Fraud Detection: A Data Science Approach

Thunradee Tangsupakij, Geeratigan Arsanathong, Yu-Chieh Wang

***Abstract*— We conduct an analysis of fraud detection stemming from the Kaggle competition in 2019. Using a mix of modern exploratory data analysis techniques along with predictive models such as Logistic Regression and Support Vector Machines, we were able to accurately conduct fraud detection. Along the way, we developed a possibly unique method for dealing with missing values, as well as created a simplistic metric for model comparison. Finally, in an attempt to model the problem from the perspective of time-series, our novel Sliding Window Decision Tree Ensemble approach significantly outperformed existing popular learning algorithms on this data-set.**

I. INTRODUCTION

Throughout history, humans have made numerous attempts to commit fraud in order to gain financial or similar advantages at the expense of others. Typically, fraudulent transactions involve someone either physically stealing a credit/debit card, or stealing enough of a card holder's identity in order to fraudulently use the victim's credit in order to make transactions. As time goes on, methods used in conducting fraud are only becoming larger, efficient and more sophisticated, leading to demand for new methods in fraud detection [1].

During the past decade, many financial institutions have began utilizing the power of Artificial Intelligence (AI) and Machine Learning (ML) techniques in order to aid in detecting fraudulent transactions. According to a study conducted by the Association of Certified Fraud Examiners, while only 13% of organizations currently use AI and ML, 25% plan to adopt these techniques within the next two years [1]. In fact, the market for fraud detection and prevention is expected to exceed 80$ billion by 2025, thus becoming one of the largest industries demanding new AI and ML techniques [2]).

This paper presents a concrete data-science approach to the Fraud Detection problem as outlined in the Kaggel competition. On top of using a mix of powerful methods commonly used in the field, *we also develop our own novel methods throughout*. In particular, we develop methods as well as test them alongside traditional methods whenever it is appropriate in our attempt to properly solve the fraud detection problem.

This article is broken into four major components. In section 2, we conduct extensive Exploratory Data Analysis (EDA) on the IEEE-CIS fraud detection dataset. This analysis begins with describing the data-set with basic statistics. This immediately leads us to our first hurdle: how do we manage the rampant amount of missing values across all features in our data-set? **After attempting both simplistic and global approaches, our team came up with a creative idea for what we believe to be an optimal solution to the problem**. This is followed by a brief comparison of the method with an overly-simplified linear regression approach in an attempt to highlight predictive accuracy gained by the method. Having implemented our missing value-filling approach, we begin feature pruning and feature engineering. In particular, by developing a careful understanding of the data-set, we create three features as combinations or transformations of existing features that we believe will be helpful in correctly classifying fraud. Finally, by noting our computational power limitations, we create measures of model accuracy that we believe we can use to accurately compare predictive models.

Section 3 is our Confirmatory Data Analysis (CDA) stage, where we test predictive algorithms on the transformed data-set obtained from our analysis outlined in section 2. We present our fraud detection accuracy results using logistic regression, multiple linear regression, support vector machines and Naive Bayes algorithms on a given testing set. For each model, we briefly describe both the strengths and the limitations behind using these approaches. Notably, we discover many of these models to be far from accurate, motivating us to

look for a new approach.

Having conducted our section 3, all while feeling unsatisfied with our results, we choose to take an even deeper look into our problem. After a long discussion around how we may find stronger methods, we began to suspect that there may be a strong time-series based dependency in our data-set. We loosely elucidate on this idea by developing a simple test for time-series based dependency, leading us to develop a new approach that we deem to be our "novel" approach. Finally, we test this approach extensively across our measures of model accuracy. Having conducted a thorough testing of the Kaggle 2019 competition data-set using multiple prediction algorithms, we conclude this research article with a final model comparison and an outlook to possible future research avenues.

II. Exploratory Data Analysis

A. Data Wrangling

In the first step, we gather the IEEE-CIS Fraud Detection dataset from Kaggle. There are two datasets, which are identity and transaction data set. Both datasets contain an ample number of features which we found some engaging describes as follows: 1.) Transaction data: This is a historical online transaction data that presents 590,540 transactions and 394 columns. We have found that there are 11,318 frauds out of 590,540 transactions or about 7.85% of the data, which leads to the problem of unbalanced data. Also, there are several features we found relevant such as transaction time, transaction payment amount, payment, payment card information, and the email domain. The data-set also contains many features without the description, but we believe that there might be useful for our model. 2.) Identity Data: The identity data-set contains only 144,233 transactions with 41 columns. As the description, all features (id_01 to id_37) present the network connection information or the digital user agent associated with the transactions. These have been collected by Vesta's fraud protection system and digital security partners, which we are not given.

Having merged our data-sets into one combined feature-set, we immediately notice the extensive

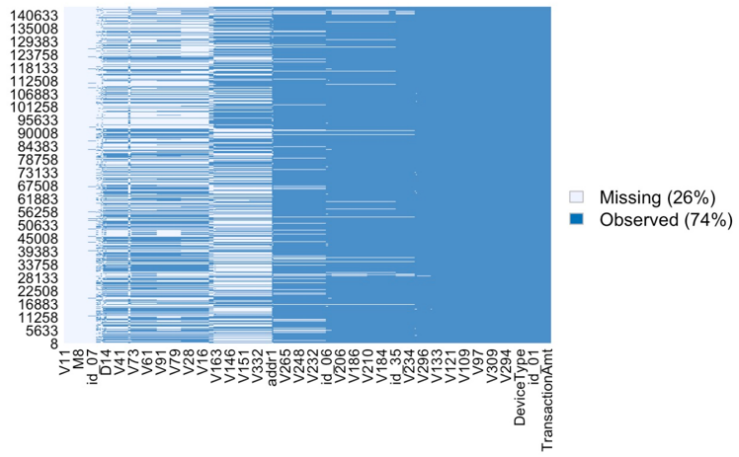

Fig. 1. Depiction of the missing values in our data-set

amount of missing data prevalent throughout all of our data-set's features. To make matters worse, the missing values rarely coincide across multiple features - forcing us to look for ways to handle this missing value problem. To illustrate this issue, figure 1 depicts areas where we have missing values within a slice of 140,063 rows of our data-set.

Our first approach to handle the missing values is to fill the missing values with the average (in the case of a quantitative variable) of the values in the same column tied to each individual account identification, or the mode (in the case of a qualitative variable) of the values in the same column tied to each individual account. By using means (modes) confined to each individual account, we are better able to localize our estimates for the missing values NA in each position in comparison to using a global mean (mode) approach.

Despite this idea working in many practical problems in the field of data science, it is also known that averaging to replace missing values in high-dependency data situations can easily lead to issues of destroying inherent relationships between variables. In our case, this issue can be amplified by the fact that our fraudulent cases (corresponding to the variable "isFraud" taking the value 1) are extremely infrequent in comparison to our not-fraudulent cases (corresponding to the variable "isFraud" taking the value 0).

As a second approach, we chose to simply delete all features in the column space of our data-set that contain missing values. Using this approach, we have the advantage of avoiding the issue of ruining the relationships inherent in our data-set. At same time, we suffer from the disadvantage of deleting a high number of features. Using this approach, of the original 426 columns that we started with, we ended up having only 30 columns remaining.

B. Novel Method for Missing Observations

To deal with the cumbersome task of handling the missing values in our data-set, we developed the unique idea of deleting the optimal number of rows and columns such that our defined measure "Matrix Area", denoted **MArea**, is maximized. This measure is abstractly defined as the following:

$$MArea := max_{n,m}\{nm | \#NA = 0\} \qquad (1)$$

Where $\#NA$ denotes the number count of missing values "NA" in our dataset. The inspiration for this measure comes from the idea of maximizing the area of a rectangle: by deleting the appropriate combination of rows and columns of our data matrix such that the "base" (column space) multiplied by our "height" (row space) is maximized conditioned on there being no missing values (NA) in our data-set. In order to create our reduced matrix that maximizes **MArea**, we developed a simple algorithm with logic as follows:

- Initialize with i=1
- Delete the first i columns detected to contain a row with a missing value
- Scan over all remaining rows and delete any row containing a missing value
- Output the value $n_i m_i$
- Repeat until $(n_i m_i) - (n_{i-1} m_{i-1})$ is negative

We found that dropping columns that contain greater than or equal to 11% of missing values outputs the highest **MArea**. This corresponds to deleting 182 columns and 19,952 rows from our initial data-set. In such rows were 1,605 frauds and 18,347 not frauds. Therefore, from our initial data-set containing 144,233 rows (7.85% of which were fraud with 4,456,006 N/A values), we ended up having 124,281 rows, 7.82% of which are fraud, 237 columns and 0 missing values. With only a minor change in the percentage of missing values ($.03\%$), we believe our new data-set is a good representation of the original data-set.

	Drop All Columns Contain NA	Impute By Card Number Modes And Global Modes	Novel Approach
Recall	NA	0.16	0.19
Precision	NA	0.76	0.67
Accuracy (%)	NA	97.43	96.73
Bias	NA	0.22	0.3

Fig. 2. Recall, Precision and Overall Accuracy as obtained from using a multiple linear regression for each of the three methods. Note: the method corresponding to "Drop All Columns Contain N/A" returned "NA" due to never predicting isFraud=1

C. Brief Comparison of Missing Value Methods

In order to judge which of the previous methods was most effective, while simultaneously dealing with constraints in our available computational time and power, we chose to run a simple linear regression over all three data-sets corresponding to the three methods outlined above. We understand that a simple linear regression is no-where near the optimal model to use when dealing with a binary response variable; however, we believe that it works purely for use in comparing the three methods. Figure 2 depicts the Recall, Precision and Accuracy (definitions in section IV.A) across each of the three methods using a multiple linear regression.

The results of Figure 2 can be deceiving. At first glance, we may be inclined to choose the method corresponding to imputing the missing values by the modes corresponding to each individual identification (card number). This is due to the method having a higher Precision (75%) and overall accuracy (97.43%) relative to our novel approach's 66.67% and 96.73%, respectively. However, our novel approach managed to have an advantage in recall (19.31%) in comparison to

the standard imputation method (16.38%) - a very important measure in the case of fraud detection (discussion on this aspect can be found in section IV.A). On top of this, our novel approach leads to a highly reduced data-set in comparison to the original, giving the method a clear advantage in terms of computational complexity for our predictive models. With these two facts in mind, we opt to choose the reduced data-set created from our novel approach as the data-set we will use in the next sections.

D. Feature Reduction and Engineering

Having finished pre-processing and dealing with missing value issues in our data-set, we now chose to look for variables that may be redundant in our data-set. To begin, we first looked at the variables C1 through C14, noted as being "similar identifying count values" in the competition. We emphasize here that the supplier of the data-set did not give any other specific information regarding what any of the features referred to in the data-set due to privacy issues. With the knowledge that the variables C1 through C14 are quantitative variables that may be closely related, we computed a correlation matrix between these variables. A table of the correlation matrix is depicted in Figure 3. In the figure, we highlighted variables that had an extremely high correlation (correlation > .9). Due to this redundancy in information, we deemed the variables C2, C4, C6, C7, C8, C9, C10, C11, C12 and C14 should be removed from the data-set.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
C1	1.00	1.00	0.00	0.97	0.17	0.98	0.93	0.97	0.18	0.96	1.00	0.93	0.77	0.95
C2	1.00	1.00	0.00	0.97	0.13	0.97	0.94	0.98	0.13	0.97	0.99	0.94	0.75	0.94
C3	0.00	0.00	1.00	0.00	-0.01	0.00	0.00	0.00	-0.01	0.00	0.00	0.00	-0.01	-0.01
C4	0.97	0.97	0.00	1.00	-0.01	0.96	0.90	0.96	-0.02	0.95	0.97	0.89	0.64	0.91
C5	0.17	0.13	-0.01	-0.01	1.00	0.23	-0.01	-0.01	0.93	-0.01	0.17	-0.01	0.72	0.38
C6	0.98	0.97	0.00	0.96	0.23	1.00	0.86	0.92	0.25	0.91	0.99	0.86	0.81	0.98
C7	0.93	0.94	0.00	0.90	-0.01	0.86	1.00	0.98	-0.01	0.99	0.92	1.00	0.63	0.79
C8	0.97	0.98	0.00	0.96	-0.01	0.92	0.98	1.00	-0.01	1.00	0.96	0.98	0.65	0.86
C9	0.18	0.13	-0.01	-0.02	0.93	0.25	-0.01	-0.01	1.00	-0.01	0.18	-0.01	0.70	0.40
C10	0.96	0.97	0.00	0.95	-0.01	0.91	0.99	1.00	-0.01	1.00	0.96	0.98	0.65	0.85
C11	1.00	0.99	0.00	0.97	0.17	0.99	0.92	0.96	0.18	0.96	1.00	0.92	0.78	0.96
C12	0.93	0.94	0.00	0.89	-0.01	0.86	1.00	0.98	-0.01	0.98	0.92	1.00	0.63	0.79
C13	0.77	0.75	-0.01	0.64	0.72	0.81	0.63	0.65	0.70	0.65	0.78	0.63	1.00	0.88
C14	0.95	0.94	-0.01	0.91	0.38	0.98	0.79	0.86	0.40	0.85	0.96	0.79	0.88	1.00

Fig. 3. Correlation matrix of the variables C1 - C14

Next, upon inspection of the remaining variables, we used the information we did have about a limited number of the variables to come up with new ones. The first such variable that we created is one that we called "velocity". In our data-set, we had a variable denoted as "DT", and another denoted as "Distance". Upon reading the information provided in the competition, it seemed that "DT" corresponded to "change in time", while "Distance" corresponded to the distance between the places where each transaction was taken place. Therefore, we defined our variable "velocity" simply as the following:

$$velocity := (DT)\,x\,(Distance) \tag{2}$$

The motivation behind this variable is simple: naturally, we don't expect to see someone make a transaction in Pullman, Washington and a minute later make one in Bangkok, Thailand. When this variable is high *relative* to when either DT or Distance is small, this would indicate a person would have had to go faster than is physically expected of that individual. In other words, we believe the *interaction* effect between DT and Distance can be a significant indicator of a fraudulent transaction.

The next variable that we created involved a transformation on the variables "P_emaildomain" and "R_emaildomain" denoted the "purchaser's" emailing address and the "reciever's" emailing address. Upon inspecting these variables, we noticed that often times the email addresses ended with a common ".com" or ".net", while occasionally ending with a country-specific prefix like ".uk". With consideration towards this data-set being based out of the United States, we felt

it was safe to assume that consumers would very rarely have transactions going through other countries. At the same time, we felt that it could be possible that fraudulant transactions would be routed through other countries. Therefore, out of these two categorical variables, we created two new ones as follows:

$$P\_dotCom_i := 1 | P\_emaildomain_i = 1\,;\ 0\ o.w. \tag{3}$$

$$R\_dotCom_i := 1 | R\_emaildomain_i = 1\,;\ \ 0\ o.w. \tag{4}$$

Despite having a couple other ideas in regards to variable creation, in the end we deemed our other ideas as being either infeasible due to missing value issues as outlined in the previous subsection, or simply redundant to another variable already included in the data-set. At this point, we have concluded our Exploratory Data Analysis stage and choose to begin testing models on our data-set.

III. Confirmatory Data Analysis

For our Confirmatory Data Analysis, we begin by defining which models we will choose to use for our problem. In particular, noting that our response variable "isFraud" is a binary qualitative variable, we instantly reduce our feasible model choices down to classification approaches. On top of this, the vast majority of variables contained in our data-set are qualitative, ruling out some modelling frameworks that assume quantitative or Gaussian independent variables. Thus, we chose to initially try three popular binary classification methods: Logistic Regression, Support Vector Machine (SVM) and Naive Bayes. In each of the models we use on our data-set, we will critique them by their performance as measured by our performance metrics, which we define in the next section.

A. Performance Metrics

In the field of Fraud Detection, there is a smaller emphasis on overall accuracy of a model than what is typical in machine learning problems. This is due to the importance of accurately predicting when a transaction is fraudulent: there is a much heavier loss to the money lender when a fraudulent transaction is classified as "not-fraud" in comparison to when a non-fraudulent transaction is classified as "Fraud". This is precisely why we choose to put a heavier emphasis on the performance metric "Recall", defined as follows:

$$Recall := \frac{\#TruePositives}{\#TruePositives + \#FalseNegatives} \tag{5}$$

Our next performance metric is a standard one used in classification problems known as precision, defined as the following:

$$Precision := \frac{\#TruePositives}{\#TruePositives + \#FalsePositives} \tag{6}$$

On top of this, the occurrence of a fraudulent transaction is much more rare in comparison to non-fraudulent transactions. This issue falls under the area in data science known as the skewed data problem. In short, due to the rare occurrence of the minority class in a skewed-data problem, a machine learning model is commonly biased in predicting the majority class even more frequently than the majority class occurs. In fact, it is even common for a learning algorithm to simply choose the majority class as it's prediction for every instance, even when the number of instances falling under the minority class can be as high as 10% of the dataset. Therefore, we invented a simple metric that we could use to properly evaluate our models in regards to this issue. We denoted this variable as "Bias", and is defined as the following:

$$Bias := \frac{\#PredictedPositives}{\#Positives} \tag{7}$$

With the way we have defined it, the best score a model can achieve for our metric "Bias" is 1. Finally, we will include the metric "overall accuracy" in evaluating our models. We note that the reasoning behind choosing these simple metrics is also in part due to computational complexity

of computing many other popular metrics such as Area Under the Curve (AUC). To properly use this metric, it would require us to run our models over this large data-set too many times, leading to a nightmare in terms of computational complexity.

B. Logistic Regression

In choosing Logistic Regression as one of our methods, we then needed to choose a method for selection of the predictor variables. One concrete method popular in the literature is using *Backwards Selection*. In essence, we began by running a logistic regression on our reduced data-set obtained from our approach outlined in section II.B. After reviewing the model summary, we removed the predictor variable corresponding to the variable indicated as having the highest p-value. Having removed said variable, we repeat this process again - over 100 times in fact. Admittedly, this is probably not something we would choose to do again, as there is most likely a much more efficient method of variable selection for Logistic Regression when the number of variables is very high.

Having conducted our thorough backward selection process, afterward we decided to attempt to include interaction effects within our remaining variables. In particular, we attempted including interaction effects between C1, C3, C5 and C13 on the rest of the remaining variables, and subsequently repeated backward selection one more time to reduce our model to it's final state. An ANOVA table of our final model can be seen in figure 4.

In the ANOVA table, for the variables that we are capable of interpreting the coefficients, we believe they make good sense. For instance, we would expect that when the variable "TransactionAmt" is large, it is more likely to be a fraudulent transaction. This just follows common sense - if someone finds a way to steal money, one would expect them to attempt to steal as much as possible. Conversely, as noted in our variable engineering section (section II.D), we would expect that when the Purchaser's email address is a .com, the probability of a fraudulent charge to be lower than otherwise - justifying the negative coefficient pertaining to the variable "P_isDotCom". Performance metrics for our final logistic model can be viewed in Figure 5.

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-6.4899  -0.4210  -0.3240  -0.2456   8.4904

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)       -4.501e+00  3.318e-01 -13.564  < 2e-16 ***
TransactionDT      5.306e-08  2.305e-09  23.022  < 2e-16 ***
TransactionAmt     2.170e-03  8.915e-05  24.336  < 2e-16 ***
card6charge card  -8.291e+00  5.061e+01  -0.164 0.869854
card6credit        1.198e+00  3.294e-01   3.638 0.000275 ***
card6debit         7.028e-01  3.294e-01   2.134 0.032883 *
C1                -8.748e-03  4.246e-04 -20.603  < 2e-16 ***
C3                -5.315e+00  1.514e+00  -3.511 0.000446 ***
C13                1.637e-02  6.978e-04  23.461  < 2e-16 ***
M4M0               9.836e-01  3.982e-02  24.701  < 2e-16 ***
M4M1               7.508e-01  1.845e-01   4.070 4.71e-05 ***
M4M2               1.107e+00  2.598e-02  42.601  < 2e-16 ***
P_isDotCom        -1.055e-01  3.426e-02  -3.080 0.002070 **
TransactionDT:C3   2.164e-07  1.147e-07   1.886 0.059268 .
TransactionDT:C13 -1.807e-08  4.114e-10 -43.921  < 2e-16 ***
TransactionDT:C1   1.027e-08  2.332e-10  44.035  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Fig. 4. ANOVA table corresponding to our reduced model Logistic Regression

	Logistic Regression
Recall	0.49
Precision	0.41
Accuracy (%)	92.32
Bias	0.8

Fig. 5. Performance Metrics of the reduced logistic model

In our performance metrics for the logistic regression, we notice that our "Bias" metric returned a .8, indicating the model suffers from a bias in predicting a transaction to be fraudulent less often than what occurred in reality.

C. Support Vector Machine (SVM)

Support Vector Machines have become a very popular classification method in recent history. Despite the power of these models, they can also suffer from issues of convergence and high-computational complexity. In attempting to use

	SVM
Recall	0
Precision	0
Accuracy (%)	90.4
Bias	0.01

Fig. 6. Performance Metrics of the reduced logistic model

	Naive Bayes
Recall	0.56
Precision	0.4
Accuracy (%)	87.76
Bias	1.4

Fig. 7. Performance Metrics of the reduced logistic model

SVMs, we encountered numerous convergence issues. This led us to only being able to reliably compute an SVM using a linear kernal. Despite attempts to vary our cost parameter "C", we found cost=1 to be the most effective classifier. Figure 6 shows the result of our SVM in terms of our performance metrics:

One can quickly deduce that our SVM model performed very poorly as noted by our Bias metric. This tells us that the SVM predicted transactions to be non-fraudulant almost 100% of the time.

D. Naive Bayes

The Naive Bayes Algorithm is another commonly used classification method due to it's simplicity and low computational complexity. It's simplicity rests within it's over-arching assumption that each of the predictor variables are pair-wise independent, thus leading the Naive Bayes algorithm to estimate the conditional probability of each observation's class as the following:

$$P(C_k|x_1, ..., x_n) = P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \qquad (8)$$

Noting that $P(C_1)$ is very small (where 1 denotes fraud), it is clear that Naive Bayes should lead to poor classification. On top of this, it is highly unlikely that many of the predictor variables are pairwise independent - in fact, due to their paired-type dependency on each individual, it is provably an incorrect assumption. Nevertheless, we included it here to see how well it could perform. Figure 7 shows the table of the Naive Bayes classifier performance.

Upon inspection of the Naive Bayes performance by our metrics, we were quite surprised that it's "Bias" measure being greater than 1. In fact, we would have expected it to be near 0 due to the $P(C_1)$ prefactor as noted in the previous paragraph. Despite having a somewhat low overall accuracy, this model seems to be a better performing classifier than we had anticipated.

IV. A Novel Approach

In the previous section, we attempted to model our problem using Logistic Regressions, Support Vector Machines and Naive Bayes Classifiers. In all of these cases, we felt fairly unsatisfied with our results. In an attempt to more accurately deal with our problem, we decided to look into a more time-series based approach. To begin, we first attempted to estimate whether or not our problem can be considered a time series in the first place. In order to do this, *we needed a way to estimate whether or not the probabilities were dependent on time-ordering.* In a loose attempt to answer this question, we asked the question: what is the probability of a fraudulent transaction by a specific user, given that the previous transaction was fraudulent? If indeed the problem *is not* a time-series based dataset, we would expect the probability of a fraudulent transaction by a specific user, given that the previous transaction was fraudulent, to be roughly equal to the probability of a fraudulent transaction. Mathematically, this means:

$$P(C_i = 1|C_{i-1} = 1) \approx P(C_i = 1) = .0783 \quad (9)$$

However, in estimating our conditional probability, we found the following from our dataset:

$$P(C_i = 1|C_{i-1} = 1) \approx .4312 \quad (10)$$

In other words, our empiricle conditional probability of fraud given the previous transaction was fraud was approximately 43.12%. In our opinion, this easily justifies developing a time-series based approach to this problem.

A. *Sliding Window Decision Tree Ensemble*

Taking our inspiration from random forest techniques, we decided to try using a decision tree classifier that uses as training data-set a sliding window. In other words, we are using a decision tree to estimate the following probability:

$$P(C_i = 1|x_{i-1,\ldots,i-m\,;p}) \quad (11)$$

Where the value "m" denotes the length of the window. The idea here is simple: we train a model over the previous m rows, and predict the following value. We iterate this procedure over the entire data-set, retraining the model each time. Therefore, for a user-specified window size "m", we will re-train the model n-(m+1) times, giving us a total of n-m predictions, where "n" is the total number of rows in our dataset.

So far, we haven't done anything new: our model is simply a sliding window approach with it's base classifier being the basic decision tree. Methods such as these can be seen in a recent paper that uses similar approaches [3]. We generalize this concept one step further: instead of only considering 1-tree per iteration, we instead use multiple trees. In an attempt to prevent this model from getting too crazy in terms of computational complexity, we find a modest short-cut to the computations by simply storing each of the tree models trained in the previous windows. We then use each of the previous k models, each trained on the previous k windows, to make a vote for the prediction of the next transaction. Outputs from running many sequences of models with differing number of trees and differing sizes of sliding windows can be seen in figures 8 and 9 for the cases of 1 and 2 trees. The same figures for 3,4 and 5 trees are provided in the section Appendix.

Window Sizes	N Models	Recall	Accuracy	Bias
100	1	56.96	89.85	0.96
125	1	56.89	90.43	0.91
150	1	56.52	89.91	0.94
175	1	55.01	89.96	0.90
200	1	57.33	89.97	0.94
225	1	58.61	89.37	1.01
250	1	59.95	89.31	1.04
275	1	60.57	89.59	1.03
300	1	60.53	89.64	1.02
325	1	60.95	89.99	1.00
350	1	60.58	90.18	0.97
375	1	63.00	90.82	0.98
400	1	64.09	91.19	0.98
425	1	63.61	90.69	1.01
450	1	62.25	90.96	0.97
475	1	62.15	90.92	0.96
500	1	64.20	91.01	0.99
525	1	60.63	90.46	0.97
550	1	64.14	91.11	0.99
575	1	62.17	90.47	1.00
600	1	64.01	90.42	1.04
625	1	66.46	91.08	1.06
650	1	61.51	90.96	0.98
675	1	63.02	91.10	1.01
700	1	61.84	91.13	0.99

Fig. 8. Sliding Window Decision Tree Ensemble with 1 Tree

Window Sizes	N Models	Recall	Accuracy	Bias
100	2	51.65	88.81	0.96
125	2	51.53	89.42	0.91
150	2	52.17	89.11	0.94
175	2	49.61	89.02	0.90
200	2	52.19	88.90	0.94
225	2	54.24	88.42	1.01
250	2	57.62	88.45	1.04
275	2	55.50	88.42	1.03
300	2	56.84	88.93	1.02
325	2	55.15	88.67	1.00
350	2	55.82	89.42	0.97
375	2	57.91	89.54	0.98
400	2	60.50	90.18	0.98
425	2	59.17	89.67	1.01
450	2	56.34	89.97	0.97
475	2	58.19	90.16	0.96
500	2	57.39	89.68	0.99
525	2	57.18	89.59	0.97
550	2	62.10	90.59	0.99
575	2	58.94	89.54	1.00
600	2	60.47	90.00	1.04
625	2	63.08	90.40	1.06
650	2	57.73	90.19	0.98
675	2	58.20	90.29	1.01
700	2	57.24	90.16	0.99

Fig. 9. Sliding Window Decision Tree Ensemble with 2 Trees

Somewhat surprising to us, it seemed that the ensembles corresponding to the simple case of 1 tree tended to outperform the other ensemble groups. In figures 8 and 9, highlighted in red is the model corresponding to the highest Recall metric for the case of respectively 1 and 2 trees. Likewise, highlighted in yellow is their respective strongest performer in terms of overall accuracy, and finally light-blue denotes the best performer in terms of our bias metric.

B. Choosing the Best Sliding Window Decision Tree Ensemble

In order to select the best Sliding Window Decision Tree Ensemble, we developed a new criteria for selecting models. In particular, we ranked each model in terms of their placement in each of the 3 categories. The final score would be simply adding up these ranks as a sum-total. For example: if a tree was the 20th best in Recall, 10th in Accuracy and 1st in Bias, it's overall score would be 31:

$$Overall := Rank_{Recall} + Rank_{Accuracy} + Rank_{Bias} \tag{12}$$

We don't believe this metric to be completely optimal - some of the performance metrics should be weighted more heavily than others. However, for simplicity we kept the weightings equal across all three categories. A deeper comparison with a stronger understanding of the metrics is left for a future study. In computing $Overall$ for each of the models, we found our winner - shown on figure 11 in the Appendix.

	Logistic Regression	Naive Bayes	SVM	Novel Approach
Recall	0.49	0.56	0	0.64
Precision	0.41	0.4	0	0.54
Accuracy (%)	92.32	87.76	90.4	91.01
Bias	0.8	1.4	0.01	0.99

Fig. 10. Comparison of the models by our Performance Metrics

V. Final Comparison and Concluding Remarks

We will conclude this article with a brief comparison of all the models we test with our performance metrics outlined in section III.A. A table directly comparing the models is shown in figure 10. It's clear from the table that our "Novel Approach", which we have named "Sliding Window Decision Tree Ensemble" with a sliding window size of 500, and only with one tree was the best performer. We are proud to showcase that our model significantly outperformed the other models used in this data-set, in particular in the areas of Recall, Precision and Bias. Having achieved a near-perfect Bias score, along with over 10% gains in recall and precision, the sliding window decision tree completely outperforms any of the competing models.

Despite the promising results we have shown in this article, we still feel that there are many ideas left on the table. In particular, one thing we noticed late in our project was that in some of the categorical variables, new categories were showing up in the later part of the data-set. For example, newer versions of operating systems or phone versions such as the IPhone 8 only began showing up later in the data-set. This indicates to us that the variables in this data-set are evolving in time to an even higher degree than what we may have accounted for in this paper. Possible future works may be able to address these ideas.

VI. Appendix

window	N	Recall(rank)	Accuracy(rank)	Bias(rank)
500	1	0.6420455(3)	0.9101366(6)	0.9943182(11)

Fig. 11. Sliding Window Decision Tree Ensemble model with best Overall score

Window Sizes	N Models	Recall	Accuracy	Bias
100	3	51.78	89.06	0.96
125	3	51.28	89.45	0.91
150	3	52.43	89.20	0.94
175	3	48.84	88.99	0.90
200	3	51.67	88.96	0.94
225	3	54.76	88.41	1.01
250	3	57.62	88.91	1.04
275	3	55.76	88.69	1.03
300	3	55.94	88.96	1.02
325	3	55.15	88.90	1.00
350	3	55.97	89.62	0.97
375	3	58.06	89.71	0.98
400	3	60.50	90.17	0.98
425	3	59.17	89.81	1.01
450	3	56.50	90.04	0.97
475	3	57.91	90.41	0.96
500	3	57.95	89.89	0.99
525	3	58.05	89.76	0.97
550	3	62.68	90.81	0.99
575	3	59.82	89.94	1.00
600	3	60.77	90.15	1.04
625	3	64.31	90.62	1.06
650	3	58.36	90.30	0.98
675	3	59.16	90.48	1.01
700	3	56.25	90.00	0.99

Fig. 12. Sliding Window Decision Tree Ensemble with 3 Trees

Code available at

https://drive.google.com/open?id=1q2kE8jFDoJ36RmQAz524oefx6SW4jN8M

REFERENCES

[1] "fraud detection," A Passion for Research. [Online]. Available: https://softwarestrategiesblog.com/category/fraud-detection/. [Accessed: 12-Dec-2019].

[2] By, "Fraud Detection and Prevention Market Worth 41.59 Billion USD by 2022," MarketWatch, 08-Sep-2017. [Online]. Available: https://www.marketwatch.com/press-release/fraud-detection-and-prevention-market-worth-4159-billion-usd-by-2022-2017-09-08. [Accessed: 12-Dec-2019].

[3] R. J. Frank, N. Davey, and S. P. Hunt, "Time series prediction and neural networks," J. Intell. Robot. Syst. Theory Appl., vol. 31, no. 1–3, pp. 91–103, 2001.

Window Sizes	N Models	Recall	Accuracy	Bias
100	4	49.24	88.74	0.96
125	4	47.57	88.81	0.91
150	4	49.62	89.10	0.94
175	4	47.81	88.66	0.90
200	4	48.33	88.41	0.94
225	4	51.67	88.44	1.01
250	4	55.81	88.41	1.04
275	4	53.66	88.22	1.03
300	4	51.72	88.28	1.02
325	4	52.51	88.32	1.00
350	4	53.32	88.84	0.97
375	4	54.84	89.22	0.98
400	4	58.45	89.68	0.98
425	4	56.67	89.35	1.01
450	4	51.69	89.08	0.97
475	4	54.52	89.59	0.96
500	4	55.97	89.13	0.99
525	4	56.03	89.47	0.97
550	4	60.82	90.25	0.99
575	4	56.60	89.16	1.00
600	4	58.58	89.03	1.04
625	4	60.92	89.98	1.06
650	4	56.47	89.77	0.98
675	4	55.95	89.67	1.01
700	4	53.62	89.45	0.99

Fig. 13. Sliding Window Decision Tree Ensemble with 4 Trees

Window Sizes	N Models	Recall	Accuracy	Bias
100	5	49.24	88.58	0.91
125	5	48.34	88.99	0.85
150	5	49.62	89.07	0.87
175	5	47.04	88.82	0.83
200	5	48.07	88.37	0.88
225	5	51.67	88.50	0.94
250	5	55.81	88.47	1.02
275	5	53.40	88.31	0.99
300	5	51.98	88.31	0.96
325	5	52.77	88.45	0.96
350	5	53.85	89.17	0.92
375	5	54.57	89.22	0.93
400	5	58.45	89.65	0.99
425	5	56.94	89.45	0.98
450	5	52.54	89.36	0.90
475	5	54.80	89.73	0.91
500	5	56.53	89.49	0.96
525	5	56.20	89.61	0.95
550	5	59.82	90.40	0.96
575	5	56.60	89.16	0.99
600	5	57.99	89.10	1.02
625	5	60.00	89.97	1.02
650	5	57.41	89.99	0.98
675	5	56.27	89.78	0.98
700	5	53.62	89.37	0.97

Fig. 14. Sliding Window Decision Tree Ensemble with 5 Trees