Delft University of Technology
Bachelor Project - TI3806

bunq

Final Report

Attribution modelling

Sytze Andringa
s.p.e.andringa@student.tudelft.nl

Daan van der Werf
d.j.vanderwerf@student.tudelft.nl

Job Zoon
j.zoon@student.tudelft.nl

TU Delft coach: **Matthijs Spaan**
bunq client: **Wessel Van**
Project Coordinator: **Huijuan Wang**

July 4, 2018

Preface

This report is written by Sytze Andringa, Daan van der Werf and Job Zoon for the course Bachelor Project of the Bachelor Computer Science at Delft University of Technology. During ten weeks, we carried out a marketing attribution project at bunq. This project consists of implementing a sytem that prepares data for a machine learning model and can communicate with the model, in such a way that the bunq marketing department can use it. The problem, research, process, results and some other important parts of the project are described in this report.

We thank everybody at bunq who was involved in the project or helped us in any way. Especially Wessel Van for his supervision, Ali el Hassouni for working closely with us on the project, Iñez Eppinga for showing us around at bunq and making us feel welcome, and André de Roos, Esan Wit, Sander van den Oever and Kevin Hellemun for reviewing our code.

Finally, we thank Dr. M.T.J. Spaan for his supervision as a TU Delft professor and for all the advice, critical questions and thinking along in the meetings we had during the project.

Summary

One of the greatest challenges in marketing is measuring the return of investment of a marketing campaign and translating that into a strategy. Companies spend a lot of money on marketing without knowing how effective certain marketing campaigns are. To solve this problem for bunq, we will be using machine learning to create a marketing attribution system which outputs the optimal parameters for advertisements, based on data from all previous bunq advertisements. This tool can be used by the marketing department of bunq to increase its efficiency. The marketing attribution project consists of three parts: the machine learning model itself, the input data of the machine learning model and the system through which people can get output of the model.

The machine learning model is created by a data scientist at bunq. The model uses supervised learning, a method that uses a set of annotated training data as a supervisor for learning patterns. We specifically make use of deep learning models that use regression to find either the expected amount of clicks or the cost per acquisition of an advertisement. The results of these models are presented as a JSON file containing the best n advertisement options and their features.

The input data to train the machine learning model was created by us. One component of the input data are the so-called touchpoints from Adjust. Adjust is an advertisement tracking company, which helps bunq with gathering data about all online encounters people had with bunq, like clicks on bunq advertisements or visits to the bunq website. The Adjust data gives the machine learning model information about how often an advertisement has been seen or clicked on, but it does not give information about how efficient an advertisement was in terms of the gained users. To solve this, we wrote an algorithm that anonymously matches the Adjust data to user data in the bunq database, based on IP-address and timestamps. The more links an advertisement has with users, the more efficient it is since it has been part of a process that convinced many users to become a bunq user. With this input data the machine learning model can be trained.

The second part of the project is creating a connection to the machine learning model in such a way that the marketing department can use it. We created a python server that accepts calls from the bunq backend and sends the calls to the model, which is written in Java. It will then pass on the response of the model back to the bunq backend. In the python server, we use a bayesian technique to determine the best inputs for the marketing attribution machine learning model, to finally get the best possible parameters for a certain advertisement.

All code in the backend is written in PHP and .json in a very clear Model View Controller structure, with strict bunq coding guidelines. Testing is done with PHPUnit tests.

Contents

1 | Introduction

bunq is a company that delivers digital financial services and received its banking license in 2015. bunq focusses on being innovative and is much more agile in comparison to the traditional banks. Ali Niknam, the founder of bunq, describes it as an 'IT company with a banking license' [1].

For bunq's marketing department, it would be desirable to know how effective certain campaigns are in terms of gaining users. Most users have multiple touchpoints with the company before they decide to download the app and start using it, but we do not know which of the touchpoint really made the difference in convincing a person to become a user. If we would know how profitable an advertisement is, it would result in a more efficient marketing strategy.

During this project we worked at the bunq office to create a system which outputs the optimal parameters for advertisements. It uses a machine learning algorithm, which processes data about advertisements of the past. The marketing department can use this system to make decisions about advertising.

In this report, you can read all about the different parts of the process of creating this system. In chapter 2, the results of our research from the first two weeks are presented. The third chapter contains the ethical implications our project could have and in the fourth section the methodology of our development is explained. Then, in chapter 5, the general structure of the bunq backend is described, to give an insight in the way we needed to code. Chapter 6, 7 and 8 contain the different programming parts that together make up the end product. In chapter 9 the results can be found and chapter 10 contains the conclusions. Finally, in chapter 11 you can find some recommendations for the future. Discussions about the different parts of the project can be found throughout the report, as sections of the chapters about those parts.

2 | Research phase

2.1 Introduction

In this chapter, the research report of the bachelor project is presented. It contains the problem definition and our solution to that problem with background information. Further information about implementations of the different components of the project can be found in other chapters, this chapter only contains the information we discovered during the first two weeks of the project.

2.2 Problem definition and analysis

Measuring the revenue of a marketing campaign is an established problem within the marketing business. We are able to make an approximation of the effectiveness of an advertisement, but it is very hard to come up with an accurate metric. Even if we would know the amount of visitors, which is sometimes possible, it is hard to determine how much we have gained out of this.

The other side of the equation however, the activity metrics, are easy to determine [2]. These metrics include which marketing campaigns have been set up, which is data such as "X articles are published" or "an advertisement was published on Y websites". These metrics only say something about the cost side of the equation and nothing about the effectiveness.

To get the most out of a marketing budget, we need to develop tools to evaluate the effectiveness of marketing campaigns. To quote John Wanamaker (1838 -1922): "Half the money I spend on advertising is wasted; the trouble is I don't know which half." [2] What makes this problem interesting is that users can have multiple kinds of *touchpoints*, moments in which the customer comes in contact with the company, before a purchase is being made. The root of the problem is that it is difficult to determine which touchpoints are most effective. Knowing what the most profitable touchpoints are is valuable information, since it shows the quality (how much profit can be made) and quantity of the users being attracted to each marketing channel.

2.2.1 Current state

Currently, bunq's marketing department makes advertisements and uses data and experiments to find out what the best parameters are for those advertisements. There are multiple kinds of parameters: the way of distributing the advertisement (through for instance facebook or instagram), but also which kind of customers are targeted (by for instance gender or age) and how the advertisement looks. They base the decisions of the values of the parameters partly on their own instinct and experience, with help of the available statistics. These statistics are not intelligent, and therefore do not recommend any marketing parameters or strategy. If they are provided with easy to read data that shows which parameters work best for which ads, this could greatly improve the efficiency of future marketing campaigns.

2.2.2 Current systems

We are going to work with the following systems during this project:

- **bunq backend** The code behind bunqs systems is in the backend. The backend is written in PHP.
- **Machine learning model** Ali El Hassouni, a data scientist from bunq, will build a machine learning model which computes optimal parameters for advertisements. It is in the form of a *POJO* (Plain Old Java Object).
- **Attribution page** The marketing department uses this page to see stats about user behavior.
- **Adjust data** Every hour *Adjust*, a mobile app measurement company, dumps a .csv file in an AWS bucket with data about bunq-related actions on the internet. This data can be used to analyse a large number of touchpoints associated with bunq.
- **bunq database** This database contains all data of the bunq users. We have limited access to this for security and privacy reasons, but when our code is merged with production it can use the database.

2.2.3 Solution

bunq is a data driven company, so it would be a great improvement for marketing if data would be available on what the best parameters are for certain advertisements. Our goal is to integrate the machine learning model within the bunq backend to give marketing the information they need.

The first thing we need to do is to combine the Adjust data and the user database, to link touchpoints with users. By combining data from Adjust and the bunq database, we will obtain the input of the machine learning model.

When we have the model and input, we need to start writing code around the model to integrate it in the bunq backend. To do this, we have to make sure all the input is in place, the model can be called upon and the output can be shown to the marketing department. After ten weeks, when the whole system is in place, marketing should be able to use it.

2.3 Structure and connection to bunq backend

bunq holds a lot of sensitive data. This makes security a top priority at bunq. Also, bunq has written a lot of code over the last couple of years, and its core repository is huge. To maintain this, bunq needs to hold some very strict custom guidelines regarding programming. This is all documented on a private wiki page only accessible to bunq employees. We were advised to let other bunq emplyees review pieces of our code as quickly as possible, to avoid the need to rewrite a lot of code in the end. In week 6 we need to upload some code to SIG for a code review. Since we are obliged to follow the strict programming standards of bunq, we do not expect SIG to reject our code because of inadequate code quality.

We expect the most computationally intensive task to be the training process of the machine learning model. The more frequently we update it, the more accurate the model will be. Convertro, a marketing attribution platform, reported that an attribution system should never be more than one day behind [3]. Adjust delivers new marketing data every hour [4], which leaves us to choose an update interval between one hour and one day. In the end we decided to not update the machine learning model automatically, because the data scientist at bunq prefers to train the model manually at first, to be able to validate it thoroughly.

The structure of the whole project as we expect it to turn out can be found in figure 2.1.

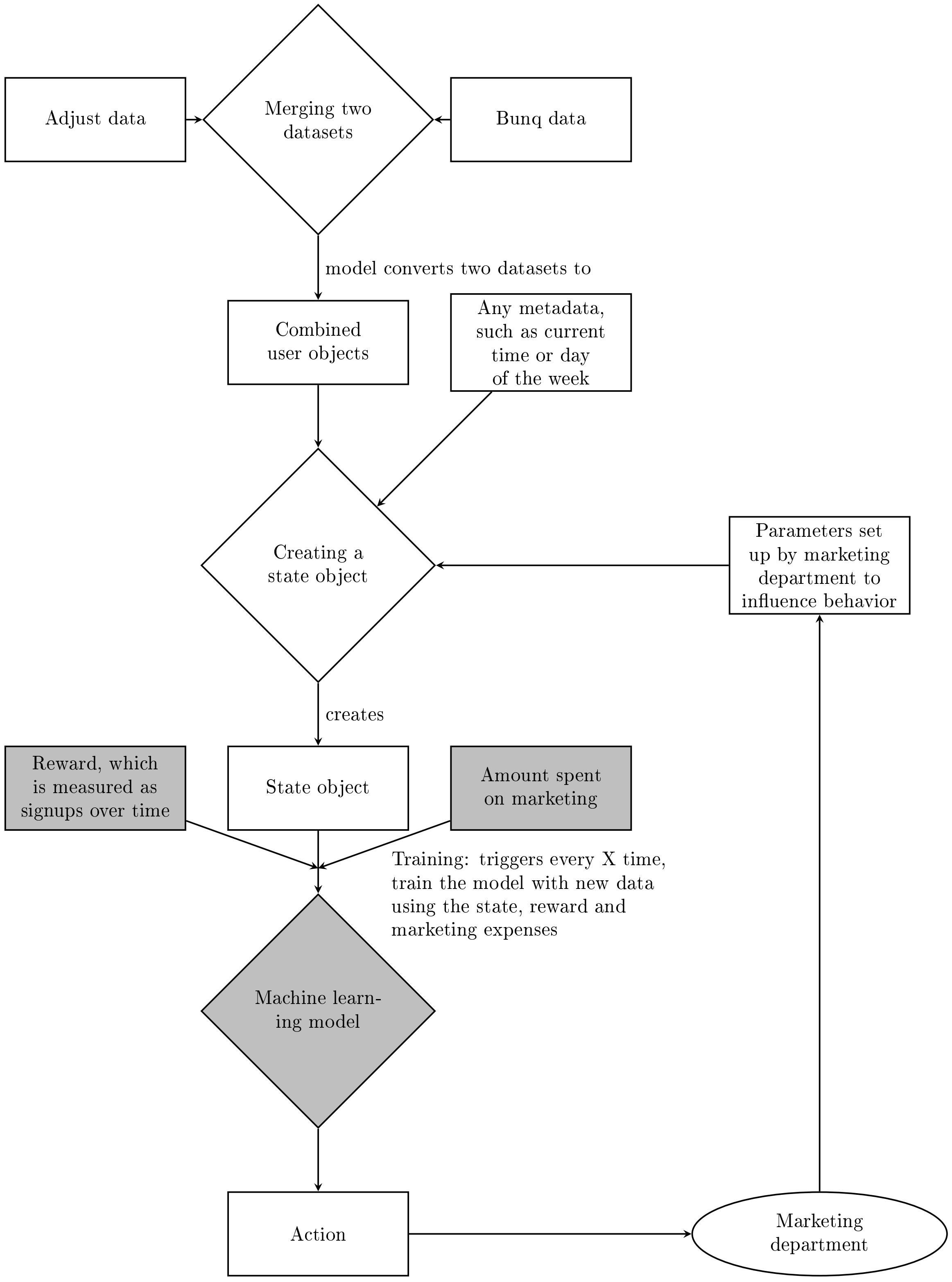
Adjust data
Merging two datasets
Bunq data
model converts two datasets to
Combined user objects
Any metadata, such as current time or day of the week
Creating a state object
Parameters set up by marketing department to influence behavior
creates
Reward, which is measured as signups over time
State object
Amount spent on marketing
Training: triggers every X time, train the model with new data using the state, reward and marketing expenses
Machine learning model
Action
Marketing department

Figure 2.1: A schema we have created to show the data flow. A rectangle, diamond or ellipse respectively represent an object, proces or entity. The creator of the machine learning model will work on anything filled up with the color gray while we will focus on the rest.

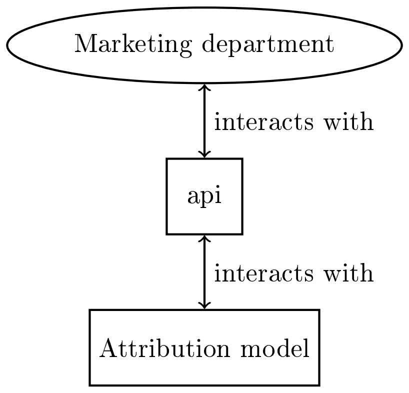

Figure 2.2: This diagram shows how the marketing department interacts with the model through an api.

2.3.1 Where to place the project within the back-end

In concept the model that we will be implementing behaves semi-independently of the rest of the back-end. No other component in the backend will directly depend on our project. The project, however, will depend on the bunq backend. Since the attribution model will use components from the backend, it is placed inside bunq's core repository.

We want the marketing department to be able to do at least the following things:

- Receive an output, which is an action output from the ML model. This output can be the basis of a marketing strategy.
- Adjust the model's training behavior, or reset it to default settings. For example, we might want it to only train over the last X months, or only find the best advertisement for a specific age group.

Marketing should be able to interact with the model through an API. A good option would be to add a panel to the marketing's statistics page. Through this panel the marketing department can interact with the attribution system. Creating a panel is not in the scope of our project, so we will ask a front-end developer to do this after we finish our project. Figure 2.2 illustrates the way the marketing department should be able to interact with the model.

2.4 Data collection

2.4.1 Data from Adjust

Adjust is a mobile measurement company that provides companies with marketing attribution insights [4]. They provide an overview of the interactions between a person and bunq. Their attribution system matches, for every user, its app installs and advertisements visited. This means that if a user installs the app because of an interaction with an advertisement on Facebook, the source (Facebook) will be attributed to the install. Their attribution also goes on after the initial install, allowing for understanding in-app behavior.

Adjust provides this data on an hourly basis. Their clients can access their raw data via so called *callbacks*, which provide in-depth information about a user event. The data that is collected by Adjust is delivered straight to Amazon S3 buckets. These buckets can then be accessed from the bunq back-end and combined using the data that we find in the bunq database.

2.4.2 Combining data from both sources

To ensure that the machine learning model is able to learn correctly and efficiently, we need information from both Adjust and bunq. Adjust provides us with data on how a certain person reaches the bunq website and installs the app. This data constists of many parameters that describe the journey of the user to the bunq website [5]. Next, we require some high-level characteristics of the person in order to train the machine learning model, which is stored by bunq.

Both datasets can be linked to get a wide variety of information that correspond to the same person. This linking can be done by creating an interval around a user's time of signing up to bunq from the bunq database. We use this interval to find the timestamp of an install from the Adjust dataset. On top of this, we can use features from the Adjust data like IP-address, geolocation or device type. This set of features is not confined to the ones mentioned above and may be expanded during implementation. Both datasets have unique identifyers to define a user. This means that if we have found a match from both datasets, we can use these identifyers to track this user in the future. This is particularly useful to construct a user journey leading to a sign up from the Adjust dataset.

2.4.3 Privacy and anonymity

Both datasets that we are working with contain data about users and thus privacy plays an important role, especially in the bunq user database, which contains sensitive data about bunq's users. Adjust adopts a pro-user approach to privacy to ensure that they can be trusted with private data [6]. What we will essentially be doing is deanonymizing the Adjust dataset by linking it to the bunq database. By doing this, certain personal data will be accumulated. Although this data can be used to link a certain user back to any sensitive data contained in the bunq database, the data in itself is not sensitive. It will contain cases like gender, age and location of signing up. This is the type of data that the model needs to train and potentially sensitive data like SSN, address or even a name are not needed. It comes down to the fact that the data will exclusively be used for marketing research purposes. The result of this research will be used to determine which ad campaign will give the highest reward. If the data would be used for instance for targeted personal advertisements based on geoloation or IP-address this would be in conflict with the *EU General Data Protection Regulation* (GDPR) [7]. More information about the ethical implications of this project can be found in chapter 3.

2.4.4 Touchpoint weight assignment

There are multiple ways to evaluate the quality of touchpoints [2]. We consider two important touchpoints.

- The *first touch* is the touchpoint that drives the first visit. For bunq, this would be the touchpoint that drives a visitor to visit a bunq related web page.
- The *lead Conversion* is the touchpoint which drives the visitor to become a lead (i.e. give you their contact information). For bunq this can be defined as the point at which the user decided to sign up.

In a single touch model, we assign 100% of the revenue credit to a single touchpoint. However, for the model we are creating we are going to look at a multi touch model. This means that we assign different weights to touchpoints to obtain an optimal result. If we put heigher weights on the first touches, which is a *top-of-the-funnel* (TOFU) market strategy, it means the marketing strategy is primarily focused on getting more people to know about bunq. Using lead conversion as the main metric is called *bottom-of-the-funnel* (BOFU) strategy and is more focused on making more people sign up at bunq.

Touchpoint weight in our model

We can use touchpoint weight assignment to change the behavior of the model. This can be done by adjusting the weights of touchpoints in the combined user objects. This is done before the user objects are being sent to the machine learning model. We define a parameter p which the marketing department is able to adjust. A low p assigns higher weights on early touchpoints (TOFU strategy) and a high p vice-versa (BOFU strategy). By default we want to measure the point where the user became a lead, so signed up. However, if the marketing department is interested in a strategy that is based on getting as much website visits as possible, we can set p to 0 and obtain a marketing strategy for that scenario.

Changing p means we should also change how historical rewards are measured. This reward is, by default, measured by the number of signups. An alternative way to measure the reward would be the number of website visits or premium accounts gained. We want our input and output to correlate as much as possible. Therefore, when p is low, the reward will be based more on the amount of website visits, whereas a high p will result in a reward being more based on signups. By default, the reward should be evaluated as the number of signups.

This p value is optional, and can be considered as a could have. If it will not be part of the project, a BOFU strategy will be applied.

2.5 Machine learning model

The machine learning model is the core of the project. We will not build the model by ourselves, but one of our supervisors from bunq, Ali el Hassouni, will build it over the next couple of weeks. Together with him, we have set up some guidelines regarding the in- and output of the machine learning model to make sure our architectures match. Our task will be to integrate this into the bunq architecture. We will provide an efficient and correct data flow to the model in such a way that the marketing department is able to use it. The purpose of this section is to discuss the inputs and outputs, the chosen approach and relevant considerations that were taken into account regarding the machine learning model.

2.5.1 Reinforcement learning

Reinforcement learning methods turn out to be widely used for these kinds of problems. This category of machine learning methods develop a model/policy/agent that tells us what to do in a given situation so that a cumulative future reward is maximised. This is different from the standard modelling approach: *supervised learning*. Supervised learning maps inputs to a label/target. For our case that would be mapping marketing parameters to amount spent and number of signups. There are some problems however if we would use supervised learning. Marketing is a sequential decision problem with a temporal dimension among many other dimensions/factors/parameters. Also, user actions have delayed effect into the future which would be completely missed using a standard modelling approach. Because of these limitations, we think reinforcement learning is a more suitable approach than supervised learning.

2.5.2 How it works

For every hour, the total amount of money spent on online marketing and the number of signups are evaluated. The input is a state of the current environment, which contains marketing profiles of users and any other correlated data. The output is the corresponding expected cumulative reward for all possible marketing actions. These actions are defined beforehand and can be interpreted as marketing activities, which is a concept explained in chapter 2.2. The marketing department will use these values to help them plan marketing campaigns.

LSPI algorithm for reinforcement learning

The model that we will be using is based on a reinforcement learning algorithm called the *Least Squares Policy Iteration algorithm* (LSPI). LSPI is used to develop a *policy* that maximises the cumulative future reward, given a batch of historical data. This historical data is presented as hourly $(state, action, reward)$

samples from the last four months. A policy in the context of reinforcement learning is a function that dictates what action to take given a particular state [8].

LSPI is very suitable for prediction problems where we are interested in learning a good control policy for a task. It uses a related algorithm called *LSTDQ*, which learns the approximate state-action value function of a fixed policy, thus permitting action selection and policy improvement without a model [8]. LSTDQ is very data efficient, which LSPI combines with the policy-search efficiency of policy iteration. Because LSPI is a completely off-policy algorithm, it can (re)use the data that is continuously collected by Adjust in each iteration to evaluate the generated policies. Because of this, the LSPI algorithm enjoys stability and soundness of approximate policy iteration and has no need for learning parameters, such as a learning rate, that often need careful tuning. Figure 2.3 illustrates the flow of the LSPI algorithm.

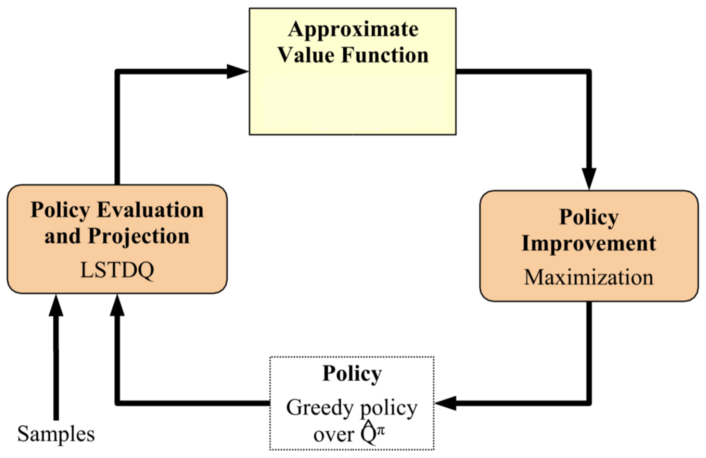
Approximate Value Function
Policy Evaluation and Projection
LSTDQ
Policy Improvement
Maximization
Policy
Greedy policy over $\hat{Q}^\pi$
Samples

Figure 2.3: A diagram that shows the flow of the LSPI algorithm. $\hat{Q}^\pi$ is the state-action value function computed by the policy evaluation of the current policy π [8].

2.5.3 First version

A first very basic version for the model was already developed and tested. It is based on a simple setting. In this setting, the state is purely based on 'day of week' and 'hour of day'. The action is the amount we need to spend during each tuple $(day, hour)$. The reward is the total number of signups, divided by the amount spent during a certain hour during a certain day of the week. The state space and the action space will be expanded in further versions of the model. Expensions of the action space consist of possible marketing investments and targeting parameters for our campaign. Any expansions of the state space consist of features that provide information about the current marketing state, such as any data from adjust or running marketing campaigns.

2.6 Code development

In this section, the choice of programming language and the way of testing will be presented.

2.6.1 Programming language

During the first meeting we had at bunq, the choice of programming language already came up. Bunq recommended us to use Python or PHP. Because of our experience with Java we added that language to the possible options we had to choose between. The three options all have some advantages.

Some of us already have experience with *Python*, and the language is fairly easy to learn for the others. Also, Python is the most popular machine learning programming language: 57% of data scientists and machine learning developers use it and 33% of them prioritise it for development [9]. Although we are not going to develop the machine learning model ourselves, using a language that is widely used for machine learning might still be benificial.

The most used programming language in the bachelor Computer Science is *Java*. This means that all teammembers have a lot of experience with it. Java is also one of the most used languages for machine learning [9]. Moreover, the machine learning model we are going to use will be in the form of a POJO (Plain Old Java Object). When we use Java, it will be easier to deal with the model itself, because it is written in the same language.

The last option is *PHP*. None of us had any experience with PHP and it is not commonly used for machine learning. However, the entire bunq back-end is written in it, so if we write our project in PHP it can be joined seamlessly with the other bunq code. Also, in PHP there already are ways in place to communicate with the bunq databases, so we do not have to make those connections ourselves.

At first we decided to use PHP, to make sure the model could be integrated in the back-end. But in the end, it turned out we needed to program in all three above mentioned programming languages. The connection to the back-end and the merging of data is written in PHP, the machine learning server is written in python and part of the connection to the Java machine learning model is written in Java.

2.6.2 Testing

Testing is a crucial part of software development, because you never want to commit code that might not work properly. We will make use of two types of tests: first we have all code tests, such as unit integration tests and end-to-end tests, and secondly we use manual tests.

We will use *PHPUnit*, a programmer-oriented testing framework for PHP [10], to test the code. This framework is used to test all code at bunq, so we have to use it too. For manual testing, we will ask the marketing department to try our program to see if it works and whether it contains all functionality needed. Also, we will do some manual testing ourselves when working on the project, to watch if everything works as intended.

3 Ethical implications

3.1 Our project and the GDPR

Because our project deals directly with user data, we had to keep in mind that everything we did had to be privacy compliant and up to standards with the EU General Data Protection Regulation (GDPR) that became active during our project. The GDPR is designed to protect and empower the data privacy of EU citizens by reshaping the approach to data privacy of European organizations [11].

Because we are developing a new and not yet widely implemented marketing strategy based on user data, these new regulations might have an impact on what we can and cannot do in our work. This section will give a brief overview of how we dealt with these issues.

3.2 GDPR for data matching

At the core of our project lies the data matching part. Anonymous data collected by the company Adjust is combined with data from the bunq database, which essentially deanonymizes the data. The data from Adjust does not contain inherently sensitive data about individuals, but rather about the events in which a person makes contact to something related to bunq. This means that having this new combined data does not pose any new privacy threats compared to the data that was already kept by bunq.

Something that does potentially conflict with the GDPR is the way in which the data is used. Because we are using the data to improve bunq's marketing strategies, we could come into contact with article 21.2 of the GDPR, which states:

> "Where personal data are processed for direct marketing purposes, the data subject shall have the right to object at any time to processing of personal data concerning him or her for such marketing, which includes profiling to the extent that it is related to such direct marketing." [7]

With the data that we have collected by matching bunq users to Adjust touchpoints we could potentially target users directly with marketing. The keyword here is 'direct'. The data that we gather from matching is used for research purpose only and will not be used for any activities directly targeting the individual. This is because all the collected data will be fed to a machine learning model that only learns patterns in the data and does not retain any information about an individual. The results of this model are used for improving marketing campaigns on a global scale and not on an individual scale.

3.3 GDPR for machine learning

Not only targeting individuals may conflict with the GDPR, there are also machine and deep learning consequences following the new regulations. One of the major effects of the GDPR on machine learning is the Right to Explaination, of which article 13 and 15 are most relevant to our project. These articles describe how the data controller must provide the data subject with any stored information about him and how this information is used by the data controller. Since the collected data is only going to be used for marketing research, these articles will not have a big impact on the project. The most important thing to take from all of this is that the machine learning side of bunq should be able to not only try to embed their algorithms with privacy protecting measures, but also make sure that the data inventory, classification and maintenance is well documented [12].

4 | Development methodology

4.1 Location and communication

We worked five days a week from 9 to 6 at the bunq office in Amsterdam, where we had our own office. Whenever we were in need of something from another bunq employee, for example if we had a code related question, we could ask them through Rocket.Chat, bunq's internal communication program. The communication within the company was very efficient because of the clear communication guidelines within the company, as well as the fact that we were close to everybody we needed help from.

We had meetings scheduled regularly throughout the week. On Wednesdays we had a meeting with the owner of our project at bunq, where we discussed the progress made and any updates about the project. Every week at Thursday morning, we had a meeting at the TU Delft with our TU Delft supervisor. By seeing him face to face every week, we assured that he was always up to date on what we were doing and could provide us with help if we needed it.

4.2 Agile

We worked using an agile-like approach. Agile is defined as "a lightweight software engineering framework that promotes iterative development throughout the life-cycle of the project, close collaboration between the development team and business side, constant communication, and tightly-knit teams." [13]

During the research phase, we split the project up in parts to get an overview of what has to be done. Every week we discussed what we would like to achieve that week. Weeks 3, 4 and 5 were mostly based on creating something that works. After that, we focused more on code quality, additional features and a stable endproduct.

Since we worked together in the same room at the same time, we were flexible in terms of assigning tasks. Whenever new priorities popped up or tasks took either longer or shorter than expected, we could directly discuss this together and come up with a plan to adjust our schedule.

Sometimes whenever a teammember finished something, there was something he had to wait for before it was possible to work for him on a next planned task. The most common causes for this were code reviews or missing components. Among teammembers we made an agreement that whenever this was the case, the teammember would work on the final report. That way we already had a lot of texts written, saving us a lot of work near the end of the project.

4.3 GitLab

GitLab is a Git-repository manager which has planning tools and issue tracking features. bunq uses GitLab for all development and other issues that need to be handled. We opened an own issue for our project which we always referenced when committing code.

Every time when we extended our own branch with more code, we assigned a merge request of that code to one of the junior developers at bunq. They then gave us comments on how to improve our code to match it with bunq's strict coding style. Some of the comments were just about coding style, others about the general structure. These reviews were very beneficial regarding improving our code.

When we finally had code ready that could be added to the backend, we also assigned the merge request to the senior developers. They also added some notes about coding style and structure, which improved our code even more.

4.4 Coding language and IDE

We can split up the project in two parts that are implemented in a very different way. The first part consists of anything related to preparing any data in such a way that it is possible to train the model. This training data consists of merged users and other touchpoints. The second part consists of anything related to communicating with the model. Since training the model is not an automatic process (yet), and the trained model is delivered manually, the two parts do not need to communicate with each other. They differ quite a bit in terms of architecture, programming language and functionality.

The first part is implemented in PHP. As stated in the research chapter, we used PHP because this way it fit right in the bunq core backend. PHPStorm was used as the IDE, because this is the IDE used at bunq and a couple of tools are written to specifically work with PHPStorm. We used a codesniffer with PHPStorm to have our code automatically checked according to the bunq coding guidelines.

The second part contains both Java and Python code for several reasons. The model could only be connected by calling it with Java code. Also, a machine learning model server already existed in python, as described more extensively in chapter 8. The PEP-8 coding style [14] was used for Python and the Google Java Code Style for Java [15]. Having the code in perfect shape is more important for the core backend than for the python server. We noticed this in terms of how strict bunq asked us to code both.

5 | Code Architecture

This chapter will describe how the bunq backend works and what the important components are that we need to deal with. First, the use of composer, a tool that generated .json files, is described. After that the different parts of the Model View Controller (MVC) structure are explained. Daemons and Libraries are treated next and then we discuss how these parts were used to solve our problem. Finally, a discussion is added about the architecture.

5.1 Use of composer

A lot of programming is done in .json files. In those files, definitions are given for Model, View and Controller classes. The specific definitions per class type will be explained in the following section.

After creating such a .json definition file, a package manager called composer is used. Composer is called through a terminal and will generate code and classes based on the definitions given in the .json files. The generated classes make sure there is a general structure in all of the code and they support and call on the PHP classes that are written by the developers.

5.2 MVC-structure parts

A MVC systems consists of three different parts: a Model, a View and a Controller. In this section the use of the three different parts will be explained with their respective use in the bunq backend. A simple overview of the MVC-structure can be found in figure 5.1.

5.2.1 Model

Models are used as representations for database rows, and are a way of reading and writing from and to different database tables. For every table in the database, a *Tablename*Model exists. Those models have get and set functions to retrieve and change values of specific rows in the database, but also static functions like 'createQuery' which are used to perform queries on the entire database table. This can be used to find specific rows in the database based on the parameters that are inserted in the query.

Almost all of programming in Models is done through the .json definition files. These files contain the table name, description, the keys of each element in the table (columns), the relations with other tables and some table settings. After defining the .json files and running composer, a lot of different classes are created. For example, SQL queries that create the table, classes that contain get and set functions for elements of the table and query/search objects to create queries on both SQL and ElasticSearch. ElasticSearch is an efficient JSON-based search and analytics engine [16], that the bunq data scientists use.

5.2.2 View

The View classes consist of two parts: pre-processors and post-processors. When an API-call is sent to the backend, the pre-processor creates objects for the controller based on the parameters given in the call. After the objects are created, the view calls the right controller to start a workflow. The output of the controller is then handled by the post-processor to create an output that can be sent back as response to the API-call.

The .json definition file contains the Models that are used, the inputs, the outputs and the name of the pre- and post-processor. The pre- and post-processor files contain methods to parse the inputs and create outputs from a Model, as well as some other functionality that needs to be done before and after the worflow needs to be called.

5.2.3 Controller

The controller consists of workflow classes. Those classes contain the actual functionality needed to transform the inputs sent by the pre-processors to the outputs which the post-processors can send back. Some of the workflows are called directly from view classes, others (called sub-workflows), are called from other workflows. A workflow is divided in different states, which are functions that are part of a workflow.

The .json files are the definition of the workflows. They contain the input, output and flow through the states of the workflow. This means that states do not call eachother within themselves, but the definition takes care of the flow from one state to another. There is also a possibility to add logical statements to the transition between two states to decide to which state the workflow should go next.

5.3 Other parts

Except for the MVC, there are two different kinds of code structures in the backend that we need to use. They are explained in this section.

5.3.1 Libraries

Libraries contain code that does not fit in one of the three MVC catagories. This is mostly functionality that is not really part of a workflow, or objects that are used to support functionality. Library classes can be called both in Controllers and Views. The structure of libraries within the MVC-structure can be seen in figure 5.1.

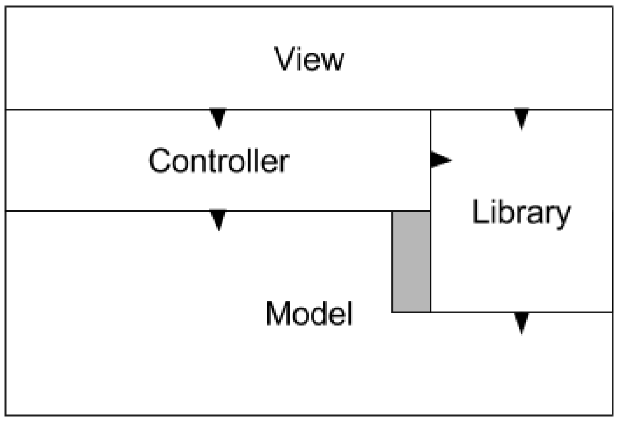

Figure 5.1: A diagram that shows the MVC structure with libraries. The dependencies are shown with the little arrows.

5.3.2 Daemons

Daemons are background processes that can assist the regular code. bunq has multiple Daemons running at all times. Daemons are in a way a replacement for view classes: instead of making API-calls to the backend to get the code running, Daemons will run themselves without interaction from the outside. There is no real functionality in the Daemon classes, all the functionality can be found in the workflow class that a Daemon calls.

5.4 Usage of all different parts

The project is clearly split up into two different parts: the first part is about collecting and merging data and the second part is about connecting the machine learning model to the bunq backend. The architecture of both is discussed here, but the architecture of the python server itself is discussed in chapter 8, because the python server is not a part of the bunq back-end.

5.4.1 Collecting and merging data

This part consists of two subparts: collecting data and merging data. Their architecture is really similar, but the code is mostly seperated, except for the MarketingAttributionAdjustTouchpointModel, which they both use. The structure is first explained for each part, then in figure 5.2 an overview is given.

Model

We created two Models: the MarketingAttributionAdjustBucketModel and the MarketingAttributionAdjust-TouchpointModel. The former is used to represent the CSV-files which contain all the Adjust data. Those are saved to know which files are inserted into the database and which are not. The latter model is used to represent the touchpoints from Adjust. Because every Adjust touchpoint has around 140 columns, we decided to only add the useful columns for matching data to the Model to keep the Model clear. All the other colums of a touchpoint are added to just one column in the AdjustTouchpointModel, in .json format. The collecting data part of the code creates instances of the Models and writes those to the database. It also adds two empty columns: a User column and a RequestMetadata column. The merge data part of the code then takes all MarketingAttributionAdjustTouchpoint entries and tries to match them with Users using the RequestMetadata table. If a match is found, both colums are filled with the data of the match.

Controller

The Controller consists of workflows that contain most functionality that is used to create and merge the data. This is, for example, reading the CSV-files, saving the touchpoints to the database, and finding users by the touchpoints. Some of the more specific functionality is moved to library classes, to keep the workflow classes clean with a clear flow. We have two upper workflow classes: one for the collecting of data and one for the merging of data. Those classes all have some subworkflows, which are called once or looped over.

Choice between View and Daemon

The two upper workflow classes need to be called from somewhere to let the code run. We had two choices to do this: with a View class or with Daemons. We decided to go for Daemons for two reasons. First, there is no need to have an input on the process, which views are made for. Daemons do not require any input from 'outside' the back-end. This makes them more fit for the job. Secondly, Daemons have a built in database looping functionality which makes them very efficient when you want to loop through a database. This is exactly what we need to merge the MarketingAttributionAdjustTouchpointModel database with users, because the Daemons loop through that database and try to find a match for every row.

5.4.2 Connection to python server from backend

The connection from the bunq backend to the python server is quite simple in terms of functionality, but the architecture is more complex. The following sections explain the different parts and in figure 5.3 an overview is given.

View

Views can communicate with the 'outside world', which means that someone who wants to use the code can send an API-call to the view. In our case, the API-call needs to be redirected to the python server. This means that the view only saves the body of the API-call, but does not process it in any other way. The same applies to sending back the response. When the view receives the response, it only adds the body to its own response for the API-call and then immediately sends it back. We created two views, one which handles info calls and one which handles predict calls.

Controller

For both views a workflow class exists. They are really simple, they only call on a library object which connects to the python server and then send back the response from that library class.

Library

We wrote a library class which actually makes the call to the python server using curl, a tool that can transfer data through URL's [17]. When a curl response is received, the class sends back the response to the workflow classes. The library class both accepts info and predict calls and will handle them in two different ways.

5.5 Discussion

The bunq back-end architecture provided us with a very strict coding style. In the beginning, this took a lot of time to learn which could have been spent on writing new code, but in the end it ensured that the quality of our code is way higher than it would have been without this architecture. Especially the MVC-structure resulted in high quality code. The other bunq back-end part that was really useful were the Daemons. In our initial code, a for-loop was used which took the itemId of every entry in the database and looped through them in that way. The Daemon has its own way of looping through a database which is much more efficient. All in all, the end product is much cleaner and better working than it would have been without the strict guidelines of the bunq back-end.

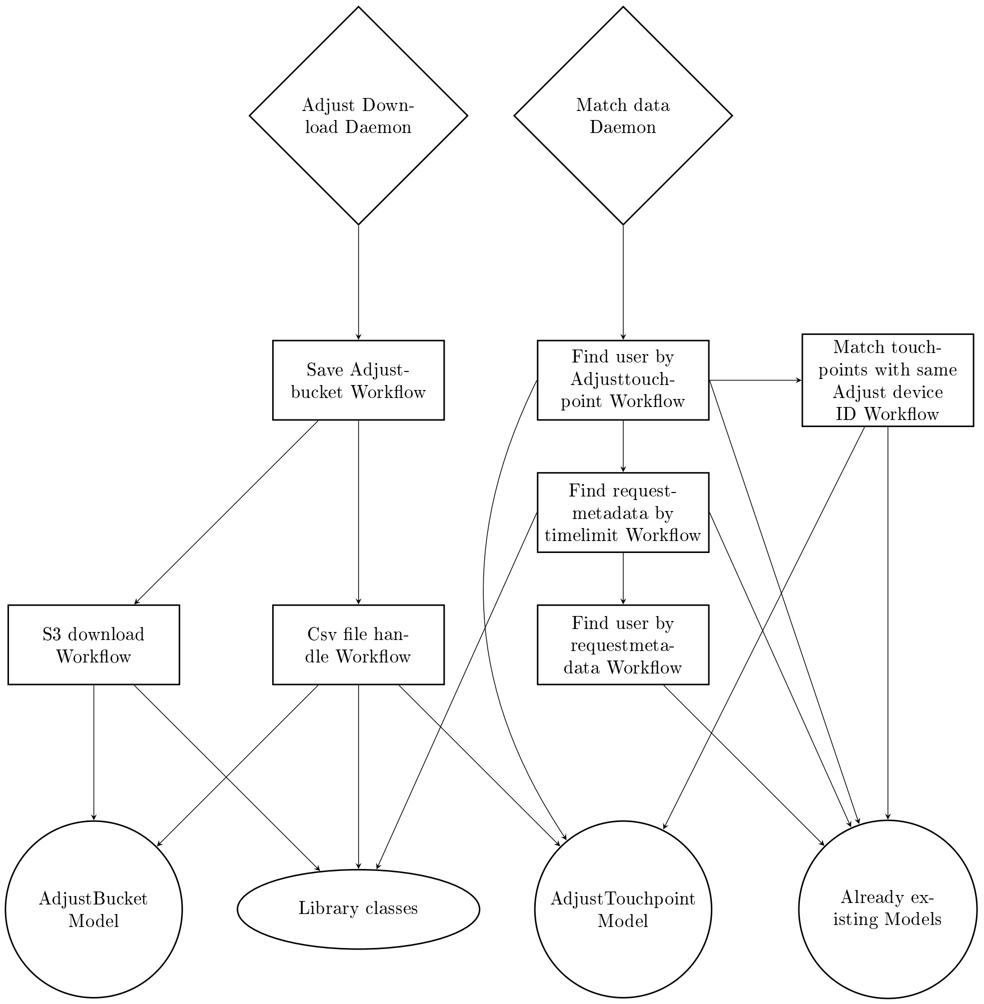
Adjust Download Daemon
Match data Daemon
Save Adjustbucket Workflow
Find user by Adjusttouchpoint Workflow
Match touchpoints with same Adjust device ID Workflow
Find requestmetadata by timelimit Workflow
S3 download Workflow
Csv file handle Workflow
Find user by requestmetadata Workflow
AdjustBucket Model
Library classes
AdjustTouchpoint Model
Already existing Models

Figure 5.2: A schema that shows all different types of classes. Diamonds are Daemons, rectangles are controller workflow classes, circles are models and ellipses are libraries. An edge from A to B means that class A calls to class B.

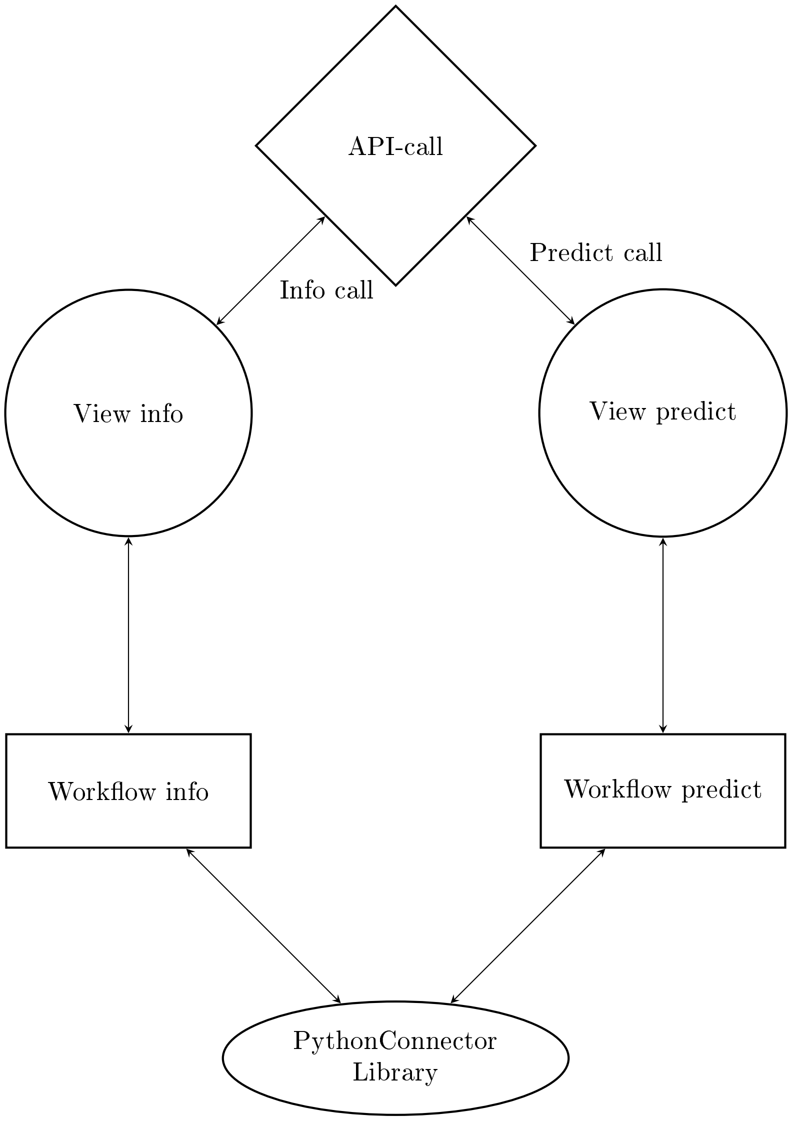

Figure 5.3: A schema that shows the structure of the backend connection the the python server. Diamonds are API-calls, circles are view classes, squares are controller classes and ellipses are library classes.

6 | Preparing data for the model

For the duration of the project we have been working with two major data sources. The first one being the touchpoints collected by the company Adjust and the second the bunq database. The way in which we collect data from both sources proved to be two very different processes. In this section we will discuss how we efficiently collect data from both of these sources and elaborate on our choices towards creating an algorithm that combines both sets of data.

6.1 Adjust Data

The first and main data source that we had to work with is the collection of touchpoints from the company Adjust. These touchpoints aggregated by Adjust are dumped on a Amazon web service called S3 (simple storage service). These dumps were in the form of csv files containing between 500 and 5000 rows on average which have about 100 attributes. The amount of rows per csv file are dependent of the hour of the day they correspond to. There is a clear pattern where the files contain very little touchpoints during the night and more during the day. There were also very large outliers on special occasions. An example of this is when there was a bunq update, which led to a huge number of touchpoints per csv file in the hours following the release.

The rows describe various characteristics that make up a touchpoint. Examples are the Adjust Device Id, the creation time of the touchpoint, the geolocation of the touchpoint and the ip-address. Some of the attributes in the table did not contain values, as they were not activated (added to the list of attributes monitored by Adjust) by bunq yet. To be able to use the data from the Adjust we first needed to download the objects from the S3 buckets, unzip them, read the csv files and write them to the database.

6.1.1 S3 Bucket

The hourly data dumps by Adjust are stored in what Amazon calls a *bucket*. This bucket contains every csv file as a zipped file, which are called *objects*. To be able to efficiently download these objects from the bucket there were some steps that needed to be taken. The first step was how to download the bucket object. Here we had three options: to download the entire object into memory and do operations on it, stream the entire object and do operations on chunks or download the entire object to a cache and use PHP file handlers to perform operations on it. The first option proved to be unsuitable as storing an entire object in memory while performing operations on it was too much for the PHP working memory. On top of that, this solution does not scale well at all, which means that as bunq grows the code will become increasingly slow and will have memory issues. The second solution scales a lot better than the first, but also came with a problem. This approach would make use of the streamWrapper class of the S3 PHP sdk [18]. While this approach seemed like a good solution to the scalability problem, streaming an object that was still zipped meant that the streamed chunks could not be used independently. This meant that we still had to stream the entire object before being able to unzip and use it. This is why we chose to use the S3 sdk to download the entire object into a cache and perform operations on it from there.

6.1.2 Downloading from S3

The first step in downloading an object is to determine which bucket object needs to be read next. Every object in the bucket has an unique key which could be used in conjunction with a get function from the S3 sdk to retrieve a specific object. To persistently keep track of which objects we have read we store metadata on every object that has been processed in the bunq database. When a download is complete the object is stored in the database. When downloading a new object, the database is queried in order to find the key of the latest bucket object that was read. Using this key the next object from the bucket is collected and downloaded. The object is stored as a zipped csv file in the default temporary folder that PHP uses. We chose to not remove the object from the S3 bucket after downloading, as storage at the AWS side is not an issue and the objects are still a valuable source of information that might be used by another source later.

6.1.3 Unzipping and storing

When the zipped csv file is succesfully downloaded it can be unzipped in the next step. PHP has a native function that can unzip files with the *.gz* extension called gzopen(). If all goes well, an unzipped version of the file is created, after which the original file is deleted. This unzipped file can now be used to extract data from and when this is done the unzipped file is removed as well. This process is described in the next section.

6.1.4 Storing the data

Because csv files contain a lot of different fields and Adjust dumps a csv file with an avarage of 3000 rows to the amazon bucket every hour we needed to think of an efficient way to store all these rows in the database. This comes down to about 10 million rows in 4 months. Our first choice was to read every csv file and directly translate every row to an associative array. Using the header of the csv file we combined every row with this header to split up a csv file with X rows into X associative arrays. These arrays could then efficiently be added to the database. While this method did work fairly well, it did not conform to the bunq coding style, as it resulted in a large amount of setter methods being called in one function, making the function too long. On top that, it is not allowed for models, which are further described in chapter 8, to have a large amount of attributes. At this point we thought of 3 different alternatives, which will be explained in the following paragraphs.

Alternative 1 was to still create an associative array out of every row, but instead of going over every element in the array and adding them to a model we only added the values from the array that we needed to index. These indexed attributes were chosen by the data scientist in order to make his searches on the database table more efficient. We could then encode the entire array as a JSON string and add that as one attribute to the model in a MYSQL *LongBlob* format. The data could then be extracted quite efficiently by getting this attribute from the database and decoding the JSON string. This method saves us more than 100 rows over our first attempt and does not result in more (sub)tables or rows.

The second alternative was to use what in the bunq back-end context is called an extended model. What an extended model does is use a certain amount of base attributes and on top of that use attributes from a collection of submodels. For our problem this would mean that we would divide all the original attributes into categrories and create a submodel for every attribute. While this method would decrease the size of the touchpoint model, this method would still result in the same code style difficulties, as the total amount of colums stays the same.

The last alternative was to decrease the size of the model to basically only contain an id and other standard bunq database metadata. The rest of the touchpont values would then be stored in a different table containing one column with the touchpoint id and another with one of the touchpoints values. This would result in a separate table containing all the touchpoint values with one touchpoint value per row. The problem with this method is that it will result in a table with an immense amount of rows. This amount of rows is equal to the amount of csv attributed (120) times the amount of rows in our first attempt.

Because of the (dis)advantages of all 3 methods we have chosen the first method to use for the touchpoint model. The reason is that it is the best scaling option of the 3 and because the raw data in JSON format can still easily be used to extract data without any losses. This all leaves us with two tables that we have added to the database: one for every object in the S3 bucket and one for every touchpoint (csv row) from the bucket objects. Using these tables we match the touchpoints to bunq users, which will be described below.

6.2 Matching Adjust touchpoints with bunq users

To train the machine learning model, we need to know which specific advertisements were efficient and which were not. One of the inputs to determine whether an advertisement is efficient is the amount of users gained by that advertisement. The Adjust data does not provide us with any information about whether the person who clicked on an advertisement became a user in the end. That is why we need to match the Adjust data with the data bunq has about their users.

To do this, we wrote two algorithms that try to find a match with a user for every touchpoint. If a match is found, the UserId of the user is added to the AdjustTouchpoint table. The way the algorithms work and the choices that are made are explained in the two following subsections.

6.2.1

[redacted]

7 | Machine learning model

During our research phase we discussed how reinforcement learning is widely used for problems like marketing attribution and how the standard approach for these kinds of problems is usually by using supervised learning models. In the end, the models that were trained on the data collected and matched by us are supervised learning algorithms. In this chapter we will elaborate on the concept of supervised learning, how the trained models are used within our system and what the models produce as results.

7.1 Supervised deep learning

Supervised learning is a machine learning method that distinguishes itself from other methods by using annotated training data as a 'supervisor'. The supervised learning algorithm aims to learn a mapping function $Y = f(x)$, where x represents the input variables and Y the output variable computed by the function f. Using the training data the algorithm approximates the mapping function with the goal of using the function to accurately compute Y with new, previously unknown, input data x [19]. The problem of approximating this mapping function can be grouped into $regression$ and $classification$ problems. Both these kind of problems have the goal of finding an accurate mapping function, but differ in that regression uses numerical attributes and classification uses categorical attributes. In this context numerical data is data that is represented by a number, for instance a persons age or the amount of times a person clicks on an advertisement. Categorical data represents characteristics that can be regarded as an element of a category, for instance a persons gender or hometown. The final models use regression, because it works towards finding numerical values like the expected amount of click or the cost per acquisition.

Deep learning algorithms are a form of machine learning that allows a computer program to learn from experience and are inspired by biological neural networks [19]. Without being programmed to perform a specific job, these neural network inspired programs learn to replicate certain tasks by considering a large amount of examples. Neural networks consist of an interconnected collection of nodes, similar to the neurons in a brain. The difference between deep learning neural networks and standard neural networks lies in the amount of internal (hidden) nodes that the network uses, which is shown in figure 7.1. Because of the large amount of layers of internal nodes, deep learning algorithms are able to learn from complex nonlinear data, like consumer touchpoint behavior. In the next section we will elaborate on what this means for the models that were trained on the data that we collected.

7.2 Training the model

The dataset that was described in chapter 6 was used to train the model. As of writing this report the size of this dataset was about [redacted] rows, of which about [redacted] rows were matched with a bunq user. The matched rows gave the most valueable information, but both the matched and unmatched rows were used to train because even unmatched rows give information about consumer's touchpoints. As stated before, we did not design or create the machine learning model, but we did work closely together with the data scientist who did. We will discuss some interesting statistics about the training of the model in this section.

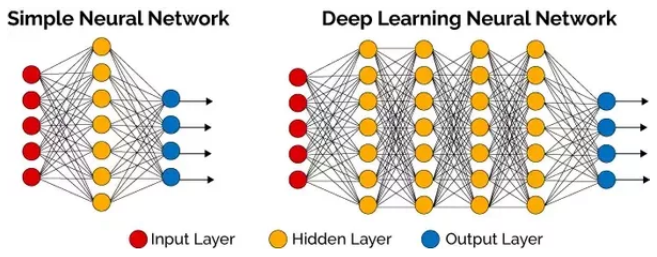

Figure 7.1: Difference between a simple neural network and a deep learning neural network

7.2.1 Training parameters

Table 7.1 shows a list of training parameters for the model that outputs the expected amount of clicks. The first parameter we see is the number of folds used for K-fold cross-validation. K-fold cross-validation is a method for evaluating machine learning models using resampling. The parameter k represents the number of groups in which the dataset is split up. Carefully choosing a value for k is important as it might lead to a non-representative evaluation of the model [20].

The amount of hidden nodes per layer is also an interesting parameter to look at. Using too many nodes in the hidden layer can lead to overfitting, making the model less generally applicable. It will also lead to a longer time to train the model. Too few neurons in the hidden layers also causes problems. An amount of hidden nodes per layer that is too small will lead to underfitting, which means that the layers do not contain enough neurons to adequately learn patterns in complex data [21].

Another parameter that is interesting to take a look at is the hidden_dropout_ratios parameter. This parameter specifies the ratios in which hidden neurons will be ignored during training. This essentially means that their influence on neurons down the stream and vice-versa is temporarily disabled. Using dropout can improve generalization and will prevent overfitting to a certain extent.

Parameter	**Value**
nfolds	4
response_column	TotalClicks
activation	RectifierWithDropout
hidden	300, 400, 500, 600
epochs	836,25
hidden_dropout_ratios	0.15, 0.1, 0.05, 0.01
max_w2	10
initial_weight_distribution	Normal Distribution
stopping_rounds	0

Table 7.1: Training parameters of the expected amount of clicks model

On top of these machine learning parameters, there were also model specific variables that influence weight of certain factors of the training process. Figure **??** shows the scales of importance for a multitude of input variables. For example, we can see that installs are the most important to bunq in this model as these values have very high importance. All of these variables are decided on by the data scientist, who has a good idea of where the focus of the model should be.

When training the deep learning model, the deviance statistic was used to test the goodness-of-fit of the current state of the model. As for the training of this model, we can see from figure **??** that the deviance starts off very high and then converges to a lower value. The model stops training when either the stop value for the deviance has been reached or the maximum amount of epochs is reached. In this case we can see from figure 7.1 that no stop value is set, but that there is a maximum amount of epochs set. From figure **??** we can see that the model stopped training when this number of epochs was reached.

7.3 Using the trained model

We got a few different models delivered. Our first fully trained model outputted the expected amount of clicks of an advertisement. The second one outputted the cost per acquisition of an advertisement. Both models take a set of advertisement parameters as input. The input fields are largely the same for both models. The difference between the two models is that the cost per acquisition model contains a `cost in euro` field, which the other model does not. Figure **??** shows the parameters of the model calculating the expected amount of clicks:

The output consists of a single value, representing either the expected amount of clicks or the cost per acquisition depending on the model. This value is used by the server to evaluate the effectiveness of a call.

8 | Connecting the model to the bunq backend

A few months before we started this project, an employee from bunq programmed a machine learning server (ML-server) in python to fetch predictions out of machine learning models. By sending parameters over a route towards the server it is possible to obtain predictions from a model. Different routes correspond to different machine learning models. At bunq we were told that we could use and adjust the server, in such a way that we can use it for our project. The purpose of the ML-server is that it is able to load a lot of different models which can then be accessed in a rather easy way. Unfortunately, the ML-server was not created with marketing attribution in mind, but with transaction monitoring instead, and was therefore in its delivered state not suitable for our project. Therefore, we extended the server with two major improvements: functionality to support plain old java object (pojo) models and functionality to get predictions based on a search space as input.

The software used to create machine learning models by data scientists at bunq is called H2O. H2O is Java-based software for data modeling and general computing [22]. It contains a lot of analysis tools which leaves a lot of expansion possibilities open for our server. For example, we could be able to show more details about the found prediction, such as confidence intervals.

The ML-server is connected with an endpoint in bunq's core back-end through a view, only accessable to those with admisitrator priveledges. The marketing department can interact with the ML-server through this endpoint. The ML-server is placed in the back-end in such a way that it is only connected to the core back-end. Any security measures are dealt with by the core back-end. This structure provides the marketing department with an endpoint which should be callable from the Admin page. bunq's front-end department can extend the Admin page with a simple panel to make it possible to fetch predictions in an easy way.

The readme of the repository is included in appendix **??** and provides instructions of how the server should be used.

8.1 Configuring the server

The ML-server holds a settings .json file. In this file we define any models the server should include, their feature types (which is equivalent to input parameters types, for instance for advertisements), any thresholds that are used to label returned predictions and any other relevant configuration data. An example is shown in figure 8.1.

When the server tries to access a model, it does so by checking whether its corresponding binary or jar file is included in its `./data` folder. Each model should have its own file. Next to these model-specific files, we should also add a jar containing any non-model-specific Java code. This jar is called `machine_learning_java_adapter.jar` by default.

```
{
  "app": {
    "name": "bunq_mlserver",
    "data_dir": "../../data"
  },
  "server": {
    "bind_port": 54321,
    "bind_host": "127.0.0.1",
    "gunicorn_options": {
      "log-syslog": false
    }
  },
  "daemon": {
    "pidfile": "/tmp/bunq_mlserver.pid"
  },
  "api": {
    "namespace": {
      "madls": "MarketingAttributionDeepLearningSimple"
    }
  },
  "model": {
    "MarketingAttributionDeepLearningSimple": {
      "model_filename": "All_features__5epoch_450x250x450_1",
      "type": "pojo",
     *****
      "threshold": [
        {"group": "low", "min": 0.0},
        {"group": "mid", "min": 13.2},
        {"group": "high", "min": 40, "max":100}
      ]
    }
  }
}
```

Figure 8.1: Example of a settings json file, containing a single model.

8.2 The Java Adapter

A problem we directly faced is that it is not possible to run a pojo model directly through a python command. Therefore we created an adapter in Java. Its job is to provide communication to any H2O machine learning model that is defined in pojo format. Communication between python and Java is done through Pyjnius, which is a python library for accessing Java classes [23]. Pyjnius allows us to call methods from a jar through python.

We could have also chosen to replace the python code by Java code and implement a server in java. But doing so would require to reimplement a lot of functionalities. The ML-server already was, for example, already able to work in threads, have a config file and create routes for models according to their namespace. Next to that: we would also have to figure out how to connect to binary models in Java if we wanted to keep that functionality.

The Java adapter can be used to build model-jar files for the server's data folder. Such a generated model-jar file should at least contain three classes, next to any dependencies these classes use. First, the pojo model itself, which is generated code from H2O and usually the deliverable from the data scientist. Second, the `ModelClassContainer` class, which is a class that our main predictor is able to recognize. Third, we need to manually create a custom class that extends `ModelClassContainer`, whose name should be equal to the filename of the jar and the "model_filename" field in the settings .json file. Since this custom class is an extension of `ModelClassContainer` but contains a specific model, we can use this as input for the main callable predict function.

Exceptions are thrown with messages that are as clear as possible when something is missing. This makes it easy to see if something went wrong regarding adding a new model or generating the jar.

8.3 Fetching predictions from the server

When the model is included in both the data folder and the settings file, we can call it by taking the corresponding namespace, which is also defined in the settings file, as part of an API REST call. `HTTP GET <model_route>/info` responds with information about the model. `HTTP POST<model_route>/predict` responds with predictions based on the parameters within the body.

Initially the predict route was built in such a way that it can make a prediction from a single combination of advertisement parameters for a binary model. We extended this with array support, which means that instead of assigning a single value to an advertisement parameter, we are able to assign an array of values. This enables the use of a search space. From this search space, the server finds the best X results, where X is defined as the field `amount_of_predictions` of the input.

By default, the server tries to approach an optimal set of advertisement parameters in an iterative way using bayesian techniques. To do this, we use the hyperopt python package. It evaluates which parts of the search space work best by sending a variety of different advertisement parameters to the ML-model. For each call made to the model, the system evaluates the output of the model to choose a new set of advertisement parameters. With this feedback loop, it tries to approach the optimal solution within Y iterations, where Y is defined as an input field. This creates a list, which includes the found optimal solution and any other iterated combinations of advertisement parameters after these Y iterations. From this list, we then return the best X results.

A different technique to fetch the best X results is through a brute force search among all possible combinations of advertisement parameters. This process is triggered whenever the field `process_all_combinations` is set to `True`. The server constructs a collection of all possible parameter combinations, sends each of them to the model and returns the best X results. When using this technique instead of the bayesian one, increasing the length of the arrays will exponentionally increase the runtime since the server brute forces all possible combinations.

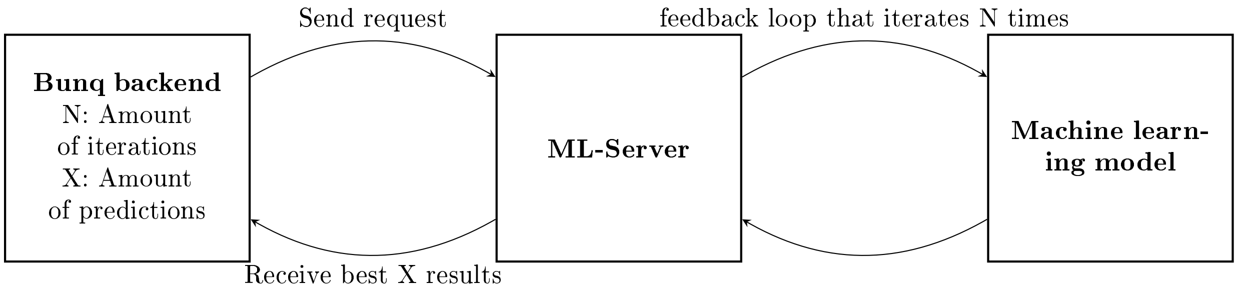

Figure 8.2: A schema that shows the flow from the client to the ML-model and back.

In almost every case the bayesian method will provide the most efficient answer. We expect this to be the most often used route. Brute forcing all combinations might be useful when we want to evaluate some or a few specific cases as it might reveal new insights. Therefore we did not remove the brute force method from the server and left it as optional, but we did set the default method to be bayesian.

8.4 Architecture of the ML-server

The ML-server contains the `model_controller_base.py` file, which can create `ModelBase` objects. Those objects contain functionality to fetch a prediction from a model. A prediction request can be of two different types: pojo or binary, and can also be performed either brute force or bayesian. This sums up to four possible paths to take into account. The difference between brute force or bayesian is handled inside the `ModelBase` class by splitting our process function up in two. The difference between calling a binary model or pojo model is made in extended classes from `ModelBase`, which are `ModelJava` and `ModelBinary`. These child classes should contain a `predict_single` function and may contain an optional `custom_pre_process` function. Since the output should be of the same format for each process, every predict is going through the same post-process. Any configuration data, which is discussed earlier, is retrieved through `config.py`.

The `api_controller` class creates a `Server` object which holds an extension of `ModelBase` for each model. Requests for this model are handled by calling `process()` on the model and returning its response. Due to the use of Daemons, multiple predictions can be run simultaniously.

During the creating of both the server and java adapter, we set the focus on easy expansion. In the future bunq might want to load models of unsupported catagories (so non-pojo or non-binary models) in the server. Since we will not be around the bunq office after our project is finished to help people expand on our code, we had to make sure it would not take up a lot of work for someone to extend our code. Therefore we have also provided a detailed readme of how to use the server, which can be found in appendix ??.

8.5 Architecture of the Java adapter

The adapter is written in an object oriented way. There are some different model categories defined by H2O. Models of the same category are called in the same way and have the same output. The adapter includes the `ModelPredictorAbstract` class, which is the parent class of `ModelPredictorBinomial` and `ModelPredictorRegression`. Next we have the `Parameter` interface and `ParameterSingle` class. This class represents an advertisement parameter as input for the model. A list of `Parameter` interfaces can be send to a predictor to obtain a `ModelResponsePair`. The `ModelResponsePair` represents an input-output combination, and has `ModelCategory`-specified child classes.

Next we have the `ModelClassContainer` class. This class contains a model and its corresponding category. If we want to add a new model, we should create a custom child of `ModelClassContainer` with no constructor parameters. This class, together with its model is put into a seperate jar. The custom `ModelClassContainer` class will form one of the inputs to fetch a prediction. The other inputs are the parameters. The call to the main function is made in `ModelCaller`. `ModelCaller` picks the corresponding `Predictor` class and returns its output in .json format.

The UML of this Java-adapter can be found in figure 8.3.

8.6 Discussion

Fetching optimal solutions from the ML-model is quite a complex and computationally intensive task. It was therefore important this would happen efficiently, because for such a huge model, long runtimes might result in the system being infeasable to use. Connecting the model to the bunq backend was done using both Java and python. Having the process run efficiently through multiple languages raises difficulties, such as that a JVM is generated in python, Java is heavily object-oriented and errors are not always easy to recognize when passed on between Java to python. This section explains some of the decisions we had to make, and a few problems and limitations we have faced.

8.6.1 Necessity to expand the ML-server

There were two main reasons for which we had to extend the server with extra functionality.

First of all, the server did only support binary models and no pojo models. Such models are represented by executable binary files. In python, binary models can be called rather easily by using functions from the H2O python package. The marketing attribution ML-model we obtained was in pojo format. These type of models are represented by a Java file and should first be compiled, before we can call them. The maker of the marketing attribution machine learning model preferred to build the model as a pojo instead of a binary object, since a binary model would probably hurt perfomance, and because the maker is not able to inspect or adjust the model's generated code if it is a binary model. Besides that, functionality to load both pojo and binary models gives the python server an extra dimension of flexibility. To tackle this problem, we have built the Java Adapter.

Secondly, marketing attribution is based on inserting combinations of different parameters in the model and picking the best X results from it. This is different from transaction monitoring, for which the server was initially built. Transaction monitoring takes a single transaction as input, runs it through a ML-model, processes the result by flagging it as suspicious or not and goes on to the next. A single Marketing Attribution request is performed by running multiple sets of parameters through the model and returning the X best results as the most profitable recommended actions for the marketing department. [redacted]
The first version we have build was a brute force approach and computed each possible combination of parameters to return the one with the highest value. The second version included bayesian techniques to search for an optimal solution using a feedback loop.

8.6.2 Multiple JVM instances

One of the limitations of the python server is that it is not possible to create multiple Java Virtual Machine (JVM) instances for a single python process. A JVM instance is created as soon as `import jnius` is called in python, and should therefore be called before the first java call is made since JVM options cannot be changed after the JVM starts up. As a result, if one would add or change something Java related, such as adding a new model, the server needs to be restarted in order to let it rebuild the JVM.

It is difficult to determine whether using a different package than pyjnius has the guarantee to not face the same problem. Jpype, a similar package, addressed the same issue in their docs, but explained that the issue is due to the SUN JVM, the JVM that is used by pyjnius, giving a non-specific exception when a `startupJVM()` is called after a `destroyJVM()`.

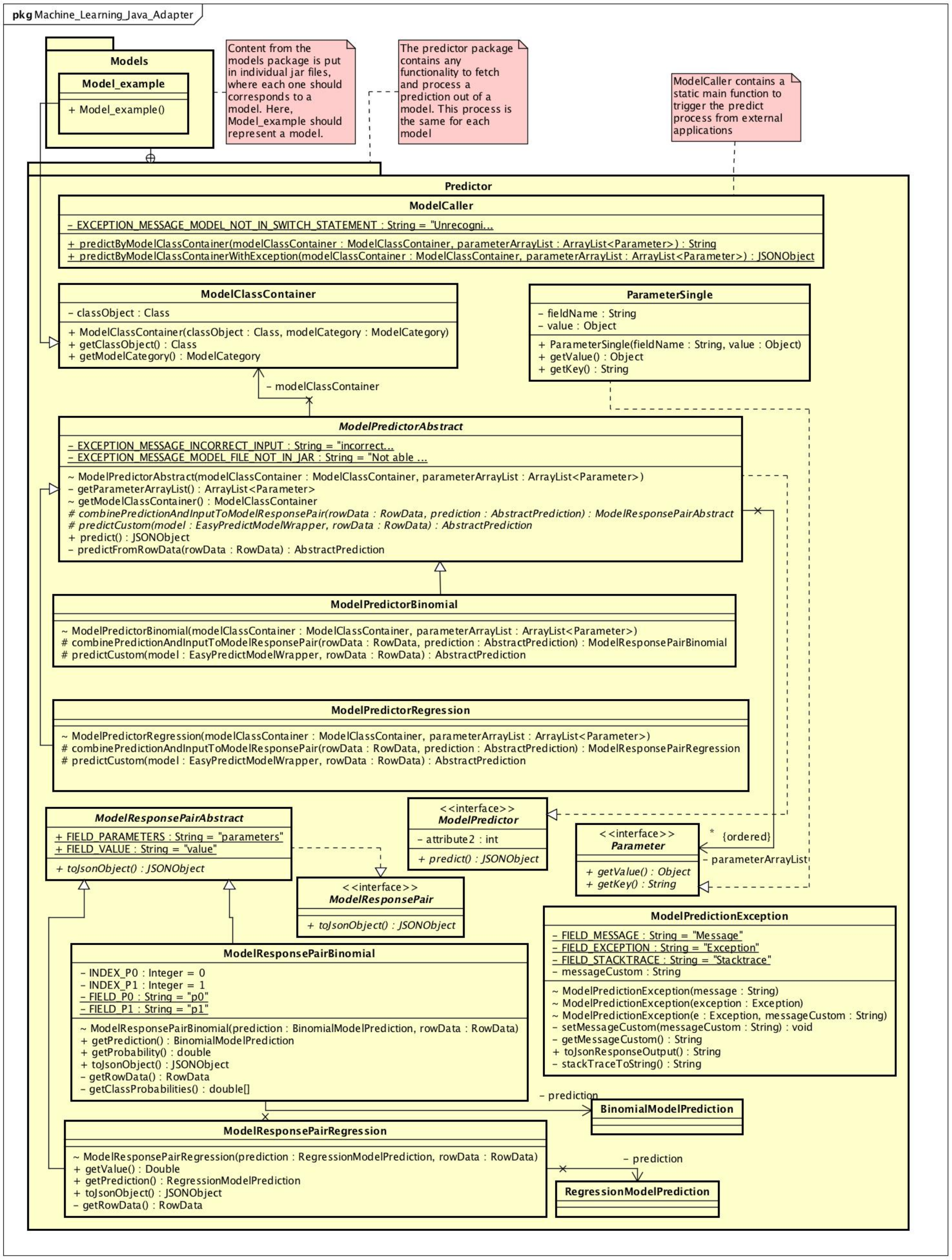
pkg Machine_Learning_Java_Adapter
Models
Model_example
+ Model_example()
Content from the models package is put in individual jar files, where each one should corresponds to a model. Here, Model_example should represent a model.
The predictor package contains any functionality to fetch and process a prediction out of a model. This process is the same for each model
ModelCaller contains a static main function to trigger the predict process from external applications
Predictor
ModelCaller
- EXCEPTION_MESSAGE_MODEL_NOT_IN_SWITCH_STATEMENT : String = "Unrecogni...
+ predictByModelClassContainer(modelClassContainer : ModelClassContainer, parameterArrayList : ArrayList<Parameter>) : String
+ predictByModelClassContainerWithException(modelClassContainer : ModelClassContainer, parameterArrayList : ArrayList<Parameter>) : JSONObject
ModelClassContainer
- classObject : Class
+ ModelClassContainer(classObject : Class, modelCategory : ModelCategory)
+ getClassObject() : Class
+ getModelCategory() : ModelCategory
ParameterSingle
- fieldName : String
- value : Object
+ ParameterSingle(fieldName : String, value : Object)
+ getValue() : Object
+ getKey() : String
- modelClassContainer
ModelPredictorAbstract
- EXCEPTION_MESSAGE_INCORRECT_INPUT : String = "incorrect...
- EXCEPTION_MESSAGE_MODEL_FILE_NOT_IN_JAR : String = "Not able ...
~ ModelPredictorAbstract(modelClassContainer : ModelClassContainer, parameterArrayList : ArrayList<Parameter>)
- getParameterArrayList() : ArrayList<Parameter>
~ getModelClassContainer() : ModelClassContainer
# combinePredictionAndInputToModelResponsePair(rowData : RowData, prediction : AbstractPrediction) : ModelResponsePairAbstract
# predictCustom(model : EasyPredictModelWrapper, rowData : RowData) : AbstractPrediction
+ predict() : JSONObject
- predictFromRowData(rowData : RowData) : AbstractPrediction
ModelPredictorBinomial
~ ModelPredictorBinomial(modelClassContainer : ModelClassContainer, parameterArrayList : ArrayList<Parameter>)
# combinePredictionAndInputToModelResponsePair(rowData : RowData, prediction : AbstractPrediction) : ModelResponsePairBinomial
# predictCustom(model : EasyPredictModelWrapper, rowData : RowData) : AbstractPrediction
ModelPredictorRegression
~ ModelPredictorRegression(modelClassContainer : ModelClassContainer, parameterArrayList : ArrayList<Parameter>)
# combinePredictionAndInputToModelResponsePair(rowData : RowData, prediction : AbstractPrediction) : ModelResponsePairRegression
# predictCustom(model : EasyPredictModelWrapper, rowData : RowData) : AbstractPrediction
ModelResponsePairAbstract
+ FIELD_PARAMETERS : String = "parameters"
+ FIELD_VALUE : String = "value"
+ toJsonObject() : JSONObject
<<interface>>
ModelPredictor
- attribute2 : int
+ predict() : JSONObject
<<interface>>
Parameter
+ getValue() : Object
+ getKey() : String
*
{ordered}
- parameterArrayList
<<interface>>
ModelResponsePair
+ toJsonObject() : JSONObject
ModelPredictionException
- FIELD_MESSAGE : String = "Message"
- FIELD_EXCEPTION : String = "Exception"
- FIELD_STACKTRACE : String = "Stacktrace"
- messageCustom : String
~ ModelPredictionException(message : String)
~ ModelPredictionException(exception : Exception)
~ ModelPredictionException(e : Exception, messageCustom : String)
- setMessageCustom(messageCustom : String) : void
- getMessageCustom() : String
+ toJsonResponseOutput() : String
- stackTraceToString() : String
ModelResponsePairBinomial
- INDEX_P0 : Integer = 0
- INDEX_P1 : Integer = 1
- FIELD_P0 : String = "p0"
- FIELD_P1 : String = "p1"
~ ModelResponsePairBinomial(prediction : BinomialModelPrediction, rowData : RowData)
+ getPrediction() : BinomialModelPrediction
+ getProbability() : double
+ toJsonObject() : JSONObject
- getRowData() : RowData
- getClassProbabilities() : double[]
- prediction
BinomialModelPrediction
ModelResponsePairRegression
~ ModelResponsePairRegression(prediction : RegressionModelPrediction, rowData : RowData)
+ getValue() : Double
+ getPrediction() : RegressionModelPrediction
+ toJsonObject() : JSONObject
- getRowData() : RowData
- prediction
RegressionModelPrediction

Figure 8.3: Detailed UML diagram of the java adapter. This diagram shows how classes interact with each other and gives an overview of all the classes and functions of the java code.

There are some options that might be able to fix this issue. The jpype docs suggested using a non-SUN JVM which supports JNI invocation. The writer says he believed that IBM's JVM support JNI invocation, but did not know if their destoyJVM would work properly [24].

A different solution would be to tweak the server in such a way it creates a new python process for each server call. This would require linking each thread that is created for a server call to a seperate python process.

However, we think any of these solutions would be out of the scope of the project, and therefore adressed it as a could have. Removing the need to restart the server whenever we update a model is much more useful to different kind of models. Marketing Attribution does not require to have a server up and running at all times, since any call is a result of a manual request and no automatic processes depend on it. It functions as a tool for the marketing department, which is an internal group and therefore it does not influence external users directly. This is different from transaction monitoring, where a call is made to model for each transaction.

8.6.3 Model size

Towards the end of the project we received a bigger ML-model, trained on real data. However, the size of the model was large: a single pojo of 8.8 MB. The models we had obtained so far to test were of sizes around 1 MB. When we imported this bigger model into the server and tried to fetch a response, the JVM gave no reply, and the server seemed to take an infinite amount of time to return a result. However, when we skipped the python-Java connection and predicted directly through Java, we were able to fetch a response from it within a second. What was also curious is that through a standalone python script we were able to fetch a single prediction from the model, using the same methods and packages as the server does.

We suspected it to be something memory related, since the size of this new model was around eight times as large compared to any previous models. Setting a higher maximum heap size gave no result. We did some experiments with the python script that did give results by enabling the JVM option `-XX:+PrintGC`, which enables printing messages during garbage collection. This eventually showed us the cause. For the larger model the python script increased the MetaspaceSize of the JVM, which is the maximum amount of space to be allocated for class metadata [25]. This increase of MetaspaceSize was not performed on our server when we executed the same lines of code, probably because of the fact that every server call is treated as a seperate python thread, and might behave differently. By setting the MetaspaceSize higher, specifically high enough such that the python script would not print a log that it is increasing the MetaspaceSize, the JVM's MetaspaceSize was increased and we were able to load the bigger sized model.

During the handling of this issue, we found out that the model runs a lot faster when the heap size of the Java thread is increased. By setting it on 4 GB, we went from 200 iterations in 90 seconds, to 1000 iterations in 50 seconds. This means that there is a lot of optimization possible. However, since we have found and handled this issue in the last week of the project, it was out of our scope to do this, and we will leave this as an recommendation for bunq to look at when they want to optimize it further.

9 | Results

With all the functionality described in the previous chapters we can bring our work at bunq to a full circle. We collect data from Adjust, match it to bunq users and use the combined dataset to train the machine learning model. This model is then loaded into the python server to which API-calls can be made to get a prediction from the model. In this section we will show the results of this full circle using a clear example.

Appendix E shows a list of features that we will use as an input to the ML-server. We have set `amount_of_iterations` and `amount_of_predictions` at 2000 and 10 respectively for this example, which means that after 2000 iterations, it will return 10 results. The .json response contains an array with the highest scoring advertisement options from all iterations. Figure 9.1 shows the best result found. Figure 9.2 shows an existing advertisement that bunq has used before. As of writing the report, the bunq marketeers have not yet had the time to create an advertisement based on our results.

CENSORED

Figure 9.1: Machine learning model output of the bunq advertisement. The amount of iterations was set on 2000 and CostEuro was defined to search between 0 and 100 euro. According to the found result, we should target females in Germany aged between 55-64 at 5pm. Note that the text parameter only describes the content of the advertisement, not the language. It does not imply that a Dutch text should be promoted in Germany.

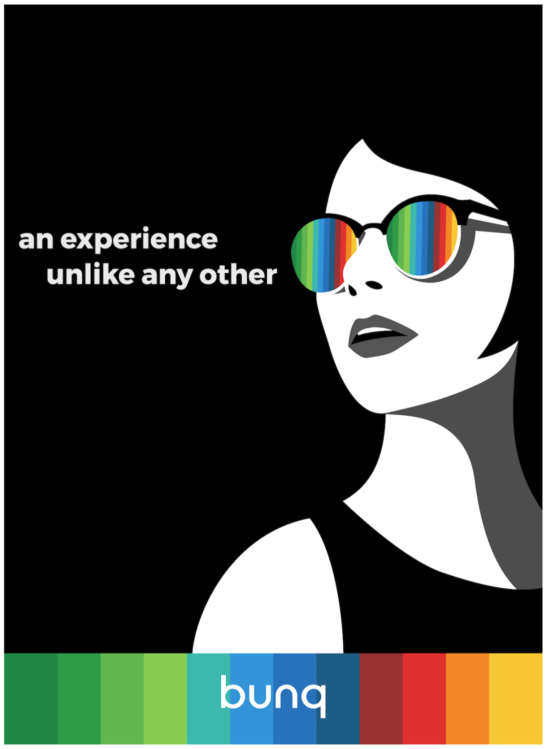

Figure 9.2: A commonly used bunq advertisement

10 | Conclusion

The marketing attribution project that we worked on for the past 10 weeks proved to be a more diverse problem than we had initially thought. We used multiple programming languages and various types and sources of data in order to reach the final product. This product consists of back-end functionality to periodically and automatically collect and store data on consumer touchpoints with bunq, back-end functionality to match existing bunq users to these touchpoints and a server that is allows for accessing multiple machine learning models trained on the aformentioned data using API calls in the back-end.

To create a dataset that is is suitable for training a supervised learning algorithm a few steps had to be taken. The first step was to efficiently collect and store the data on consumer touchpoints. This involved creating functionality to download and process the data, which came in the form of a .csv file every hour. This work is executed by Daemons, which also run once every hour.

The next step was to match the data to bunq users, before being able to be used as a training dataset. By using queries integrated in the bunq back-end we created a matching algorithm that looks at a consumers IP-address and the time of creation of the touchpoint. Like the data collecting part, this algorithm is also executed by a Daemon that runs periodically. When the algorithm finds a match, the user data from the bunq database is added to the touchpoint database row and can be used by the machine learning model.

The next step is training the model. We did not do this, as creating and training a machine learning model was outside of the scope of the project. The training was done by a data scientist from bunq, directly using the data written to the bunq database by our daemons.

The trained model can then be accessed through a python server, which is able to host multiple models. Through an API-call, we can ask the server to find an optimal set of advertisement parameters using bayesian techniques.

Finally, when a marketeer decides on the features of an advertisment, this person is now able to use these features as an input of an API call. The model will then provide the user with an expected amount of clicks or the return on investment rate of this advertisement. This brings the project to a full circle, where the consumer behavior is monitored and advertisements are adjusted based on this.

11 | Recommendations

11.1 Front-end interface

To make feeding the features of an advertisement to the model easier for people who do not have any backend experience, a simple interface for marketeers is almost a must. Bunq uses an admin web interface, which can be used for instance to handle tickets opened by users and to monitor transactions. This would be the perfect place for a marketing attribution interface. Because learning how the bunq front-end works and implementing an interface are both outside of the scope we did not create an interface. For someone with front-end experience at bunq this would probably be a very easy task, as all of the functionality we have created is accessable using simple API-calls. During the project we already had a meeting with a front-end developer to talk this through.

11.2 Improved matching

[redacted]

11.3 Adjust API

The way in which Adjust makes their data available for programatic research is not optimal. They collect data and dump all the data in an Amazon S3 bucket as .csv files. Handling these files in PHP is not impossible, but a much more elegant solution would be if Adjust would create an API to directly access their touchpoint data. This way, instead of having to download, unzip and transform to an array, we could just call functions like $getTouchpoint(...)$ or $Iterator.getNextTouchpoint()$. If Adjust ever creates this API our code could be significantly simplified and would also be more efficient.

11.4 Automatic training

Right now the bottleneck in our final product as a whole is the training of the machine learning model. At the request of the data scientist the first version of the product would not include automatic continuous training of the model. The reasons for this were that implementing this automatic training is difficult to do well and the data scientist which created the model prefers to do it himself. We think that if automatic continuous training can be implemented reliably it would be a significant improvement to the product. As explained in section 8.6.2, adding a new model to the server is currently not possible without restarting the server and changing configuration settings. One part of an automatic training process would be to replace models on the server. Therefore, fixing the problem described in section 8.6.2 is a part of adding an automatic training process.

Bibliography

[1] Van der Pol, M. (2018) *Interview met Ali Niknam*, Z Life, found at: *www.bright.nl/bright-business/bunq-banken-zijn-nu-een-pot-nat.*

[2] Con, J. (2016), *B2B marketing attribution 101*, Bizbible, found at: *www.bizible.com/hubfs/B2B-Marketing-Attribution-101-ebook.pdf.*

[3] Convertro (2016) *The definitive guide to marketing attribution*, found at: *www.convertro.com/white-papers/the-definitive-guide-to-marketing-attribution.*

[4] Adjust (2018) *Buyer's Guide to Mobile Attribution*, found at: *www.learn.adjust.com/rs/108-GAZ-487/images/Buyers%20Guide%20to%20Mobile%20Attribution.pdf.*

[5] Adjust website, found at: *www.partners.adjust.com/placeholders/.*

[6] Adjust (2018) *Mobile Attribution*, found at: *www.learn.adjust.com/rs/108-GAZ-487/images/Mobile%20Attribution.pdf.*

[7] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), *Official Journal of the European Union*, Vol. L119, 4 May 2016, pp. 1-88.

[8] Lagoudakis, M.G. & Parr, R. (2003) *Least-Squares Policy Iteration*, Duke University.

[9] Voskoglou, C. (2017) *What is the best programming language for Machine Learning?*, Towards Data Science, found at: *www.towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7.*

[10] PHPUnit website, found at: *www.phpunit.de.*

[11] Vojvodic, M. (2017) *Addressing GDPR Compliance Using Oracle Data Integration and Data Governance Solutions*, Oracle white paper.

[12] Zimbres, R. A. & Rolim, G.A. (2018) *GDPR in Machine Learning*, MLMAG.

[13] Agile Software Development, found at: *https://www.techopedia.com/definition/13564/agile-software-development.*

[14] PEP-8 coding style website, found at: *https://www.python.org/dev/peps/pep-0008/.*

[15] Google Java coding style website, found at: *https://google.github.io/styleguide/javaguide.html.*

[16] Elastic Search website, found at: *https://www.elastic.co/products.*

[17] curl website, found at: *https://curl.haxx.se/.*

[18] AWS ADK for PHP 3.x, found at: *https://docs.aws.amazon.com/aws-sdk-php/v3/api/index.html.*

[19] Kim, K. G. (2016) *Deep Learning book review*, found at: *https://www.synapse.koreamed.org/Synapse/Data/PDFData/10*
22-351.pdf.

[20] Refaeilzadeh P., Tang L., Liu H. (2009) Cross-Validation, *In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems.* Springer, Boston, MA.

[21] Heaton, J. (2008) *Introduction to Neural Networks for Java*, Heaton Research.

[22] h2o documentation, found at: *http://h2o-release.s3.amazonaws.com/h2o/master/3574/docs-website/h2o-py/docs/intro.html*

[23] pyjnius documentation, found at: *http://pyjnius.readthedocs.io/en/latest/index.html.*

[24] jpype documentation, found at: *http://jpype.sourceforge.net/doc/user-guide/userguide.html.*

[25] oracle blog, found at: *https://blogs.oracle.com/poonam/about-g1-garbage-collector%2c-permanent-generation-and-metaspace.*

Appendices

A | Info sheet

General information

Title of the project: Attribution Modelling
Client: bunq
Date of final presentation: 04-07-2018

Team

Sytze Andringa

Interests	Data Science Multimedia Computing
Contributions	Back-end Research Machine Learning Server

Daan van der Werf

Interests	Software Development Machine Learning
Contributions	Database model definitions Collecting data from Adjust

Job Zoon

Interests	Software Development Algorithms
Contributions	Matching data functionality Back-end connection to server

All members contributed to the final report and to the final presentation.

Client & Coach

Client

Name	Ir. Wessel Van
Company	bunq
E-mail	wessel@bunq.com

TU Delft Coach

Name	Dr. Matthijs Spaan
Department	Software Technology
Group	Algorithmics
E-mail	m.t.j.spaan@tudelft.nl

Contacts

Sytze Andringa	sytze@morvigor.nl
Daan van der Werf	daanvanderwerf@hotmail.com
Job Zoon	jobzoon96@gmail.com

Description

Challenge One of the greatest challenges in marketing is measuring the revenue of a marketing campaign and translating that into a strategy. One of the employees at bunq created a machine learning model in Java which can solve this by predicting how advertisements will perform. It is our job to collect data to train this model and to create a system in such a way that bunq's marketing department can use it.

Research During the research phase, we looked into the architecture of our application, the way we have to collect and match data, how the machine learning model works and the way we should develop code.

Process We worked at the office of bunq in Amsterdam for 5 days a week from 9 to 6. bunq provided us with MacBooks and access to their systems, and whenever we had any questions there was always someone to answer them. Some of the unexpected challenges we faced during the project are the difficulty of programming in bunq's back-end, not having access to the bunq database to test our algorithms and not having a server in place yet that can deal with big POJO models.

Product To train the machine learning model, which was created by a bunq employee, data from a company called Adjust was used. Adjust is an advertisement tracking company that dumps CSV-files to an AWS bucket. We collected those files and stored the rows in the bunq database, where we also matched the rows with bunq users. A matching algorithm based on IP-address and timestamps was implemented to do this.

After the creator of the machine learning model trained the model, the marketing department needed a way to access it. We created an H2O server in python, which can make calls to the Java model. Also, we implemented code in the back-end that accepts API-calls for the machine learning model and sends it through to the python server.

Outlook We opened merge request for the different components of our end product, and they are, or will be, merged into bunq's production code. Recommendations for the future are improving the matching algorithm, creating a front-end application to automate the API-calls and automatically training the machine learning model with the data we created.

The final report for this project can be found at: `http://repository.tudelft.nl`

B | Original project description

B.1 Project description

The goal of this project is to develop and build an attribution model for marketing. The model ingests very detailed data from different sources such as visits to the website of bunq, online marketing campaigns data, clicks on online advertisements and installs of the bunq app or even data about offline marketing activities and attributes bunq users to a certain marketing channel. This way of tracking allows us to find out which marketing channels deliver the most value in terms of users. Furthermore, attribution modelling allows us to improve our online marketing efforts by for instance having an ad bidding tool in place that learns to evaluate the effectiveness of marketing campaigns by looking at the combination of target group and the marketing creatives. The machine learning model becomes trainable by learning from the data outputted by the attribution model.

B.2 Company description

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.

C | SIG feedback

C.1 Initial feedback

Following is the initial feedback from SIG in Dutch:

> *De code van het systeem scoort 4.3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size.*
>
> *Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes.*
>
> *In de Java-code binnen jullie project is ModelPredictorPojo.predictFromRowData() de enige lange methode, maar omdat daar vreemde dingen gebeuren is het alsnog een goed voorbeeld. Jullie gooien daar op regel 155 een ModelPredictionException, maar vervolgens wordt die exception een paar regels later afgevangen en null teruggegeven. De vraag is dus wat de bedoeling is op het moment dat de input van de methode niet geldig is: (1) de methode geeft null terug, (2) er wordt een exception gegooid. Op dit moment doen jullie een combinatie van allebei.*
>
> *In de Python-code zien we meer lange functies. daemon() in cli.py is een goed voorbeeld, dit is ook typisch een functie die nog veel groter gaat worden op het moment dat de hoeveelheid functionaliteit gaat toenemen. Je zou daar het parsen van de command line argumenten willen scheiden van het daadwerkelijk opstarten van de daemon.*
>
> *De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid tests blijft nog wel wat achter bij de hoeveelheid productiecode, hopelijk lukt het nog om dat tijdens het vervolg van het project te laten stijgen.*
>
> *Over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.*

The English translation of this feedback is:

> *The code scores 4.3 stars out of 5 on the maintenancemodel, which means it's above average maintainable. The highest score was not achieved because of the lower scores for Unit Size.*
>
> *For Unit Size, we look at the percentage of functions that are longer than average. Splitting these kind of methods into smaller pieces ensures that each component is easier to understand, test and therefore easier to maintain. Within the longer methods in this system separate pieces of functionality can be found which can be refactored to separate methods.*
>
> *In the Java code within your project ModelPredictorPojo.predictFromRowData() is the only long method, but because strange things happen there, it is still a good example. You throw a ModelPredictionException on line 155, but then that exception is caught a few lines later and null is*

returned. The question is therefore what the intention is when the input of the method is not valid: (1) the method returns null, (2) an exception is thrown. At this moment you are doing a combination of both.

We see more long functions in the Python code. daemon() in cli.py is a good example, this is a function that will become even greater when the amount of functionality increases. You would want to separate the parsing of the command line arguments from the actual startup of the daemon.

The presence of test code is promising. The amount of tests is still a bit behind in comparison to the amount of production code, hopefully it will be possible to increase that during the continuation of the project.

In general, the code scores above average, hopefully it will be possible to maintain this level during the remainder of the development phase.

C.2 Changes according to the feedback

There are two major points that could be improved according to the feedback:

- The unit size of our function in Java and python, which are used for the connection to the machine learning model.
- The amount of tests.

Both were fixed in the weeks after the SIG feedback. The unit size was easy to fix by splitting all of the larger functions up into smaller pieces. Also, we changed the weird Java function in such a way that it makes more sense.

We also wrote more tests and every bit of code that is reasonably easy to test is tested now. Some parts require a lot of very complicated existing Models from the bunq database, so those parts were almost impossible to test for us, unless we created huge testcases. We tested those parts of the code extensively by hand and by running them over huge amounts of data, so we are sure they run well and do not give any exceptions.

C.3 Final feedback

Following is the final feedback from SIG. We did not recveive a grade, only this feedback.

In de tweede upload zien we dat het project een stuk groter is geworden. De score voor onderhoudbaarheid is in vergelijking met de eerste upload iets gedaald. Jullie zaten bij de eerste upload ook erg hoog qua score, dus die daling is daarmee wel enigszins verklaarbaar. Bij Unit Size, het enige verbeterpunt, zien we weinig verandering. Jullie hebben het voorbeeld aangepast, maar er zijn in de nieuwe code weer nieuwe gevallen geïntroduceerd. Naast de toename in de hoeveelheid productiecode is het goed om te zien dat jullie ook nieuwe testcode hebben toegevoegd. De hoeveelheid tests ziet er dan ook nog steeds goed uit. Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback op de eerste upload deels zijn meegenomen tijdens het ontwikkeltraject.

D | Machine learning server repository readme

E | Features for model prediction