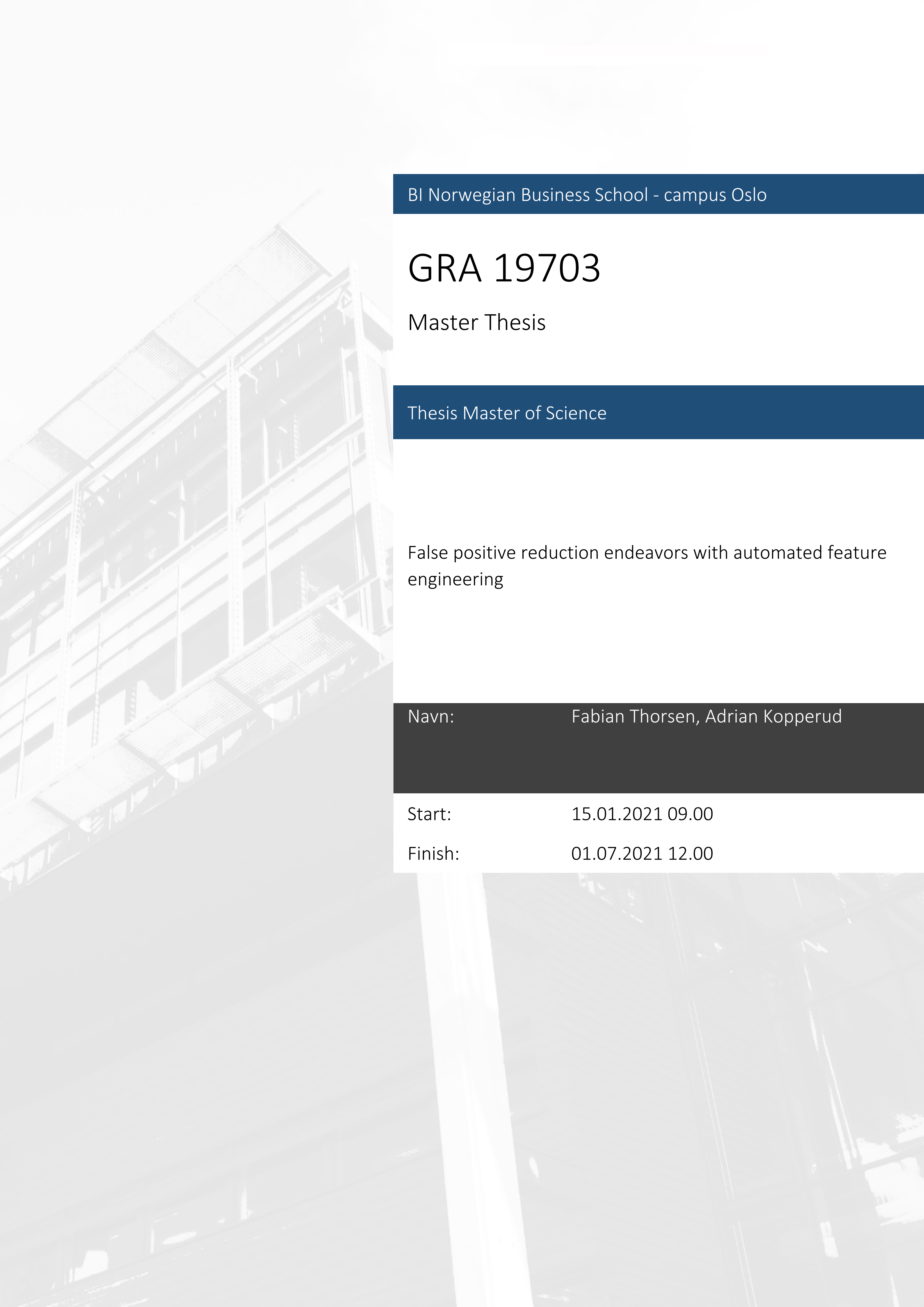


BI Norwegian Business School - campus Oslo

GRA 19703

Master Thesis

Thesis Master of Science

False positive reduction endeavors with automated feature engineering

Navn:	Fabian Thorsen, Adrian Kopperud
Start:	15.01.2021 09.00
Finish:	01.07.2021 12.00

False positive reduction endeavors with automated feature engineering

An empirical study to reduce false positives in fraud detection systems

Adrian Kopperud and Fabian Thorsen

Supervisor: Alfonso Irarrazabal

Master thesis, Master of Science in Business Analytics

Acknowledgement

This thesis is a part of BI Norwegian Business School´s Business Analytics Master of Science degree. Throughout our studies and past year we have gotten great support.

We would first like to thank our supervisor Alfonso Irarrazabal, for his great advice and guidance in difficult periods. Your valuable advice made us question the important aspects of this thesis, enhancing our work greatly.

Second, we would like to thank our lecturer John Chandler Johnson for his valuable knowledge and motivating ways of teaching. You provided us with both tools and motivation to be best equipped for this thesis.

Lastly, we would like to thank friends and family for their love, support and discussions. We could not have done it without you.

Abstract

Credit card fraud has been a problem for decades, and with the booming trend of online shopping fraud losses expected to rise for every year to come. Fraud detection systems often generate more *false positives* than *true positives* in order to attain a higher detection level of fraudulent transactions. These false positives have plagued the fraud detection industry for years as they are expensive to investigate and require extensive manual labor.

An automated feature engineering approach was implemented to address the problem of high false positives while at the same time conserving most of the true positives. We generate a high feature space (1750 features) of rich features without manual intervention other than specifying the primitives. In addition, a feature reduction method is implemented to retain the features with the highest predictive power to counteract the dimensionality problem of the method.

To compare our results, there were two additional datasets created for benchmarking purposes. The first dataset only included the cleaned original features, referred to as the *baseline*. In the second dataset, we generated manual features from the original data to reproduce the situation of a domain expert. The proposed solution was tested with the XGBoost to quantify the effect of the automated feature engineering on the reduction of false positives and was compared to the benchmarking datasets.

Our analysis of the results shows that automated feature engineering can improve false positives by 84% while managing to retain 89% of the true positives compared to the baseline dataset. In addition, we find no significant difference between automated and manual feature engineering on the discarding of false positives, and both methods are equally good. However, the results suggest that an automated approach can cut down feature engineering time a lot while providing richer features than manual feature engineering, suggesting a potential for bottom-line savings by reducing the number of domain experts and improved efficiency in the analytical life cycle.

Acronyms

TP	True Positive
FP	False Positive
TPR	True Positive Rate
FPR	False Positive Rate
TN	True Negative
FN	False Negative
ROC	Receiver operating characteristic curve
AUC	Area under the curve
XGBoost	Extreme Gradient Boosting Algorithm
RF	Random Forest
SVM	Source-vector Machine
DFS	Deep Feature Synthesis
SMOTE	Synthetic Minority Oversampling Technique
ML	Machine Learning
PCA	Principal Component Analysis
EDA	Exploratory Data Analysis
CV	Cross-validation
ANN	Artificial Neural Network
RQ	Research question
AI	Artificial Intelligence
MLE	Maximum likelihood estimation

List of Figures

List of Tables

Contents

1. INTRODUCTION AND MOTIVATION

This research investigates how automated feature engineering affects one of the main problems encountered in anomaly detection, namely the false positive problem. Our thesis uses sophisticated machine learning methods and feature engineering to rank three different models on how well they reduce false positives. Our research domain is within the e-commerce sector and we will apply our methods to transactional data.

Due to the increased development of online solutions and technologies, online fraud has increased rapidly over the last decade. With the help of the COVID-19 pandemic, a new digital norm has emerged from the disruption of regular routines. Thus, businesses are being forced to expedite digital transformation more than ever before, with consumer patterns booming within online shopping. Such a revolution however, does come with certain drawbacks. A recent report from 2020 showed that 4 out of 5 banks and financial institutions had a massive increase in fraud losses last year (FICO, 2020), and it is expected that the worldwide loss of credit card fraud will increase from $27.85 billion to $40.63 billion within the next five years (The Nilson Report, 2019).

It is often the case that fraudulent transactions are reported when the customer contacts the credit card company. However, the banks cannot rely on all their customers to report fraud. To detect fraudulent transactions and crimes committed by fraudsters, banks rely on a heavy amount of data to identify and learn customer patterns in order to predict fraud using detection systems.

Predicting fraud is nothing new and has been around for a long time. There are multiple defined supervised methods in the literature to help solve fraud (Brause et al., 1999), (Aleskerov et al., 1997). However, spotting fraudulent transactions is a challenging task due to multiple reasons. For example, imbalanced data is a big challenge as a large portion of the data is genuine transactions and only a tiny fraction fraudulent (Makki et al., 2019). A second major problem is concept drift. The constantly changing consumer pattern can be a challenge for the model, requiring consistent updates of the expert rules to keep the model relevant (Gama et al., 2014).

However, there has been little research in reducing the false positives, which has plagued the banking industry for years (Pascual, Marchini, 2018). For example, in 2017, 1 out of 15 consumers were affected by false positives, and adults below the age of 35 will most likely drop a credit card company when being declined upon purchase (Pascual, Marchini, 2018). As a result, high numbers of false positives benefit no one, and analysts have pointed out that it may cost more for the online merchant and banks than the gain from predicting fraud itself. Furthermore, merchants reported that 32% of customers stopped shopping with them after the decline from a false positive (Bannett, 2017)

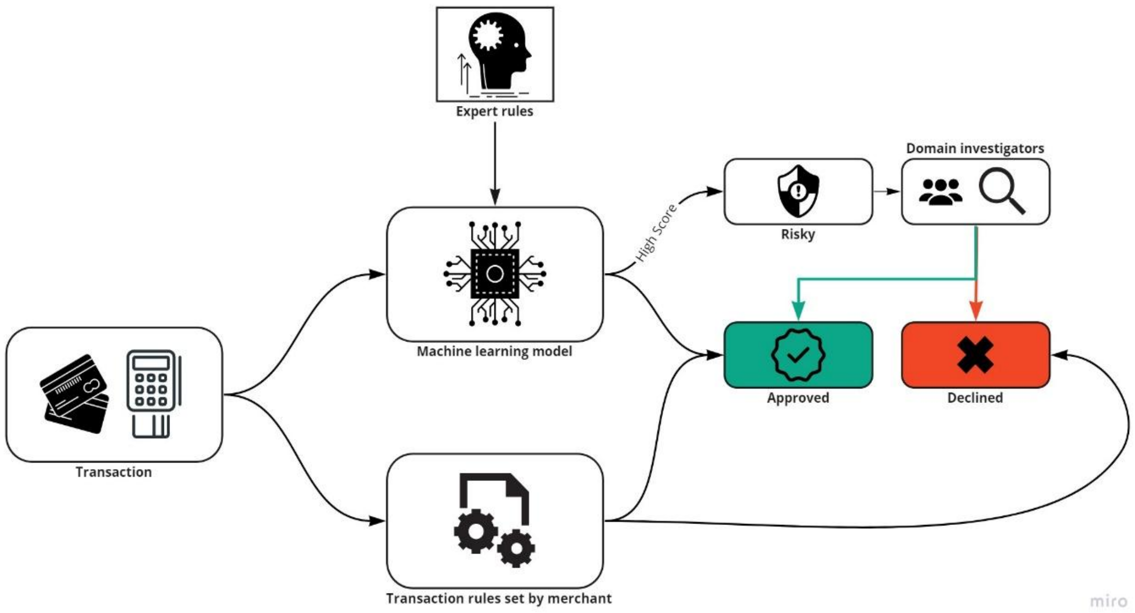


Figure 1.1: Illustration of a common practice in today's method for reducing false positives

From Figure 1.1 do we illustrate how e-commerce companies mitigate the problem of false positives today (Carcillo et al., 2018), (Ingenico Inc, 2020). The process consists of a multi-step method where the transactions are ranked through the scoring of different predefined rules.

The merchant can create their own rules through a deny list, containing blocked IP addresses and blocked regions, to name a few. This process works as a standalone filter that either declines or approves the purchase directly based on the satisfaction of the conditions.

The expert rules are feature engineering on historical data performed by domain experts. These features have the goal of scoring a transaction based on previous purchase patterns. An algorithm is used to output the predicted score based on the attributes of the transaction. This filter can work on top of the merchant-specified rules.

Suppose the transaction score is above a certain threshold. In that case, it is forwarded to a security center usually located in the company where fraud agents will judge the transaction to either approve or decline. This method is an incredibly time-consuming and costly way to reduce the false positives as we need many agents to investigate "flagged" transactions. Additionally, the expert rules in place need to be constantly updated by domain experts to reflect the change in customer activity patterns (Milo et al. 2016). Moreover, this method is error-prone and biased based on the competency of the domain expert, the data scientist that manually creates the expert features. This could potentially lead to poor performance and a higher false-positive rate.

Over the last five years, new advancement has been made in supervised learning and other AI areas, and a weave of new methods has become available. An example of such advancement is within feature engineering, more precisely the possibility to automate the feature engineering process, a process previously known to be very time-consuming. This is an exciting field as machine learning models rely heavily on the input features, and even slight configuration to the raw features can have a significant impact (Domingos, 2012).

The inefficiency of today's methods and little research in the field of new tools made available over the last years motivates us to explore an automated approach to generate new features in order to test if we can reduce the number of false positives. Our hypothesis is that manual feature engineering is a much more exhaustive and error-prone way of creating expert rules due to it being both performed manually and it is time-consuming. Thus, automating this process could potentially decrease the false positives and reduce the number of fraud agents, saving the credit card company and merchant an extensive amount of costs. Furthermore, we think it is beneficial that expert rules are updated by an automatic method since the pattern of today's customers is in a consistent change. Doing so could save a lot of time which goes to update the rules from false positives and concept drift that can be allocated to improve the model or other places in the pipeline where resources are needed.

1.1 Fraud

Before diving deeper into our problem we define what "fraud" is. The Oxford dictionary interprets fraud as a *"wrongful or criminal act that is intended to result in financial personal gain"*. In literature, we have multiple forms of fraud, but we will focus on online fraud in this thesis (Jain et al., 2019). In the domain of online shopping fraud, fraudsters commit the crime remotely through mail non-receipt card fraud, false merchant websites (phishing), credit card id theft and account takeover to mention a few (Jain et al., 2019). These methods contribute to a vast amount of losses each year, as identified in the introduction.

To mitigate these losses, detector systems are implemented as described in Figure 1.1, which classify fraudulent transactions from genuine transactions. When a detector system "flags" a transaction, it blocks the purchase of a customer and sets off an alarm in the security center of the bank. A fraud agent will then decide whether the transaction was actual fraud or not, based on some investigation. Commonly this investigation will be conducted through the agent calling the customer or collecting more information about the transaction to make a judgment.

A transaction classified as fraud after an investigation is considered the truth and is referred to as *positives*. On the contrary, transactions classified as genuine after investigation are referred to as *negatives*. The domain experts change the classification of the model based on the outcome of the investigation. In this thesis, do we look at offline data from such a detector system. Thus, the target class is fixed and cannot be updated by an agent like in a real-world scenario. We will look at the binary classification problem, and four types of cases are defined.

- **True Negative (TN)** are transactions that generate no alerts and are legit. There is a significant share of these observations than positives, thus creating an imbalance in the data.
- **True Positives (TP)** are positive transactions classified as fraudulent by the detector and validated by the agent. In a normal environment there are only a few of these compared to the number of negatives.
- **False Negatives (FN)** are positives not detected by the system. The cost of these undetected transactions can be high for the credit card company. Customers may notice the fraud by themselves and report it to the credit card company.
- **False Positives (FP)** are negatives classified as fraud by the detector and subsequently, have the agent's investigation concluded that it was a genuine transaction. Thus, the customer has gotten their purchase declined even if it was a legit purchase. It is difficult to estimate the cost of one false alert as this can be company specific. However, many false positives can create huge losses for both the credit card company and the merchant. Moreover, it will be beneficial to minimize these types of cases.

1.2 Automated Feature Engineering

Automated feature engineering has the aim of generating informative and discriminative features from the raw data. In general, feature engineering requires human insight, usually referred to as domain experts, to understand the data at hand. Thus, it is a complicated process to automate and there are only a handful of frameworks that support the automation of features today. This thesis will apply the Featuretool approach, which we will do a short introduction for below.

Featuretool is an open-source Python library that automatically generates a large set of interpretable features from a single or set of relational tables. The framework can generate new features through Deep feature synthesis (DFS) that uses dataset relationships, data types and other underlying factors to enhance and extract more information from the pre-existing features. DFS mainly uses mathematical operations called Primitives to generate these features. These primitives are generally nested mathematical operations such as sum, standard deviation or averages. The type of feature created is generally a function of the dataset architecture meaning that several relational datasets may generate different features than a single dataset.

Featuretools bring a significant amount of out-of-the-box functionality such as variable type inference and default parameters that let you quickly generate features without much work. Nevertheless, there is some prep work that needs to be done in order for the library to use DFS to generate features. First, defining the entities and relationships among the entities is required to know what datasets are mutated. Entities are simply data tables, either one or more, that contain a set of features and observations. Relationships among the entities may be predefined such as in RDBMS systems where tables are connected through ID features or keys.

Second, Featuretools needs information on the data types of each feature that is being transformed. This may be inferred directly by the algorithm but is suggested to be done manually as the automatic procedure is not very technical and cannot detect categorical features. The last operation needed to be done before the features can be generated is specifying the types of primitives to be used when running DFS. This depends on the types of relationships and data types present in the problems, as some primitives only function when there are multiple tables with relationships. Lastly, the DFS algorithm needs to be run in order to generate the new features. This part of the procedure is similar to running most ML models in python.

DFS is a complicated method that uses the relationships among observations to calculate new features (Kanter, Veeramachaneni, 2015). This is another reason why certain primitive operations can only be done when there are multiple entities joined through relationships. Another complex task DFS performs is known as primitive stacking where multiple primitives are done in succession, creating one or more particularly deep features. A feature such as *MAX(MEAN(TransactionAmt))* is an example of this, and the number of primitives used in the creation is known as depth. Featuretools also thrive when exposed to time series data as the new dimension opens up the opportunity to create features dependent on time as opposed to static datasets.

1.3 Goal and Research Question

Following our motivation and description of the problem in previous sections, this thesis explores an automated approach to generate interpretable features to discard as much of the false positives (FP) as possible in today's detection systems. Many companies use domain experts to manually update and engineer features to maintain the relevancy of the fraud detector. As previously mentioned, we see this method as limited and biased towards the competency of the domain experts as there may be features or relationships that are not thought of that could have a higher predictive power.

This project will develop new features through Deep feature synthesis (DFS). We will explore many of its functionalities to achieve a rich collection of features that could help the model generalize better. The function will create random features in a higher feature space, thus creating more opportunities for the detector to learn. Unfortunately, discarding FP could result in discarding true positives (TP), which is not desirable. Therefore, a good trade-off is necessary between reducing FP and maintaining TP, which is essential for all banks based on their respective thresholds.

Based on the goal outlines above will we formulate the following two research questions:

RQ 1: Do automated feature engineering decrease the FPR rate in fraud detection systems compared to doing no feature engineering.

RQ 2: Does automated feature engineering better decrease the FPR in the fraud detection system than a manual feature engineering approach?

This thesis focuses on implementing a new technique to automate the feature engineering process to reduce the number of false positives. This thesis focus is not to aid domain experts. However, this approach could be helpful to those creating expert rules as we will look at differences and benefits with automated compared to manual engineering of features in RQ 2.

2. RELATED WORK

Fraud detection has been around since the 90's. The first systems were very restricted boundaries in the form of expert conditions and could collect, process, and store data. These conditions check for specific attributes, such as if the card was used in a different country or the transaction amount was more significant than some threshold. The transaction was then blocked based on if these conditions were satisfied.

In 2011, a detailed comparison of methods within fraud prediction was published (Bhattacharyya et al., 2011). Here the Support Vector Machine (SVM), Random Forest, and Logistic Regression were compared within the credit card fraud domain. The study concluded that the Random Forest approach had the best accuracy and fewer false positives, followed by Logistic Regression and SVM.

In July 2019, the imbalanced class and concept drift problem was addressed (Devika et al., 2019). The paper's focus was to create a novel learning approach to address the concept drift and imbalanced class challenge in fraud detection. The paper's outcome was successful, and they managed to demonstrate the imbalanced class problem and concept drift in a real-world problem. An interesting resultas it identified and resolved two of the most common fraud detection problems; adapting to new fraud methods and the challenge of imbalanced data.

Another research paper published in January 2019 (Jain et al., 2019) introduced the concept of fraud related to the e-commerce sector. The paper explained all the different ways fraud could be conducted; stolen credit cards, mail non-receipt card fraud and account takeover, to mention a few. In addition, there were various methods listed for how one could detect fraudulent transactions. These methods include ANNs, Bayesian Network (BNN), K- Nearest Neighbor (KNN), Decision Trees and SVM. As a result, they found out that ANNs returned both the highest accuracy and the lowest false-positive rates compared to other methods. The KNN, Logistic Regression, Decision Trees, and BNN returned a medium false positive rate, and the SVM on the other had the highest observed false-positive rate. The Drawback of the ANN method was the high cost of training the model, followed by KNN, SVM and Decision Trees, which all placed somewhere in the middle in terms of training costs. The Logistic Regression approach was the least expensive model to train.

In 2017 Cornell University published a highly relevant article for our thesis (Wedge et al., 2017). This research presented an automated feature engineering approach to cope with the false positive problem in the fraud detection system. The reduction of false positives demonstrated that a lot of genuine transactions were falsely classified as fraudulent. The paper used the Featuretools approach to derive the features based on the historical transaction data automatically. In total, 237 features were generated for each transaction, and a tree-based classifier was used in the study. The model was tested on a massive dataset from a banking corporation and was compared to their existing detector solution in the bank. The model was tested on an unseen dataset of 1.852 million card transactions. The result of the automated feature engineered model was a stunning 54% cut down in false positives. Such a drop in false positives provided estimated savings of 190.000 Euros. They also investigated the possibility of deploying the model under streaming computation in a real-world situation. We think it is vital for further research and validation based on the promising results from this study on automated feature engineering.

Baader & Krcmar (2018) proposed a red flag approach combined with process mining to reduce the false positives in the domain of internal fraud detection. The red flag approach gives hints or indications of fraudulent activity by scanning the dataset for a "fraud pattern". On top of this method, they apply process mining to recreate the as-is business process to visualize the information across the organization in the form of a user interface. Their framework was applied to a purchase-to-pay business process (P2P). P2P handles the purchase of goods to the payment of the vendor of an organization. Their method aimed to detect internal financial fraud and supplier procurement fraud, while maintaining a low false-positive rate. Their method was prosperous compared to other studies, and they achieved an FPR of 0.37% and a TPR of 48.38%.

So far, we have found that neural network methods outperform other machine learning approaches in terms of reducing false positives. This could be because deep learning models automatically operate directly on the raw data at a higher representation level. However, such a performance comes with certain drawbacks. The construction and choices of these deep learning models are almost impossible to interpret for humans, and the models are very costly to train (Jain et al., 2019). On the contrary, machine learning algorithms understand operations that are native to humans, such as and, if, or operations. Furthermore, algorithms such as XGBoost, Random Rorest, Gaussian Naïve Bayes and Logistic Regression have shown promising results in terms of performance and implementation cost which have been shown in the literature (Jain et al., 2019), (Bhattacharyya et al., 2011), (Wedge et al., 2017), (Goyal et al., 2020). Thus, these algorithms are good model candidates to help us answer our research questions.

It was hard to find previous research that discussed reduction methods of false positives, which we see as crucial for a well-working detection system. Instead, many articles discuss how accuracy could be improved, selecting the best models and optimizing the hyperparameters for best results (Dornadula, Geetha, 2019), (Xuan et al., 2018), (Lakshmi, Deepthi, 2018). Additionally, several articles look at the benefit of how domain experts could decrease the false positive based on their experience and competency (Whitrow et al., 2009). However, over the recent years, techniques such as automated feature engineering have become available, and multiple studies have been conducted on the application within different domains (Kanter, Veeramachaneni, 2015), (Wedge et al., 2017), (Lucas et al., 2019). These studies have shown that the automation of feature engineering both increased performance and reduced development time, and some suggested that it outperformed the domain experts.

Based on our analysis of various articles in the literature, we observe that little research has been done in the field of false-positive reduction. We identified only two articles within this field (Wedge et al., 2018), (Baader, Krcmr, 2018). Because automated feature engineering has shown promising results in many domains, we find it interesting to implement this technique to extract useful features that could potentially reduce false positives. Furthermore, will we extend and validate work already done (Wedge et al., 2017) and supplement this narrow field with a comparison between manual and automated feature engineering effect on false positives, which has not previously been attempted to the best of our knowledge.

Trivedi et al. (2020) recently introduced a comparison study of machine learning methods within credit card fraud detection. Here the goal was to analyze the different algorithms and their performance in credit card fraud detection. Model's tested include Random Forest, BNN, Logistic Regression, SVM, KNN, GBM, to mention a few. The study was conducted on a dataset of European cardholders. On average, the algorithms return an FPR of 4.3% (Trivedi et al., 2020). For our paper, is this finding interesting before going into the experiment as it can be viewed as a benchmark.

3. DATA PROCESSING

This chapter is divided into two parts. The first section presents how we collected our data and selected the relevant features for our experiment. The second chapter is the most detailed and includes preprocessing and transformation. Here we describe how we created the three different datasets for our experiment described in the transformation section. This section is the most essential aspect of this chapter to answer our research question.

Figure 3.1 illustrates our data collection and preprocessing architecture. These are the first two stages of our methodology that will be continued in chapter 4.

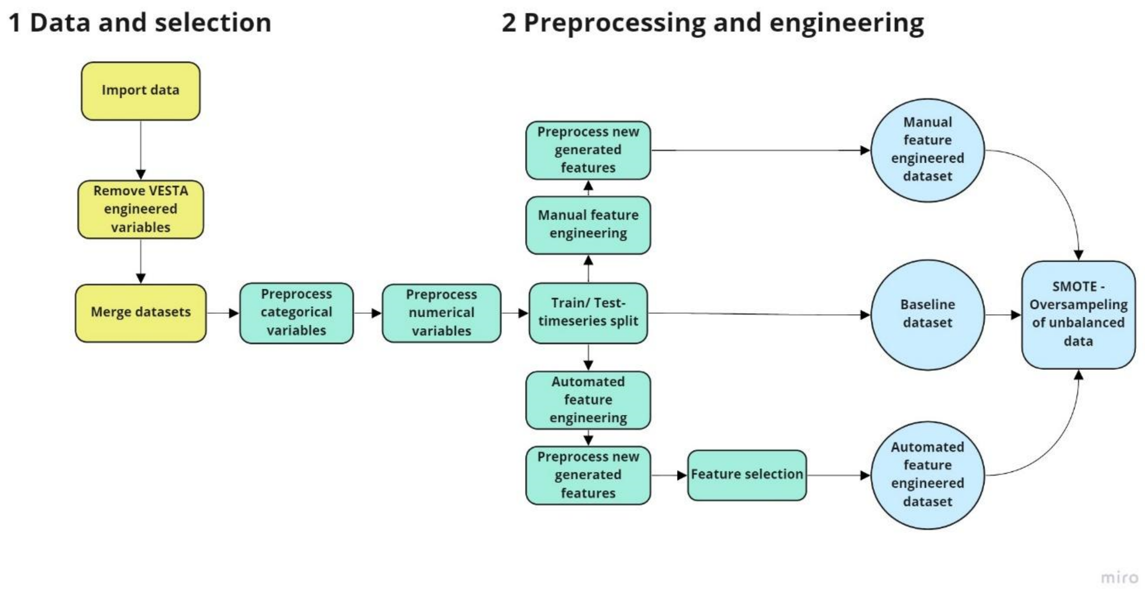

1 Data and selection
2 Preprocessing and engineering
Import data
Remove VESTA engineered variables
Merge datasets
Preprocess categorical variables
Preprocess numerical variables
Train/ Test-timeseries split
Manual feature engineering
Preprocess new generated features
Automated feature engineering
Preprocess new generated features
Feature selection
Manual feature engineered dataset
Baseline dataset
Automated feature engineered dataset
SMOTE - Oversampeling of unbalanced data
miro

Figure 3.1: Selection and preprocessing architecture

3.1 Data Collection and Selection

Our dataset was collected from Kaggle on the 01 of December 2020. Kaggle is a website where companies with various problems publish datasets through competitions where data scientists compete in creating the best performing models. The data collected was from a fraud detection competition held in 2019 by VESTA Corporation. VESTA is an e-commerce and credit card company that provides a labeled dataset and an unlabeled dataset from their detection system. We choose only to use the labeled dataset in our research, containing approximately 590.000 rows and two tables.

The transaction table contains 394 features while the identity table consists of 41 features, amounting to 435 features. The transaction table provides information on the product bought and the type of card used in the transaction. Furthermore, we are provided with a timed delta in the format of seconds between each transaction, along with the address and associated email address. In addition to these features there are many masked features included with no explanation due to privacy reasons. They provided extensive amount rich features to compensate for features that could not be included in the dataset. The identity table includes digital signatures, IP and proxy-related information, and other features related to the customer's identity. VESTA provided the start date of the data which started on 2017-12-01.

We chose this dataset because it was the only available dataset we could find from a banking domain with raw features and not pre-engineered features, often being PCA computations of original features. The decision of using the dataset was based on the amount of features and its large number of observations. Although the dataset contains a mix of pre-engineered and natural features, we find it sufficient to use the most naturally occurring features to best ensure reproducibility, thus discarding most of the pre-engineered features. Based on our research questions it was important to include a lot of raw features in our baseline model. We merged both the identity and the transaction table based on the unique “TransactionID” key.

We filtered out the VESTA rich features (denoted V_xx) because they were pre-engineered features combined from provided and not provided features. Based on our research question, we only include the raw and masked features. The engineered features could potentially leak information from other features already included which could potentially invalidate our research. Appendix 1 includes a description of all the raw features from our baseline dataset.

3.2 Data Cleaning

"Pre-processing is an important step in the machine learning process. The pre-processing step is necessary to resolve several types of problems including noisy data, redundancy data, and missing data values" (Kotsiantis et al., 2006)

This section will follow general changes to the data table as it is where the information converges to become the data we will train our model on, finally. The aim is to remove all redundant information and clean up the data through the imputation of missing values, remove outliers, categorical encoding and correct structural errors, to mention a few. The primary purpose is to detect inaccurate, inconsistent, and irrelevant data and modify or delete this useless information to form a dataset that provides quality to the other modeling steps down the value chain (Agarwal, 2015).

First, we overview the data structure and patterns through an exploratory data analysis (EDA). Second, we cleaned the raw features by removing irrelevant and redundant information present in the data. The python code for all our work is included in Appendix 2.

3.2.1 Removing Redundant Information

Dist1 and dist 2

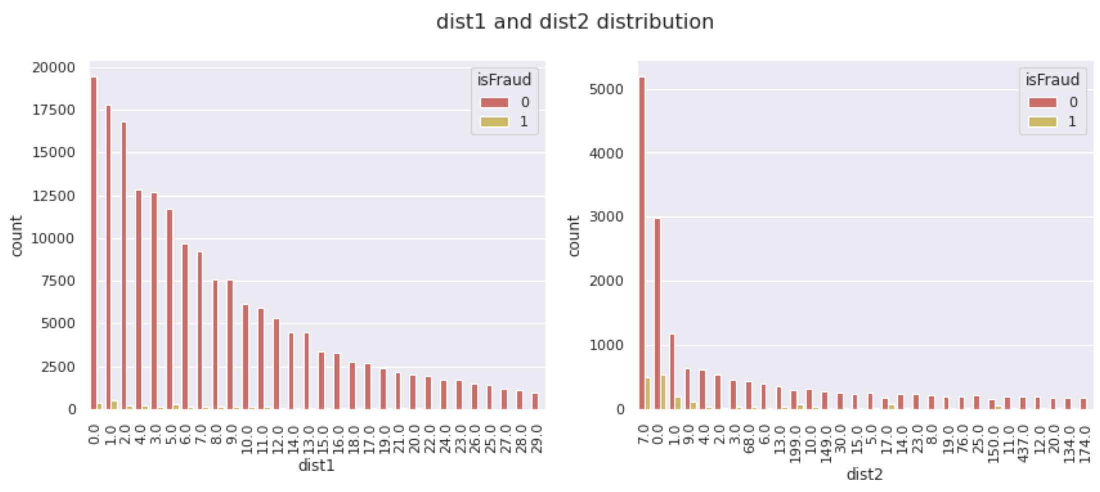


Figure 3.2: Distribution of *dist1* and *dist2*

The dist features describe the distance between different objects such as zip-code, IP address and phone area. As shown in Figure 3.2 the dist1 feature contains more information than *dist2*, which could be explained by the fact that *dist2* has 45% more missing values than *dist1*. As a result of the amount of missing data in *dist2*, it was deleted from the table. One argument for the removal is that we get more accurate data and results (Kotsiantis et al., 2006). In Appendix 3, we included an extensive analysis of the missing data we base this judgment on.

TransactionID

The identification variable (*TransactionID*) we used to merge the two tables is removed from the dataset as this variable is no longer needed for our modeling part.

3.2.2 Missing Values

The missing data is one of the common problems found in data today. Imputing the missing values makes the analysis more manageable by making the dataset complete as it eliminates the problem of handling complex patterns of missingness (Chhabra et al., 2019).

There are several ways to eliminate missing values in the data. A data science article proposed the following methods (Badr, 2019)

- **Mean imputation.** Calculate the mean of the non-missing values and use this to impute the missing value observations. This method only works for numerical data.
- **Zero/ constant approach.** Impute missing values with a new value different from all other values. This method can be used for both categorical and numerical features. A drawback is that it can introduce bias to the data.
- **Imputation using deep learning.** This method can impute missing values using the other features in the dataset to predict the missing feature. This method works well for both categorical and numerical features. A drawback is that this solution is prolonged and time-consuming.

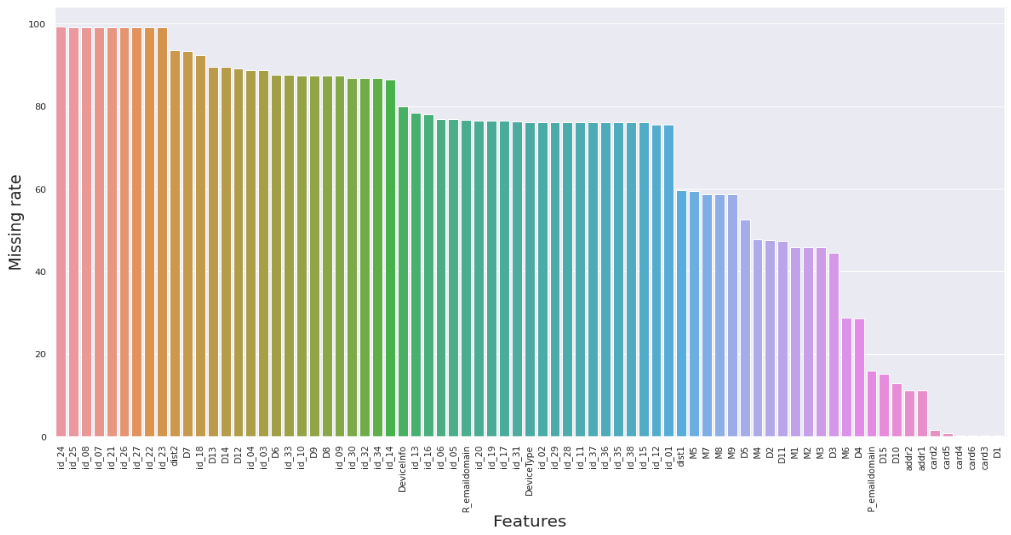


Figure 3.3: Missing data percentage by features

As shown in Figure 3.3, our data have a substantial number of missing values, especially in the identification features that have on average 80% missing values in each feature. Due to our lack of domain expertise, we find it hard to conclude if these values are missing at random or if there is a reason for the data to be missing. Therefore, we impute the missing numerical features with a zero/constant approach, filling the missing numerical values with a number significantly different from any other value in the dataset (Bhaya, 2017). This imputation was done because of the average high percentage rate of missing data. On the contrary, dropping them could potentially lead to loss of fundamental observations and feature-specific information, something we did not want.

The choice of method to impute missing values varies and depends on what kind of data you have, and there is no defined rule for this process. We are aware that using a constant-value approach to fill the missing values has its limitations and may not be ideal. In this specific case, it does not make sense to fill the missing values with the feature mean or use deep learning to predict the input value based on other similar features because the missing value percentage is too high on average.

The categorical features are imputed the same way. If the number of missing values is vast, it can be replaced with a new category (Kumar, 2020). We therefore impute all categorical features with a new category, "None" for each feature.

3.2.3 Outliers

Outliers are defined as values that excessively deviate from the feature mean (Kotsiantis et al., 2006). The transaction amount (*TransactionAmt*) is such a feature in our dataset, most likely due to special-case transactions or fat finger errors. There were in total three observations, none of them fraudulent that were above the threshold of 10000. We remove the outlying values from the dataset.

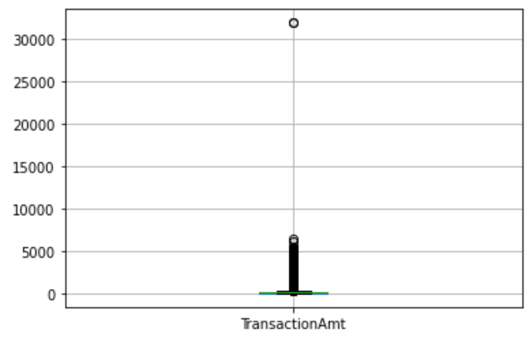


Figure 3.4: Boxplot of TransactionAmt

3.2.4 Categorical Features

"Unlike quantitative attributes, categorical attributes typically have no natural ordering or distance between values that fit quantitative definitions of outliers. One key data cleaning problem with categorical data is the mapping of different category names to a uniform namespace. E.g., a "razor" in one data set may be called a "shaver" in another." (Hellerstein, 2008).

We used EDA to identify structural errors and inefficient categorical variables with many categories where only a few are essential. We apply feature mapping and regrouping on those premises to make the feature more susceptible to provide information a machine learning algorithm can learn from.

The method applied was to merge all the few observations into one category called "others", thus making the feature less complex. We have illustrated below how we did the feature mapping of the categorical features.

Device Info

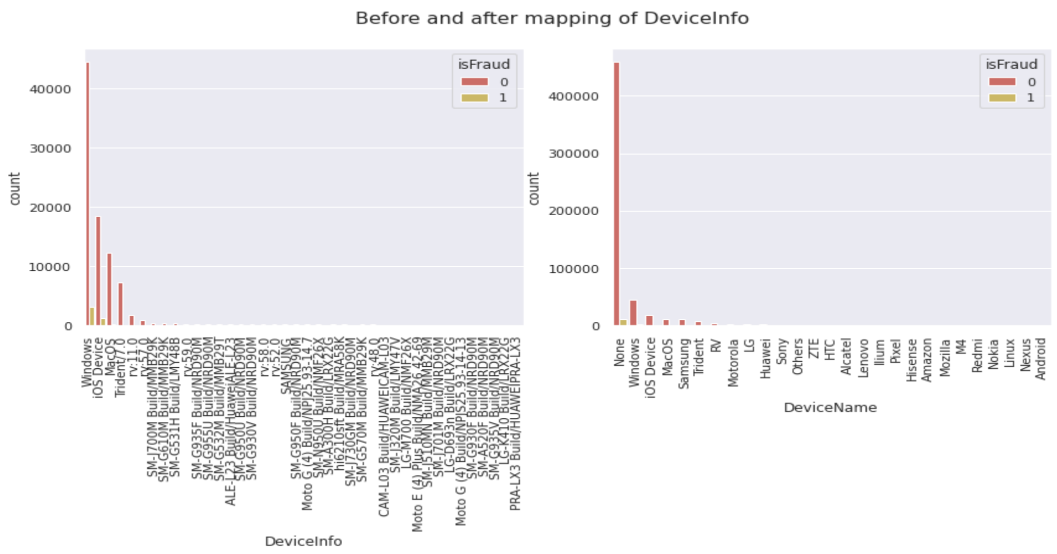


Figure 3.5: Device Info before and after mapping

The *device_info* feature had multiple categories of the same name. For example, the iPhone (IOS) had multiple categories with different software versions. We grouped all software systems with the same name into one group for each provider.

The same mapping and reorganization were done to the following features;

ID_30, ID_31, ID_33, card6, P_emaildomain and *R_emaildomain.*

We have attached the preprocessing of these categorical features and illustration in Appendix 7.

3.2.4.1 Categorical Encoding

Most machine learning models cannot handle categorical features directly as text, and thus we need to transform them into numerical values. The different model's performance varies based on what kind of algorithm we use. (Cerda et al., 2018)

In addition, it is crucial to understand what kind of categorical variables you are working with. As nominal categories have no order and label encoding could be inefficient, the model could misunderstand and treat the nominal values as a hierarchy or ordering (Shaikh, R. 2018).

From the data science article on categorical encoding (Yadav, 2019), was the following methods presented to encode text into numeric values.

- **Label encoding** is a simple approach to convert each value in a column to a number. This method uses number sequencing, meaning that different values will have a number assigned in a sequential order starting on 0. Thus, this approach is best for ordinal categorical features as an algorithm may misinterpret the data by hierarchy or order ($0 < 1 < 2$). Therefore, is this method not optimal for nominal values with no specific categorical order.
- **One-Hot Encoding** solves the misinterpretation that the numeric values have some kind of order to them. This method converts each category into its unique column with a 1/0 value. The row with the first column value will have the value 1, and the rest will be assigned 0. The drawback of this feature is that it can create a vast feature space for highly cardinal categorical features. This can lead to "the curse of dimensionality" and increase the model calculation time.

We faced multiple challenges when implementing a suitable method to encode our categorical features. The first major problem was detecting if the feature was either nominal or ordinal as most of the meaning of the categorical features was masked and not appropriately explained. The second problem was high cardinality for several of the categorical features. For example, the categorical feature *card1* had 12 000 different categories, making it impossible to encode with a one-hot encoding method. The feature space would destroy the model performance and potentially introduce the "curse of dimensionality".

Based on theory, the most beneficial solution would be one-hot encoding. However, the high cardinality present in multiple categorical features made it difficult for us to use this approach. This is because it gives rise to several other problems, such as the risk of blowing up the feature space and fighting the curse of dimensionality, leading to potential overfitting or worse performance for the model (Cerda et al., 2018).

As a result of the problems we faced, we implemented a trial-and-error approach, testing both methods. First, we implemented a count encoding strategy to reduce the cardinality for the highly cardinal features, which was transformed into numeric variables. Then we applied one-hot encoding to the remaining categorical features. After the one-hot encoding, we ran a PCA to reduce the dimensionality of the sparse matrix produced. Finally, we tested the method with an algorithm to get the AUC score and compared the results to a model that used the label encoding approach.

The outcome was that the label encoding method outperformed the one-hot encoding method significantly. Thus, we decided to go with the label encoding method even though the method has its limitations. We base our choice on the increased performance with label encoding and on the premise that we used a trial-and-error approach to see what works the best for our data.

3.2.5 Time Series Train/ Test Split

A general step in machine learning is to split the data into train and test sets. It is a crucial process as it is the only way to validate how the model will perform on unseen data. After separating the training dataset, we use this chunk to train, validate and tune the model. Furthermore, it is essential to know what data you have and choose a split method accordingly (Grootendorst, 2019).

Most commonly, we want an even distribution of fraudulent patterns in both the training and test dataset. We do not want patterns present in the test data which are not present in the training data, as it is hard for a machine learning model to predict a pattern it has never been exposed to or trained on. Thus, patterns present in test data should also be present in the training dataset. In python, do we achieve this by using a stratified split.

Since we have time-series data in our thesis, we most likely have a fraudulent pattern that have developed over time as new fraud methods have emerged, also referred to as concept drift in literature (Devika et al., 2019). If we deployed a stratified split, we would most likely get good results that reflect our model's predictive power. However, we would indirectly leak information concerning the target through the training process as it spreads information from all periods across all the datasets. Since we want the experiment to be as realistic as possible, we do not have transactions from the same period in both the training and test dataset. This is because a model that knows the former will naturally predict the latter well, returning too optimistic test scores and not generalize well to real-world applications (Miyaki, 2019).

We split based on periods since we are working with time series. The test data will be the last 20% of the period, and the train will include 80% of the data before. There are limitations to our approach as our model will predict on blind test data, potentially leading to lower accuracy and performance than doing the split more traditional with a stratified split. Nevertheless, this does not mean that our results would be invalid, but it could make our results less accurate.

In Figure 3.6 we illustrate how we split our data and we can see how the fraudulent activity drops in the testing period. This can indicate that we have a change in activity or fraud pattern, making it harder to model, and we may expect the model to have less accurate results than what is expected. In our case, we continue with this approach while being familiar with its limitations.

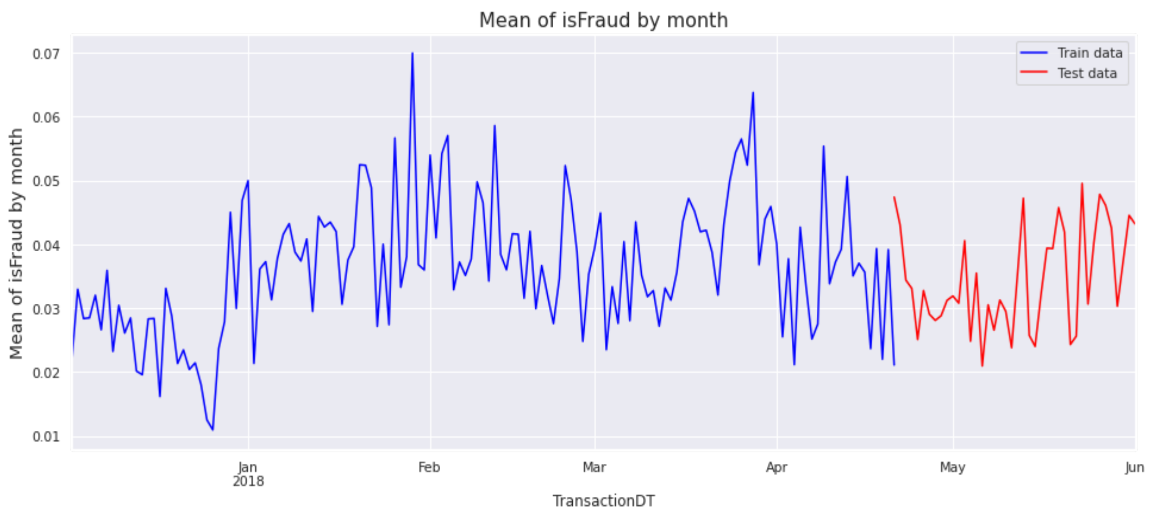


Figure 3.6: Train/test split of the dataset

3.3 Feature Engineering

"The function of mathematical modification to the value of a feature which extracts more value than in its original state summarizes the goal of transformation" (Osborne, 2002). From this paper, two forms of transformations are identified.

1. Change in the original feature
2. New features created from existing features.

In this section, we define all three datasets used in this experiment and all datasets have the cleaned raw data in common. For our two baselines, no feature engineering and manual feature engineering will be applied. Finally, automated feature engineering in the form of DFS will be applied to a clean dataset which later will be evaluated against the two baseline datasets in the result chapter where we will quantify the overall performance of automated feature engineering.

3.3.1 Dataset 1 – Baseline

Our baseline dataset will only contain a clean copy of the natural raw features available directly from customer interaction when making a purchase. This dataset will be used as a benchmark against the automated engineering method to answer our research questions.

3.3.2 Dataset 2 – Manual Feature Engineering

For this dataset we use the baseline as the foundation for further feature engineering. We craft new features based on the features present in the cleaned baseline. The goal is to build new features based on our knowledge, attempting to recreate how a domain expert would craft new features by hand.

Change in original features

TransactionDT

TransactionDT was initially given as the number of seconds, and we transformed this feature into a DateTime feature based on the starting date of 2017-12-01. The credit card company that provided the original dataset also provided this date.

New features created from existing features

Date Features

From the DateTime feature, do we create additional time-based features. We generate the following features*; weekdays*, *hour of the day*, *day of the month* and *month of the year*.

High risk and low risk feature

From the EDA, we find that most fraudulent transactions happened from 05:00 at night to 10:00 in the morning. In Figure 3.7, we illustrated that time of day strongly depends on whether the transaction is fraudulent. The grey trendline represents the amount of fraud while the bars represent transaction activity.

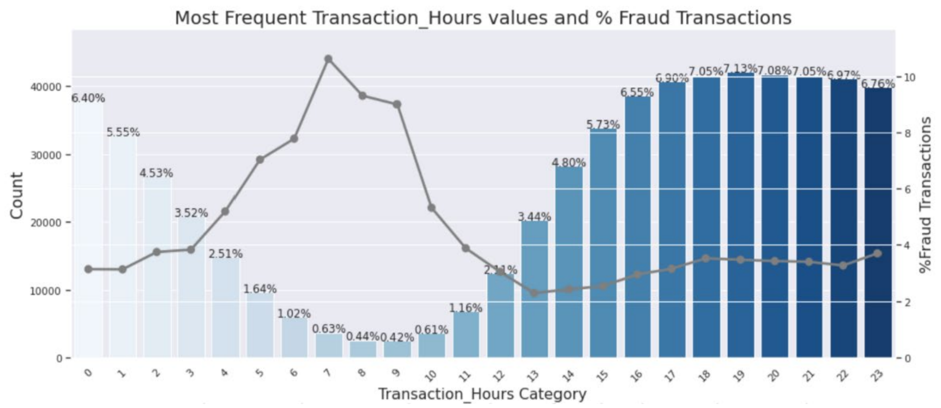


Figure 3.7: Most frequent transaction hours of the day

A binary feature was created which is 1 if the time of day is between 05:00 and 10:00, and 0 otherwise.

TransactionAmt

From the transaction amount we generate two additional features. First, the transaction amount is highly skewed. Thus, we transform the *TransactionAmt* into a new feature taking the log of the transaction amount.

Second, we create an additional feature which only extracts the decimal number from the *TransactionAmt*. This could be useful as the EDA showed that the mean fraud is higher for transactions with three decimal points, as illustrated in Figure 3.8.

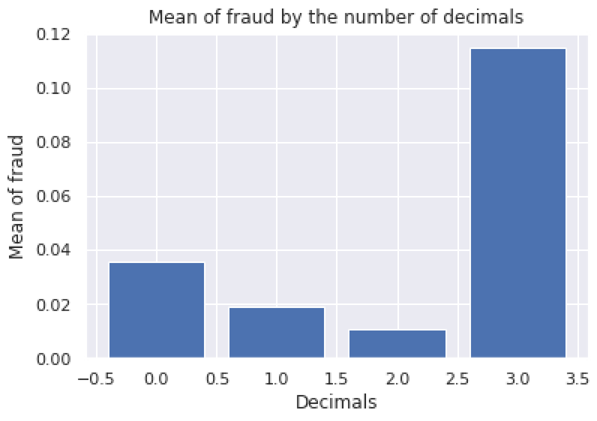


Figure 3.8: Mean fraud by decimals

Random aggregation of some essential features

We implemented a random aggregate method on the different card types with the transaction amount (*TransactionAmt*), including various max, min, skew, var, and std operations.

Count encoding

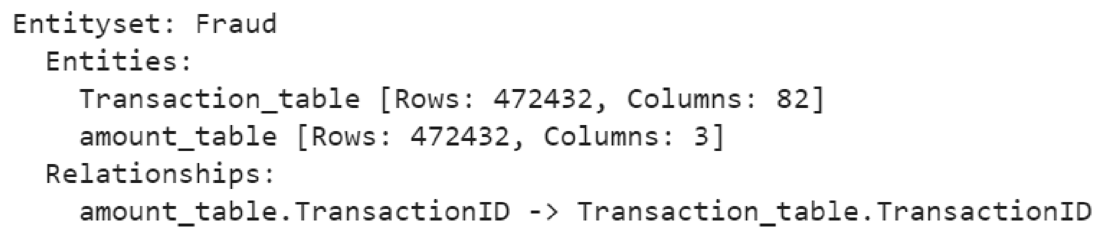
Count encoding is sometimes used for replacing highly cardinal categorical features. It is performed by replacing the categorical value with its count of instances. In our case we had multiple card features with high cardinality. For example, *Card1* has over 12 000 categories as previously mentioned. Hence, this method transforms the categorical features to a numerical format and can have helpful information for the model to learn. We apply count encoding for the features *card1* to *card6*.

3.3.3 Dataset 3 – Automated Feature Engineering

This section provides a detailed description of how we implemented automated feature engineering through a python library called *Featuretools* on the baseline dataset to create new features. This approach can be used for both a set of related tables and single tables. In our research, we focus on how Featuretools perform on a single table.

Entity set and entities

We start by creating an entity set for the transaction table. The entity set can be interpreted as the contained table(s) data structure and allow us to group multiple tables if we have more than one table. We specify each entity for the entity set where an entity being one data table. In our case, we have one table but want to split the transaction amount (*TransactionAmt*) into a single table to use aggregation over the entire transaction table. Thus, we create two entities in the *fraud* entity set, the *transaction_table* and the *amount_table*.



```
Entityset: Fraud
  Entities:
    Transaction_table [Rows: 472432, Columns: 82]
    amount_table [Rows: 472432, Columns: 3]
  Relationships:
    amount_table.TransactionID -> Transaction_table.TransactionID
```

Figure 3.9: Output of entity set

From Figure 3.9, we can see that we manage to create the entity set *Fraud* which holds the entity *transaction_table* and *amount_table*. The corresponding dimensions of the dataset are listed. Since we split out the transaction amount to form a new table for the purpose of aggregation, we have to specify the relationship between the two tables as seen in the output.

Specification of variable type

We had to specify what kind of features were categorical, time-based and numerical in each entity set. The default setting of Featuretools is to specify all features as numeric unless we input otherwise. Since we had many categorical features, did we update the entity information with the correct specification. The time delta (*TransactionDT*) was specified as the time index to create new features based on the time.

Feature Primitives

Featuretools operate using primitives. Primitives are operations that are applied to our dataset in order to generate new features. There are two forms of primitives.

> **Aggregation** primitives' groups features from all related data tables to form one main table. Operations such as max, min, st.deviation and skew are a few examples of operators to choose from.
>
> **Transformation** primitives are applied to multiple features in a single data table. Operations such as the difference between two features or absolute value are some examples of transformative operations.

Our primary focus is on the transformative primitives in our experiment, but we also include aggregation primitives for the *TransactionAm*t feature. Commonly are transformation primitives applied for single tables. Multiple tables are usually aggregation primitives applied to aggregate the information from all tables into one entity before transformation primitives are applied to the entire table.

Furthermore, we specify what type of transformation and aggregation primitives we want to apply from a list of available primitives. Based on our data and the information present in the table(s), we choose to use the following set of primitives to be applied to our entity set.

Primitive	*Type*	*Description*
Divide numeric	Transformation	Divided numeric features
Multiply numeric	Transformation	Multiply numeric features
Diff	Transformation	Compute the difference between the value in feature and the previous item in that feature
Hour	Transformation	Determine the hour value from the timedelta
Day	Transformation	Determine the day value from the timedelta
Month	Transformation	Determine the month of the year from the timedelta
Week	Transformation	Determine the week of the year from the timedelta
Time since	Transformation	Calculate the time from one transaction to another using the timedelta
Is weekend	Transformation	Return boolean value of true/false if the timedelta falls on a weekend

Time since previous	Transformation	Compute the time since the previous transaction using the timedelta
Max	Aggregation	Calculate the highest value
Min	Aggregation	Calculate smallest value
Median	Aggregation	Determine the middlemost number in the feature
Mean	Aggregation	Compute the average for the feature

Table 3.1: Primitives applied in Featuretools (DFS)

As a result of limited domain knowledge of the raw features, we let Featuretools run primitives on all our features without specifying any limits. An attribute of Featuretools is that we can specify which feature we want to perform the operations on, but by default will a primitive that is selected be applied to all features in the entity set.

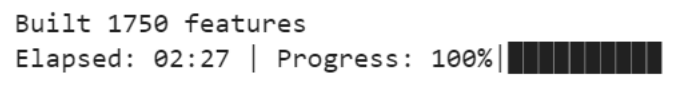
However, we specify what features we want to apply the multiplication primitive on since we do not have enough computing power to create all the interactions.

We implement multiplication to the following randomly selected raw features; *TransactionAmt, dist1, D2, D4, D10, C1, C5, C6, C11* and *C13*.

Deep Feature Synthesis

After specifying all details required to use Featuretools we run the DFS, binding everything we have specified up to this point. DFS uses primitive stacking in order to generate the deep features. The depth is defined as the number of primitives that are used to make a new feature. An example of this is that if we took the absolute value of one feature and multiplied it with another feature, the newly generated feature would have a depth of two because two primitives are used. An example of such a feature is; *TransactionDT*(ABSOLUTE(TransactionAmt))*. In our experiment we use a depth of two.

Simple preprocessing was applied after the DFS. Single value features were removed as they had low variance and no predictive power. Additionally, we impute new missing values that have arisen, and label encodes newly generated boolean categorical features to a numeric format.



```
Built 1750 features
Elapsed: 02:27 | Progress: 100%|██████████
```

Figure 3.10: Output from the DFS function

From Figure 3.10 have we illustrated the output after running DFS on our dataset. It took us only two and a half minutes to generate 1750 features, fast and efficient.

3.3.4 Feature Scaling

A common practice within supervised learning is to scale and normalize the different features to the same range. For example, transaction amount would have a higher interval of values than age. Normalization will help ensure that all the features are in the same range. Some learning algorithms are sensitive to scaling, whereas others are not.

In our case we do not normalize the dataset for Naïve Bayes, Logistic Regression or the tree-based ensemble methods which are not sensitive to variance in the data (Thenraj, 2020). Furthermore, it is proven in research that the accuracy of the

XGBoost becomes worse when normalizing rather than using the raw data (Borkin et al., 2019).

3.4 Reduction Methods

Feature selection is an essential topic in classification as it may have a considerable effect on the accuracy of the classifier (Karabulut et al., 2012). We add another layer of complexity to our supervised methods when doing feature engineering because of the dimensionality problem. It is vital to realize the trade-off between model complexity (number of features) and accuracy. A reduction in features increases accuracy and performance because the excess features can be noise (Belkin et al., 2019). Automated feature engineering generated an exhaustive amount of new features as previously shown. To reduce dimensionality, we attempt to implement various reduction methods described below (Koehrsen, 2018).

- **Collinear feature selection** is a deterministic method that finds collinear features in the dataset. For each pair of collinear features, the method identifies and deletes one of them. We specify a threshold for collinearity for where we want the model to delete features.
- **Zero important feature selection** is a non-deterministic method that uses gradient boosting to assess the feature importance of each feature in the data. In a tree-based model, these features are not used to split any nodes, and thus we can remove them without losing model performance.
- **Low importance removal** builds on the zero-importance feature selection method. It finds the lowest important features which do not contribute to the total importance based on a predefined threshold. For example, we set a threshold to find how many features we need to achieve a certain amount of variance in the data. This is a trade-off between complexity and variance, an important topic within supervised learning.
- **PCA** is a dimensionality reduction method that aims to enhance strong patterns in data. Through the use of a technique called *eigenvalue decomposition*, PCA aims to create features that maximize the information captured, while also keeping the dimensions to a minimum.

We first attempted to implement PCA to the data, but it was impossible to fit in memory due to the large size of the dataset. Secondly, we tried to implement different batch sizes to the PCA to control memory usage. We see a significant dip in model performance from this method and a significant increase in the false positives, thus resulting in the discarding of this method.

Going further, we tried to implement another reduction technique called zero importance feature selection. This method uses an implemented algorithm for feature selection, typically a decision tree algorithm and in our case it was based on XGBoost. Implementing this algorithm we managed to cut down from 1750 variables to 400. However, at the cost of a significantly lower area under the curve (AUC), potentially due to the high correlation that can misguide the feature ranking for these algorithms (Tolosi, Lengauer, 2011).

Going further, we implement a cut-off to remove highly correlated features. Through this method, we drop the highly correlated features above our pre-set threshold of 0.9 (90% correlation). From the literature, it can be shown that algorithms such as Random Forest or gradient boosting models can generate misleading feature ranking when the training dataset contains large groups of correlated features (Tolosi, Lengauer, 2011). This method was also addressed in other research (Haixiang et al., 2017), where collinearity was reduced through removing highly correlated features. On the contrary, it can be argued as a naïve method to drop all highly correlated features above a certain threshold as there may be good relationships or features we lose among all the noise. In our case, this method was the only way to reduce the features to go ahead with other selection methods, but we are aware of the limitation this method could possess.

After cutting down the feature space with the correlation method, we were left with around 700 features. At this point we implemented the zero important feature selector again. Additionally, we implemented low importance removal to keep the variables that explain 98% of the total variation in the data, leaving us with 306 features. Appendix 6 illustrates the cut-off graph for the number of features to keep.

Our selection methods were performed on a trial-and-error approach to see which method worked the best for our automated feature engineering dataset as there is no "silver bullet" method for feature selection (Jović et al., 2015). We select our method based on the AUC score and number of false positives by trying different approaches, making us choose the most beneficial method based on performance gain. An important notion is that we only perform reduction techniques to the automated engineered data for our experiment because the manual and baseline dataset has a much lower feature space.

3.5 Class Imbalance Problem

Imbalanced target class

"A dataset is imbalanced if the classes are not approximately equally represented." (Chawla et al., 2002)

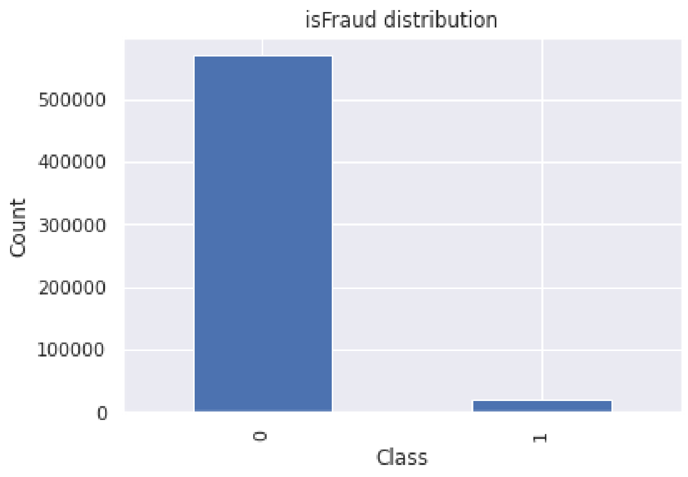


Figure 3.11: Target class (*isFraud*) distribution

As for most fraud datasets, we encountered the problem of class imbalance illustrated in Figure 3.11. After the preprocessing, fraudulent transactions accounted for no more than 3.67% of the observations in the data. Most supervised algorithms learn best when the target class is equally distributed. When there is a high imbalance, the algorithms tend to be biased towards the majority class and predict almost none of the observations from the minority group. If none of the fraudulent transactions were predicted would the model still return an accuracy of 96%, falsely suggesting that the model is performing exceptionally well.

3.5.1 Handling Class Imbalance Problem

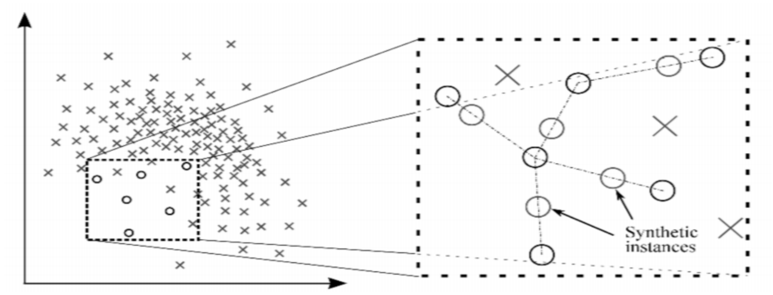


Figure 3.12: Illustration of SMOTE (Walimbe, 2017).

SMOTE is an oversampling technique that uses information about the already known anomalies and attempts to generate new observations of the minority class up to a given percentage (often a 50/50 distribution). This way, the model fits the data to reflect the underlying information better and more accurately detect actual anomalies. A typical outcome for highly imbalanced data when not using a sampling technique is that the model may believe that the minority class is an outlier.

The feature that makes SMOTE different from other over-sampling techniques that use replacement is that new observations are generated using nearest neighbor techniques to the minority class (Chawla et al., 2002). This way, the new observations are related to the central sample and no outliers are generated, thus lowering the risk of inducing any overfit from adding new observations (Liang et al., 2020).

We apply *SMOTE* after preprocessing to combat the problem of class imbalance, as shown in research. By not implementing any sampling strategy, the result will be inaccurate and not reflect the actual patterns in the data (Caldeira et al., 2014). Research has shown that datasets with many observations have better accuracy (Elreedy, Atiya, 2019). SMOTE is performed as the last step before modeling and is only applied to the training data. The complete Python code can be found in Appendix 2

4. RESEARCH METHODOLOGY

This chapter is divided into two parts. The first section presents the theoretical framework for the supervised algorithm and how we selected the model to use in our evaluation. The second section includes what kind of metrics we used to evaluate the performance of the different models.

In Figure 4.1, we have illustrated the architecture for this chapter. This chapter will introduce the last part of our methodology, which makes us quantify and interpret the results from automated feature engineering.

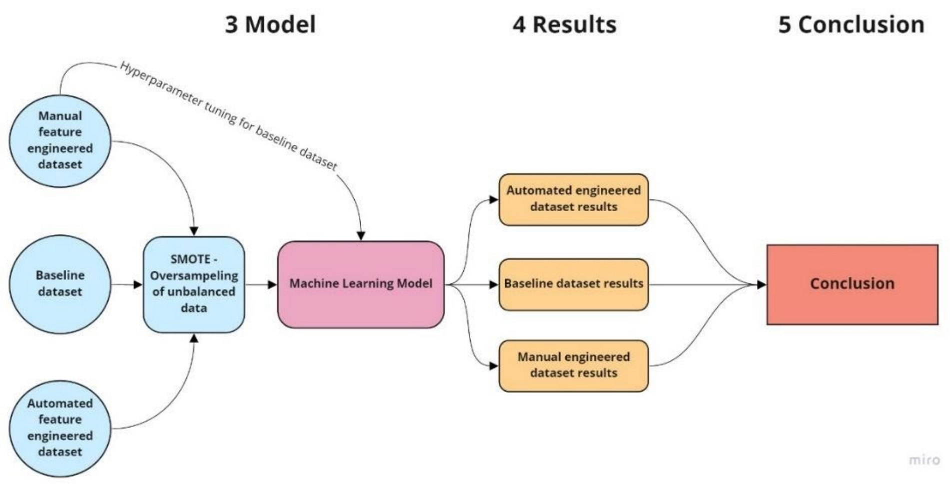

3 Model
4 Results
5 Conclusion
Hyperparameter tuning for baseline dataset
Manual feature engineered dataset
Baseline dataset
Automated feature engineered dataset
SMOTE - Oversampeling of unbalanced data
Machine Learning Model
Automated engineered dataset results
Baseline dataset results
Manual engineered dataset results
Conclusion
miro

Figure 4.1: Model and evaluation architecture

4.1 Machine Learning

Machine learning can be grouped into four categories; Supervised Learning, Semi-Supervised Learning, Unsupervised Learning and Reinforcement Learning (Pedregosa et al., 2019). For this thesis, we only consider supervised learning.

For this experiment, our goal is to identify a supervised model that can be used to evaluate if additional features generated by automated feature engineering would make a difference in discarding false positives. The target feature can either be classified as "0", a genuine transaction, or "1", a fraudulent transaction. It is essential to choose a supervised method that adapts well to our data's characteristics and can generalize to perform well for new, unseen data.

Different machine learning models serve different purposes and make different assumptions about data. Based on the previous application within related literature, we choose to include four different models in our test (Jain et al., 2019) (Wedge et al., 2017). The following section introduces each of the algorithms evaluated.

4.1.1 Logistic Regression

The Logistic Regression model was evaluated for our experiment as it performed on a moderate level concerning the reduction of false positives and had the lowest training costs of all the tested models (Jain et al., 2019).

The Logistic Regression method is standard within classical statistics and is considered one of the best methods for a binary classification problem (Geron, 2019). The Logistic Regression is based upon the logistical probability function described by Equation 4.1.

In detail, the model assumes that for each potential outcome of the dependent variable (y), the probability of y = 1, is P and y = 0 is equal to (1 – P).

$$P(X) = \frac{e^{(b_0 + b_1 X_1)}}{1 + e^{(b_0 + b_1 X_1)}} \quad (4.1)$$

$$log\left(\frac{P(X)}{1 - P(X)}\right) = b_0 + b_1 X_1 \quad (4.2)$$

Consider an example; if we attempt to predict if there will be rain tomorrow, the outcome is limited to the number of potential outcomes (the number of classes). In this case, the outcome is either rain (y = 1) or there will not rain (y = 0), namely a binary classification problem. When the logistical regression model estimates the probability of an event, it transforms the problem into a categorical form based on a threshold value being 0.5 as default (For example, "1" if the probability > 0.5, and "0" if the probability is < 0.5) (Hosmer, Lemeshow, 2000).

The logistical regression coefficient must be estimated using maximum likelihood estimation (MLE) (Brownlee, 2016), which is illustrated in Equation 4.2. The idea behind MLE is to find the coefficient of β_0 and β_1, such that the probability predicted $\hat{p}(x_i)$, using Equation 4.1 corresponds to the observed probability in our data (James et al., 2019).

Logistic Regression is one of the simplest and fastest algorithms to implement and train in machine learning and can be viewed as a baseline for many classification problems. The low variance makes it less prone to overfit where the classes are clearly separated. Moreover, the model can generalize to multiple classification problems instead of binary, and it does not consider the distribution of the classes within the feature space. The main drawback of the model is the risk of overfitting when many of the features in the training data are highly correlated. (Howbert, 2012).

4.1.2 Naïve Bayes

Bayes theorem

The Naïve Bayes method is based on Bayes Theorem, a formula illustrated in Equation 4.3 that determines the probability by estimating the frequency of values and a mix of values in the previously collected data. Moreover, it provides the probability of an event happening, given the probability of another event that already occurred (Tan et al., 2013).

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)} \quad (4.3)$$

Bayes theorem is commonly used to solve classification problems and thus we evaluate its performance on our dataset in this thesis. We let X denote a set of attributes, and Y denote the class. P(Y) is the *prior probability* calculated from the training dataset by the fraction of data associated with each class. We then define the *class conditional probability* denoted P(X|Y). Finally, we need to learn P(Y|X), which is the *posterior probabilities* for all X and Y combinations based on information drawn from the training data. However, it is not a straightforward task estimating P(X|Y). Thus, we introduce the Naïve Bayes to solve this issue.

Naïve Bayes Classifier

The Naïve Bayes classifier uses prior knowledge of the classes combined with new information gathered from the data. The approach can be considered a relatively simple method but may still outperform the more advanced classification methods. Another key attribute is the speed and accuracy when applied to a large dataset (Han et al., 2011).

Naïve Bayes classifier has conditional independent assumptions, meaning it assumes conditional independence between the attribute values P(X_i|Y). Based on this assumption, we calculate P(X|Y) using Equation 4.4 with much less effort than Equation 4.3.

$$P(X) = \frac{P(Y)\pi_{i=1}^{f} P(X_i|Y)}{P(X)} \quad (4.4)$$

f superscript - the number of features.

Because the machine learning classifier is supervised, both probabilities P(Y) the prior, and P(X_i|Y) the attribute probabilities can easily be calculated by counting the occurrences from the training data.

$$P(Y) = \frac{1}{\sqrt{2\pi\sigma_y^2}}\, exp^{-\frac{(X_i - \mu_y)^2}{2\pi\sigma_y^2}} \quad (4.5)$$

There exist many forms of Naïve Bayes, and among these, the Gaussian Naïve Bayes is one of the most popular methods. Compared to the Naïve Bayes, Gaussian Naïve Bayes assume that each feature's likelihood obeys the normal distribution rule (Tan et al., 2013). Equation 4.5 illustrates how this is calculated.

The Gaussian Naïve Bayes makes classification easier as it only uses the mean and standard deviation from the training data. Furthermore, the advantage of the classifier is that it works better with less data than other methods, and it provides faster computational time. Additionally, Gaussian Naïve Bayes is limited when dealing with highly correlated features as it assumes independence (Vadapalli, 2020)

4.1.3 Random Forest

Bagging

Bagging is an ensemble technique also referred to as Bootstrap Aggregation. This technique reduces the predictions' variance by combining multiple model predictions on different sub-samples of the same dataset. One item is chosen as part of the sampling volume and the item is then introduced back to the original dataset so it may be chosen in the next pass. After repeating this process multiple times, we get many different models on different volumes. We combine all these learners through a deterministic averaging process in the end.

Random Forest

Random Forest is a classifier consisting of many single decision trees. It uses bagging (Bootstrap Aggregation) and randomness when constructing individual trees to build an uncorrelated forest of trees. The combination of all the tree's single predictions is aggregated into one result which is more accurate than the individual decision of the predictions from a single tree. All the individual trees in the Random Forest return a class prediction and can be trained in parallel. The class with the most votes from all the trees combined becomes the model prediction. Figure 4.2 illustrates this case, here class 0 has six votes, and class 1 has three, and thus will the final prediction become 0 for the model (Yiu, 2019).

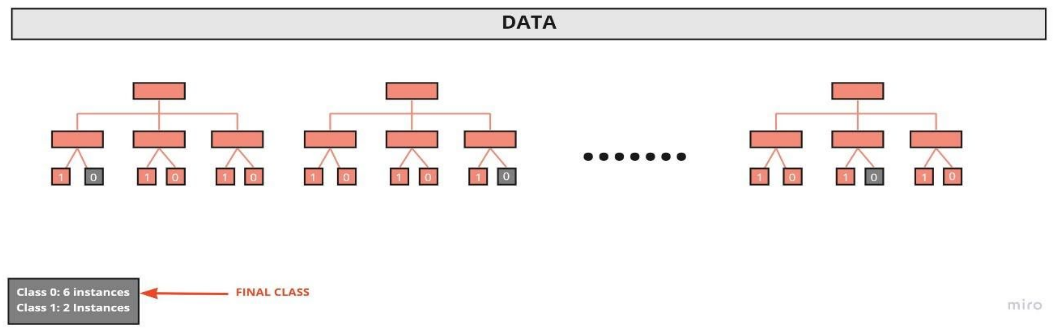


Figure 4.2: Random Forest Voting Scheme

In general, performance increases with RF since the variance is reduced through combining low-correlation decision trees. Additionally, the data is partitioned at random in each split and not by any measure of information gain contributing to further randomness.

Furthermore, Random Forest requires very few parameters. There are only two parameters that need consideration; the number of trees to grow and the number of variables to consider at each node. It is worth mentioning that the default settings tend to perform so well that it hardly needs any parameter tuning (Bentéjac et al., 2019).

4.1.4 XGBoost

Boosting

Boosting involves using multiple weak learners (i.e. decision trees and similar) to create one strong learner that can return a better result than each individual learner. The main difference compared to the bagging method mentioned above is how the elements are weighted and chosen. Bagging operates with equal-weighted elements, meaning they have the same probability of appearing again, while boosting increases the weight for misclassified data, encouraging the model to learn complex patterns better. The boosting method trains the weak learner sequentially. Thus, each learner tries to do a better job than their predecessor by adding more weight so the next hypothesis is more likely to classify the sample correctly. Lastly, the final prediction is the weighted average of all sequential predictors (James et al., 2013).

XGBoost

XGBoost is short for eXtreme Gradient Boosting. The method is an advanced form of gradient boosting published by Chen and Guestrin (Tianqi, Guestrin, 2016). XGBoost tries to combine weak learners to create a strong learner, much like the boosting method above. During the learning process, weak learners are generated. At each stage the weak learner calculates the value or class label and returns a loss (difference between the actual value and the predicted value). The magnitude of the loss creates a new weak learner, which again trains on the remaining errors. This process continues until a threshold is met. Moreover, this process is referred to as gradient descent optimization, where the gradient boosting name comes from.

The benefit of the XGBoost is that it has better regularization than normal gradient boosting, reducing overfitting. Since XGBoost allows for parallel processing through GPU, it is much faster than the normal gradient boosting (Brownlee, 2016). Moreover, a normal gradient boosting is a greedy algorithm since it stops splitting the node when encountering a negative loss. XGBoost on the other hand stops splitting when it reaches the predefined limit of max depth. A helpful attribute built into the XGBoost is cross-validation, making it better at determining the number boosting rounds. It is important to note that when using XGBoost, a range of hyperparameters needs to be tuned to achieve the desired results, a potential downside compared alongside algorithms such as Random Forest and Logistic Regression.

4.2 Model Selection

4.2.1 Cross-validation

Cross-Validation (CV) is a statistical method used to estimate the performance and skill of the ML model (Brownlee, 2018). It is a commonly used tool to assess and select different ML models for a given classification problem.

Since we deal with time-series data in our thesis, we implement a time series split with the k-fold method. For example, if the dataset contains [1, 2, 3, 4, 5] and we do a k-fold CV with a time series split, we would follow these rules (Shrivastava, 2020).

- Every test set contains unique observations.
- Observations from the training set occur before their corresponding test set. In other words, the test data must be ahead of time compared to the training data.

For example, we get the following if a dataset contains five observations.

- Training: [1] test [2],
- Training [1, 2] test [3]
- Training [1, 2, 3] test [4]
- Training [1, 2, 3, 4] test [5]

  By averaging the k-fold's, we get the average of the performance metric used, for example, AUC.

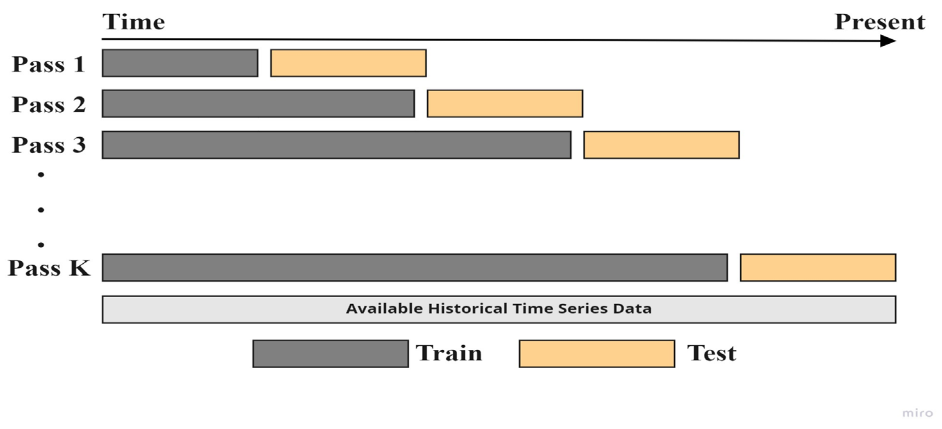


Figure 4.3: K-fold CV with time series split

In Figure 4.3, we illustrate how cross-validation could be performed.

Keeping our research question in mind, the mission is to find a model that could work as a tool to quantify the results from using automated feature engineering, to see how this approach affects the FPR. Therefore, k-fold cross-validation was implemented to select the model which best fitted our raw data.

Our CV makes different Receiver Operating Characteristics (ROC) curves based on different folds from the training data. Next, we average all the folds to get the mean AUC, an excellent metric for validating and comparing different models (Forman, Scholz, 2010).

The reason for performing the ROC validation was the massive imbalance in the dataset. Other more standard and straightforward metrics such as accuracy, recall, and precision could potentially be misleading, which will be explained more extensively in the section of model evaluation.

None of the models applied in the CV will be tuned. That way, we ensure the performance is out of the box performance for all models. A notation, the cross-validation was only performed on the baseline dataset. This way, we could select the model for use in the rest of the evaluation.

4.3 Model Tuning

For machine learning models one can generally say that discovering the optimal tuned parameters is both time-consuming and computationally exhaustive. Applying gradient boosting methods requires a lot of parameters tuning as the default parameters of XGBoost are not optimal. In gradient boosting, the number of boosted trees, learning rate, and the maximum depth of the trees make the model more robust. Not paying attention to the parameters can make wrongful conclusions from the model training as it may be overfitted and unable to learn. The parameters can be divided into two groups; hyperparameters and consistent parameters.

Consistent parameters

Some parameters stay constant through the whole training process and are usually training specific and chosen based on your objective goal and what we want to predict. In particular, we need to specify a binary loss function. In addition, specifying what type of hardware we want to run the model on is essential. Some models run on GPU and others on processors. Through the use of GPU parallel processing, the runtime is drastically reduced.

Hyperparameters

Hyperparameters are parameters that control the model and may have a substantial impact on performance. As machine learning models get more sophisticated, the number of hyperparameters to tune increases. Ideally, a hyperparameter optimization method like *grid-search* would be applied to find the best parameter configuration since this method finds the global optima. Due to grid-search being a brute force algorithm, employing it requires extensive computational powers. Only people with supercomputers could perform such a task (Chollet, Allaire, 2018), and thus sub-optimal for our purpose. As a result, suitable tuning methods have increased as there are many methods to choose from when tuning a model rather than using grid-search. Essentially, the tuning algorithm can be seen as an optimization tool trying to decrease the loss as much as possible. The decrease happens when an optimizer loops over multiple values of different hyperparameters and reports the lowest loss of all the combinations it tried at the end of the search (Bissuel, 2019).

4.3.1 Tuning Gradient Boosting Machine

Consistent Parameters

We use the XGBoost library in python to build a well-performing model. For the constant parameters, there is only “boosting” we need to specify. This parameter is set to *gbtree,* which is the gradient descent of tree types that penalize complexity.

Hyperparameters

We utilize *Bayesian Optimization* to tune the XGBoost model. The reason for choosing this method is the ability to include more features to tune and its low computation time. We were limited by using standard computers for this experiment and could not use other methods such as *Grid-Search* or *Randomized-Search*. Bayesian Optimization considers past evaluation when choosing which hyperparameter set to evaluate in the next iteration. It then chooses the combination in an informed way which makes it able to focus on the parameter space which is believed to bring the most promising validation score. In other words, the Bayesian optimizer makes “bets” on which mix of hyperparameters are more likely to achieve the best objective function until it has reached the pre-specified limit of iterations (Kapil, 2019). Consequently, this method requires fewer iterations than other methods because it disregards areas of the parameters space it believes will not bring any extra performance to the evaluation. It also provides faster results than similar methods and surpasses human experts at selecting hyperparameters (Snoek et al., 2012).

Methods such as *Grid-Search* and *Randomized-Search* use much time to evaluate different hyperparameters completely uninformed of previous iterations it has made. This forces the methods to spend a lot more time and a considerable amount of computing power evaluating inferior hyperparameters (Koehrsen, 2018).

The hyperparameters tuned for the XGBoost models are:

1. *Number of estimators (N_estimators)*
2. *Max tree depth (max_depth)*
3. *Learning rate (learning_rate)*
4. *Minimum child weight (min_child_weight)*
5. *Colsample by tree (colsample_bytree)*

4.3.2 Controlling for Parameters

In our experiment, model settings are controlled for both constant parameters and hyperparameters. We only tune the model for the baseline dataset, and the best parameters will then be applied to all models. We find this beneficial because we get a more accurate image of the isolated effect automated feature engineering has on the FPR. If we had different hyperparameters for each model, it could potentially return too optimistic results regarding automated feature engineering, naturally something we do not want.

4.4 Model Evaluation

4.4.1 Bias-Variance Dilemma

One key decision to make when working with machine learning is how to validate your models. Validation gives you insight into the model but can also return reasonable indications for how unbiased and generalized the performance is.

Bias is the case where the model makes assumptions that are far from reality. This could happen using the wrong learning algorithm, which is different from the relationship between the dependent and independent features. A high bias model will not learn the underlying pattern of the training data and therefore return a high loss during training and validation. For example, a linear model is less flexible to more complicated problems and thus often results in poor performance for more complex problems. Bias is popularly referred to as the state of underfitting.

Variance is the case where the error is caused by fluctuation in the training data. Ideally, do we want the model to perform the same regardless of the data the model has trained on. However, this may not be the case if the model is too sensitive and captures random patterns that only appear in the training data. Thus, variance occurs when the learning function varies significantly with the data used for training. This state is referred to as overfitting.

Mean squared error/ Total error represents the general noise in the data resulting from zero important features or randomness. This error can be reduced during the preprocessing stage.

The bias-variance tradeoff is the level of bias and variance in the underlying model. A complex model usually has high variance and low bias and can for example be a tree model. On the contrary, a low complex model has low variance but high bias and can be a regression for example. A model cannot be both high and low on complexity, therefore is called a trade-off between bias and variance.

Figure 4.4 can we see how bias and variance change based on the complexity of the model. There is usually an optimal model complexity that minimizes the square error by balancing bias and variance (Doroundi, 2020). Finding the equilibrium is often a difficult task due to the underlying target function being unknown, and it would not be possible to estimate the exact bias and variance errors (Singh, 2018).

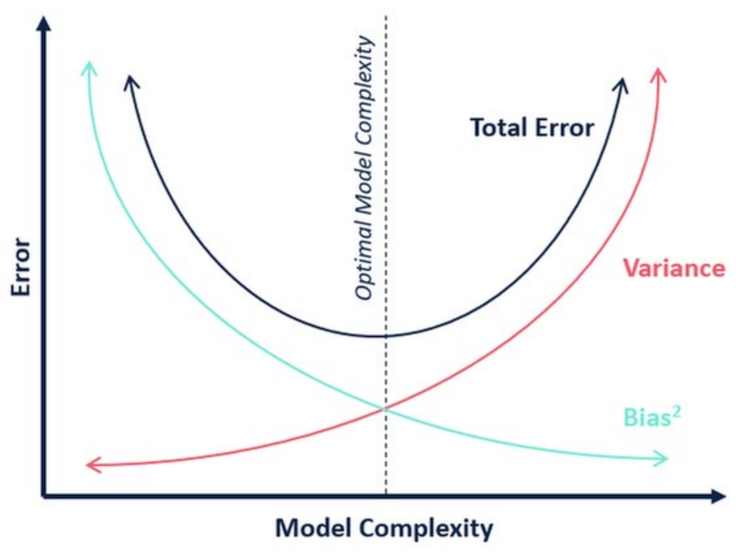


Figure 4.4: Illustration of bias-variance tradeoff (Hulgol, 2020).

How can we deal with over- and underfitting, and how to detect it?

K-fold cross-validation is an excellent way to investigate the bias-variance tradeoff and also ensure that the model has a low error. It is vital to choose the proper value of *K* so the testing procedure returns the best possible estimation of K.

To assess the bias-variance characteristics with the k-fold cross-validation, we check the error output with each calculated fold being one error. We find the bias from the mean of all the k-fold errors. To check for variance do we compute the standard deviation of the errors. A resulting high number indicates that the performance overfits the data greatly, something we do not want.

To deal with the overfitting we employ techniques such as feature reduction. Manually removing irrelevant features by removing collinear features and applying regularization methods can help the model generalize better. Another approach would be adding more trees to the Random Forest or the gradient boosting model, which could potentially help.

In our case, we include a gradient boosting model that is robust against bias and variance. The prediction from the gradient boosting model is the weighted average predictions yielded from multiple weak learners where the best model iteration with the lowest variance is chosen of all the weak models (Bühlmann, 2012), thus contributing to tackling high variance. Additionally, the error prediction is reduced by focusing on the bad predictions and then modeling them better in the next iteration, thus reducing the bias.

Another important consideration is the presence of imbalanced data. The reduction of bias requires equal representation of all group outcomes. As discussed in the data processing chapter this was solved by implementing an up-sampling technique (*SMOTE*) to balance the target class for the train data.

4.4.2 Performance Measures

In our research, AUC is implemented to evaluate the classifiers. We look at the change in FPs (ΔFP) and the change in TPs (ΔTP) as performance measures for evaluating how the different datasets impact the model in the domain of false positives. In supervised learning literature, standard metrics are accuracy, precision, recall and F1. However, in our case these performance metrics are not a good fit as we are dealing with highly unbalanced data. In this domain there are a few positive targets per million transactions. Consider a situation with 10 positives and 1 million transactions. If the algorithm classified every transaction as “normal” the accuracy metric would be above 99%. This is not a helpful detector as it detects none of the fraudulent transactions. On the contrary, if the classifier detects all the 10 positives but has 200 FPs, then the precision of that model would be 0.024, which may be seen as a bad result at first sight but the model can still be good. Recent research also supports this claim that precision, accuracy, recall, and F1 score should be avoided as they could be biased when classes are highly unbalanced (Luque et al., 2019).

The **Confusion matrix** returns the output of a machine learning classifier for binary or multi-class problems. It is helpful to get an overview of model performance by looking at the difference between predicted and actual values. In most cases it is used for further calculations of the FPR, TPR, accuracy, precision, recall and F1 score. Appendix 5 includes a brief description of the measures not applied but have context.

Confusion Matrix	0 (Predicted Negative)	1 (Predicted Positive)
0 (Actual Negative)	True Negative (TN) Classified as not fraud and was not fraud.	False Positive (FP) Classified as fraud but was not fraud
1 (Actual Positive)	False Negative (FN) Classified as not fraud but was fraud	True Positive (TP) Classified as fraud and was fraud

Table 4.1: The confusion matrix scheme

The difference in FPs and TPs

It is beneficial to know the difference between the FPs and TPs after implementing the new feature compared to the initial baseline. The differences are estimated as described in Equations 4.6 and 4.7. The subscripts F and B denote the FPs and TPs from the manual and automated engineering (F), and the FP's and TPs from the baseline (B). Note that the difference can be either negative or zero.

$$\Delta FP = FP_F - FP_B \quad (4.6) \qquad \Delta TP = TP_F - TP_B \quad (4.7)$$

False Positive rate (FPR) and True positive rate (TPR)

After computing the confusion matrix from both the manual and automated feature engineered models we compare the FPR and the TPR rate. This metric comparison works as if the different models formed a single model. The metrics are described in Equations 4.8 and 4.9.

$$FPR = \frac{FP}{TN+FP} \quad (4.8) \qquad TPR = \frac{TP}{TP+FN} \quad (4.9)$$

ROC and the TPR/ FPR tradeoff

The ROC curve can be used to illustrate the classifier performance in the domain with unbalanced data (Fawcett, 2006) and (Phua et al., 2004). The x-axis represents the FPR and the y-axis the TPR. Since each dimension of the graph has a strict ratio, it does not depend on the class distribution. Thus, the plot does not change as the class distribution changes (Fawcett, 2006). In addition, it can be said that ROC curves examine a single classifier for multiple classification thresholds.

Figure 4.5 is an example of an AUC with two classifiers (Fawcett, 2006). We see that the curved line represents the different thresholds between TPR and FPR. The shaded area under the line is represented as the AUC. The diagonal line represents the random chance and has an AUC of 0.5. In other words, a model which follows the diagonal is no better in detecting something than a random flip of a coin. Thus, a working model will have a higher AUC than 0.5.

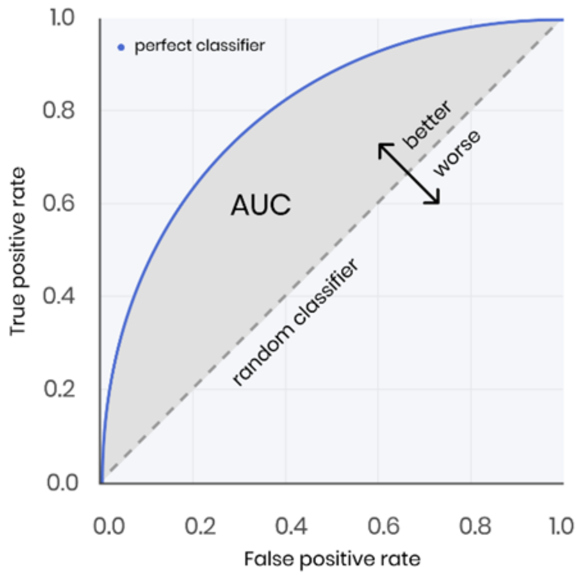


Figure 4.5: ROC curve of two classes

AUC

AUC is a single scalar value that is transformed from the ROC performance (Fawcett, 2006). The measure is a portion of the unit square and always has a range between 0 to 1. The better the AUC, the better is the model performance and ability to distinguish between negative and positive classes. As a reference for future results, an AUC between 0.90 and 1 is usually considered an excellent result, while between 0.50 and 0.60 are considered failures (Hanley, McNeil, 1982).

Matthews Correlation Coefficient

Matthews Correlation Coefficient **(**MCC) considers all four values in the confusion matrix. The measure takes values ranging from -1 to 1, where 0 means that the classifier is no better than a random flip of a coin. A value of -1 means a negative correlation, thus misclassification, and 1 means a perfect classifier. The Matthew measure is applied because it is perfectly symmetric and no class is more important than others. Hence, it can be seen as a more reliable statistical measure which only produces a higher score if the prediction obtained performs well in all four categories of the confusion matrix (Chicco, Jurman, 2020).

We choose to use MCC as recent studies have shown that this method was the best metric for highly imbalanced classes (Luque et al., 2019). As a reference for future result comparison, we find that 0.30 - 0.39 is a moderate positive relationship, 0.40 - 0.60 strong positive relationship and 0.70 or higher is a solid positive relationship (Powers, 2011). This metric is described in Equation 4.10.

$$MCC = \frac{TP\ x\ TN - FP\ x\ FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (4.10)$$

5. RESULTS

We have created three datasets to be compared against each other to conclude the viability of the automated feature engineering framework and its effect on the false positives compared to the baseline and manual engineered features.

This chapter will first choose which algorithm to use in our evaluation part, and this is done through a model selection process using CV. This way we can evaluate different models to make sure generalization and performance is equally good for out-of-sample use, and to be sure that the selected model is optimized. The test partition remained untouched through the whole model selection, tuning and training phase. Secondly, we go through the experimental setup for the chosen model. Finally, we will go through the result for each dataset and make a comparison between the results.

5.1 Model Selection

The first stage of the evaluation was to find which ML model that best fit our data. We calculate the ROC-AUC of all the four candidate models from out-of-box performance (no tuning). Using 10-fold cross-validation on the clean baseline data we receive the following results:

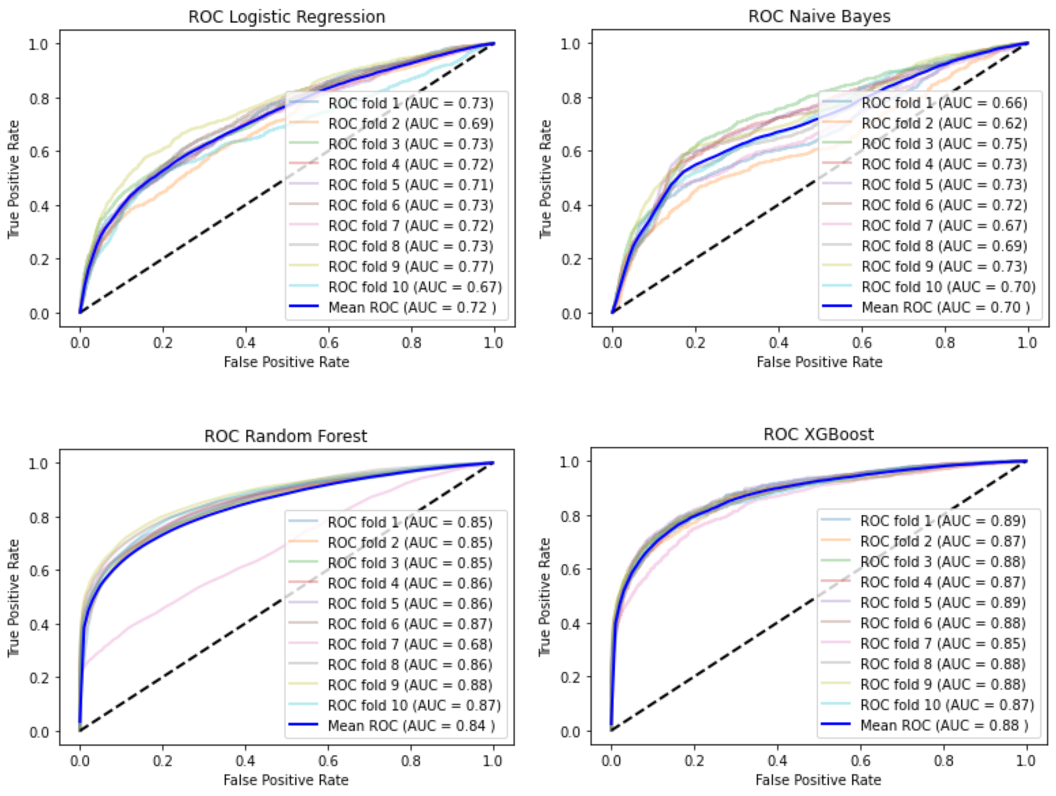


Figure 5.1: ROC-AUC 10-fold CV with time series split

From Figure 5.1 Random Forest and XGBoost demonstrate to be two worthy contenders as they have the highest AUC score on average compared to the four models. We can see that the different thresholds between FPR and TPR are better for these two, indicating that they have an overall lower FPR, which is what we want to see. Based on the results from the CV alone we choose XGBoost as the model to use in our experiment due to it having the highest AUC of **0.88,** indicating that in 88% of cases it can distinguish between the two classes.

5.2 Experimental Settings for the Benchmark Model

To evaluate the performance, we implement the model selected from the cross-validation stage. This model will be used as a reference to evaluate how the additional features generated by the automated approach influence false positives compared to the two other methods.

XGBoost was tuned according to the features in the baseline data, and we keep these hyperparameters constant for all models. We control for the hyperparameters to keep consistency and comparability throughout the research. Additionally, we only utilize one algorithm in the evaluation phase as our goal was to explore additional features' effects on FPR, rather than different algorithm effects on performance. Our benchmark model has the following setup.

Machine learning model:

XGBoost

Input features:

Baseline – 81 features

Manual engineering – 124 features

Automated engineering – 306 features

Total amount of observations:

(455 832) Original

(911 664) Upsampled using SMOTE

Hyperparameter Tuning:

Bayesian optimization for XGBoost (Appendix 4 contains the optimized values)

Objective:

Binary

Loss:
AUC
Early stopping:
200 rounds
Regularization:
L1

Table 5.1: Experimental settings for the XGB model

Summarizing Table 5.1; the model was first fed the preprocessed training data before adding the engineered features from each manual and automated method separately.

SMOTE was applied to the datasets in order to reduce the majority class bias. Using *Bayesian optimization* the model was trained and validated with the best combination of parameters to optimize the tree structure.

Furthermore, regularization measures to control for overfitting as previously discussed are implemented. Early stopping avoids overfitting by monitoring the performance of the test data. When the model does not improve after a fixed number of iterations the training is stopped. The L1 regularization is controlled through the alpha constant. We tried multiple values through a trial-error approach for the best results.

5.3 Experimental Setting 1 - Baseline

The baseline dataset was created using only the raw features. The dataset was only cleaned and pre-processed without adding any feature-engineered variables. In total, the baseline dataset contains 81 features.

5.3.1 Results

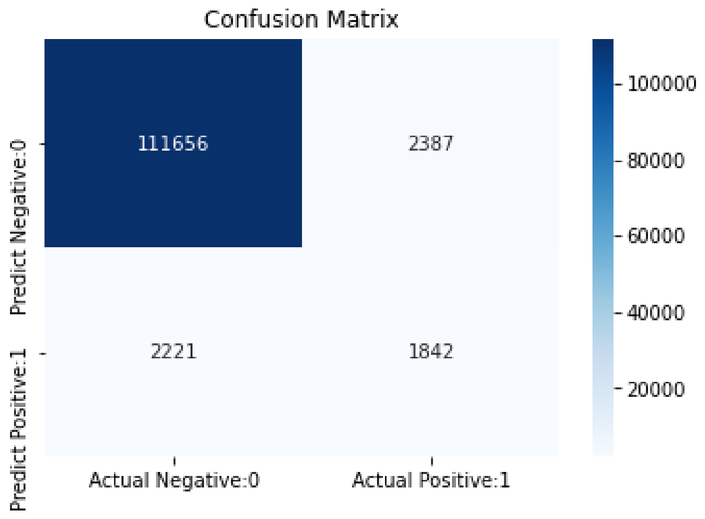


Figure 5.2: Confusion matrix of the baseline model

TPR	FPR	AUC	MCC
45.46%	**2.09%**	**0.87**	**0.43**

Table 5.2: Result metrics of the baseline model

Our baseline yielded a surprisingly good AUC score of **0.87**, indicating that our baseline can distinguish between the two classes in **87%** of the cases. Additionally, we got a strong correlation score of **0.43** with the MCC metric. These scores combined suggest that our classifier as an entity is considered to do a good job. From Table 5.2 we achieve an FPR of **2.09%**, which may seem small as a percentage but with 2387 FP values we expect it to be reduced through further feature engineering. Furthermore, we got a TPR of **45.46%**, which is not a high value, but it may be affected by the threshold between FPR and TPR, set to the default of 0.5. Appendix 8 includes a description of the details of thresholds.

It was interesting to get such a good result without any feature engineering, and the results exceeded our expectations. Reflecting on these results, it may suggest that the model has managed to pick up on some trend in the data despite the high sparsity of many of the original raw features, especially with the identification features containing on average above 80% missing values.

5.4 Experimental Setting 2 - Manual Feature Engineering

For our second dataset, we utilized the preprocessed baseline data and manually engineered new features to potentially improve the machine learning algorithm performance, thus decreasing the FPs. From the best of our knowledge, we leverage the raw features through our own insight, coupled with research done in the credit card domain. After the engineering process the dataset consisted of 124 features, an increase of 43 new features compared to the baseline.

The process of exploring and creating the additional features by hand was an exhaustive task, and we used the equivalent of 30 hours in total to explore and create these domain-engineered features.

5.4.1 Results

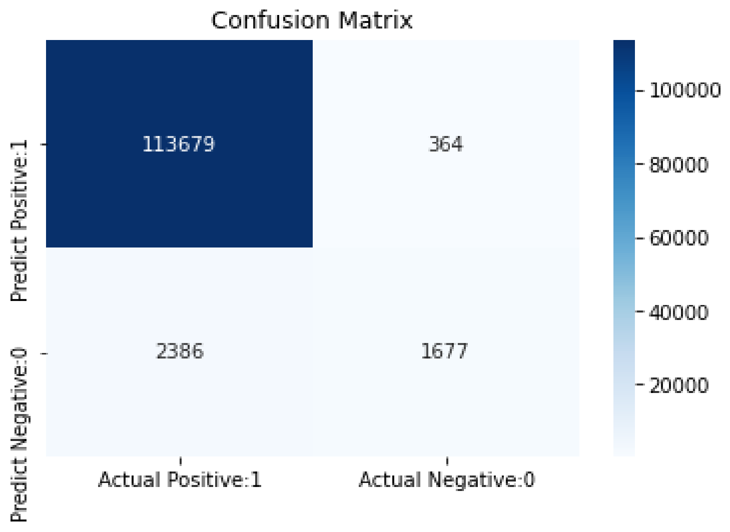


Figure 5.3: Confusion matrix of the manual model

TPR	FPR	AUC	MCC
41.28%	**0.32%**	**0.92**	**0.57**

Table 5.3: Result metrics of the manual model

Baseline		Manual Feature Engineering	
FP	TP	FP (Δ FP)	TP (Δ TP)
2387	**1842**	**364 (-2023)**	**1677 (-165)**

Table 5.4: Comparison between baseline and manual model

With respect to the results, the manual engineered model yielded an AUC of **0.92** (which is considered excellent), a moderate increase by **5.8%** compared to the baseline results. Furthermore, the model performed better as a whole compared to the baseline as the MCC increased to a correlation score of **0.58**, an increase of 0.15 from the baseline. As shown in Table 5.3 the manual feature engineering can discard a huge amount of the FP (-2023), but it has a penalty of discarding TP as well (-165). Thus, we manage to decrease the FP compared to the baseline by **84%** with the manual engineered approach. Looking at the results, the manual engineered model outperformed the baseline, suggesting that the new features give more information to the model, indicating that it managed to generalize better to the holdout data.

5.5 Experimental Settings 3 - Automated Feature Engineering

For our third dataset, we applied automated feature engineering to the baseline data. Our objective was to let the DFS algorithm create new features based on the raw data from the baseline, using the pre-selected primitives. The DFS algorithm yielded in total 1750 new features. Furthermore, we resolved the problem of the huge feature space by applying reduction techniques previously discussed in the model framework chapter. This left us with 306 features, an increase of 225 features compared to the baseline.

The creation process of the new features was done in approximately 2.5 minutes, although we used the equivalent of 5 hours to learn the basics and default settings of Featuretools, can it be said to be highly effective especially compared to the time we used on manual features engineering. Note that the five hours used is a one-time investment and further application of the Featuretool method in the future will be much more efficient.

5.5.1 Results

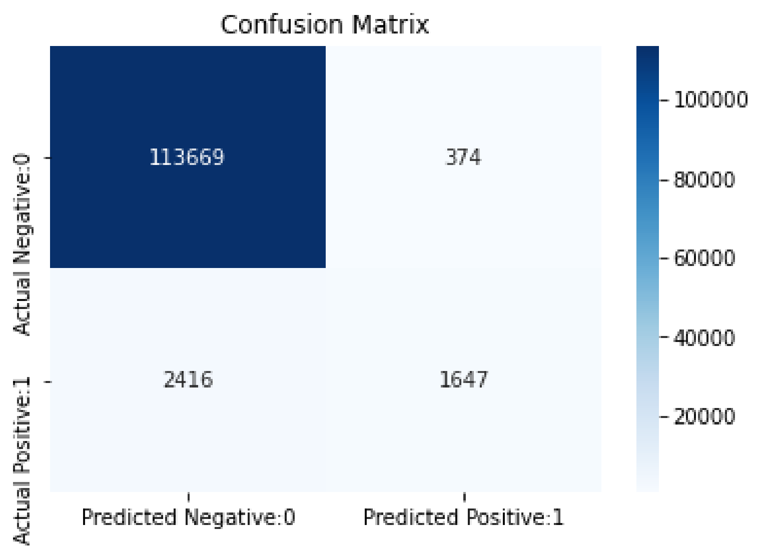


Figure 5.4: Confusion matrix of the automated feature engineering model

TPR	FPR	AUC	MCC
40.54%	**0.33%**	**0.92**	**0.57**

Table 5.5: Result metrics of the automated feature engineering model

Baseline		**Automated Feature Engineering**	
FP	TP	FP (Δ FP)	TP (Δ TP)
2387	**1842**	**374 (-2013)**	**1689 (-195)**

Table 5.6: Comparison between baseline and automated feature engineering model

As shown in the above tables the automated engineered method outperforms the baseline on all levels. The model yields an AUC of **0.92,** a moderate increase over the baseline by **5.8%**, although an equal score compared to the manual engineering approach. The MCC yields a correlation score of **0.57** which is a good improvement over the baseline and equal compared to the manual engineered approach. These results indicate that the automated model performs better for all groups in the confusion matrix than the baseline, but similar to the performance of the manual engineered model. From Table 5.6, we have discarded a lot of FPs from the initial baseline (-2013), but we have also lost some of the TPs (-195). In total, we reduced the false positives by **84%** compared to the baseline results. However, this reduction is equal to the manual engineered approach.

5.6 Comparison of the Results

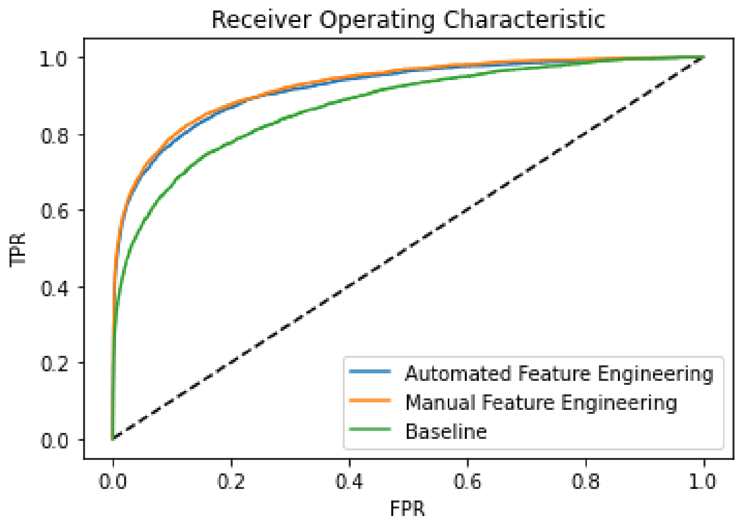


Figure 5.5: ROC curve comparison of the models

There is a clear distinction between performance of the baseline and the two other engineered approaches in respect to the trade-off between FPR and TPR, which can be seen in Tables 5.6 and 5.4. However, it is difficult to separate the two curves of the manual and automated engineered methods as seen in Figure 5.5 due to their almost perfect overlap.

It is interesting to observe that both approaches using feature engineering, in different ways and scales have almost equal performance on all levels in our comparison. The key takeaway from the results is that both the manual and automated approach is superior to the baseline regarding discarding false positives. However, both methods have similar performance when compared. In the light of this one might say that both methods help the model to learn and generalize better to the holdout data. However, it is impossible to divide between automated and manual approaches to say that one is better than the other just based on the quantitative results.

6. CONCLUSION

6.1 Discussion

Before discussing our results, we reintroduce our research question from the introduction chapter.

RQ 1: Do automated feature engineering decrease the FPR rate in fraud detection systems compared to doing no feature engineering?

RQ2: Does automated feature engineering better decrease the FPR in the fraud detection system than a manual feature engineering approach?

The results indicate that automated feature engineering reduced the FPs with **84%** compared to the baseline going from an FPR of **2.09%** to **0.33%.** Additionally, we managed to preserve most of the TPs as we only discarded **11%,** going from a TPR of **45.46%** to **40.54%** with the automated approach. The AUC and the MCC increased from **0.87** and **0.43** to **0.92** and **0.57**, indicating that the automated feature engineering model is an overall better classifier than the baseline. When starting this project our hypothesis expected that the automated engineering method would outperform the baseline. As seen from our results, this theory holds for our quantitative interpretation, aligning with a positive outcome for our RQ 1, that automated feature engineering decreases the FPR from the baseline of not doing any feature engineering.

Second, the results from automatic engineering compared to manual engineering were quite similar. Both methods reduced the FPs by **84%**. However, the manual method had the edge on preserving the TPs with a drop of **9%** (TPR of **41.28%**) compared to the automated approach at **11%** (TPR of **40.54%).** This minimal difference could be caused by random variation in the data. The AUC and the MCC scores are equal for both methods indicating that they have an equivalent classification performance. Based on the results, we do not find a significant difference between the two approaches in terms of reducing false positives or classification performance in general. This was surprising as our hypothesis expected that the automated feature engineering approach would have an edge on the manual engineered method. Based on the result from our experiment, and with RQ 2 in mind, there is no difference between manual and automated feature engineering in terms of decreasing the FPR.

Our results build on existing evidence from the study presented by Cornell University, (Wedge et al., 2017). Their method achieved a reduction in FPs of **54%** compared to their baseline of the study. From our results we have achieved a reduction of **84%** in FPs. Although not all factors are equal in terms of experimental framework and trade-off between FPR and TPR, we argue that both results significantly reduced the FPs from their respective baseline using DFS.

Furthermore, those our FPR results outperform those of Trivedi et al. (2020) comparison study on ML models within the fraud detection domain. Their study achieved on average an FPR of **4.3%** from all models tested with no feature engineering. Our results show a FPR of **0.33%** after automated feature engineering was applied. These results can be argued to be somewhat comparable as both methods are within the domain of credit card fraud detection, and show the benefit automated feature engineering can have when applied.

While previous research has focused on reducing false positives from the current solution of a banking corporation, our thesis has tried to explore the difference between manual and automated feature engineering within the reduction of false positives. As previously discussed, our results state that the difference between the manual and automated methods in relation to the FPR is insignificant. However, there is a significant difference in the number of hours used and the number of features created, influencing cost and labor, making one approach more reasonable than another for a banking corporation.

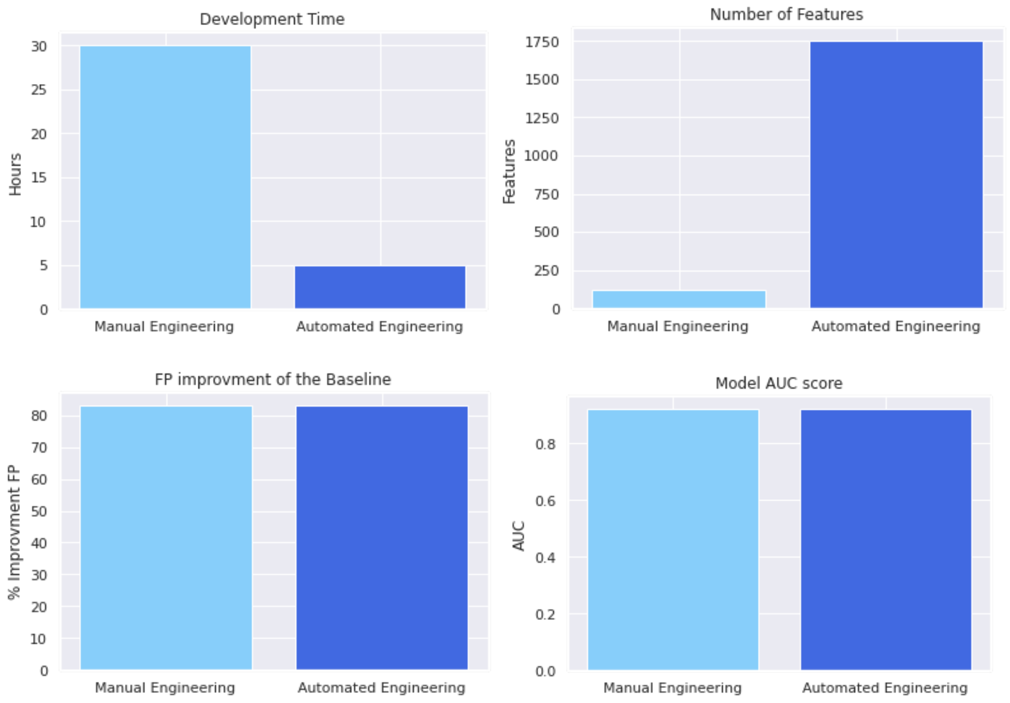


Figure 6.1: Summary of the result from the manual and automated approach

From Figure 6.1, we can see a summary of the results of both the automated and the manual approaches. An important observation is the difference in development time and the number of features created from the two approaches.

Engineering the manual features took a significant amount of time. We had to perform EDA to analyze the feature pattern of the raw data to figure out what kind of features would make sense to engineer. Thus, resulting in a very time-consuming approach to creating additional features. On the contrary, the DFS approach required that we specified the data source and the respective categories of the features. After the first pass of the data, the DFS algorithm generated a huge feature space within 2.5 minutes, a lot less than the approximately 30 hours we used to engineer a few features by hand manually. About 5 hours were used to learn the Featuretool basics, but this is considered a one-time investment.

Analyzing the results supports the theory that automated feature engineering is a better approach to reduce the false positives when taking the time it takes to perform manual compared to automated engineering into account. From a cost-saving standpoint, the use of automated feature engineering is a significant opportunity for larger cost savings. At the same time, are we able to get features a domain export may never have thought of creating or had time to implement. Although not all the features generated from the DFS are helpful, as our results suggest using feature space reduction methods seems very effective in dealing with this. Moreover, it is usually better to have more features than a few as it provides the option of using various selection methods to retain the most relevant features to the respective domain.

All of the above results should be considered when building a detection system for fraud prediction. As the results have shown, the DFS method is superior to the baseline but yields the same result as the exhaustive and time-consuming manual feature engineering approach. Most banking corporations utilize manual engineering today when updating expert rules, which are inefficient, humanly biased, and costly. Creating an automated pipeline may lower FPR and cut down on manual labor costs for companies within the financial industry as well as other sectors.

6.2 Limitations and Further Work

Although the results have shown that automated feature engineering could be helpful when discarding FP, there are some limitations to the solution presented, along with some exciting areas for further research.

The generalizability of our results is limited to the single-table approach, as we did not collect data from other sources with multiple related tables. From the discussion earlier, the Featuretool technique works both on single and multi-table situations and desirably both methods should have been covered to get the complete picture of the potential and how it may differ working with other dataset compositions. The performance and results may vary based on whether we have multiple tables or just one single table because we apply more aggregation primitives when dealing with multiple tables.

Nonetheless, from the results presented the findings validate our hypothesis for both of our research questions. Additionally, we touch upon the aggregation primitives by splitting out the transaction amount into a separate table. Further research is needed to establish if there is a difference in using automated feature engineering on multiple related tables and single tables.

Our knowledge and competency do have an impact on the reliability of the manually engineered features. Since neither of us are domain experts, there may be additional features that we have not thought of. We have done extensive research and created manual features that best fit our domain to the best of our knowledge. For further research, it could be interesting to include actual domain expertise to get a more honest picture of how the difference will unfold between automated and manual approaches.

Additionally, it may be interesting pairing domain experts and machine intelligence, where domain experts would craft features along with automated feature engineering as a tool. We expect that their superior knowledge would reduce feature space as they know which features to apply different primitives on. In contrast, we apply primitives at random as we had limited competency within the domain, resulting in a high number of features.

Due to the lack of computing power, we could not produce more than a certain amount of features with the DFS function. As Featuretools have many primitives, we have to choose wisely as we could not store all these features in memory. Additionally, we could not create a feature beyond the depth of 2, as it would not have fit in our available memory. Even though there were limitations on computational power we expect that the average person or company does not have supercomputers, making these results somewhat generalizable for most people.

In terms of XGBoost we would like to emphasize the limitations of tuning iterations for the hyperparameters. Due to the memory bottleneck as previously stated, we ran into computational- and time constraints. The number of iterations we ran could have affected the performance of the XGBoost. This indicates that there may be potential for further tuning of the hyperparameters. Furthermore, since XGBoost is so sensitive to changes in hyperparameters, results may change drastically with the optimal parameters, and further research with suitable supercomputers could investigate this.

For further research, it could be interesting to apply multiple datasets from various domains to see if automated feature engineering performs better in some business domains than others. This thesis was limited to exploring the domain of fraudulent transactions within e-commerce.

It was beyond the scope of this study to compare different supervised algorithms with the effect of automated feature engineering on false positives. However, it could be interesting for further research to try multiple models and see if the result changes based on algorithms applied.

6.3 Conclusion

In this thesis have we considered a framework of automated feature engineering for ML model within e-commerce fraud detection, with the aim of reducing the number of false positives. What distinguishes our approach from previous studies is that we consider the comparison between manual and automated feature engineering. In addition, we implement selection methods that are compatible with many types of ML models. Thus, our framework is generalizable to many types of financial fraud detection problems and could aid domain experts and other people within the industry. Using a dataset consisting of e-commerce transactions, we have shown that our automated feature engineering framework improves the AUC by **5.8%** and reduces the FPR by **84%** from the baseline solution with no feature engineering. Furthermore, our comparison between the manual and automated feature engineering methods did not find any significant differences of the effect on FPR. However, our results suggested that the automated feature engineering is significantly more time efficient and by implementing an automated strategy the deployment time may be heavily reduced.

REFERENCES

Zhou, Z.-H. (2012). Ensemble methods : foundations and algorithms. Taylor & Francis.R. Brause, T. Langsdorf, M. Hepp.: Neural Data Mining for Credit Card Fraud Detection, Proc. 11th Int'l Conf. Tools with Artificial Intelligence (1999) 103-106.

Aleskerov, E., Freisleben, B., and Rao, B. (1997). "CARDWATCH: a neural network based database mining system for credit card fraud detection," Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering

Pascual, A. Marchini, K. (2018). Overcoming false positive declines in e-commerce.In Javelin strategy and research reports.

Bahnsen, A., Aouada, D., Stojanovic, A., Ottersten, B. (2016). Feature engineering strategies for credit card fraud detection. Expert Systems with Applications Volume 51, 1 June 2016. Pages 134-142

Yu, W. F., and Wang, N. (2009). Research on credit card fraud detection model based on distance sum. IJCAI.

Kovach, S., and Ruggiero, W. (2011). Online banking fraud detection based on local and global behavior. The Fifth International Conference on Digital Society. Page 166–171.

Bhattacharyya, S., Jha, S., Tharakunnel, K., Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. Decision Support Systems, 50(3), 602–613.

Huang, J., and Liu, J. (2012). Intrusion detection system based on improved BP neural network and decision tree. 2012 IEEE 5th International Conference on Advanced Computational Intelligence, ICACI 2012. Page 188–190.

Devika, S. P., Nisarga, K. S., Rao, G. P., Chandini, S. B., Rajkumar, N. (2019). A research on credit card fraudulent detection system. International Journal of Recent Technology and Engineering. Page 5029– 5032.

Jain, Y., Tiwari, N., Dubey, S., Jain, S. (2019). A comparative analysis of various credit card fraud detection techniques. International Journal of Recent Technology and Engineering. Page 402–407.

Wedge, R., Kanter, J. M., Veeramachaneni, K., Rubio, S. M., Perez, S. I. (2017). Solving the false positives problem in fraud prediction using automated feature engineering. Machine Learning and Knowledge Discovery in Databases. Page 372-388

Davis, J., Foo, E. (2016). Automated feature engineering for HTTP tunnel detection. Computers & Security Volume 59, June 2016. Pages 166-185.

Bannett, C. (2017). Overcoming False Positives: Saving the Sale and the Customer Relationship. JAVELIN corporation.

Makki, S., Assaghir, Z., Taher, Y., Haque, R., Hacid, M., Zeineddine, H. (2019). An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection. IEEE Access Volume 7. Pages 93010 – 93022.

Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A. (2014). A survey on concept drift adaptation. ACM Computing Surveys vol 46.

Kotsiantis, Sotiris & Kanellopoulos, Dimitris & Pintelas, P. (2006). Data Preprocessing for Supervised Learning. International Journal of Computer Science. 1. 111-117.

Hellerstein, J. (2008). Quantitative Data Cleaning for Large Databases. UC Berkeley

Osborne, J. W. (2002). Notes on the Use of Data Transformation. - Practical Assessment, Research &amp; Evaluation, 8(6).

Chhabra, G., Vashisht, V., Ranjan, J. (2019). A Review on Missing Data Value Estimation Using Imputation Algorithm. Journal of Advanced Research in Dynamical and Control Systems. 11. 312-318.

Cerda, P., Varoquaux, G., Kegl, B. (2018). Similarity encoding for learning with dirty categorical variables. Machine Learning.

Kanter, J., and Veeramachaneni K. (2015). Deep feature synthesis: Towards automating data science endeavors. IEEE International Conference on Data Science and Advanced Analytics 2015, pages 1–10.

Chawla, N., Bowyer, K., Hall, L., Kegelmeyer, W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. J. Artif. Intell. Res. (JAIR). 16. 321-357. 10.1613/jair.953.

Belkin, M., Hsu, D., Ma, S., Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. PNAS August 6, 2019 116 (32) 15849-15854

Elreedy, D., Atiya, A. (2019). A Comprehensive Analysis of Synthetic Minority Oversampling Technique (SMOTE) for handling class imbalance,

Fawcett, T. (2006). An introduction to roc analysis. Pattern recognition letters, 27(8):861{874, 17

Phua, Clifton & Alahakoon, Damminda & Lee, Vincent. (2004). Minority Report in Fraud Detection: Classification of Skewed Data. SIGKDD Explorations. 6. 50-59.

Walter, SD. (2005). The partial area under the summary ROC curve. Stat Med. 2005 Jul 15;24(13):2025-40. doi: 10.1002/sim.2103. PMID: 15900606.

Bentéjac, C., Csörgő, A., Martínez-Muñoz, G. (2019). A Comparative Analysis of XGBoost.

Chollet, F. (2018). Deep learning with Python. Manning Publications Co.

Borkin, D., Nemethova, A., Michalconok, G., Maiorov, K. (2019). Impact of Data Normalization on Classification Model Accuracy. Research Papers Faculty of Materials Science and Technology Slovak University of Technology. 27. 79-84. 10.2478/rput-2019-0029.

Snoek, J. Larochelle, H. Adams, R. (2012). Practical Bayesian Optimization of Machine Learning Algorithms

Doroundi, S. (2020). The Bias-Variance Tradeoff: How Data Science Can Inform Educational Debates. AERA Open

Bühlmann, P. (2012). Bagging, Boosting and Ensemble Methods. Handbook of Computational Statistics. 10.1007/978-3-642-21551-3_33.

Bhaya, W. (2017). Review of Data Preprocessing Techniques in Data Mining. Journal of Engineering and Applied Sciences. 12. 4102-4107. 10.3923/jeasci.2017.4102.4107.

Whitrow, C., Hand, D.J., Juszczak, P., Weston, D., Adams, N.M. (2009). Transaction aggregation as a strategy for credit card fraud detection. Data Mining and Knowledge Discovery 18(1), 30–55

Breiman, L. Friedman, J. Stone, C. Olshen, R.A. (1984). Classification and Regression Trees

Toloşi, L., Lengauer, T. (2011). Classification with correlated features: unreliability of feature ranking and solutions, Bioinformatics, Volume 27, Issue 14, Pages 1986–1994,

Jović, A., Brkić, K., Bogunović, N. (2015). "A review of feature selection methods with applications," 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015, pp. 1200-1205, doi: 10.1109/MIPRO.2015.7160458.

Karabulut, E., Özel, S., İbrikçi, T. (2012). A comparative study on the effect of feature selection on classification accuracy ,Procedia Technology, Volume 1, Pages 323-327,

Haixiang, G., Yijing, L., Shang, J. (2017). 'Learning from class-imbalanced data: review of methods and applications', Expert Syst. Appl., 73, pp. 220–239

Agarwal, V. (2015). Research on Data Preprocessing and Categorization Technique for Smartphone Review Analysis. International Journal of Computer Applications. 131. 30-36. 10.5120/ijca2015907309.

Forman, G., Scholz, M. (2010). Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement ABSTRACT. SIGKDD Explorations. 12. 49-57.

Luque, A., Carrasco, A., Martín, A., Heras, A. (2019). The impact of class imbalance in classification performance metrics based on the binary confusion matrix, Pattern Recognition, Volume 91, Pages 216-231,ISSN 0031-3203,

Hanley, J.A., Mcneil, B. (1982). The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve. Radiology. 143. 29-36. 10.1148/radiology.143.1.7063747.

Powers, D. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

Chicco, D., Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics 21, 6

Carcillo, F., Pozzolo, A., Borgne, Y., Caelen, O., Mazzer, Y., SCARFF, G. (2018). A scalable framework for streaming credit card fraud detection with spark,Information Fusion, Volume 41, Pages 182-194, ISSN 1566-2535

Milo, T., Novgorodov, S., Tan, W. (2016). Rudolf: interactive rule refinement system for fraud detection. Proceedings of the VLDB Endowment. 9. 1465-1468. 10.14778/3007263.3007285

Chen, T., Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. 785-794. 10.1145/2939672.2939785.

Cutler, A., Cutler, D., Stevens, J. (2008). Tree-Based Methods. 10.1007/978-0-387-69765-9_5.

James, G., Witten, D., & Hastie, T., Tibshirani, R. (2013). An Introduction to Statistical Learning.

Baader, G., Krcmar, H. (2018). Reducing false positives in fraud detection: Combining the red flag approach with process mining, International Journal of Accounting Information Systems, Volume 31,Pages 1-16, ISSN 1467-0895,

Hosmer, D. Lemeshow, S. (2000). Summary The prelims comprise: Half Title Wiley Series Page Title Copyright Contents Preface. pp. 1-30, Introduction to the Logistic Regression Model

Tan, P., Steinbach, M., Kumar, V. (2013). Introduction to Data Mining-Pearson

Han, J., Pei, J., Kamber, M. (2011). Data mining: concepts and techniques. Elsevier

Liang, X., Jiang, A., Li, A., Xue, Y.Y., Wang, G.T. (2020). LR-SMOTE — An improved unbalanced data set oversampling based on K-means and SVM, Knowledge-Based Systems, Volume 196, ISSN 0950-7051,

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., Stolcke, A. (2017). The Microsoft 2017 Conversational Speech Recognition System.

Domingos, P. (2012). A Few Useful Things to Know About Machine Learning. Commun. ACM. 55. 78–87. 10.1145/2347736.2347755.

Ikeda, C., Ouazzane, K., Yu. (2020). A New Framework of Feature Engineering for Machine Learning in Financial Fraud Detection, London Metropolitan University, UK

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

Dornadula, V., Geetha, S. (2019). Credit Card Fraud Detection using Machine Learning Algorithms, Procedia Computer Science, Volume 165, Pages 631-641, ISSN 1877-0509,

Xuan, S., Liu, G., Li, Z., Zheng, L., Wang, S., Jiang, C. (2018). "Random Forest for credit card fraud detection," 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), 2018, pp. 1-6, doi: 10.1109/ICNSC.2018.8361343.

Caldeira, E., Brandao, G., Pereira, A. (2014). Fraud Analysis and Prevention in e-Commerce Transactions. Proceedings - 9th Latin American Web Congress, LA-WEB 2014. 42-49. 10.1109/LAWeb.2014.23.

Lakshmi, K., Deepthi, S. (2018). Machine learning for credit card fraud detection system. Online Journal. Adaptive Machine Learning for Credit

Maes, S., Tuyls, K., Vanschoenwinkel, B. (2002). Credit Card Fraud Detection Using Bayesian and Neural Networks. University of Brussel – Department of Computer Science

Trivedi, N., Simaiya, S., Lilhore, U., Sharma, S. (2020). An efficient Credit Card Fraud Detection Model Based on Machine Learning Methods. International Journal of Advanced Science and Technology, Vol 29, No. 5, pp 3414 – 3424

Vestas Corporation. (2019). IEEE-CIS Fraud Detection (Version 1.0) [Data Folder]. Retrieved 01 November 2020, from https://www.kaggle.com/c/ieee-fraud-detection/data

Walimbe, R. (2017). Handling imbalanced dataset in supervised learning using family of SMOTE algorithm. Picture retrieved 10 April, 2021, from https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family

The Nilson Report. (2019). Annual fraud statistics. Retrieved 01 December 2020, from https://www.prnewswire.com/news-releases/payment-card-fraud-losses-reach-27-85-billion-300963232.html

Ingenico, Inc. (2021) Fraud Expert Checklist. Retrieved 01 Mars 2021, from https://support-uat.direct.ingenico.com/en/security/fraud-prevention/fraud-expert-checklist

Kumar, S. (2020). 7 Ways to Handle Missing Values in Machine Learning. Retrieved 07 May 2021, from https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e

Shaikh, R. (2018). Choosing the right Encoding method-Label vs OneHot Encoder. Retrieved 15 May 2021, from https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b

Miyaki, K. (2019). Time Series Split with Scikit-learn. Retrieved 11 April 2021, from https://medium.com/keita-starts-data-science/time-series-split-with-scikit-learn-74f5be38489e

Grootendorst, M. (2019). Validating your machine learning model. Retrieved 19 May 2021, from https://towardsdatascience.com/validating-your-machine-learningmodel-25b4c8643fb7.

Featuretools, Inc. (2021). Retrieved 04 February 2021, from https://featuretools.alteryx.com/en/stable/

Koehrsen, W. (2018). A conceptual explanation of bayesian hyperparameter optimization for machine learning. Retrieved 07 April 2021, from https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-basedhyperparameter-optimization-for-machine-learning-b8172278050f.

Bissuel, A. (2019). Hyper-parameter optimization algorithms: a short review. Retrieved 09 May 2021, from https://medium.com/criteo-labs/hyper-parameter-optimizationalgorithms-2fe447525903.

FICO Report (2020). Real-time Payment fraud. Retrieved 03 January 2021, from https://www.prnewswire.com/news-releases/fico-survey-real-time-payments-platforms-have-increased-fraud-losses-for-4-out-of-5-apac-banks-300991019.html

Badr, W. (2019). 6 Different Ways to Compensate for Missing Values In a Dataset (Data Imputation with examples). Retrieved 07 April 2021, from https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779

Yadav, D. (2019). Categorical encoding using Label-Encoding and One-Hot-Encoder. Retrieved 17 April 2021, from https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd

Vadapalli, P. (2020). Naive Bayes Classifier: Pros & Cons, Applications & Types Explained. Retrieved 18 May 2021, from https://www.upgrad.com/blog/naive-bayes-classifier/

Breiman, L. (2001). RANDOM FORESTS. Retrieved 22 May 2021, from http://www.math.univtoulouse.fr/~agarivie/Telecom/apprentissage/articles/randomforest2001.pdf

Yiu, T. (2019). Understanding Random Forest. Retrieved 16 April 2021, from https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Thenraj, P. (2020). Do Decision Trees need Feature Scaling? Retrieved 01 June 2021, from https://towardsdatascience.com/do-decision-trees-need-feature-scaling-97809eaa60c6

Brownlee, J. (2018). A gentle introduction to k-fold cross validation. Retrieved 05 May 2021, from https://machinelearningmastery.com/k-fold-cross-validation/

Brownlee, J. (2016). A Gentle Introduction to XGBoost for Applied Machine Learning. Retrieved 11 April 2021, from https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Hulgol, P. (2020). Bias and Variance in Machine Learning – A Fantastic Guide for Beginners!. Image retrieved 07 June 2021, from https://www.analyticsvidhya.com/blog/2020/08/bias-and-variance-tradeoff-machine-learning/

Pedregosa. F, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. (2019). An introduction to machine learning with scikit-learn. Retrieved 03 June 2021, from https://scikit-learn.org/stable/tutorial/basic/tutorial.html#an-introduction-to-machine-learning-with-scikit-learn

Kapil, D. (2019). Hyperparameter search: Bayesian optimization. Retrieved 03 June 2021, from https://medium.com/analytics-vidhya/hyperparameter-searchbayesian-optimization-14be6fbb0e09

Singh, S. (2018). Understanding the bias-variance tradeoff. Retrieved 07 June 2021, from https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

Kanter, M. (2018). Deep Feature Synthesis: How Automated Feature Engineering Works. Alteryx Innovation Labs. Retrieved 12 January 2021, from https://innovation.alteryx.com/deep-feature-synthesis/

MLMath.io. (2019). Math behind Decision Tree Algorithm - Retrieved 19 January 2021, from https://medium.com/@ankitnitjsr13/math-behind-decision-tree-algorithm-2aa398561d6d

APPENDIX

A1 Orginal Feature Description

Raw feature	*Type*	*Description*
TransactionDT	Timeindex	Time delta for a given reference data represented in seconds.
TransactionAmt	Numeric	Transaction payment amount in USD
ProductCD	Categorical	Product code for the product in each transaction.
Card1 – card6	Categorical	Payment card information such as car type, card category, issue bank and country etc
Addr1 – addr2	Categorical	Address. Could for example be billing region
Dist1 – dist2	Numeric	Distance between zipcode, billing address, IP address, phone areas etc
P_emaildomain and R_emaildomain	Categorical	Purchaser (P) and recipient (R) email domain

M1 – M9	Categorcial	The actual meaning is masked. But it provide information regarding match between card and address for example
C1 – C14	Numeric	The actual meaning is masked. But it provide a count of how many payments that are associated with the card for example.
D1 – D15	Numeric	The actual meaning is masked. But it can be thought of as time delta features on a numeric format.
DeviceType	Categorical	The actual meaning is masked but information is provided from multiple sources of the company's security partners.
DeviceInfo	Categorical	The actual meaning is masked but information is provided from multiple sources of the company's security partners.
Id_01 – id_11	Numeric	The actual meaning is masked. But we can think of these numeric features as proxy rating, IP address domain, digital signature, number of failed login attempts etc

Id_12 – id_38	Categorical	The actual meaning is masked but information is provided from multiple sources of the company's security partners.
V1 – V339	Numerical	VESTA engineered features by feature engineering. Not all features the V variables are created from are included as raw features in this dataset.

Table A1: Description of original raw features in the dataset

A2 Python Code

All python code used in this project is included in the GitHub repository.

https://github.com/AdrianKopperud/automatedfeatureengineering

Backup

We have attached the links to the original python files in google colab as a backup if something happens to the GitHub repository.

Dataset 1 - Baseline:

https://colab.research.google.com/drive/1W-17PHNOsA1_RW4rM0ZQC3YATme9Z_Fg?usp=sharing

Dataset 2 - Manual feature engineering:
https://colab.research.google.com/drive/1lasDGYLYlmxWwmQp6wnIw5pxUHi0oZJg?usp=sharing

Dataset 3 - Automated Feature Engineering:

https://colab.research.google.com/drive/1iP9U5-y9vWdoMESEaSgboNxCKUBdanTY?usp=sharing

Exploratory data analysis:

https://colab.research.google.com/drive/1jAg0S-RMBv1uYP4_2rDJJffBlmk18pdi?usp=sharing

A3 Missing Data Exploration and Interpretation

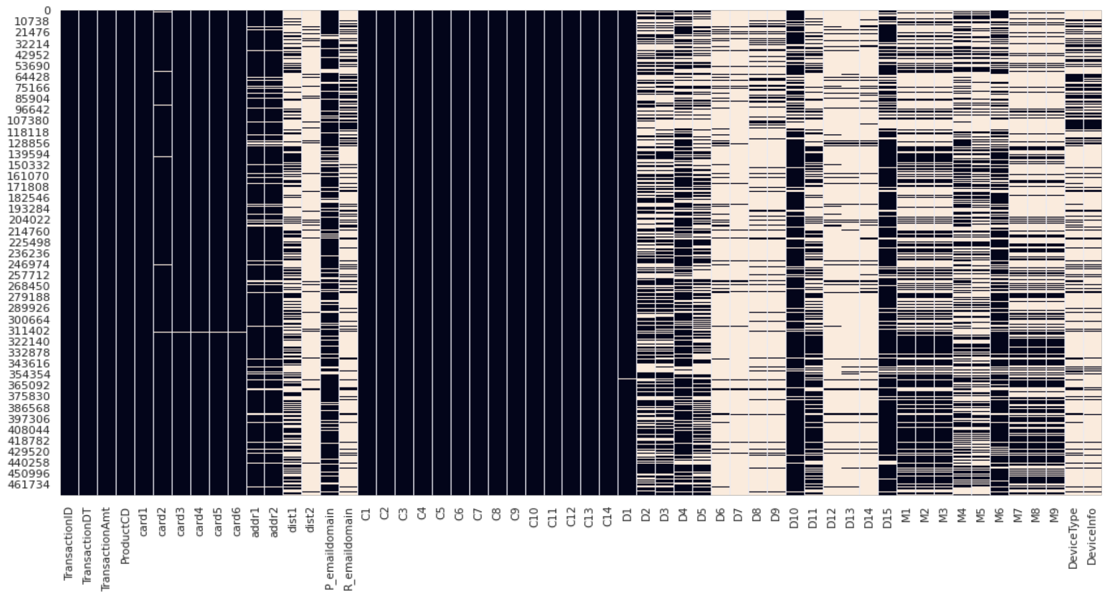


Figure A3.1: Missing data heat map of the transaction table

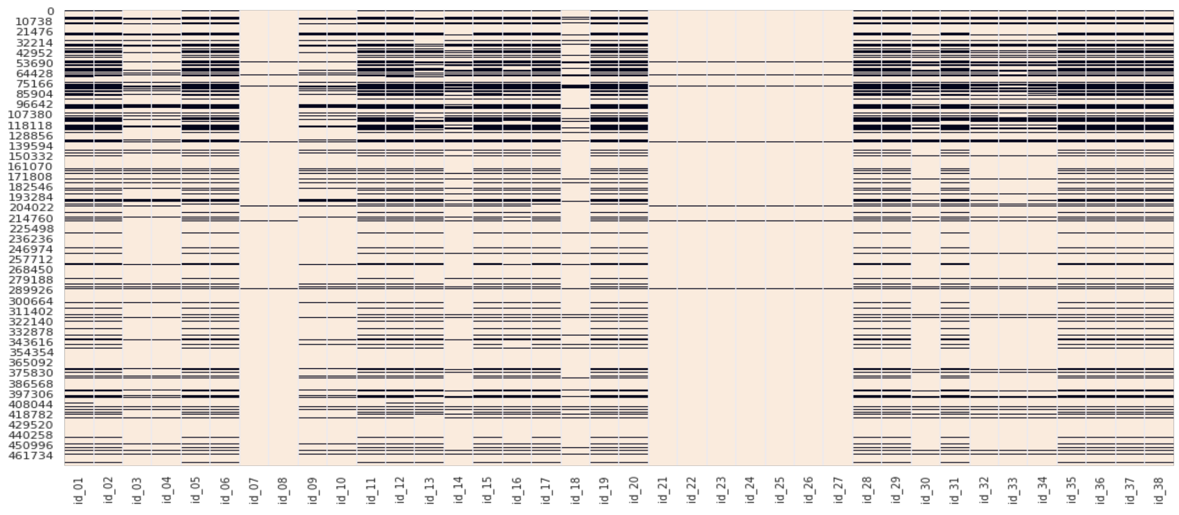


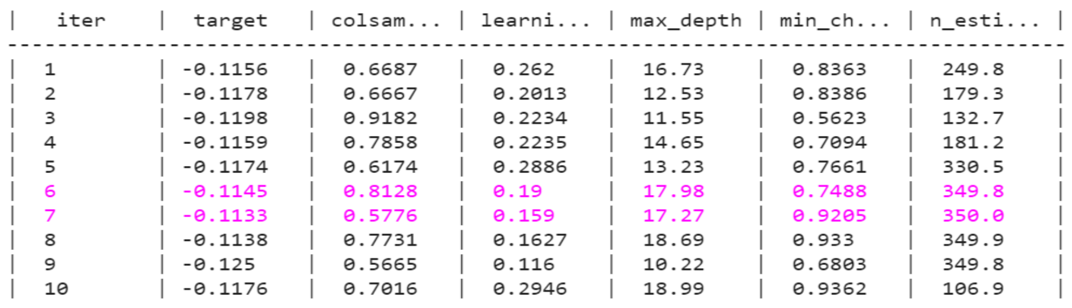
Figure A3.2: Missing data heat map of the identification table

The above plot A3.1 and A3.2 can be interpreted as a heatmap where the dark fields describe where there is data and the light field describe where we have missing values. Thus, we can get a quick overview of missing patterns and which features are complete and who is missing.

Following observations were made;

1. The observations we are making are that all the ID columns have the most missing values, but most of them have the same pattern of missingness.
2. Standard information about transaction amount, ID and transaction time is complete.
3. The M columns are missing almost all the data.
4. *Dist1* and *dist2* are both sparse, but it seems like *dist2* contains more information than dist1. Maybe we can drop one of them?
5. All the C columns are complete.
6. The D columns are sparse except for D1.

A4 Tuned Hyperparameters for XGBoost



```
|   iter    |  target   | colsam... | learni... | max_depth | min_ch... | n_esti... |
-------------------------------------------------------------------------------------
|  1        | -0.1156   |  0.6687   |  0.262    |  16.73    |  0.8363   |  249.8    |
|  2        | -0.1178   |  0.6667   |  0.2013   |  12.53    |  0.8386   |  179.3    |
|  3        | -0.1198   |  0.9182   |  0.2234   |  11.55    |  0.5623   |  132.7    |
|  4        | -0.1159   |  0.7858   |  0.2235   |  14.65    |  0.7094   |  181.2    |
|  5        | -0.1174   |  0.6174   |  0.2886   |  13.23    |  0.7661   |  330.5    |
|  6        | -0.1145   |  0.8128   |  0.19     |  17.98    |  0.7488   |  349.8    |
|  7        | -0.1133   |  0.5776   |  0.159    |  17.27    |  0.9205   |  350.0    |
|  8        | -0.1138   |  0.7731   |  0.1627   |  18.69    |  0.933    |  349.9    |
|  9        | -0.125    |  0.5665   |  0.116    |  10.22    |  0.6803   |  349.8    |
|  10       | -0.1176   |  0.7016   |  0.2946   |  18.99    |  0.9362   |  106.9    |
```

Table A4.1: Optimized hyperparameters for XGBoost

From Table A4.1 we can see that iteration seven yielded the best result out of the ten iterations that were performed. It was a very exhaustive and time-consuming task and therefore was not more than ten iterations done.

A5 Recall, Precision, F1-Score and Accuracy

Accuracy

Accuracy is simply defined as the number of correct predictions over the number of total predictions made. Using the terminology from the confusion matrix, accuracy is defined as the true values divided by n. A major concern using accuracy when dealing with imbalanced datasets is that due to the potential low number of minority targets, the metric may yield extremely high numbers close to 1. However, the explanation to this may simply be that the model only predicted 0 for each prediction. Due to this accuracy is generally not used when dealing with imbalance as other metrics adjust for this class imbalance.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Recall

Recall is defined as the ratio of predicted positives over the total amount of actual positives. Recall is adjusted through adjustments made to the threshold value, discussed in the previous section on the Logistic Regression model. Although a better measure for use on imbalanced data, recall usually has to be measured alongside precision as it only measures the ratio concerning true values without taking the other predictions into account. The F1-score that will be discussed below is a metric that functions as a combination between recall and precision.

$$Recall = \frac{TP}{TP + FN}$$

Precision

As mentioned above precision is quite similar to recall but instead of measuring the ratio in relation to actual values, it measures the ratio of positive predictions to the total number of predicted positives, the sum of TP and FP.

$$Precision = \frac{TP}{TP + FP}$$

F1-Score

The F1-score is a metric that uses the harmonic mean of both precision and recall. The metric is widely popular, especially for problems with large class imbalance as it both takes the sensitivity to misclassifications into account and the fraction of positive predictions into account. Although somewhat criticized regarding the equal weighting scheme of precision and recall, the metric helps alleviate some of the problems that may be present in imbalanced datasets.

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

A6 Illustration of Feature Importance Parameters

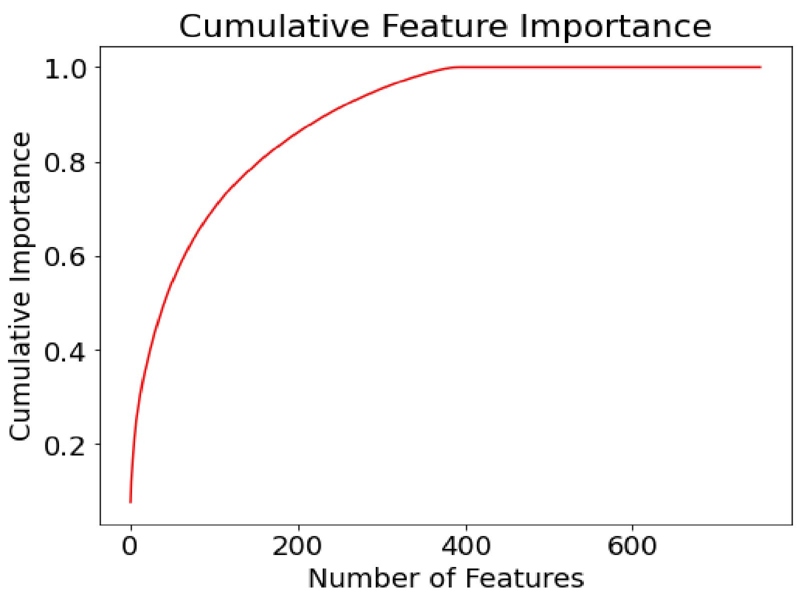


Figure A6.1: Cumulative feature importance plot of zero important features

Figure A6.1 shows the graph which illustrates where to cut off based on non-important features. When the line starts to flat out there is no more gain from those additional features that can improve the performance. Rather it can be smart to remove those as they only cause our model to be more complex than it must be.

A7 Mapping and Cleaning of Categorical Features

ID 30

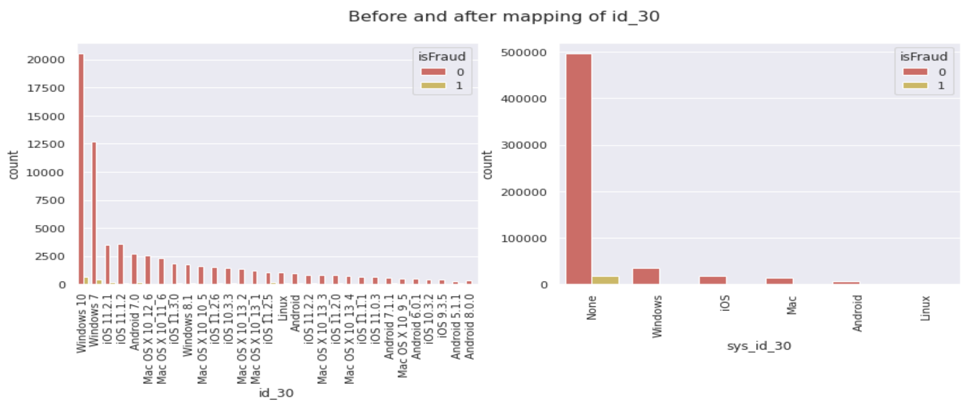


Figure A7.1: *ID_30* before and after mapping

The *ID_31* feature included the different kind of operating system the customer used when making the transaction. There were multiple different categories for various editions of the same operating systems. We reduced the cardinality of id_30 through grouping all operating systems to their respective provider, thus creating one category for each provider.

ID 31

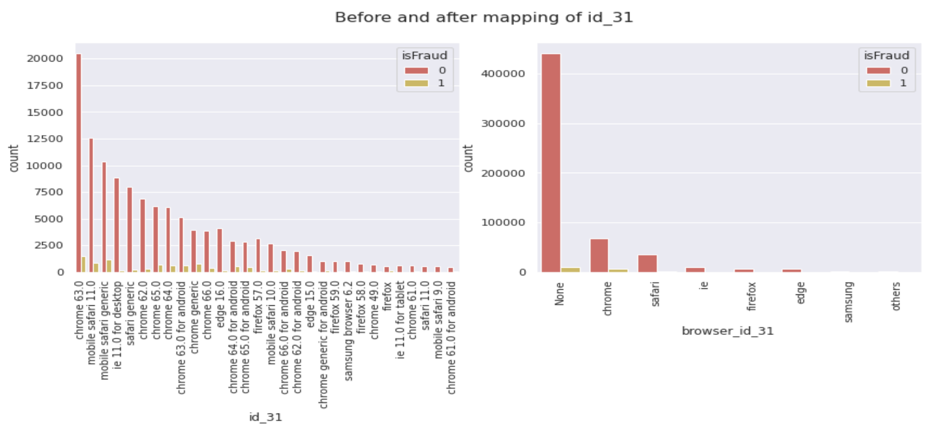


Figure A7.2: *ID_31* before and after mapping

The *ID_31* feature included what type of browser the customer used for the transaction. As the categories only differ on version number we chose to discard this information and to create categories only containing the browser name as this reduces the cardinality significantly

ID 33

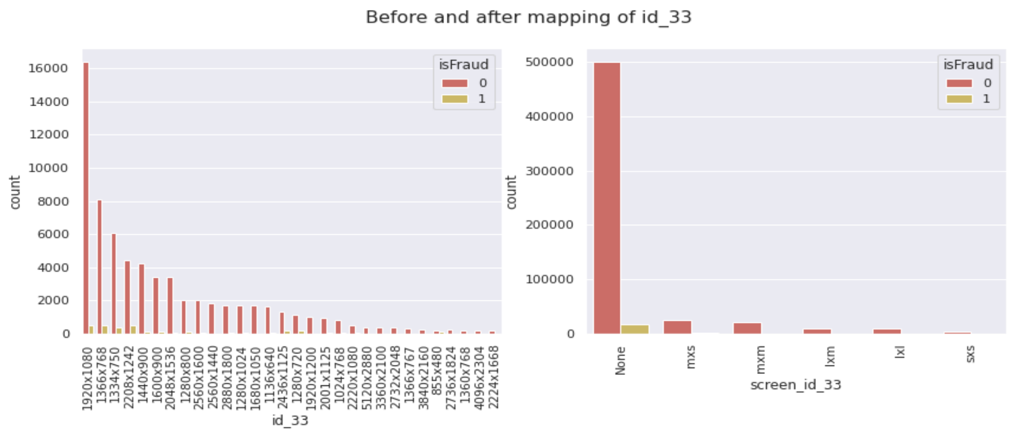


Figure A7.3: *ID_33* before and after mapping

The *ID_33* feature included ìnformation on the screen size of either the phone or computer that the transaction was made from. We map these different screen sizes into 5 categories which include medium/small (mxs), large/medium (lxm), medium/medium (mxm), small/small (sxs) and large/large (lxl).

Card6

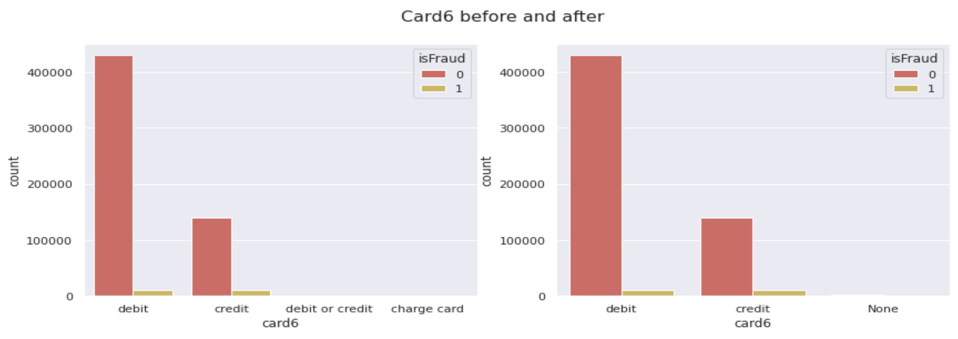


Figure A7.4: *Card6* before and after mapping

The *card6* feature originally contained four categories, but the two categories "charge card" and "debit or credit" were removed. We found that it does not make sense to have an own category which could be either credit or debit when those categories were already present. The inconsistency was removed and put into the new category "none" as they in total only account for 45 observations out of the complete 590.000 observations.

P_emaildomain and R_emaildomain

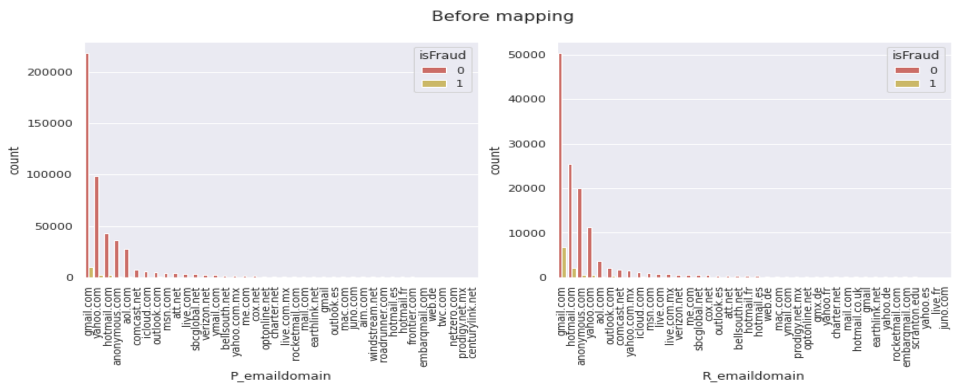


Figure A7.5: *P_emaildomain* and *R_emaildomain* before mapping

The *P_emaildomain* and *R_emaildomain* features included the purchase and recipient's email domains. From Figure A7.5, we see over 60 different domains used while only a few accounts for most customers. From the plot, we can see that gmail.com and hotmail.com are the most used domains. To account for all the redundant domain categories, we transform the original *P_emaildomain* and *R_emaildomain* into two new features that can be described as "the application of mathematical modification to the value of a variable" to extract more information than in its original state (Osborne, 2002). We transformed the features into two new categories containing the web domain and the bin, which is the email provider name. Redundant domain names with low observations were mapped into a common category named "other". The mapping had the aim of creating representative and heterogeneous categories through a simple method. Below in Figures A7.6 and A7.7, can we see the results.

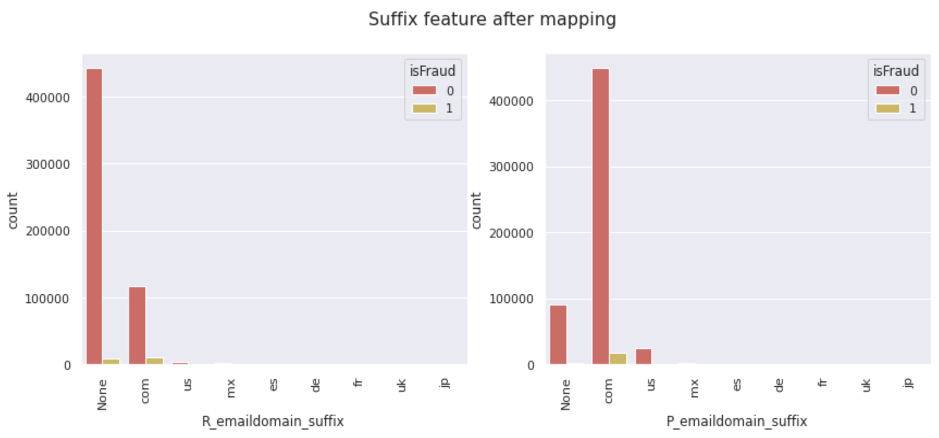


Figure A7.6: *P_emaildomain_suffix* and *R_emaildomain_suffix after mapping*

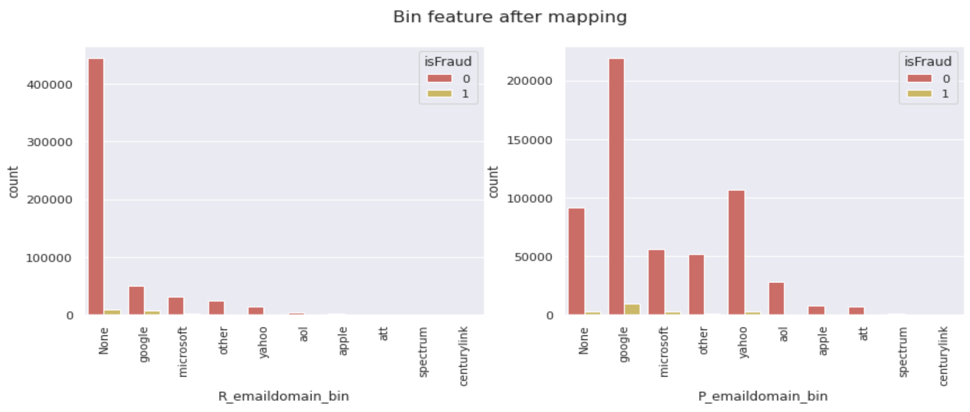


Figure A7.7: *P_emaildomain_bin* and *R_emaildomain_bin after mapping*

A8 The Thresholds between FPR and TPR

Threshold moving is another method to select the desired level between FPR and TPR. From the ROC curve, is it possible to optimize this relationship. By default, the threshold always is 0.5, meaning that a predicted probability greater than 0.5 will be classified as fraudulent, and a probability lower than 0.5 will be classified as genuine in our experiment.

For example, a credit card company may adjust the FPR and TPR levels to achieve their desired strategy of minimizing FPR or maximizing TPR. By lowering the rate from the standard default of 0.5 will both the FPR and TPR increase. This will benefit in predicting more fraudulent transactions but at the same time increase FPs. On the contrary, moving the threshold above 0.5 will, in most cases decrease the FPR and decrease TPR, which will benefit lower FPs but also lower the rate of TPs.

Threshold moving is highly relevant for a company with a cost-saving strategy where either FRP or TPR is more important. In our problem, we are not working with a specified threshold from a company and we will therefore not pay attention to moving this threshold and keep the default of 0.5 for our experiment.