Automated Transaction Monitoring

Bachelor Thesis

R. Hageman
B. Kostense
B. van Walraven
H.J. van der Wilk

TI3806 - Bachelorproject

TU Delft

Automated Transaction Monitoring

Bachelor Thesis

by

R. Hageman
B. Kostense
B. van Walraven
H.J. van der Wilk

to obtain the degree of Bachelor of Science
at the Delft University of Technology.

Project duration:	April 22, 2019 – July 5, 2019
Supervisors:	ir. Sander van den Oever, TU Delft
	ir. W. Van, bunq
	MSc. A. el Hassouni, bunq
Bachelor Project coordinators:	ir. O. W. Visser
	dr. ir. H. Wang

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

For the past 10 weeks, we have been tasked with improving the performance of the transaction monitoring system of bunq, an internationally active mobile bank. bunq has requested that we improve this system by automating the training of the machine learning model, providing better input data for this model and creating additional machine learning models. During this project, we have been working at the offices of bunq on this system. This thesis will give an overview of our research, software design process and implementation.

Our thanks go to everyone at bunq who has helped us, especially Wessel Van for being our client coach, Ali el Hassouni for guiding us through the transaction monitoring system, Niels Tiben for answering our countless questions, André de Roos for helping us with backend related issues, Esan Wit for helping us with DevOps related issues, Hannah van Drunen for human resource related topics and Ali Niknam for his continuous feedback. Finally, we would like to thank Sander van den Oever from the TU Delft for being our TU Delft coach.

R. Hageman
B. Kostense
B. van Walraven
H.J. van der Wilk
Amsterdam, July 2019

Summary

For 10 weeks, bunq, a Dutch mobile bank, tasked us to improve its transaction monitoring system that is used to detect fraud. The current system is a combination of a rule based system and a machine learning model. Every transaction goes through this system and if either one of the two components generates a hit, the transaction gets sent to the compliance department for manual checking.

The first two weeks consisted of a research phase, where we interviewed the client to get a better overview of the problem, looked at the possible solutions and selected the ones that we thought were most optimal. The last eight weeks consisted of implementing the actual improvements. These improvements consisted of three major milestones: automating the training and deployment of machine learning models, improving the quality of the input data for these models and developing additional models.

The first step of the implementation consisted of refactoring the previous transaction monitoring system backend. This allowed for multiple machine learning models to generate predictions on transactions. Database migrations have been designed to prevent loss of data while changing the database design. Also, the machine learning server hosting the machine learning models has been changed to support multiple models. To make use of this refactor, new models have been trained for different types of mutations and to detect the greatest outliers per day.

Secondly, a system has been designed to train these machine learning models automatically at fixed intervals. The required data is fetched from the database, enriched with different sources of other information and sent to a Python training server in batches. On this server the different models are trained with optimal parameters found using GridSearch. Once trained, information about the machine learning model such as name and status is stored in the database using an API. After inspection of a data scientist, a model can be set to an active state to generate fraud scores for transactions.

Finally, the data used as input for the machine learning models has been improved by almost quadrupling the amount of features the machine learning model uses to train on. Also, retrospective labelling of transactions that are in hindsight fraudulent has been applied to improve the quality of the data.

Contents

1 Introduction

A bank is obligated to report unusual transactions according to article 2a of the *Wet ter voorkoming van witwassen en financieren van terrorisme* [11], a Dutch law to prevent money laundering. This law, of course, also applies to bunq, who call themselves an ‘IT company with a banking license‘. Since this law only requires banks to use a simple rule based system to detect fraudulent transactions, bunq already uses an adequate form of transaction monitoring, as they combine their rule based system with a machine learning model. bunq chose to deploy this machine learning model to more accurately catch fraudsters in order to protect her customers.

Since bunq is growing fast and is expanding to other countries, they must keep constantly improving their transaction monitoring system. Because transactions from different countries have different characteristics, such as the words they use in the description, transactions with different characteristics are fed into the machine learning model with each country bunq adds to its market. Because the current machine learning model was not trained on such data, it gives less accurate predictions about these transactions. Therefore, the machine learning model should be trained on new data more often.

However, labelling data and training a new model is currently a laborious and time-consuming task. Therefore, over the span of ten weeks, we have been tasked with reworking the machine learning part of the transaction monitoring system to improve its performance. Our goal will be to automate the training and deployment of new machine learning models, provide better input data to train these models on and create additional machine learning models.

This thesis describes the process of improving the current transaction monitoring system used by bunq. The research phase is documented in Chapter 2. In Chapter 3 and Chapter 4, the current situation will be discussed. In Chapters 5 through 7 we will discuss the product we have developed. Chapter 8 and Chapter 9 contain the discussion, conclusion, recommendations and ethical aspects of the project.

2 Research Phase

In this chapter we will present the research phase as performed at the beginning of the project. This includes a formulation of the problems as stated by the company and our considerations for solutions to these problems. First, we give a brief description of the company in Section 2.1 and define the problem in Section 2.2. Next, the current system that bunq uses for transaction filtering and transaction monitoring will be discussed in Section 2.3. As bunq is obligated to keep their systems up to date, several improvements have been proposed by them which are listed in Section 2.4. The way we aim to implement the suggested improvements is described in Section 2.5. Section 2.6 elaborates on the methods used to assess the performance of the developed models. The requirements and the roadmap on how we plan to complete these requirements can be found in Appendix C.

2.1. What is bunq?

bunq is a Dutch bank which focuses on ease-of-use financial services, mainly through their mobile app. The company was founded by Ali Niknam, known as the founder of the web-hosting company TransIP. The company is founded in 2012 and bunq obtained their banking license in 2014 from the Dutch central bank (DNB) [2]. bunq, like every other bank, offers financial services at the cost of a monthly subscription. However, unlike others, this is the only source of income for bunq, as other banks also generate money through (often risky) investments.

2.2. Problem Definition

Since bunq is a bank, they provide the service of transferring money between accounts or from accounts to cash. In recent years bunq has experienced a growth in users, meaning the amount of transactions has also increased. This increase in total transactions also means an increase in fraudulent transactions, which bunq, like any other bank, has to monitor and report to the appropriate authorities. Authorities require Dutch banks to do this to prevent money laundering, credit and debit card fraud, identity theft and various other types of fraud, as stated in the *Wet ter voorkoming van witwassen en financieren van terrorisme* [11]. Detecting potential fraudulent transactions and investigating them can be a costly activity, which has led not only Dutch banks [15], but also large banks worldwide [19] to be negligent in monitoring these transactions.

bunq already has a system that is responsible for monitoring transactions. However, as their user base grows, transactions are not only more numerous, but also differ in their characteristics. [redacted]. Also, fraudulent transactions take on new forms, which bunq must be able to detect. To effectively monitor the growing number and growing types of transactions, bunq wants to automate this process as much as possible. This is desirable as it becomes infeasible for all transactions to be checked manually by the compliance department, which is responsible for detecting fraud and reporting it to the Financial Investigation Unit (FIU).

2.3. Current System

When a transaction is made, it goes through a sequence of processes before and after it is executed. An overview of the processes relevant to our problem is depicted in Figure 2.1.

[redacted]

Figure 2.1: Overview of the systems a transaction passes through where bunq checks for fraud.

2.3.1. Transaction Filtering

[redacted]

2.3.2. Transaction Monitoring

[redacted]

Figure 2.2 [redacted]

[redacted]

Figure 2.2: Structure of the machine learning part of the transaction monitoring system.

2.4. Assignment

To improve the described system on the previous page, bunq has proposed a set of solutions to improve the transaction monitoring system. Before describing the way we plan to implement these solutions in Section 2.5, we will describe the improvements that could be realised as stated by bunq.

2.4.1. Automatic Training

bunq is currently expanding to different countries where the characteristics of the transactions differ to the ones the machine learning model is trained on. To ensure proper accuracy on all transactions, training on recent data is required.

Right now, training the machine learning model has to be done manually, which is a laborious process [redacted]. This manual process consists of labelling transactions as either fraudulent or not, converting all the data to a format that the model can train on and finally training a machine learning model on this dataset. Therefore, the first milestone is to automatically train the current supervised learning model using recent transactions. This improvement is also suggested by the builders of the current system [10].

2.4.2. Improving Labels

To make sure that the data the machine learning model trains on is an as well as possible reflection of reality, we are tasked to improve the labelling of the transactions sent to the machine learning model. The task of improving the labels consists of identifying transactions that have not been labelled as fraudulent by compliance, but could in hindsight be considered fraudulent because the user is a known fraudster.

To accomplish this, the labels of some earlier transactions made by a fraudulent user should be changed according to a given set of rules. Then, when the model is retrained as described in Section 2.4.1, the model should have been improved, since the training data is more accurate.

2.4.3. Training a Machine Learning Model for Businesses

[redacted]

2.4.4. Implement More Features

As the fourth milestone, bunq would like to see their current supervised learning model being improved by adding more input for the machine learning algorithm to train on. [redacted]. However, there are still potential usable features out there that are not being used by the transaction monitoring system. The compliance team will help us to come up with these features as they have the required specific domain knowledge to be able to identify these features.

Therefore, it is our task to find usable features and add them to the data that the machine learning algorithm uses to train on.

2.4.5. Outlier Detection

The current transaction monitoring system is created upon known instances of fraud. This means that new, unique and undetected fraud may remain unnoticed. Supervised machine learning algorithms are known to perform suboptimal when receiving significant outliers [13] and thus other techniques are required to detect unknown types of fraud.

[redacted]

2.4.6. Detecting Fraud in Transaction Sequences

The current machine learning algorithm is an implementation of a supervised learning model, where the algorithm maps the information about the transaction to the chance of the transaction being fraud. A minor drawback of this approach is that this algorithm does not find links between previous transactions, i.e. the algorithm is myopic. [redacted] [3], [redacted]. If we were to implement this, we would not have to rely on implemented historical features. Therefore, our last milestone is to enhance the current supervised learning model with another machine learning model that is capable of learning and recognising these patterns. This is, however, a big task and the possibility of finishing this task depends on the time left after reaching the previous milestones.

2.5. Design Choices

Several decisions regarding the implementation, such as choosing algorithms, training methods, and frameworks will have to be made. In this section, we will elaborate on these decisions, and how we plan to actually achieve the goals described in Section 2.4.

2.5.1. Automatic Training

To implement the automatic retraining of the algorithm, we need to perform a few steps. The first step is to make sure that the data is prepared for training. When the compliance team manually checks a transaction, it passes through a few different phases, as seen in figure 2.3. In the end the transaction is set to one of the final states. We need to map these different labels to a Boolean state, namely 0 (not fraud) or 1 (fraud). Green states in figure 2.3 correspond with not fraud, while red states correspond with fraud. Then, to train the machine learning model, we need to map the features of the checked transactions to these labels. The third and last step is to perform the retraining with the newly acquired data.

[redacted]

Figure 2.3: Possible labelling of a transaction as a result of transaction monitoring.

Training the model with all available data would take several days or even weeks on consumer infrastructure. Therefore, it is not feasible to do this locally. Since bunq already uses Amazon Web Services, we are going to use a large EC2 instance where we can train our machine learning models.

Figure 2.4 gives an overview of the global structure that has to be designed for this project, altering the design from Figure 2.2. It only provides a general structure. The specific software design can be found in Section 6.2. When we compare Figure 2.4 to Figure 2.2, we can see that we plan to add a new functionality to the transaction monitoring system, which is a machine learning trainer. This trainer is going to send training data to a Python training server running on an EC2 instance. This training server will train a new model and deploy it to the Java machine learning server.

[redacted]

Figure 2.4: Global structure of the new transaction monitoring system.

We will train the same type of supervised machine learning model that is currently being used, namely a Gradient Boosting Machine (GBM). H2O is a machine learning framework that supports GBMs and is available in Python and R. Since we all have programming experience in Python but not in R, we will be using H2O in Python for machine learning related programming.

2.5.2. Improving Labels

The task of the manual retrospective labelling of transactions is a laborious task and can be automated. This is done by setting up a set of rules for which transactions should be flagged, as not every transaction from a user which has committed fraud, should be flagged as fraudulent. For example, not every previous transaction of a fraudster is necessarily fraudulent as they also have to, for example, do groceries and pay their bills. However, some transactions that are made by a fraudster can definitely be considered fraudulent in hindsight. The rules to decide for which transactions their state should be changed are developed by the compliance team using their specialised domain knowledge. Each time a user gets tagged for fraudulent behaviour, these rules are followed and the previous transactions which might be involved in this behaviour are tagged as fraudulent.

2.5.3. Training a Model for Businesses

[redacted]

2.5.4. Implement More Features

To implement more features, we need to figure out which features could potentially be useful for our algorithm to train on. A way to figure this out is by discussing this issue with the compliance team and the data scientists at bunq, as they have a lot of experience with fraudulent transactions, so they should have knowledge about which features occur often in fraudulent cases. Lastly, we need to test our newly added features by checking whether the accuracy of the model is increased with the added feature.

2.5.5. Outlier Detection

To catch outliers, various approaches are possible. First, we will discuss several of the possible approaches. Then, we make and explain our choice.

Clustering is a technique in which the algorithm tries to find patterns in the data. It tries to find groups in the data where the elements are in the same group have comparable values for their features [12]. Outliers can be detected by applying this technique to the dataset and stopping the process when the amount of desired outliers are left. The already clustered transactions are more similar than the unclustered ones. Clustering has the disadvantage that the computational complexity is dependant on the amount of dimensions of the data. Also, the large number of dimensions in our data causes the distance measures to become increasingly meaningless [16].

Local outlier factor (LOF) [4] is an algorithm where outliers are detected using the local deviation with respect to its neighbours. For a transaction, we find the maximum distance to its k-nearest neighbours. This way, we can calculate the density of the data point, which we can use to find outliers. A property of LOF is that, because it uses a different approach of calculating distance than clusters, it also finds different outliers. A disadvantage is that the resulting values of each data point are hard to interpret, as the result of something being an outlier or not with respect to the resulting value depends on the dataset. LOF suffers from the same curse of dimensionality as clustering.

An autoencoder is a neural network architecture consisting of two separate networks, the encoder and the decoder, which learn efficient encoding of data in such a way that the decoding is similar to the original data. The autoencoder trains its network b encoding data to reduce noise (dimensionality reduction) and decoding data to reconstruct the original input. Such networks are mainly used for compression and noise reduction, but as autoencoders learn how the majority of the data looks like, it is also often used for detecting outliers [5]. At bunq, an experiment with autoencoders has already been performed with results indicating the ability of detecting other transactions than the rule-based and supervised learning models. Of those transactions, a significant amount turned out to actually being fraud [8]. A study by Schreyer et al. [17] has shown that auto-encoders detect relevant transactions for a follow-up audit. Kaggle provided an open credit card fraud detection dataset where an autoencoder has been trained upon [9] showing a decrease in total operation costs.

To conclude, both clustering and LOF are usable for detecting outliers but suffer from the curse of dimensionality. Autoencoders are not susceptible to this and are shown to produce promising results on both the bunq and other datasets. Therefore, to implement our unsupervised learning algorithm, we have chosen to use autoencoders, as they best fit our needs in this specific context. The goal is to automatically train the autoencoder at the same moment as the supervised learning algorithm. It should be trained on the most recent data, just like the supervised learning algorithm, [redacted].

Since we already use H2O to train the GBM, we will also be using H2O to train the autoencoder.

2.5.6. Detecting Fraud in Transaction Sequences

To improve the existing supervised learning algorithm such that it can find links between transactions, reinforcement learning is the machine learning class that we want to take a look at. A reinforcement learning approach is one of the three major machine learning classes, along with the supervised- and unsupervised learning approaches. A reinforcement learning algorithm tries to take different actions in order to maximise its reward [14]. In our case, the reward would be to correctly identify fraud cases and non-fraud cases (true positives and true negatives), while the punishment would be to flag a fraud cases as non-fraud, and vice versa (false positives and false negatives). The algorithm tries different actions in order to optimise the reward and minimise the punishment, and thus finds an optimal mapping from the input data to the desired labels. As it does not try to learn a simple mapping from input to output as a supervised learning model does, but tries to generally maximise its reward, it is able to detect logical connections between transactions in the data. The importance of reinforcement learning and automatic retraining in the context of fraud detection follows from the following quote from Chen et al.: "Since the financial operations may vary from time to time, the need of a reinforcement learning that keeps on training should also be put into consideration." [6].

If we have enough time to complete this task, we will be using Facebook Horizon to train our reinforcement learning model, as this was recommended by the data scientist at bunq and H2O does not support reinforcement learning yet. Facebook Horizon is built upon PyTorch, which uses the Deep Q-Learning algorithm for reinforcement learning.

2.6. Performance Measurement

Several steps of the assignment consist of adding to or changing parts of the current supervised machine learning algorithm. All of these changes are meant to improve the performance of this model and a method to validate this is important. Such a method is not required for detecting outliers as it is solely built for the purpose of detecting odd transactions with the assumption this will potentially include fraud, but for improving labelling, adding new features and implementing reinforcement learning, it provides valuable insights.

2.6.1. Training Data

Access to Triage, a staging environment where testing before deployment is performed, is granted to build and test functionality. While this assists in building the systems, it only consists of test data created by developers. Therefore, Triage is only useful to us to test whether the functionality works, but not to assess the performance of machine learning models trained on this data. To assess the performance of a machine learning model, it needs to be used on real data.

2.6.2. Evaluation Criteria

Evaluating the performance of a classifier normally starts with measuring the area under the curve (AUC) when plotting the true against the false positives. However, AUC is not an optimal performance measurement metric, because our dataset is very unbalanced meaning that classifying every transaction as non fraudulent also provides a large AUC. Although AUC is still important, this is definitely not the silver bullet for assessing the performance of a model.

Another measurement is a confusion matrix where the true positive/negative and false positive/negative are represented. These values are used to define two types of errors, type 1 and type 2. Type 1 errors are flagged transactions that turn out to be valid, costing bunq money due to unnecessary work for the compliance department. Type 2 errors are instances of fraud that are not flagged.

The model should minimise both the type 1 and type 2 errors while optimising the AUC. While there is no concrete logic defined by bunq which states that a model is strictly better if not all the metrics are improved, when at least two of the thee improve a model is most likely better than before. The final decision on this remains at the discretion of a bunq employee.

2.6.3. Deploying a Model in Production

The goal of our system is first to allow automatically training of a machine learning model and then to improve its performance. Once a machine learning model is automatically trained, it is required to verify its performance before using it in production. Ideally, the performance of such a new model could be verified by using it besides the model used in productions. When this shows better results, it could be deployed into production.

3
Current Situation: Backend Architecture

In this chapter relevant information about the backend of bunq is provided. This is relevant as it describes the various demands and expectations that our code has to live up to and adhere by. We will touch upon the code style, backend design choices, testing and database architecture. Other relevant information that is relevant to our software development can be found in Appendix B.

3.1. Coding Guidelines

Strict rules based upon PSR-2 have been established for the bunq backend code base. These rules apply on aspects like the naming of variables, classes and methods but also on the size of functions. Because a part of our code is going to be placed directly into the backend, which is written in PHP, we have to follow these guidelines. To help adhere to these conventions, a coding style checker is available for PHP.

3.2. MVC Design

The backend of bunq is constructed of multiple request handling flows built upon the model view controller (MVC) design pattern as schematically depicted in Figure 3.1. Once a request is sent to the backend, the `Router` determines the correct route of pre-processors which extract the information of the HTTP request. Then, the request is forwarded to workflows that act as controllers. These workflows contain the business logic and communicate with the database via models. Once the workflows are executed they return a response which is parsed by the post-processors before sending it back as a response on the initial request.

Not all business logic is executed directly due to incoming HTTP requests. Several background tasks, so called daemons, are started by a time-based job scheduler (cronjob). These daemons execute workflows containing the business logic. Parallelization is also possible with daemons, by creating several workers which perform batches of the big task of the master daemon. The master daemon keeps track of the workers and creates more if required.

3.3. Workflow and Model Definitions

bunq has created its own parser for JSON files containing definitions for models, workflows, objects and views. This provides a developer with a consistent way of creating such patterns without having to write trivial code. In this section the two most used definitions are explained; workflows and models.

A workflow is similar to a finite state diagram. In the workflow definition different states are defined. Each state has different input and output which can be used by the consecutive states and boolean logic is used to define the conditions for each transition to another state. All logic for making these transitions and passing along variables is available in the abstract workflow class created by the parser. Only the business logic in each state has to be implemented.

Models are used as interfaces with the database. The definition contains all properties, their type and eventual index, together with the relation with other models. The abstract class contains all trivial code like getters and

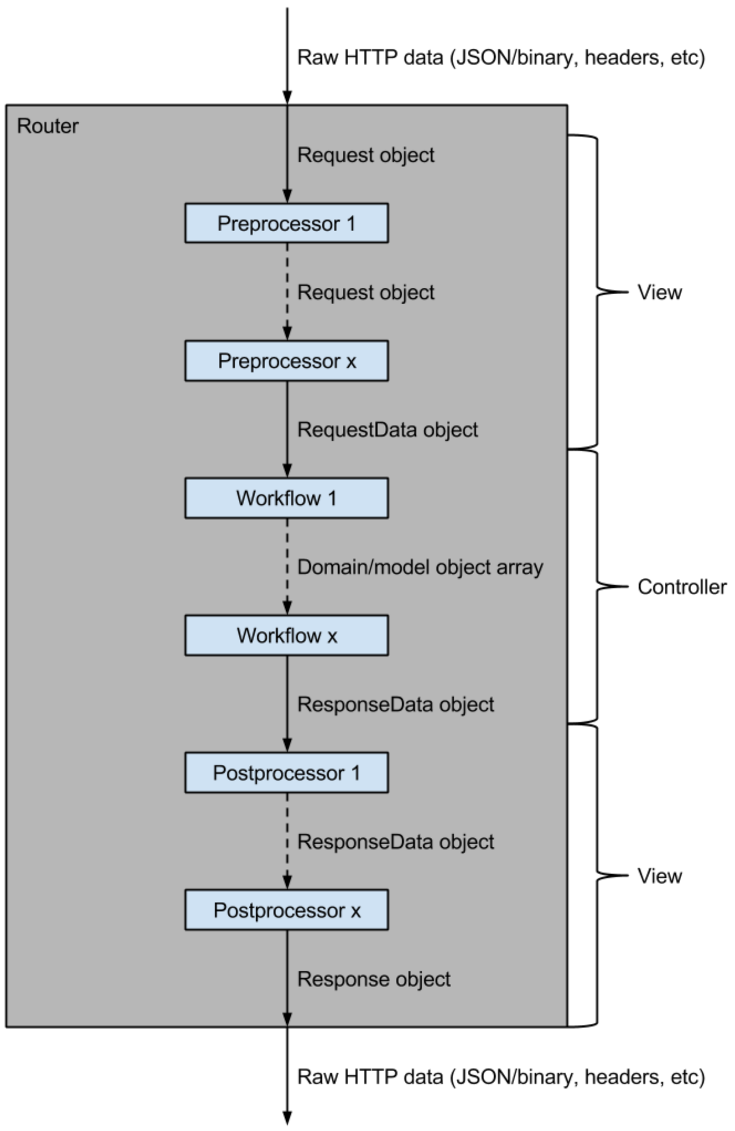

Figure 3.1: Schematic overview of a request handling flow as depicted on the internal wiki of bunq.

setters. Queries to create the table in the database are also generated. Only implementing the Access Control Layer, which defines which operations are allowed by who, is left.

3.4. Testing

Unit testing these controllers is harder as a consequence of the generated abstract workflows bases classes. Therefore, bunq uses mainly end to end testing for workflows or even test the complete MVC pattern. For this, a testing framework has been created which cooperates with the standard PHP testing frame work PHPUnit. JSON files are used to define parametrized scenarios. Once executed, standard assertions are used to assert correct behaviour. However, unit testing is still possible in some workflows and we try to do unit tests as well if we can.

On the other hand, there are also classes that are not workflows, and can therefore be tested normally. These classes will have to be both end to end tested as unit tested.

3.5. Database Architecture

The main database architecture of bunq is built upon MySQL and is backed up by another database. All tables are grouped with other tables which are frequently used together. Each group is stored on a different cluster to spread the load. To reduce the load even further, a separate database built on top of Elastic Search is created. A daemon makes sure the search database is as up to date as possible. As it could occur that the search database is lacking behind due to high load, it should therefore not used for critical applications. The MySQL database remains the single source of truth but the search database is orders of magnitude faster and thus useful for analytics and other applications where incompleteness of information is not catastrophic. [redacted].

4 Current Situation: Transaction Monitoring System

In Chapter 2 we have already briefly explained what the transaction monitoring system does. In this chapter we will provide a more detailed overview of the transaction monitoring system as currently used at bunq. This is useful to explain as it better indicates why the improvements we made matter. First we will discuss the tangible effect of our product in Section 4.1, which will be visible in the compliance dashboard, where transactions are monitored. Next, Section 4.2 will discuss the current backend architecture. Finally the current machine learning server setup is discussed in Section 4.3.

4.1. Compliance Dashboard

When a transaction is flagged for manual checking by compliance, it is shown in the dashboard which the compliance team uses and is depicted in Figure 4.1. All transactions hit by the transaction monitoring system which are not yet processed by compliance are shown. Once an employee of the compliance department selects a transaction, he or she has access to all the information about the user and the transaction to make a decision whether or not to report it to the Financial Intelligence Unit (FIU). Compliance employees are required to report all unusual transactions to the FIU [11]. Improving the predictions by the machine learning model will mean that the transactions shown here are more accurate, reducing workload for the compliance team.

[redacted]

Figure 4.1: Screenshot made of the compliance dashboard in the testing environment Triage with fake data.

4.2. Current Backend Architecture

This section will describe the current transaction monitoring architecture. First, Section 4.2.1 will explain how current model architecture is implemented and how the database tables are configured. Section 4.2.2 will describe the controller architecture.

4.2.1. Current Model Architecture

Currently, the model architecture of the transaction monitoring system consists of two tables: [redacted] and [redacted].

TransactionMonitoringMetadata table For every mutation, an entry in the [redacted] table will be created. This table includes information about the classification of the transaction monitoring system on the mutation. The table includes the following columns:

- `Featuremap` is a key value storage of all features calculated about the mutation.
- `MachineLearningName` is the name of the machine learning model used to classify the mutation. This field is optional as not all mutations are classified by a machine learning model.
- `FraudScore` is the outcome of the classification of the machine learning model on the mutation. It is a number between 0 and 1, where a larger number means a higher chance of the mutation to be 'unusual'. Again, this is optional as not all mutations are classified.
- `Status` is used to indicate what the status of the mutation is. [redacted]. [redacted]

  [redacted]

  [redacted]

  [redacted].
- `ExecutionTime` is the time it took to process this mutation. This is used to monitor the performance of the transaction monitoring system.

TransactionMonitoringRisk table Once a transaction that passed through the transaction monitoring system is hit by at least one rule or the machine learning model returns a risk score above a certain predefined threshold, the transaction is labelled as [redacted]. If this is the case, an entry in the [redacted] is created. Each entry is linked to a `metadata` entry and a user. It also contains a `RiskIndicator` which is used to determine why a mutation is shown in the compliance dashboard.

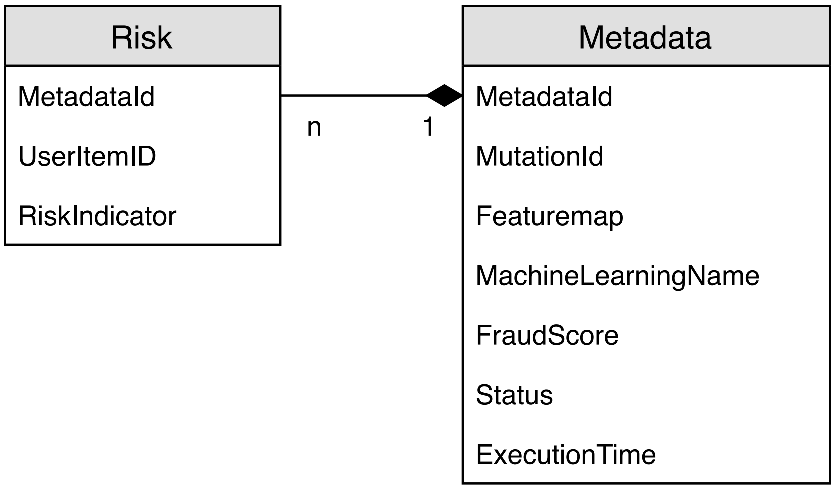

Figure 4.2: UML diagram of the previous transaction monitoring software database architecture.

4.2.2. Current Controller Architecture

[redacted]. Each mutation is processed by a single `TransactionMonitoringWorker`. This worker calls the `WorkflowTransaction MonitoringWorker` which determines whether the mutation is incoming or outgoing and performed by either a business or personal user. Depending on the type of mutation, a different workflow is executed and finally the created `metadata` is saved.

In Figure 4.3 an overview of the current transaction monitoring system is shown with a private incoming transaction as input. Each mutation type dependent workflow asserts that the correct workflow is executed. Then, all applicable rules are checked on the mutation, and if a rule applies, an entry in the [redacted] table is created and the status of the [redacted] entry is set to [redacted]_

[redacted].

[redacted]. The `executeWorkflowTransactionMonitoringMachineLearning` requests the machine learning server to provide a list of the required features. Once these features are calculated and put together into a featuremap it is sent to the machine learning server to get a prediction. The prediction is stored together with the featuremap into the [redacted] table. If the machine learning server indicates that the prediction is above a threshold, an entry in the [redacted] is also created.

4.3. Current Machine Learning Server

[redacted].

The machine learning server runs a Java Virtual Machine and the code is built upon the Spring Boot framework. As a machine learning model is exported in a Java format, it first needs to be compiled to a jar file. This compiled file can then be used to load a `MachineLearningModel` object on which a prediction can be called. Because this process of compiling and object loading is computationally intensive, the model object is cached in memory. This way, the object does not have to be loaded from a jar each time a prediction is requested. When a model is uploaded, this cache is cleared and overwritten with the new model. [redacted].

[redacted]

Figure 4.3: Overview of the current transaction monitoring software controller architecture.

5 The Product: Multiple Model Functionality

The current transaction monitoring system as explained in Chapter 4 has been designed with the rule-based system together with one machine learning model in mind. As it is part of the requirements to support multiple models and we aim to make deployment of additional models as easy as possible, a refactor is required to accommodate multiple models. In this chapter, we will first discuss various aspects of the refactor of existing code and database tables. Then, we will discuss the addition of a model for business transactions. Lastly, we will touch upon the implementation of an autoencoder for outlier detection.

5.1. Refactor

In this section, we will discuss the refactor to accommodate multiple models. We will discuss the refactor according to the MVC design pattern the bunq backend is built upon. First, the model and database refactor will be discussed, followed by the controller and view refactors.

5.1.1. Initial Design

Initially, a complete refactor of the transaction monitoring system for both the rules and machine learning models was designed. This design used the database to store all the different rules and models as classifiers and allowed the compliance department to dynamically adapt these classifiers used on transactions. However, in collaboration with developers of bunq we have decided not to continue with this design as it is not know what the impact would be on underlying mechanisms such as the compliance dashboard and generated reports for the FIU. Therefore, to allow us to complete the project in the limited timespan, we have decided with bunq that we have to focus the refactor on building upon the current system. This means that existing database tables must remain largely preserved.

5.1.2. Model Architecture

To accommodate multiple machine learning models while preserving existing tables that are related to transaction monitoring, we chose to add two tables. These will be explained in this section. A figure of all tables related to transaction monitoring is presented in Figure 5.1.

Result Table One of the introduced tables is [redacted]. This table is meant to take over functionality that was previously present in the [redacted] table. While the metadata table contained columns for fraud probability and machine learning model name, the table was still unfit to host predictions for multiple models. As the feature map is saved in the metadata table, you would have to store it twice for multiple predictions. Furthermore, we wanted to have a relation with the table we will explain in the next section where machine learning models are stored as entry. This would mean altering the metadata table, which we preferred not to do as explained at the start of this section.

The new table has a relation with both metadata and predictor. A fraud probability and execution time is stored for each prediction made on a metadata model. This probability will replace the one in the former metadata table and determine whether an entry is stored in the risk table.

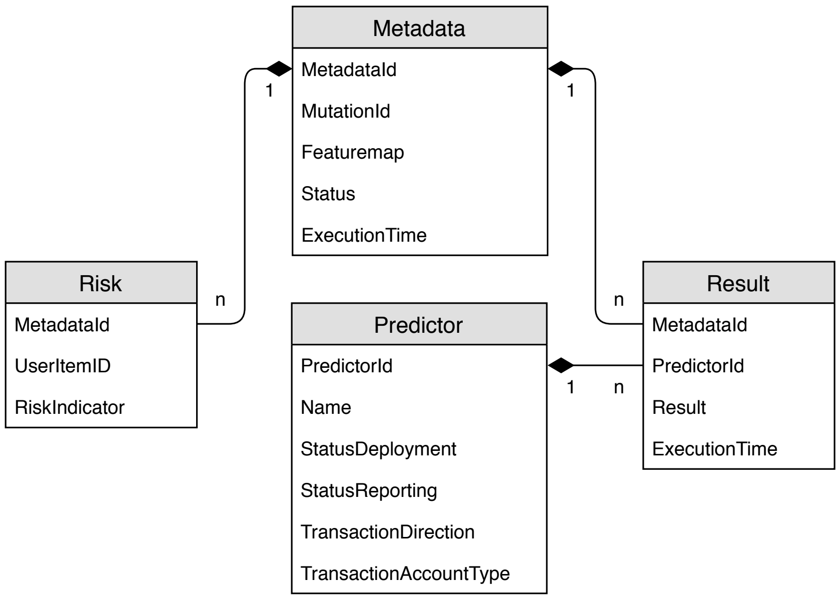

Figure 5.1: UML diagram of the new transaction monitoring software database architecture.

Predictor Table The other table is [redacted]. This table stores information about all machine learning models that have been or will be deployed. The table includes the following columns:

- `Name` is the standardised unique name of the machine learning model. This is used to make it easy to determine the type of machine learning model for humans.
- `StatusDeployment` is used to determine whether a model is still training, active or deprecated. This is used to determine whether predictions should be made using that model.
- `StatusReporting` is used to determine whether a model reports hits directly if they are above a certain threshold, or aggregates the highest scoring hits over a period and sends them to compliance only then. It is also possible to not report any hits of a machine learning model for validation purposes.
- `TransactionDirection/Type` is used to separate models that predict only for business or private transactions and incoming or outgoing, or both of either.
- `RiskIndicator` is used to determine with which `RiskIndicator` the prediction is scored if it is placed in the risk table. This is the indicator used on the compliance dashboard to indicate why the mutation is hit.

Migration of Current Tables As the result of storing the machine learning models in a new table, the same type of information is scattered around multiple places. To prevent this, a migration which moves the results from the metadata table to the new table has been performed. The migration consisted of three steps to be completed.

First, a `PredictorModel` is inserted connected to the currently used machine learning model. It is set to directly report to compliance and it only applies on private transactions, to mimic the previously hardcoded behaviour. Secondly, a new `PredictorModel` is created for each unique `MachineLearningModelName` in the `MetadataModel` table. These models are added to store all the data in a single and to prevent loss of data. As they are not operational any more, their deployment status is set to deprecated. Finally, the [redacted] table is filled with all the predictions made earlier. Each result has a connection with a metadata model and the corresponding predictor. The columns `MachineLearningModelName` and `FraudProbability` could now in principle be dropped from the table, although this is not scheduled to occur soon as the new system must demonstrate its performance first.

For deployment, only the first step is critical, because as soon as the refactor is deployed, only [redacted] will be created. The two other migration steps can be executed at any moment as no new entries in the metadata table will contain the two migrated columns. This turned out practicable, as while tested on smaller datasets the migrations took significantly long for the complete metadata table [redacted]. [redacted]. After this was reduced to $\mathcal{O}(n)$) by us, it took [redacted] to perform the second and [redacted] for the final migration step. The third step took significantly longer as besides looping over each metadata entry, a lot of insertions were executed.

5.1.3. Controller Architecture

In the previously active system, transaction monitoring mainly consisted of both a workflow that assessed a transaction based on rules and machine learning. The implementation of rules was similar to how we wanted to use machine learning models, where it is possible to add and remove machine learning models for certain types of transactions. However in contrast to how rules work, prediction made by machine learning models consist not just of logic and require communication with a separate server. While the models described in previous sections were needed to accommodate for this, the controller logic is fairly similar. Figure 5.2 gives a visual representation of the new controller architecture but omits the details of the `WorkflowTransactionMonitoringRule` as it remained the same.

[redacted]

Figure 5.2: Overview of the new transaction monitoring software controller architecture.

Similar to how it worked before, a transaction enters the transaction monitoring system in `Workflow TransactionMonitoringWorker`. It then goes through the applicable workflow for its type, private or business, incoming or outgoing. However, while earlier the rule workflow was called followed by a single call to the machine learning workflow, multiple calls to the machine learning workflow are now made. The amount of calls is determined by a query to the `TransactionMonitoringMachineLearningPredictor` model that fetches all predictors that are applicable to that transaction. This workflow gets a prediction from the machine learning server, saves an entry to the result table, saves an entry to the risk table if the threshold is exceeded and updates the metadata status to either be not flagged or needs investigation.

To get a prediction, a featuremap containing all the features for the predictor must be sent to the machine learning server. Before it is sent, an HTTP GET request is sent to the server to retrieve a list of required features. If multiple predictors are applicable on a single metadata model, some features will be required for both predictors. By passing along the created featuremap, calculation of duplication features can be prevented causing a decrease in execution time. Feature engineering has more implications which do not directly concern this workflow, which will be discussed in Section 6.3.1.

5.1.4. View Architecture

PHP code executed on bunq servers have access to the database via the models. Code outside of the backend must communicate with the database via an API. As the training of machine learning models is performed on python servers, it must use such an API. To accommodate the automatic insertion of new predictors, a RESTful API has been created. [redacted]. Using an API allows also for end-to-end testing of the complete workflow by inserting data and verifying the behaviour.

5.1.5. Machine Learning Server

The server that runs the machine learning model was implemented for running only a single supervised learning model. To be able to run different machine learning models in parallel, the server had to be refactored. There were a few things needed to accommodate this change.

First of all, when a model is uploaded, the name of this model should be unique, since other models can be run on the same machine. In the old server, each uploaded model was given the name `model.jar`. When a new model would be uploaded, this would cause the older model to be overwritten by the newer model. To solve this problem, a model is given a unique name based on a timestamp each time a model is uploaded to the server. This way, multiple models can exist on the same server. After uploading a model to the machine learning server it is also stored on a S3 bucket. This allows the server to fetch all models in case it crashes.

To make the possibility of having multiple models useful, the server should also be able to run predictions on a specific model. Therefore, the HTTP call for getting predictions has been changed to include the model name. This way, the backend is able to get predictions from specific models and compare results given by them.

During deployment of the refactored server and corresponding backend, there is a time where one of the systems is using an older version than the other. In the case that the machine learning server is already deployed and the backend is not, the server should still work as before. Therefore, the endpoints used previously to get a prediction from the server, should still exist. These endpoints return the predictions given from a default model, which is the model defined before updating the machine learning server.

5.2. Model for Business Transactions

[redacted]. By implementing the functionality for multiple models as explained in Section 5.1, a new model that is specialised for business transactions could be deployed and added to the table of active machine learning models and the transaction monitoring system would automatically make predictions. In Section 6.4 will be discussed how the training of a specialised model for business transactions works.

5.3. Outlier Detection

As explained in Chapter 2, supervised machine learning models are less accurate when the incoming transactions have different characteristics than what it has been trained on. To also detect outliers, we chose to use an autoencoder. The autoencoder will be trained at the same time the regular model will be trained, and it will classify every transaction as well. The prediction will be saved in the [redacted], just like predictions made by other machine learning models. However, this predictor is saved to be reporting to compliance in an aggregated manner. [redacted].

6
The Product: Automatic Training

As described in Chapter 2, the current process of training a machine learning model is a laborious task. It is done manually by collecting a dataset, labelling every transaction as either fraud or non fraud, converting it into a for the machine learning model readable format and running the training scripts. If the process would be automated, the required time and effort can be spend more productively. In this chapter we will discuss the design and implementation of automating this process.

6.1. Overview

The process of automatically training a model is going to require a few steps. Most of these steps run directly in the backend of bunq. The first step is collecting a dataset, which is the featuremap for each transaction ever made at bunq. The second step is to label all these transactions as either fraudulent or non-fraudulent. Then the model has to be trained on this dataset. Because training a machine learning model on datasets of this size requires a enormous amount of computing power, this is not done directly in the backend of bunq. A separate, very powerful EC2 instance will be used to do the heavy training computations. After the model is trained, the fourth step is to export the model as a POJO to another separate server, which hosts all machine learning models. The fifth and last step is to write an entry to the database indicating that the model is ready to use.

6.2. Software Architecture

At the start of the project, we designed a workflow structure for the process of automatically training machine learning models. This is depicted in Figure 6.1. As seen in this diagram, there is one main workflow called `WorkflowTransactionMonitoringMachineLearningTrainer`. This workflow is responsible for the whole training process in the backend, from preparing and storing the dataset to executing a request for training on the external server. To make things more clear, this workflow delegates some of its responsibilities to another workflow called `WorkflowTransactionMonitoringMachineLearningLabelMapping`. The responsibility of this workflow is to get all features of all transactions ever made, assign a label of either fraudulent or non-fraudulent to them and then store these mappings in JSON into an S3 bucket. The advantage of delegating these responsibilities to a separate workflow is that this process can then be batched. This is a requirement, as there are, at time of writing, millions of metadata models in the transaction monitoring metadata database. As the number of metadata entries is enormous, the trainer call the `WorkflowTransaction MonitoringMachineLearningLabelMapping` in batches.

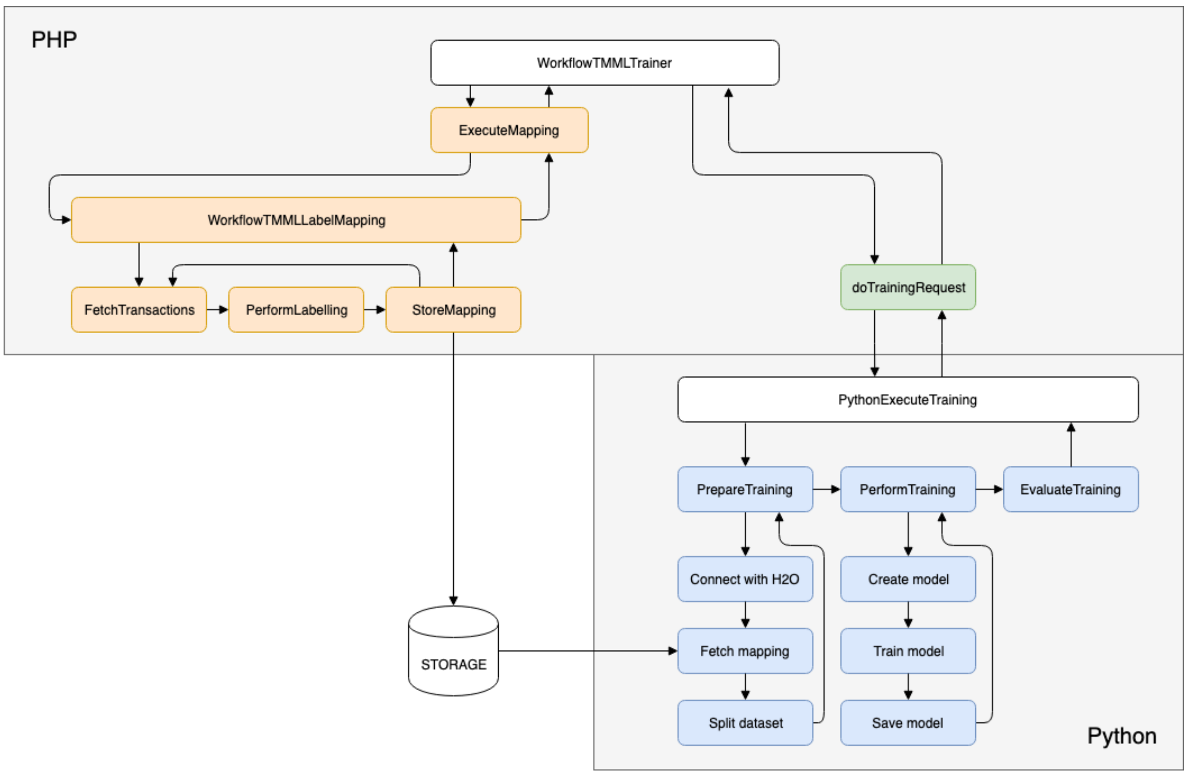

Figure 6.1: Software design diagram for automatic retraining of the ML system.

When all transactions have been processed and stored, the main workflow makes a call to the EC2 server to get the training process started. On this remote server, a Python script is run. This script is responsible for training the machine learning model. First it starts a local H2O-server. This is a Java server, running on the same machine as the script, which handles all the training and model generating. After this server is started, the dataset, which was previously uploaded to the S3 bucket, is then loaded from this bucket. For the H2O-server to read in this dataset, the JSON format of the dataset is changed to a DataFrame format. The dataset is then split into a training set and a validation set of respectively 80% and 20% of the data, after which the training is started on the training dataset. After this process has completed, a model in the form of a POJO is created, which is sent to the server where machine learning models are hosted using a unique name to be able to differentiate between the models on the server. From this server, these models can be called by the backend to get a prediction for a specific transaction. Because the backend does not know the name of the newly generated model, a database table, containing all machine learning models on the machine learning server, is updated with an entry for the new model.

6.3. Implementation: Version 1

When we started implementing the workflow as depicted in Figure 6.1, we came to the conclusion that the system was sub-optimal. This was mainly caused by factors that we did not take into account, because we were not as familiar with the system at the time of creating the software architecture. Therefore, we made some changes to the workflows and how they work. The resulting system is depicted in Figure 6.2. The diagram will be discussed in more detail below.

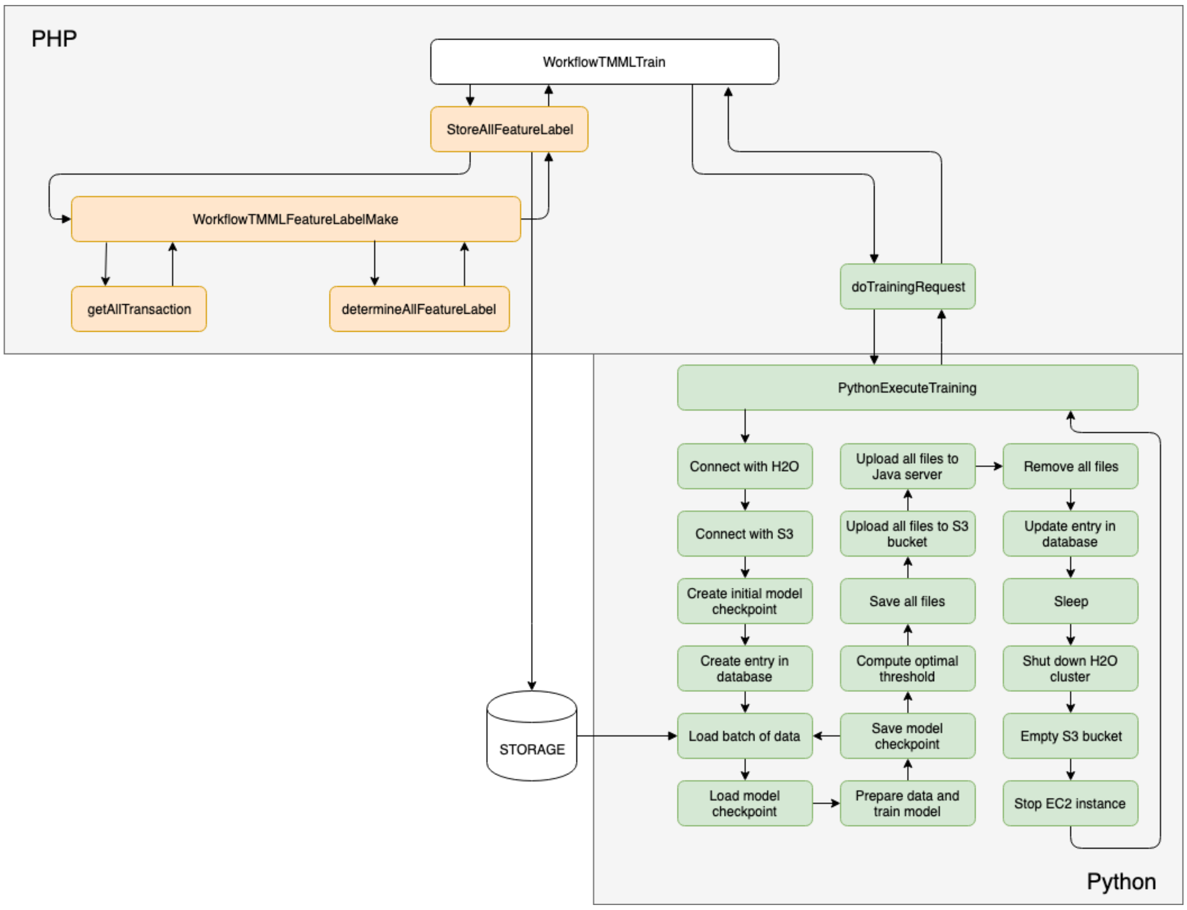

Figure 6.2: Software implementation for automatic retraining of the ML system.

6.3.1. Implementation Details: Feature Engineering

The automatic training is done by running a script in the backend, but before the training workflow can be started, the script ensures that all features for all transactions are calculated. Currently, these features are calculated when a transaction comes in and are then stored in a database object called `TransactionMonitoringMetadataModel`. This ensures that the feature engineering does not have to be done every time a machine learning model is trained. However, when new features are added, they should be calculated for all previous transactions as well. To make sure all features can be used in training the machine learning model, the features which are not yet calculated will be determined before any training will happen.

As feature engineering can be time consuming on large datasets, we parallelised this process. Because calculating a feature of a transaction is not dependant on any other feature, this process is easily parallelisable by executing the feature engineering of each transaction on a separate thread. However, there are a lot of transactions these features should be calculated for and creating threads uses a lot of overhead. Therefore, we decided to parallelise batches of transactions. Each batch runs on a seperate process and once a batch is finished, a new batch will be started. This continues until all features have been determined.

6.3.2. Implementation Details: Automatic Training Preparation in Backend

When the process of automatic training is started, the `WorkflowTransactionMonitoringMachineLearningTrain` is called by a script. This script is called by a cronjob that executes in a given interval of time, determined by bunq. The workflow calls the method `storeAllFeatureLabel`, which queries the transaction database to find its size. It then calls the workflow named `WorkflowTransactionMonitoringMachineLearningFeatureLabelMake` over batches of `TransactionMonitoringMetadataModel` objects. The method `getAllTransaction` fetches the required batch of transactions from the database. Then, in a call to `determineAllFeatureLabel`, for each transaction it is checked what metadata status it has. Examples of these statuses are [redacted]

[redacted]

[redacted], [redacted], and [redacted] [redacted]. Each of these statuses has a mapping to either fraudulent or not fraudulent. A JSON object is created with the featuremap and the a label of either [redacted]. Then back in `storeAllFeatureLabel`, the mapping is stored in the S3 bucket. This is done for all metadata models in the database.

When this workflow is completed, `doTrainingRequest` is called. When this request is done, an EC2 instance, containing the Python training scripts, is started.

6.3.3. Implementation Details: Python Training Scripts

The Python scripts that train the machine learning model are hosted on a remote EC2 instance. The script first initialises H2O and connects with the S3 bucket where the training data is stored. It created an entry in the database indicating that a model is training. Then, it trains the machine learning model in batches. Afterwards, it saves the POJO, jar, snapshot of the model, and threshold. It uploads these files to the Java server and S3 bucket before removing them again. It then enters a predefined period of sleep, such that a bunq employee has a chance to take a look at the data and snapshot if they want to. When the sleeping period is over, the H2O cluster gets shut down, the S3 bucket containing the training data is emptied, and finally the EC2 instance is stopped.

6.3.4. Notable Differences

The implementation as depicted in Figure 6.2 differs substantially from our initial design as seen in Figure 6.1. First of all, some of the names have changed. This is done to match the style enforced by bunq. Secondly, the underlying processes in `PythonExecuteTraining` are substantially different. The script trains the model in batches instead of on the whole dataset at once to reduce memory usage and increase training speed. Also, the model and other files are now stored as a backup on a seperate S3 bucket as well. Lastly, we also removed the `storeMapping` and moved the logic that was in there up to `storeAllFeatureLabel`, as it did not make sense to have the uploading of the files in the workflow that is supposed to map labels to either fraud or not fraud. Instead, we added the functionality of `storeMapping` to `StoreAllFeatureLabel`.

6.4. Implementation: Version 2

During the development of the automatic training process, we also started implementing retrospective labelling. The implementation details of that are explained further in Section 7.2, but it also meant a complete rewrite of the automatic training process. The first implementation was not designed to accommodate multiple ways of labelling the training dataset. Together with some other improvements in mind, we designed a new system for creating a training dataset and executing the training process. The feature engineering part has not been changed since that is a separate process which did not require improvements at this time.

6.4.1. Implementation Details: Automatic Training Preparation in Backend

There is one large difference between the first system and the newly designed automatic training system. This difference is in the way the training dataset is created. Where in the old system dataset batches were created and the training script executed afterwards, in the new system there is a complete seperate workflow for creating a training dataset batch. This worfklow is depicted in Figure 6.3.

As seen in this diagram, the workflow `WorfklowTransactionMonitoringMachineLearningTraining DatasetBatchCreate` has four methods. It starts by determining dataset entries from the feature maps and status codes in the `TransactionMonitoringMetadataModel` objects stored in the database. This process is run in a separate workflow again and is comparable to how the datasets were created in our initial design. After creating these dataset entries, another workflow is called: `WorkflowTransactionMonitoring MachineLearningDetermineBatchRetrospectiveLabellingDetermine`. This mouthful of a workflow will be explained in detail in Section 7.2, but in general it creates dataset entries for transactions which do not have a fraudulent status code, but retrospectively can be seen as fraudulent. The third method of the dataset batch creation workflow combines the dataset entries created in the previous two methods. The dataset entries created in the first method comprise all transactions possible in this batch, however the entries created by the retrospective labelling workflow only contain the transactions which have been retrospectively labelled. To combine these entries, the retrospective labelling entries overwrite the entries created from feature maps and status codes. When the complete dataset batch is created in memory, it is stored in an S3 bucket in the fourth method of the workflow. This process is also comparable to our initial design.

Another factor we did not incorporate in our first version of the automatic training workflows was having the possibility to create datasets of only private users or company users. To accomplish this, we filter by these

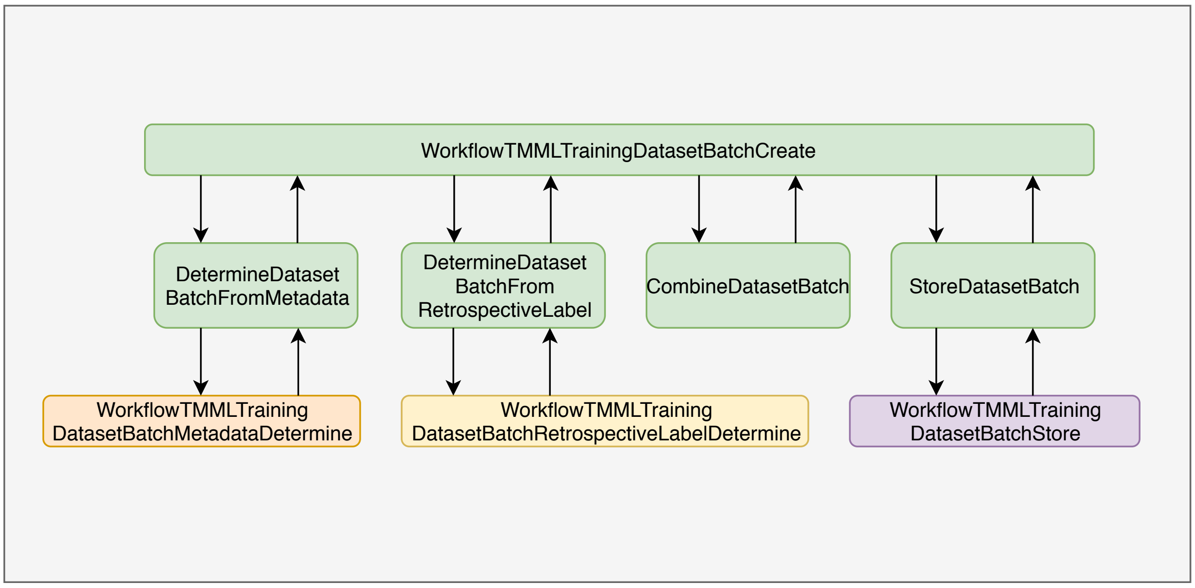
WorkflowTMMLTrainingDatasetBatchCreate
DetermineDataset BatchFromMetadata
DetermineDataset BatchFrom RetrospectiveLabel
CombineDatasetBatch
StoreDatasetBatch
WorkflowTMMLTraining DatasetBatchMetadataDetermine
WorkflowTMMLTraining DatasetBatchRetrospectiveLabelDetermine
WorkflowTMMLTraining DatasetBatchStore

Figure 6.3: Software implementation version 2 for automatic retraining of the ML system.

specific user types when retrieving the `TransactionMonitoringMetadataModel` objects. The datasets are then stored on the S3 bucket with a prefix containing either [redacted]_
[redacted] or [redacted]_
[redacted] describing a dataset of data of private users or company users respectively.

6.4.2. Implementation Details: Python Training Scripts

In the second version of our automatic retraining workflow, the Python scripts that train the model have been altered as well. The single original scripts has been split up into several scripts such that we can easily add new models and functionalities. The diagram that describes the new main training script can be found in Figure 6.4.

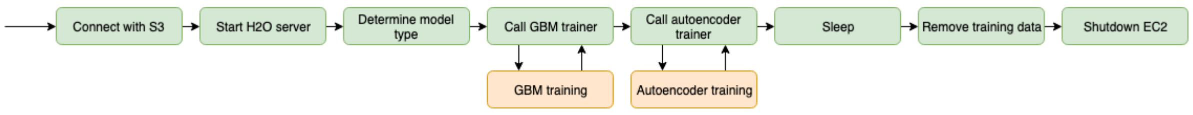

Figure 6.4: Python main script version 2.

The script that is depicted in Figure 6.4 is the main training script. It connects with the S3 bucket where the training data is stored and starts an H2O server. The script then determines the model type by checking whether all files in the bucket contain either the prefix [redacted]_
[redacted] or [redacted]_
[redacted] as discussed in Section 6.4.1. If not all the files in the S3 bucket have the same prefix, an exception is thrown. It then calls the `GBM trainer`, which handles the training and uploading of the GBM. Afterwards, it calls the `Autoencoder trainer`, which trains and uploads the autoencoder. Lastly, sleeps for a set period of time, removes all training data, and shuts down the EC2 instance where it runs on.

The `GBM trainer` is described in Figure 6.5 below.

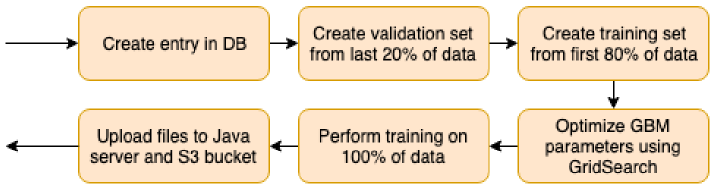
Create entry in DB
Create validation set from last 20% of data
Create training set from first 80% of data
Optimize GBM parameters using GridSearch
Perform training on 100% of data
Upload files to Java server and S3 bucket

Figure 6.5: Python Gradient Boosting Machine training script version 2.

The script first creates an entry in the database for the model indicating that it is currently training. Then, the script retrieves the newest 20% of transactions from the S3 bucket and creates a validation set from that data. It also creates a training set from the rest of the data, being the 80% oldest transactions. Then, it uses GridSearch to build and train models with different sets of randomly chosen GBM parameters, such as amount of trees and maximum tree depth, and returns the parameters of the most successful model based on F2 score. The script then trains a model on 100% of the data using the optimal parameters found by the GridSearch. After the training, the script uploads a POJO of the model, jar file, threshold and snapshot to the S3 bucket and the POJO, jar and threshold to the Java server. The `Autoencoder trainer` goes through the same steps, but does not upload a threshold as it does not require one. It also uses different parameters for the GridSearch to optimise.

6.4.3. Considerable Differences

As shown in Section 6.4.1, the main difference between our first implementation is that training dataset batches are created in a separate workflow. We decided in favour of this design, because it abstracts the logic needed to create a batch of dataset entries better than before. This abstraction makes the code base more readable and easier to maintain. It also allows for better implementation of using different data sources in the dataset; in our case the feature map with its status code and the retrospective labelling workflow.

6.5. Development and Testing

During development we used Localstack to host a local AWS S3 bucket. We then stored data from our local database in the local S3 bucket. When everything worked nicely, we wrote a Python script to access this bucket and store it in a data frame, and trained our classifier on the retrieved data. When everything worked locally, we were granted access to AWS for further testing. Here, we created our S3 bucket and EC2 instance, and got everything to work once again. Then, our merge requests were thoroughly checked by colleagues to ensure code quality and correctness. After a meeting with DevOps to configure the EC2 instance, S3 bucket, Java server changes and cronjob, our code was uploaded on the test environment Triage, where we could do further testing. Once everything was working correctly, our code was deployed in production.

Of course did we not only test our code by seeing how it behaves in our local and the Triage environment, we also wrote a large amount of tests. Our tests consisted of both unit tests, where we tested whether the individual states in the workflows returned the correct values, and end to end tests, where we checked whether a workflow returned the correct results.

7 The product: Improving Data Quality

In this chapter we will discuss the changes we made to improve the data that is used to train the machine learning models used for transaction monitoring. First, Section 7.1 will describe the process of designing more features. This essentially gives the machine learning model a greater variety of data to train on. After that, Section 7.2 will describe the design of a system which labels transactions of users that later turned out to be fraudulent. The process of designing new features was a continuous one in the sense that over the course of the entire project we implemented new features as they were requested. This is in contrast to the retrospective labelling implementation, which primarily happened at the end of the project.

7.1. Feature Engineering

During the project, we have increased the amount of features [redacted]. This is a significant increase in data that not only has to be calculated and stored, but also sent to the training server to be processed. This section will first give a general overview of what types of features have been implemented in Section 7.1.1. Next, Section 7.1.2 will discuss the process of developing these features. Finally, the complete list of features can be found in Appendix F.

7.1.1. Overview

To give a rough idea what features are used to train the machine learning model, Section 7.1.2 provides an overview of all types of features that have been implemented. All features types are associated with a transaction. For example, monetary account type features relate to the monetary account that is associated with the transaction. [redacted]

7.1.2. Development and Testing

Prior to our project, the transaction monitoring system already had a software architecture for calculating feature values. Since we have not changed the general structure of this, we will not go into depth of how this is implemented. However, even though the development of new features did not require a large intervention in the software architecture, the development required a significant amount of time. This had two main causes, which we will discuss in this section.

[redacted]

[redacted]

[redacted]

7.2. Retrospective labelling

To further improve the data used for training the machine learning model, we implemented a process we chose to call retrospective labelling. This name comes from the notion that transactions of users that turned out to be fraudulent under certain circumstances have to be labelled as fraud in hindsight. This section will first describe the initial design in Section 7.2.1. Next, the software design that was implemented will be discussed in Section 7.2.2.

7.2.1. Initial Design

In the initial design, the retrospective labelling workflow was called before the automatic training workflow. This section will shortly summarise the behaviour of that workflow, followed by an explanation of why this design was not implemented.

Behaviour Summary Each time before training, the retrospective labelling workflow would consult a table where which transactions had been retrospective labelled before was stored. Using this table, the workflow would be executed with the transactions that had not gone through this workflow before. If such a transaction was from a user that had a fraudulent transaction at any point, the transaction was tested against a couple of rules which are composed by compliance. If any of these rules applied to the transaction, such as whether the amount of the transaction was significantly high, this transaction would then also automatically be labelled as fraudulent. This would then be stored in the `TransactionMonitoringMetadata`, just like when a transaction is manually flagged as fraudulent.

Encountered Problems The consequence of this design is that large quantities of transactions could be automatically labelled as fraudulent. This has a serious downside, in that when this process goes wrong it makes a big impact that could be difficult to be undone. Furthermore, storing this information is only needed and used for training the machine learning model. Together with bunq we decided to make a new design that would not have this problem.

7.2.2. Implemented Software Architecture

The new design works similar to the old design in the sense that before training, transactions will be labelled as fraudulent if the user has had a fraudulent transaction at any other point and if certain rules apply to that transaction. However, instead of storing this data, it is determined while the training process happens. The integration in that process is described in Section 6.4.

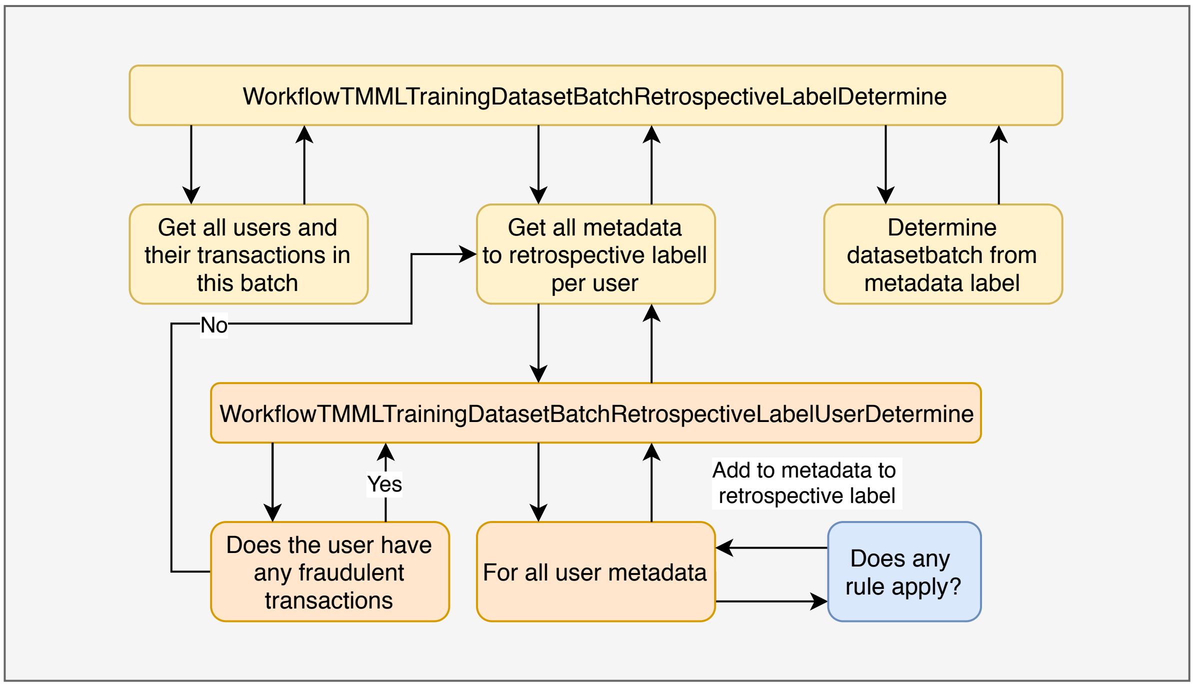
WorkflowTMMLTrainingDatasetBatchRetrospectiveLabelDetermine
Get all users and their transactions in this batch
Get all metadata to retrospective labell per user
Determine datasetbatch from metadata label
No
WorkflowTMMLTrainingDatasetBatchRetrospectiveLabelUserDetermine
Yes
Add to metadata to retrospective label
Does the user have any fraudulent transactions
For all user metadata
Does any rule apply?

Figure 7.1: Software architecture of the implemented retrospective labelling workflow.

The retrospective labelling process is depicted in more detail in Figure 7.1. For each batch of transactions that is prepared for training, `WorkflowTMMLTrainingDatasetBatchRetrospectiveLabelDetermine` is called. This workflow calls `WorkflowTMMLTrainingDatasetBatchRetrospectiveLabelUserDetermine` for each user with their transactions in this batch. If the user has had a fraudulent transaction at any point, all their transactions in this batch are assessed according to a set of rules. These transactions will be labelled as fraudulent and overwritten in the original batch before sent to training.

8 Discussion and Conclusion

In this chapter we will discuss several issues and assess whether the project can be considered a success. Furthermore, the value added by our product for bunq will be evaluated. We will also discuss the knowledge we have gathered during this project.

8.1. Discussion

In this section we will briefly discuss the accomplishments of this project, as well as some of the issues we encountered.

8.1.1. Accomplishments

The product we delivered is a fully functioning automated training system, capable of training both private and business machine learning models. We also tripled the amount of input features for the model to train on, and we implemented retrospective labelling to further increase the quality of the input data. Lastly, we implemented an unsupervised learning model to catch new and undiscovered types of fraud. Every system is completely integrated into the existing bunq backend, and thoroughly tested through unit tests and integration tests. The code has been deployed to the production environment.

8.1.2. Issues

The main issue we encountered was that merging our software into the `develop` branch can take a large amount of time, sometimes up to three weeks. Often, our software was functional fairly quickly, but due to the strict coding guidelines we were not as familiar with as regular employees and high requirements with regards to code, various revisions were often needed before our software would be merged.

Due to this lack of time, we were not able to experiment with various machine learning parameters and input features to verify performance. Since training a model takes a lot of time and making a comparison between models requires them to classify transactions in parallel for a significant period, it would be difficult to compare various slightly different models with each other in our limited timespan of ten weeks.

8.2. Conclusion

In this section we will assess whether the project can be considered a success. We will also discuss the value created for bunq with our project, as well as the lessons we learned during the project.

8.2.1. Measurement of Success

To assess the success of our project, we are going to take a look at the requirements we initially set in Appendix C.1. We can see that all must haves have been satisfied by our project. As for should haves, we did not implement a reinforcement learning algorithm as there simply was not enough time. We did however implement the automated optimisation of parameters. Our could have has also been satisfied. The would have of using the most optimal reinforcement learning algorithm was not achieved during our project. We also added several other implementations that were not mentioned in the initial requirements. These additional implementations can be found in Appendix F.

All things considered, we believe that our project can definitely be considered a success, as all must have requirements are implemented together with both listed and other new requirements.

8.2.2. Value

Although bunq is obligated to monitor all transactions for suspicious behaviour, it is not an activity providing revenue. This causes an incentive to keep the costs minimised while still performing this important task for society. False positive hits by the machine learning models require the compliance department to do unnecessary work. Frequently training new models on recent data could decrease the false positive rate, but this required manual labour to collect, process and label the data. By automating this process and improving the quality of training data together with additional models classifying other mutations, the costs will drop and the quality will improve.

8.2.3. Lessons

Working on a project like this, with impact on critical systems of a fast growing mobile banks is exciting. All our merge requests were reviewed by at least two developers of bunq, allowing us to learn best practises from them.

Also, development of software in a company differs from university projects in terms of deployment, especially at a bank. We had never experienced the deployment of a piece of software into an existing system before. This was a very interesting and informative experience.

Another very exiting aspect of our project was working with enormous amounts of data. We had all followed various courses on big data and computational intelligence, but we had never truly experienced it with such amounts of data in practice. This really taught us the importance of creating efficient code.

9 Recommendations and Ethics

In this chapter we will give recommendations on what could be improved upon and discuss several of the ethical issues at stake.

9.1. Recommendations

In this section we will discuss things that could be implemented to further increase the quality of our product.

9.1.1. Reinforcement Learning Algorithm

Reinforcement learning could improve the accuracy of the predictions generated by the machine learning model, since it is able to generate predictions while taking sequences of transactions into account, instead of just looking at the single current transaction. Although it was a moonshot, one of our milestones for this project was implementing a reinforcement learning algorithm. Since we did not have the time to implement this, a recommendation would be to implement this algorithm instead of the Gradient Boosting Machine that is being used for supervised learning.

9.1.2. Speeding up the Training Process

Speeding up the training process could save bunq a lot of money on servers costs. Faster training times would also lead to more convenience and easier experimentation with different settings and parameters.

There are multiple ways to speed up the training process. The current system only allows training the models sequentially. An upgrade would be for the machine learning server to allow for training in parallel, such that multiple models could train at the same time. This way, the training scripts could load a data batch where all models train on, instead of fetching all batches separately for all machine learning models. Also, frameworks designed to handle big data like Spark could be used for batch processing to speed things up as well.

9.1.3. Business Feature Optimisation

As it stands now, a business specific model can be trained, and there are several business-specific features. However, more business features could be created to potentially increase the accuracy of the model.

9.1.4. Parameter Optimisation

Both the Gradient Boosting Machine and the autoencoder use GridSearch to find the optimal combination of parameters from predefined possibilities. However, as time continues and bunq gains a better insight in the models and how they perform, further possibilities for the GridSearch could be added to boost the performance of both models.

9.2. Ethics

In this section, we will take a look at the ethical issues that arise from our project.

9.2.1. Bias in Unsupervised Learning

[redacted]

9.2.2. Impact on the Environment

Training a machine learning model on large amounts of data requires significant computational power. All computing hardware requires energy and depending on the way it is generated a significant footprint on the environment can be made. Training another type of machine learning model is calculated to cause an CO_2 emission equal to the lifespan of a car [18].

Although Amazon states their goal is to achieve 100% renewable energy usage [1], in February 2019 Greenpeace again accused Amazon to not fulfil their promise [7].

Previously, bunq owned several servers which required a lot of maintenance and were not used at full potential throughout the day. Moving to the cloud eliminated the need of in-house server maintenance and decreased the idle time of the servers as scheduling can optimize the usage of computing power. Moving bunq's complete software stack from Amazon Web Services to another more environmental friendly cloud computing provider could decrease its environmental footprint further, although this is probably not worth the resources as Amazon is working to reach the 100% renewable energy usage.

A
Project Description

In this appendix we will show the original project description and other relevant information as found on BEPSys.

A.1. About the Project

We're a bank, which comes with certain obligations. Among others, we need to filter all incoming and outgoing transactions. We built a transaction monitoring model using state-of-the-art machine learning techniques to make sure that suspicious transactions are being filtered and checked.

We obviously want to work as efficient as possible, which means that we want to decrease the number of false positive hits (= manual work) while always staying one step ahead of fraudsters. To achieve this it is required that the machine learning model is retrained frequently to learn about new types of fraud.

It's up to you to automate the process of retraining the model (while considering technologies such as AWS Sagemaker, Apache Spark for scalability) which means that our compliance team should only monitor the behavior of the system instead of actively retraining it. Furthermore, we want you to improve the performance of the current system through better choices of algorithms (e.g. unsupervised learning using auto-encoders or reinforcement learning) and better input representations.

A.2. About bunq

Some years ago a bunch of coders decided they would challenge the status quo. Together we set out to create the true alternative to traditional banking. A bank that would be different. The bank of the free.

You will join a team of over 80 passionate bunqers who've come from all over the world (13 nationalities and counting) to build something amazing together. Waking up with brilliant ideas and going to bed knowing that you've made them happen. That's working at bunq. We are using the unlimited power and possibilities of code to bring innovation to a place where stagnation is the status quo: the finance industry. We don't limit ourselves to what has been done or could be done: we make shit happen.

B Software Development

In this chapter, various approaches to software development and programs that we will use during this project will be discussed. As the purpose of this project is deploy a better transaction monitoring system in production, the process of this at bunq will also be explained.

B.1. Usage of Git

Git is a version-tracking tool that is widely used among programmers. For this project, we will also be working with Git. GitLab is the web-based development lifecycle application that we will use to manage our Git-repository. GitLab was provided to us by bunq.

Our Git-workflow was as follows: We had our own branch `feature/CIT/incremental#3552_improving_transaction_monitoring_model`, branched from the `develop` branch, which was used as bunq's master-branch. We would than make merge requests to that branch in small iterations that needed to be approved by at least two senior developers before merging. When a functionality was ready to be deployed, the branch `feature/CIT/incremental#3552_improving_transaction_monitoring_model` would be merged in to `develop`. From `develop`, releases would then be deployed to the production version.

B.2. Deployments and Hotfixes

Deployment of newly added features is done on Tuesdays. This is done by branching the `develop` branch into a newly created `release` branch. Then, DevOps runs all the needed scripts, migrations, and tests to get everything working on the `release` branch. When everything is working as it should, the deployed version gets substituted by the `release` branch. After each release cycle, a list of checks is performed to verify that the core business is not affected.

Hotfixes are similar to deployments, but they do not necessarily happen on Tuesdays only. Another difference is that, instead of branching `develop` into a newly created `release` branch, `develop` is cherry-picked for the specific needed commits to a new `release` branch.

B.3. Hard- and Software

We were all set up with a MacBook running macOS Mojave to use during the project. The bunq core on our computers runs in a Docker container to ensure proper configuration. We will be using PhpStorm as our IDE to develop PHP code. For Python, we will be using PyCharm. We will be using PHP version 7, and Python version 3.

B.4. Triage

Before features are released to the production environment, they are tested on a separate environment called Triage. All the different processes of production are also executed here and a large, but not equal to production size, database is available with fake data. To allow us to thoroughly test our workflows before deployment, also a separate machine learning server and python training server were made available.

C Research Phase: Project Plan

C.1. Requirements

After going through the research phase as documented in Chapter 2, we concluded that the final product has to meet the requirements as specified below. The requirements are described using the MoSCoW model.

Must haves
The transaction monitoring system must have the functionality to be automatically retrained with new data every month.
A data scientist at bunq should be able to select which model to use for the actual risk predictions.
Labels assigned by compliance must automatically be mapped for use by the transaction monitoring system.
Labelling transactions as fraudulent should be improved by analysing older data associated with newer transactions being marked as fraudulent by compliance.
A separate model must be able to handle business transactions by incorporating new features that are specific to businesses.
The transaction monitoring system must incorporate an unsupervised learning model to detect outliers.
The models must use more features to improve predictions.
Should haves
The models should be trained using a reinforcement learning approach to better analyse sequences of transactions.
The system should have an automated way to optimise parameters.
Could haves
Newly trained models could run next to the current model for one week to assess its performance while it does not effectively labels incoming transactions before deciding to use it in production.
Would haves
The system would use the most optimal reinforcement learning algorithm.

Table C.1: Requirements of a functional solution defined according to the MoSCoW method.

C.2. Roadmap

This section will propose a roadmap of the product that has to be developed. This will be done by estimating the feasibility of solving specific sub-problems and taking into account the time constraints of this project.

Week	Date	Description
1	Friday 26-4-2019	Get familiar with the company, analysing the problem and writing research report draft.
2	Tuesday 30-4-2019	Finalise research report and discuss with company and university supervisor.
	Friday 3-5-2019	Draft software design and start implementation.
3	Friday 10-5-2019	Finish prototype of automated labelling and retraining.
	Friday 10-5-2019	Meet with company- and university supervisor together (if available).
4	Monday 13-5-2019	Interview compliance on new features regarding both personal and business transactions.
	Friday 17-5-2019	Finish prototype of training with new feature-maps.
5	In week 5	Start implementing unsupervised learning model.
	Friday 24-5-2019	Finalise automated retraining with improved features and labelling.
	Friday 24-5-2019	Update software design and report.
6	In week 6	Start implementing reinforcement learning model.
	Friday 31-5-2019	Finish prototype unsupervised learning model.
	Friday 31-5-2019	First upload to SIG
7	In week 7	Complete working combination of automated retraining and unsupervised learning model.
		Establish to what extend current product is production-ready.
	Friday 7-6-2019	Finish prototype of reinforcement learning model.
	Friday 7-6-2019	Process feedback from SIG and update report.
8	In week 8	Finish reinforcement learning model and interview company on what further product features have priority.
9	Wednesday 19-6-2019	Make product production ready.
	Friday 21-6-2019	Final upload to SIG.
	Friday 21-6-2019	Send final draft report to company- and university supervisors.
10	Tuesday 25-6-2019	Deliver final report.
11	In week 11	Final presentation.

D

Software Improvement Group (SIG)

To check the quality of our code, we have uploaded it to the Software Improvement Group. The feedback we received can be found below.

D.1. First Feedback by SIG

The original feedback by SIG is in Dutch and can be found below.

> De code van het systeem scoort 3.1 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Module Coupling en Duplication vanwege de lagere deelscores als mogelijke verbeterpunten.
>
> Voor Module Coupling wordt er gekeken naar het percentage van de code wat relatief vaak wordt aangeroepen. Normaal gesproken zorgt code die vaak aangeroepen wordt voor een minder stabiel systeem omdat veranderingen binnen dit type code kan leiden tot aanpassingen op veel verschillende plaatsen. Dit soort code wordt niet alleen vaak aangeroepen, maar wordt vaak ook vanzelf groter omdat alle functionaliteit wordt gecentraliseerd. Om zowel de grootte als het aantal aanroepen te verminderen zouden deze functionaliteiten gescheiden kunnen worden, wat er ook toe zou leiden dat de afzonderlijke functionaliteiten makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden worden. Bij grote bestanden met veel verantwoordelijkheid kan de mate van coupling tussen bestanden ook worden verbeterd door de interface en implementatie van elkaar te scheiden. Door de toevoeging van de interface ontstaat meer flexibiliteit om de implementatie op een later moment aan te passen.
>
> Voorbeelden in jullie project:
>
> - [redacted].[redacted]
> - [redacted].[redacted]
>
> Bij Duplication wordt gekeken naar de hoeveelheid gedupliceerde code. We kijken hierbij ook naar de hoeveelheid redundantie, dus een duplicaat met tien kopieën zal voor de score sterker meetellen dan een duplicaat met twee kopieën. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om de hoeveelheid gedupliceerde code zo laag mogelijk te houden. Na verloop van tijd zal de gedupliceerde code moeten worden aangepast. Dit leidt niet alleen tot extra werk, aangezien op dat moment alle kopieën tegelijk moeten worden veranderd, maar is ook foutgevoelig omdat de kans bestaat dat één van de kopieën per ongeluk wordt vergeten.
>
> Voorbeelden in jullie project:
>
> - [redacted].[redacted]
> - [redacted].[redacted]
>
> De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

> Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

The translation to English is as follows:

> The code of the system scores 3.1 stars on our maintainability model, which means that the code is market average maintainable. We see Module Coupling and Duplication as possible points of improvement due to their relatively low part scores.
>
> For Module Coupling, the percentage of the code that is relatively often invoked is considered. Normally, code that is often called causes a less stable system because changes within this type of code can lead to changes in many different places. This type of code is not only often invoked, but often also increases automatically because all functionality is centralised. To reduce both the size and the number of calls, these functionalities could be separated, which would also make the individual functionalities easier to understand and test, and therefore easier to maintain. For large files with a lot of responsibility, the degree of coupling between files can also be improved by separating the interface and implementation. The addition of an interface gives more flexibility to adjust the implementation at a later time.
>
> Examples in your project:
>
> - [redacted].[redacted]
> - [redacted].[redacted]
>
> Duplication looks at the amount of duplicated code. We also look at the amount of redundancy, so a duplicate with ten copies will count more strongly for the score than a duplicate with two copies. From the point of view of maintainability, it is desirable to keep the amount of duplicated code as low as possible. After a while the duplicated code will have to be adjusted. This not only leads to extra work, since all copies must be changed at the same time, but is also prone to error because there is a chance that one of the copies will be accidentally forgotten.
>
> Examples in your project:
>
> - [redacted].[redacted]
> - [redacted].[redacted]
>
> The presence of test code is certainly promising. The amount of test code also looks good, hopefully it will be possible to keep writing new tests as well as adding new production code.
>
> In general, some improvement is still possible, hopefully it will be possible to achieve this during the rest of the development phase.

D.2. Our Reaction

Most of the feedback is positive about the design of the system and the quantity of tests written for the code. We can relate to the fact that some code is used in a lot of different places as pointed out in the first part of the feedback about Module Coupling. However, in our opinion the classes indicated to suffer from the problem are not related, as both classes are relative small, 46 lines of code each, only implementing a function of the abstract class [redacted] and only called once in the workflow where the features of a mutation are determined. A reason for this could be that only relevant parts of but not the complete codebase of bunq is shared with SIG. For the next feedback we could include more classes which could be relevant to comprehend the complete view, giving SIG a better opportunity to give valuable feedback.

Code duplication was indeed present in some classes indicated by SIG. We solved this by creating a library containing functions with the repeated code. There were other places where code duplication was happening as well. However, some of these occurrences were unsolvable because we simply followed the style of the existing bunq backend, and we did not want to break this style. We did keep the feedback from SIG in mind when creating additional code. For example, when adding the training of additional models to the Python script, we transferred all of the more general functions to a `util` file to avoid duplication.

E Additional Implementations

During our project, bunq employees often visited us asking for help regarding various tasks, often related to transaction monitoring. Although these tasks were not necessarily a part of our project, we chose to implement a few of these requests as they were related to transaction monitoring and we saw it as our responsibility to catch as many fraudsters as possible. The various extra features we implemented can be found below.

- [redacted].
- [redacted].
- [redacted].
- [redacted].

F
Machine Learning Features

In this appendix the features created to improve the performance of the predictors are listed.

- [redacted].
- [redacted].
- [redacted]
  - [redacted].[redacted]
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].

- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].
- [redacted].

G
Glossary

Transaction A transfer of money in any form between bunq users or from bunq users to other banks or vise versa.

Private/Person transaction Used to refer to transactions made by persons in contrast to businesses.

Compliance Team within bunq that ensures that transactions are compliant with the law.

H2O A framework to train machine learning model.

Dataframe Dataformat that the H2O instance uses to train with.

EC2 Amazon Elastic Compute Cloud. Place where our machine learning train script is ran.

S3 Amazon Simple Storage Service. A place to store data.

Feature(map) A feature refers to a property of a transaction which is used to train the machine learning model and classify transactions. A featuremap is the collection of features that belongs to a transaction.

Fraud probability/score A value between 0 and 1 that a machine learning model produces. A higher value indicates a higher probability of the transaction being fraudulent.

Machine learning model/predictor A model trained to classify transactions as fraudulent or non-fraudulent. Machine learning predictor is how it is named in the backend.

Transaction monitoring metadata The name of the model that contains information about the transaction, such as the featuremap. This information is used by the transaction monitoring system. Also simply referred to as metadata.

Triage Test environment used by bunq.

Financial Intelligence Unit The Financial Intelligence Unit, often abbreviated to FIU, is an authority where various instances and companies have to report potential financial fraud.

De Nederlandsche Bank The central bank in the Netherlands. Often abbreviated to DNB.

Gradient Boosting Machine A supervised machine learning technique. Often abbreviated to GBM.

Autoencoder An unsupervised machine learning technique.

GridSearch A technique for optimising hyperparameters for machine learning models.

Automated transaction monitoring

Presentation date: 03-07-2019

Project description

Challenge
For the past weeks, our project has been to automate and improve the training of the machine learning model used for transaction monitoring at bunq, a Dutch mobile bank. The ultimate goal of this project was to increase the accuracy of the machine learning model, which would result in a decrease of false positives, enabling bunq to catch more fraudsters. Concretely, we were tasked with automating the retraining process, improving the quality of the input data by implementing additional features and applying retrospective labelling to previous fraudulent transactions, and implementing additional machine learning models. Additionally, to achieve these goals, some refactoring of existing code had to be done.

Research
During the research phase, we focused on software designs, finding the optimal frameworks, defining measurements of success, and selecting the best types of machine learning models for our context.

Process
During the project, we have worked to achieve our goal at the office of bunq in Amsterdam.

Product
The product we delivered is a fully functioning automated retraining system, capable of training both private and business machine learning models. We also tripled the amount of input features for the model to train on, and we implemented retrospective labelling to further increase the quality of the input data. Lastly, we implemented an unsupervised learning model to catch new and undiscovered types of fraud.

Every system is completely integrated into the existing bunq backend, and thoroughly tested through unit tests and integration test. The code has been deployed to the production environment.

Outlook
Our product is production ready. Recommendations to further improve the system are supporting training multiple models in parallel and starting the training while data is still uploading to increase training speed, implementing reinforcement learning for better accuracy, adding more business features and further parameter optimisation.

Project team

Rico Hageman
Interests
Machine learning, blockchain, algorithms

Contributions
Refactor transaction monitoring, migrations

Bastijn Kostense
Interests
Data mining, machine learning, data science

Contributions
Machine learning models, automatic training workflow

Bram van Walraven
Interests
Software architecture, machine learning

Contributions
Automatic training, retrospective labelling, machine learning server

Hilco van der Wilk
Interests
Software engineering, data mining

Contributions
Refactor transaction monitoring, automatic training workflow

Contributions by all team members
Report writing, code reviewing, code testing, planning, feature engineering, preparing final presentation

Contact information

Project team

Rico Hageman	rico-hageman@live.nl
Bastijn Kostense	bastijnkostense@hotmail.nl
Bram van Walraven	bram.van.walraven@gmail.com
Hilco van der Wilk	hilcovanderwilk@gmail.com

TU Delft EEMCS Coach
Sander van den Oever s.y.vandenoever@tudelft.nl

Client (bunq)
Wessel Van wessel@bunq.com

Bibliography

[1] AWS. *AWS & Sustainability*, 2015. `https://aws.amazon.com/about-aws/sustainability/` [Accessed: 2-5-2019].

[2] De Nederlandsche Bank. *Register Depositogarantiestelsel*, 2019. `https://www.dnb.nl/toezichtprofessioneel/openbaar-register/WFTDG/detail.jsp?id=c9dc52ce0358e311b55a005056b672cf` [Accessed: 2-5-2019].

[3] Richard J Bolton and David J Hand. Statistical fraud detection: A review. *Statistical science*, pages 235–249, 2002.

[4] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[5] Jinghui Chen, Saket Sathe, Charu Aggarwal, and Deepak Turaga. *Outlier Detection with Autoencoder Ensembles*, pages 90–98. doi: 10.1137/1.9781611974973.11. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611974973.11`.

[6] Zhiyuan Chen, Ee Na Teoh, Amril Nazir, Ettikan Kandasamy Karuppiah, Kim Sim Lam, et al. Machine learning techniques for anti-money laundering (aml) solutions in suspicious transaction detection: a review. *Knowledge and Information Systems*, 57(2):245–285, 2018.

[7] Cassady Craighil. *Greenpeace Finds Amazon Breaking Commitment to Power Cloud with 100% Renewable Energy*, 2019. `https://www.greenpeace.org/usa/news/greenpeace-finds-amazon-breaking-commitment-to-power-cloud-with-100-renewable-energy/` [Accessed: 10-6-2019].

[8] Ali el Hassouni. Fraud detection using machine learning methods. Master's thesis, Vrij Universiteit Amsterdam, De Boelelaan 1081, 1081 HV Amsterdam, 1 2017.

[9] Daniel Falbel. *TensorFlow for R: Predicting Fraud with Autoencoders and Keras*, 2018. `https://blogs.rstudio.com/tensorflow/posts/2018-01-24-keras-fraud-autoencoder/`, [Accessed: 5-5-2019].

[10] Tom Harting, Sven Popping, Mathieu Post, and Daniël Swaab. Transaction monitoring, 2017. URL `http://resolver.tudelft.nl/uuid:b3c04ba0-bcb5-4283-93db-949e7673c22f`.

[11] Ernst Maurits Henricus Hirsch Ballin. *Wet ter voorkoming van witwassen en financieren van terrorisme*, 2019. `https://wetten.overheid.nl/BWBR0024282/2019-01-01` [Accessed: 17-6-2019].

[12] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[13] Kadir Liano. Robust error measure for supervised neural network learning with outliers. *IEEE Transactions on Neural Networks*, 7(1):246–250, 1996.

[14] Pierre Lison. An introduction to machine learning, 2015.

[15] Openbaar Ministerie Functioneel Parket. *ING betaalt 775 miljoen vanwege ernstige nalatigheden bij voorkomen witwassen*, 2018. `https://www.om.nl/@103953/ing-betaalt-775/` [Accessed: 1-5-2019].

[16] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *Acm Sigkdd Explorations Newsletter*, 6(1):90–105, 2004.

[17] Marco Schreyer, Timur Sattarov, Damian Borth, Andreas Dengel, and Bernd Reimer. Detection of anomalies in large scale accounting data using deep autoencoder networks. *arXiv preprint arXiv:1709.05254*, 2017.

[18] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

[19] Jonathan Watson. *Anti-money laundering: banking scandals prompt rethink of EU regime*, 2018. `https://www.ibanet.org/Article/NewDetail.aspx?ArticleUid=814cd9ae-0b41-4e45-99bf-3c64a428ccfd` [Accessed: 1-5-2019].