 5: Hyperparameter tuning for decision tree

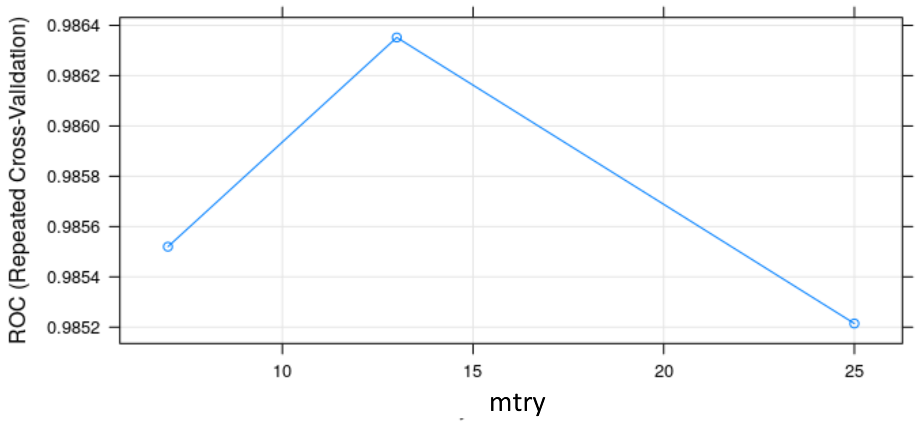


Figure 6: Hyperparameter tuning for Random Forest

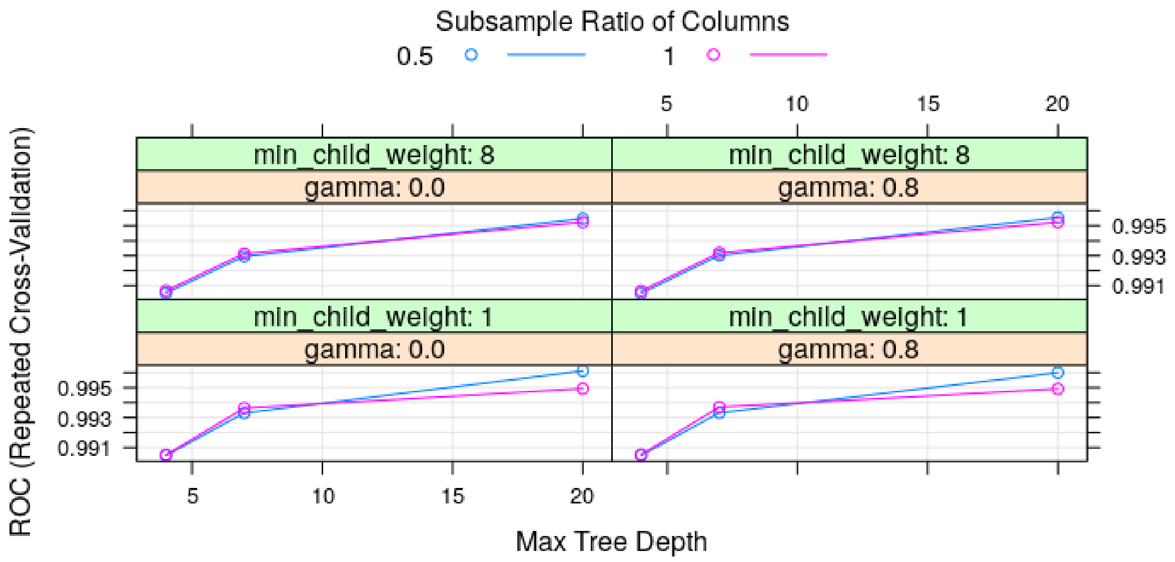


Figure 7: Hyperparamter tunig for XGBoost

```
>tuneGrid <- expand.grid(.mtry = c(7, 13, 25))
>
>fitControl <- trainControl(method = "repeatedcv",
>                           number = 10,
>                           repeats = 5,
>                           verboseIter = TRUE,
>                           classProbs = TRUE,
>                           summaryFunction = twoClassSummary)
>
>rf_model <- train(is_fraud~.,data=train,
>                method ="rf",
>                trControl = fitControl,
>                tuneGrid = tuneGrid,
>                ntree = 50,
>                metric = "ROC")
```

Figure 8: R code of the winning model Random Forest

```
>ComputeSavings <- function(amounts, pred.values, true.values) {
>  predictions <- data.frame(amounts, pred.values, true.values)
>
>  costs <- 0
>  for (i in 1:nrow(predictions)) {
>    pred.value <- predictions$pred.values[i]
>    true.value <- predictions$true.values[i]
>
>    if (pred.value == 1) {
>      costs <- costs + 20
>    } else if (pred.value == 0 & true.value == 1) {
>      costs <- costs + predictions$amount[i]
>    }
>  }
>
>  savings <- sum(predictions$amounts[predictions$true.values ==
>                 1]) - costs
>
>  return(savings)
>}
```

Figure 9: R code of the custom metric to calculate savings

```
># Time windows
>name <- c("1d", "2d", "7d", "30d")
> secs <- c(60 * 60 * 24,       # 1 day
>          60 * 60 * 24 * 2,   # 2 days
>          60 * 60 * 24 * 7,   # 7 days
>          60 * 60 * 24 * 30)  # 30 days
>time.windows <- data.frame(name, secs)
>
>clients <- unique(card.fraud$tokenized_pan)
>
>for (i in 1:nrow(time.windows)) {
>  tw <- time.windows[i, ]
>  card.fraud$Id <- seq.int(nrow(card.fraud))
>
>  cnt.attr.name <- paste("cnt_", tw$name, sep = "")
>  sum.attr.name <- paste("sum_", tw$name, sep = "")
>
>  card.fraud[, cnt.attr.name] <- NA
>  card.fraud[, sum.attr.name] <- NA
>
>  for (client in clients) {
>    client.trxs <- card.fraud[card.fraud$tokenized_pan == client,
>]
>
>    for (j in 1:nrow(client.trxs)) {
>      trx <- client.trxs[j, ]
>      current.datetime <- trx$datetime
>      related.trxs <- client.trxs[client.trxs$datetime >
>current.datetime - tw$secs &
>                                    client.trxs$datetime <=
>current.datetime  &
>                                    client.trxs$Id != trx$Id &
>                                    client.trxs$is_fraud == 0, ]
>
>      card.fraud[card.fraud$Id == trx$Id, cnt.attr.name] <-
>nrow(related.trxs)
>      card.fraud[card.fraud$Id == trx$Id, sum.attr.name] <-
>sum(related.trxs$amount)
>    }
>  }
>
>  card.fraud <- card.fraud[, !names(card.fraud) %in% c("Id")]
>}
>
>#fraud ratio per id_issuer
>issuers <- unique(card.fraud$id_issuer)
>card.fraud[, "frd_by_id_issuer"] <- NA
>for (issuer in issuers) {
>  issuer.frauds <- nrow(card.fraud[card.fraud$is_fraud == 1 &
>card.fraud$id_issuer == issuer, ])
>  issuer.count <- nrow(card.fraud[card.fraud$id_issuer == issuer,
>])
>  card.fraud$frd_by_id_issuer[card.fraud$id_issuer == issuer] <-
>issuer.frauds/issuer.count
>}
>
>#fraud ratio per merchant
```

```
>merchants <- unique(card.fraud$id_merchant)
>card.fraud[, "frd_by_id_merchant"] <- NA
>for (merchant in merchants) {
>  merchant.frauds <- nrow(card.fraud[card.fraud$is_fraud == 1 &
>card.fraud$id_merchant == merchant, ])
>  merchant.count <- nrow(card.fraud[card.fraud$id_merchant ==
>merchant, ])
>  card.fraud$frd_by_id_merchant[card.fraud$id_merchant ==
>merchant] <- merchant.frauds/merchant.count
>}
```

Figure 10: R code to create aggregated features