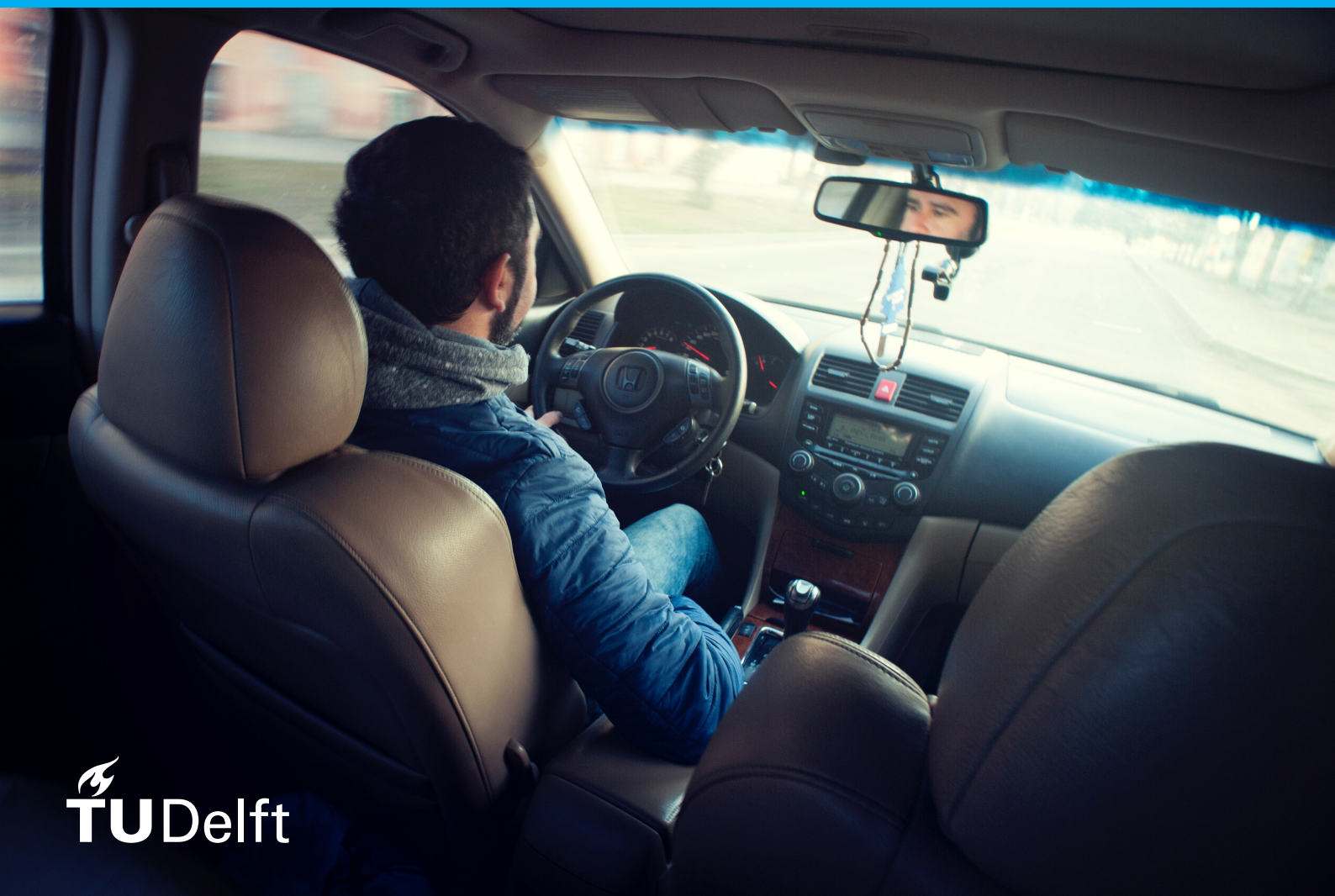
BEP TI3806

Automated Pricing Suggestions

Authors
J. Katzy
T. Rietveld
J.J. van der Steeg
E. Wiegel



TU Delft

BEP TI3806

Automated Pricing Suggestions

To obtain the degree of Bachelor of Science at the Delft University of Technology, to be presented and defended publicly on Thursday July 5, 2018 at 10:00 AM.

Authors:	Jonathan Katzy	
	Tim Rietvield	
	Jaap-Jan van der Steeg	
	Erik Wiegel	
Project duration:	April 23, 2018 – July 5, 2018	
Guiding committee:	Roel Bloo	CFO Sjauf, Client
	Stefan Dorresteijn	CTO Sjauf, Client
	Dr. M. Birna van Riemsdijk	TU Delft, Coach
	Dr. Huijuan Wang	Bachelor Project Coordinator

TU Delft

Abstract

As Machine Learning is becoming more accessible to small businesses, thanks to the rapid advance in computing power, smaller start-ups such as Sjauf (a ride sharing start-up) are starting to get interested in implementing Machine Learning solutions in their product. Sjauf needed a system that could automatically tell its customers how much a certain trip would cost them. Using this information multiple different models were developed and integrated into an ensemble. This ensemble as well as the models used by it were then used for price prediction. This project is a proof of concept to show that Machine Learning is capable of solving this problem in real time.

After researching state of the art Machine Learning models for price recommendation, the architecture of the system was designed. The supplied data was preprocessed, after which a custom Genetic Algorithm was developed for optimising models and ensembles. After validation on real-life company data, a comparison using empirical metrics was conducted. We use these empirical metrics to show that a bagging ensemble is the most efficient and accurate model for this purpose. This bagging ensemble outperformed the currently implemented functions, whilst adhering to the set boundaries on response times. Lastly, recommendations are made to the company with an overview of potential future work in this subject.

Preface

This project was conducted as final Bachelor project, part of the Computer Science Bachelor at the TU Delft and is worth 15EC. We want to thank Sjauf for their guidance and the opportunity to work in the context of a real project at their company. Special thanks go out to Stefan Dorresteijn who has invested his time in our project and has always answered our questions. We want to thank Prof. Dr. C. M. Jonker for coaching us in the early stages of the project. We would also like to show our gratitude to Dr. M. Birna van Riemsdijk who took over the coaching mid-way through the project when Prof. Dr. Jonker was unable to continue coaching us. Dr. van Riemsdijk has gone through the trouble of coaching us unexpectedly regardless of her already busy schedule.

Contents

1 Introduction

Ride sharing and Machine Learning are two topics that are currently booming in the start-up business. After companies like Uber and Lyft have both managed to reach billion dollar valuations in just a short time after being founded, there are plenty of start-ups that are attempting to get into the ride sharing market. Each start-up has their own spin on the ride-sharing business plan being used by Lyft and Uber. One of these is Sjauf which is a Rotterdam based start-up that offers its customers the ability to hire a driver that will come to them when requested and then drive the customer to their requested destination using the customer's own car. The main group of people that make use of this service are people who want to drive to an event themselves, and be driven back home at the end. This enables them to be able to drink alcohol and have the convenience of taking their own car to the location. The Machine Learning aspect of the project comes into play when a customer is requesting a quote for a trip from Sjauf.

The use of Machine Learning in the prediction of prices is not a new idea, in the past many open Machine Learning competitions have been hosted by companies. These competitions range from predicting movie ratings, such as in the famous Netflix prize [20] to predicting the price of real estate [1] or tractors [2]. The developments in Machine Learning have inspired Sjauf to use Machine Learning models to create a price prediction for their customers. The goal of this Machine Learning model is to give the customers an idea of how much they should expect to pay for a trip. In the end the driver does still set the price he wants to drive for himself. However, the customer will now have an idea of what the trip could cost before requesting a quote.

Building such a custom Machine Learning model requires the correct application of different models in order to function accurately. To start off the project report, the results of the research are documented in Chapter 2, it defines the challenges faced in this project and how Machine Learning algorithms can be used to solve the problem. After this, Chapter 3 discusses the structure of the software system that we created. In order to understand how we processed the received data to make it usable for Machine Learning, Chapter 4 is included. In Chapter 5 the Genetic Algorithm used to optimise the models is explained. The following chapter (Chapter 6) explains how the models work and how they are optimised using the genetic algorithm. This chapter concludes with a comparison of the various models. Chapter 7 discusses the ethical considerations made in this project. In the second-to-last chapter, Chapter 8, the conclusion of this project is provided. Chapter 9 concludes the report by exploring the possible future work and makes recommendations to the client.

2

Research Report

To understand the requirements of a good Machine Learning model, a research report was created. This research report discusses different techniques to preprocess data and generate a price from it. In Section 2.1 the problem will be defined and analysed thoroughly. Next, in Section 2.2 different techniques to preprocess the data are stated and explained. In Section 2.3 we explore a variety of methods to analyse the data and generate prices. Section 2.4 explains our fail-safe in case our program would fail. The choice of programming language is described in Section 2.5 and Section 2.6 concludes our findings in this research report.

2.1. Problem definition and Analysis

In order to get a good overview of what the customer wants and what exactly is expected of the program in terms of performance and accuracy, a clear definition of the problem has to be constructed. Also, an analysis has to be made of what will actually be implemented during the project. Once this is known, a set of requirements can be created which will have to be adhered to for the remainder of the project.

2.1.1. Problem definition

To improve the user experience, Sjauf wants to implement an automatic price suggestion system for both their drivers and customers. The motivation being that currently a complicated and non-transparent process is used to request a driver. The goal is to improve the user experience and ease of use.

Currently users request a certain trip, wait until there are enough offers to make a good decision about what is a good price, and then accept an offer or cancel the trip. Drivers check out currently requested trips, make their offers, and wait until they get a message whether their offer is accepted. A use-case diagram can be found as Figure B.1 in Appendix B.

An additional problem is that a lot of trips are requested just to see how the system works or to get an indication of the price, without any intention to really go on the trip. By implementing an automated pricing system we seek to improve this situation. Originally the end goal was to provide a set price, so that users simply request a trip and wait until it is provided for (or not). Drivers would simply accept a job for a certain price or not. This would make the process much easier to use, but because of the feedback problem of set prices, and the fact that Sjauf wants their freelance drivers to choose their own price, this approach is not implemented during this project.

Instead, our system should allow a price range to be indicated before the trip is requested and consequently discourage users from requesting trips they don't intend to go on. Additionally, this price range should indicate what a good price is and as a consequence users will have to wait for less offers to know which offer is acceptable. On the driver side a generated suggested price acts as guidance for their offer to make offers fall within this price range.

Sjauf wants to suggest a price through the use of Artificial Intelligence (AI) or Machine Learning (ML) techniques with the restriction that even for very rarely taken trips the price suggestions should be reasonable.

There are two datasets available to train these machines on, an older database with more, but also less accurate data from the beginning of the company, and the new database with fewer but more accurate data entries. Regretfully, within these databases, incomplete, erroneous and test data is present. Both databases also have a different structure. Thus part of our project is pulling the appropriate data from these databases and filtering this data.

2.1.2. Problem analysis

With the current system Sjauf has noticed that only 35% of the requested rides by customers are fulfilled by a driver. They attribute this low percentage to the fact that currently the customer has no indication of what a trip should cost. This means that the customer has to come up with a valuation without having a price to compare with which can lead to a large disparity between what customers want to pay and for how much the drivers are willing to make the trip. With the implementation of the price recommendation system, Sjauf hopes that giving customers an idea of how much a trip will cost will help them decide if they want to post a trip in the first place. If they do choose to request a driver, they will have a price indication beforehand.

It is important to note that this problem is in fact a multi-stakeholder problem. The different stakeholders that can be distinguished are the customer, the driver and Sjauf. Generally speaking, each of these stakeholders has a different wish for a predicted price. The customer wants to pay as little as possible, whereas the driver wants to earn as much as possible. On the other hand, Sjauf does not mind about how high the price is, but they want to make the biggest difference between the price that the client wants to pay, and the price that the driver wants to drive for, since this difference is the earning of Sjauf. One question that rises from this multi-stakeholder problem is which of the stakeholders will be favoured by the price suggestion algorithm. To answer this question it is important to note that the price suggestion will be based on trips that have been made in the past. This means that this suggestion takes both the needs of the client and the driver into account. However, the stakeholder that is favoured most by this suggestion is Sjauf. This is because they can use the suggested price to decide a price range, give the client the upper-bound of this price range as indication and give the driver the lower bound of this range as suggested price. This way, the difference between what the driver earns and the client pays is maximised, but everyone's needs are satisfied.

Another problem that needs to be addressed is the amount of data that is available for this project, as the available datasets are really small; they only contain around 3000 completed trips. In comparison, other (publicly) available transit datasets, such as the New York City Taxi Trip[1] dataset and the Taxi Service Trajectory ECML PKDD[2] dataset contain 1.1 billion and around 1.7 million completed trips respectively. Inherently, drawing conclusions from limited data will be hard as certain incorrect correlations in the data might be found, while actual correlations in data may not be deduced.

2.1.3. Project requirements

As the final product needs to be integrated into an already existing application, there are a certain set of requirements that have been posed in order to ensure that the ML program is compatible with the application. The application itself is an umbrella application where each sub-application has its own Docker[3] container making it very modular. This also means that the first requirement demanded of the product is that it can be run isolated in its own Docker container.

Due to the fact that the algorithm is intended to be used in a commercial setting, any bugs in the code could negatively impact the company, therefore all code that is written has to be tested sufficiently. This means that whereas the performance of a Neural Network cannot be confirmed with testing, it can be tested whether certain actions lead to crashes.

Furthermore, there were limitations on the languages that could be used in the program set by the CTO of Sjauf. These limitations were that the algorithm had to be callable through an Elixir API. This means that the program itself can still be written in any language as long as it can be called from an Elixir API. Furthermore, due to the required maintainability of the code the CTO did not want it to be written in Java.

[1] `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`
[2] `https://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge,+ECML+PKDD+2015`
[3] `https://www.docker.com/`

Another hard requirement is that the result of the algorithm needs to be returned as quickly as possible as the aim is to supply users in real time with suggested prices. In order to ensure that the user is not left waiting for a price we want the algorithm to return a prediction within 200ms.

Finally there are a set of loose requirements that have to be met in relation to the way in which the model is trained. Currently, only 70 transactions per week are requested. In order to incorporate this data into the model, the model has to be retrained. This leads us to the final requirement which is that the algorithm should be trainable on the available hardware within 8 hours (over night).

2.2. Preprocessing

Sjauf uses two different datasets to keep track of their past rides. As these datasets have different schemes, it is necessary to merge these sets when one wants to train on all the available data. Also, both datasets contain test entries as well as entries with missing data, which need to be filtered out before training the model. As not all data fields in the datasets are numeric (strings for example) and the input for the model can strictly be a vector of numeric values, it is necessary to transform these data fields. Finally some data fields might not be relevant for predicting a price and these should be removed to be able to train the model faster. To find the preprocessing method(s) that are best suitable for our problem, we discuss multiple relevant techniques and compare them on key characteristics. We start by discussing different data filtering techniques in Subsection 2.2.1 and discuss data integration techniques in Subsection 2.2.3. Subsection 2.2.4 concludes with multiple data transformation techniques.

2.2.1. Data Filtering

As described in the introduction of this section, it is necessary to filter out the dummy data that are present in the different datasets in order to train the model on real data. As this is as simple as ignoring the data entries posted by certain test accounts, we will not discuss techniques for doing this. However, to only train the model on valid data, it is also necessary to remove data entries with incorrect data fields. This can be done by using so called anomaly detection (also known as outlier detection), which relies on the assumption that incorrect data will result in a data entry non-similar to others. There exist several techniques for outlier detection, all with advantages and disadvantages (as identified in [8]), which will be discussed briefly in the following paragraphs.

Classification based detection

Classification based models (such as Neural or Bayesian Networks [4, 13]) are used to learn a function from a set of labelled data entries to classify such an entry into one of the labelled classes. These techniques assume that a classifier can be taught to distinguish between normal and anomalous classes in a given space. Advantages of these models are that they are able to perform multi-class classification and that determining whether an instance is an outlier or not can be done very fast. Disadvantages of this approach are that they rely on the availability of accurate labels for normal classes and that they do not return a score based on how anomalous an entry is. As Sjauf's data does not contain labels whether an entry is an outlier or not, and the fact that labelling the data by hand is unfeasible in the amount of time available for this project, we will not use this approach.

Nearest Neighbour based detection

Nearest Neighbour based detection techniques [22, 33] rely on the assumption that normal data entries occur close to other data entries, while the anomalous data entries are far from their closest neighbours. These models work either by using the distances between either the k-nearest neighbours as the anomaly score, or compute the relative density of each data entry to use as the anomaly score. Advantages of these models are that they are unsupervised, i.e. they don't need labelled data to work and that they are easily adaptable to other types of data, since only the distance function needs to be changed. The disadvantages of these models are that when the data has normal instances far from all their neighbours, it will be considered as an outlier. Furthermore, when an anomaly has close neighbours, this technique will fail to correctly classify this data entry. Also, testing a data entry is computationally intensive, as all nearest neighbours have to be computed. A last disadvantage is that the performance of these models is completely dependant on the used distance measure, which can be difficult to determine when the data is complex. As we do not care about the computational intensity of the algorithm, because filtering can be done offline, this approach is viable and will be tested for effectiveness.

Clustering based detection
Clustering based detection techniques try to group similar data entries into clusters. There are several different approaches to clustering based anomaly detection, which all rely on different assumptions [8]:

1. Normal data entries belong to a cluster in the data, while anomalies do not.
2. Normal data entries lie close to their closest cluster centroid, while anomalies lie far away from cluster centroids.
3. Normal data entries belong to large and dense clusters, while anomalies belong to small or sparse clusters.

A important difference between clustering based techniques and nearest neighbour techniques is that each instance is evaluated based on the cluster it belongs to, rather than the nearest neighbour it has. Advantages of clustering based techniques are similar to those of the nearest neighbour technique: they are unsupervised and are easily transferable to different data. In contrast to nearest neighbour techniques, testing whether an instance is an anomaly or not is fast, since the number of clusters to be evaluated is small. One disadvantage of this approach is that the performance of clustering-based techniques is highly dependent on how well the structure of clusters for normal data entries can be captured. Another disadvantage is that the main goal of clustering is not to detect anomalies, but to group similar data entries together. Therefore these approaches are not optimised for anomaly detection. An example of this are clustering techniques which require that every data point is assigned to a cluster. This approach would violate assumption 3 whenever a anomaly is assigned to a dense and large cluster. Lastly, clustering techniques are not effective when the anomalies form large and dense clusters themselves. Clustering techniques where outliers do not need to be assigned to clusters will be evaluated for effectiveness.

Statistical detection
Statistical anomaly detection uses the principle that an anomaly is a data point which is suspected to be irrelevant because it does not follow from the assumed stochastic model [3]. It relies on the assumption that normal data entries occur in high probability regions of a stochastic model, while anomalies only occur in low probability regions. Advantages of this approach are that when this assumption holds, a statistical justifiable solution for anomaly detection is provided. Also, this approach inherently results in a confidence value for whether an entry is an outlier or not, which can be used for decision making. Lastly, if the distribution estimation is robust to anomalies in the data, this approach can be used in an unsupervised manner. A major disadvantage of this approach is that these techniques assume that data is generated from a particular distribution. This does not hold often. For this reason we will not use statistical detection to test for outliers in our data.

Information Theoretic based detection
Information theoretic techniques use the information content of a dataset using different measures, such as *entropy* and *relative entropy*. These techniques rely on the assumption that anomalies in data result in irregularities in the information content of the dataset. Advantages of this approach are that these techniques operate in an unsupervised manner and that they do not make any assumptions about the statistical distribution for the data. A disadvantage is that the performance is highly dependent on the choice of information theoretic measures. Often, such measures can only detect anomalies when there is a large number of them present in the dataset. Another disadvantage is that with this technique it is hard to create an anomaly score for a certain test entry. Lastly, this technique is usually applied to sequences, which may be hard to obtain. As it is hard to divide our data into meaningful sequences, we will not use Information Theoretic based detection methods.

Spectral based detection
Spectral techniques [27] try to find an estimation of the data with a combination of attributes that capture most of the variability of the data. They rely on the assumption that data can be mapped into a subspace with a smaller dimension where normal instances and anomalies have completely different appearances. An advantage of this approach is that it performs a dimensionality reduction, which is another preprocessing step we perform before feeding the data into the price generator. Another advantage is that these techniques can be used in an unsupervised manner. Disadvantages of these techniques are that they are only useful when a lower dimensional space exists such that anomalies look different than normal entries. Another problem

is that these techniques often have a high computational complexity. This is not a big problem as the outlier detection can be done offline, therefore this technique will be tested to see whether it is possible to reduce the dimensionality of the data in such a way that outliers can be detected.

Conclusion
In conclusion, for our problem it is not possible to use a classification based technique, as no labelled data is available and hand-labelling it ourselves is unfeasible in the amount of time available for this project. Also, it is not possible to use statistical based detection methods, as we highly doubt that a distribution exists from which our data is generated. Lastly, we will not be using information theoretic based detection, as this needs to be applied to sequences and our data is not sequential. Therefore, 3 methods remain: nearest neighbour- clustering- and spectral-based techniques. Problems might occur with nearest neighbour and clustering techniques, when outliers lie close to each other in the feature space. With spectral based techniques however, it might be impossible to reduce the dimensions of our data, as its number of dimensions is already rather small. Trials with these three techniques will have to be conducted to determine the optimal technique for outlier detection on our data. To determine this optimal technique the different pre-processing techniques will be used on the same, unbiased and non-random model and the error and variance of the results are compared with each other.

2.2.2. Missing data
Another case where data might become invalid is the case where data is missing. There are multiple methods to deal with this problem[30].

- Ignore the tuple
- Fill in the missing values manually
- Use a global constant to fill in the missing values
- Use attribute mean to fill in the missing values
- Use the attribute mean for all samples belonging to the same class as the missing tuple
- Use the most probable value to fill in the missing value

It will be dependent on the importance of the missing value which of the stated methods will be applied. If the attribute that is missing has a strong influence on the price, the best solution is ignoring the tuple since incorrectly guessing an important value will have a big influence on the results. On the other hand, just ignoring all the tuples with missing data would influence the accuracy of the predictions, as less data is available. Therefore, missing attributes that are of lesser importance should somehow be filled in, or these attributes should be removed from every entry. As the data from Sjauf does not contain many attributes already, dropping attributes in which missing values occur might be harmful to the accuracy of the model. However, manually filling in these values is very time consuming and is not feasible. Therefore it is necessary to find the most probable value to fill in for attributes of lesser importance. To do this, inference-based tools using Bayesian formalism or decision tree induction might be used[30].

2.2.3. Data Integration
Since Sjauf has two databases with data, the data has to be integrated. Data integration is the act of combining data from different sources to present it in a unified way. In this application this means that the two databases have to be combined into one dataset that can be presented to the price recommending algorithm. To make this combination possible, decisions have to be made on which columns need to be selected from both databases. Ideally the important columns are present in both the databases. If this is not the case, we will need to decide on the importance of said attributes and whether the column is indeed necessary. If such a column is necessary, the missing values have to be filled in (see also Subsection 2.2.2).

2.2.4. Data Transformation
After filtering the data and integrating the two databases, it is required to transform the data to an easily comparable format that we call feature vectors. In this application, the feature vector is a vector that represents one trip. However, since a vector consists of only numbers, all non-numeric attributes need to be encoded.

This might be challenging in cases where the attributes do not have clear numeric values, such as the destination of a trip, or the day of the week. For all of these cases, a suitable transformation of the data has to be found. A location might be represented using as a latitude and a longitude. For categorical data such as the day of the week, each category can be represented as an integer.

2.3. Generating recommendations

For the purpose of generating recommendations, there are two main classes of algorithms; Recommender Systems and Machine Learning algorithms. Recommender Systems have already found a great degree of success in the prediction of ratings people would give to a movie they had not yet seen[20]. On the other hand, Machine Learning algorithms such as Neural Networks can be used in a regressive capacity in order to generate predictions of continuous values.

2.3.1. Recommender Systems

In the realm of Recommender Systems there are many different types of algorithms that all essentially achieve the same goal of recommending a product. In this case, a price should be recommended for a trip. The different algorithms will be briefly touched upon in the following sections, starting with content based filtering, followed by collaborative filtering, and ending with an overview of a hybrid approach.

Content-Based Filtering

Content-Based filtering uses the attributes of an item to find similar items. It could then use the prices of similar items to decide a price for the current item. We have the advantage that we know the exact attributes (such as distance and driving time) of rides which can be easily compared in a meaningful way. Because our items are similar, but often not exactly the same, Content-Based filtering allows us to create groups of similar rides to reduce sparsity. There are advantages of new rides being able to be compared to any similar previous ride, and it is possible to have an explanation why this value has been decided.

An important disadvantage of this approach is that the quality is dependent on a large historical dataset[5]. Where normally there are challenges with creating and maintaining user profiles to make correct recommendations, we only want to make a suggestion based on a current item and do not take user preference into account; therefore we avoid this problem completely.

Collaborative Filtering

Collaborative filtering uses user responses to find out which items are similar to each other. There are two separate approaches, user-user filtering, or item-item filtering. The first technique tries to group users with similar ratings together, whereas the latter tries to group items with similar ratings together. Notably, this technique completely omits the content of items, and only looks at rating similarity. Because we have the requirement that prices are not user-dependent we can't use the user-user filtering that looks for similar users.

When using item-item filtering we could group together similar rides and compare different sets with similar prices to decide the price of the current ride. The problem with this approach is that it does not work when a ride is the first in an otherwise empty set, also known as the cold-start problem, or when no similar rides can be found in a non-empty set, which is known as the "Gray sheep" problem[5]. This approach also suffers from the problem that the quality is based on a large historical dataset. Also, because there is a limited amount of users that have taken multiple trips using this approach is ineffective as the price matrix in which similarities have to be found is very sparse.

Hybrid Filtering

Three classes of hybrid mechanisms can be distinguished, those that run both mechanisms and combine the results, those that combine both mechanisms into one and those that use the output of the first system as input in the second system.

The effectiveness of the first class is dependent on whether both mechanisms cover cases that are not covered by another, if one algorithm is very good at new items but the other at good at common items it might be worthwhile to run both. The second class is essentially making an algorithm that uses features of both algorithms with only a single mechanism but at the cost of e.g. speed and simplicity. The last class uses the result of one system in consideration with all other attributes to make a decision.

All these hybrid approaches are very flexible and dependent on the performance of the separate parts, thus experimentation is required to know if this works efficiently in our use-case. However as it is ineffective to create a collaborative filtering model (as described above) a hybrid approach will not be used.

2.3.2. Machine Learning

Contrary to how Recommender System predict a result using the similarities of other trips, Machine Learning (ML) takes a slightly different approach to predicting an appropriate price for a trip. In Machine Learning the individual characteristics that make up the data are analysed and used to come up with a prediction. The main methods which will be discussed are Neural Networks, Support vector machines and Ensembles.

In Machine Learning, two different types of outputs can be used. The first type of output for ML models is *classification*. In classification one tries to map an input value to a discrete value that indicates to which class said input belongs. An example in which classification is used is when one wants to determine whether a picture contains either a dog or a cat. The second type of output is called *regression*, where the model tries to map an input to a continuous value, where similar items receive similar values. An example use-case of regression is predicting the price of real-estate. As our goal is to predict prices for trips, no clear maximum price is present. Consequently, it is not convenient to regard this problem as a classification problem as this would mean that we have to define how 'big' each category will be. Decisions would have to be made regarding whether to favour the precision of the recommendation (a cent/euro per category) or to favour a smaller output dimensions (5/10 euros per category) for more accurate prediction. This problem is not present when regarding this as a regression problem, because regression does not require such categories. For this reason, we will use a regression model. Although most of the models in this chapter have both a classification and a regression approach, this chapter will only go into the regression approaches.

Neural Networks

Feedforward Neural Network Systems A Feedforward Neural Network is an algorithm inspired by the human brain. The algorithm consists of nodes that simulate neurons in the human brain [24]. These neurons are organised in layers in the network, see Figure 2.1. Every neuron in a layer is connected to all the neurons in the previous layer using weighted links. This means that every connection has a certain weight, or strength. The knowledge of the network is encoded in the strengths of the connections. The first layer of a Feedforward Neural Network is called the input layer. This is were the data enters the network. The data travels through every layer of the network and in every layer the input is multiplied by the weight, after which an activation function is applied to it. The last layer of the network is called the output layer, which returns the final result. Every layer that is in between the input- and output layer is called a hidden layer[15].

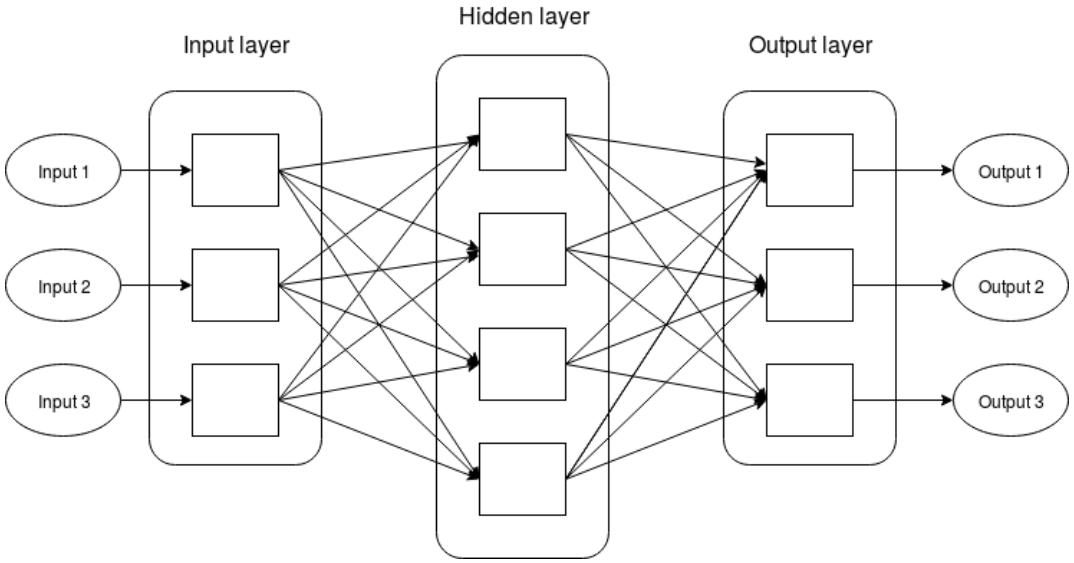


Figure 2.1: Example of a simple Feedforward Neural Network

Before the network can be used for price generation, the network needs training. During the training phase the weights in the network are updated. This training can happen in a supervised and unsupervised manner. When training in a supervised setting, every input has a known output. When training in an unsupervised setting, the network tries to find commonly occurring patterns in the input data, without knowing the correct outputs. Unsupervised learning is of no importance to us, as all data is labelled.

The steps taken in the training phase when using supervised learning are as follows:

1. Initialize all weights to a random value
2. Network predicts outputs on train data
3. Calculate error gradient
4. Update weights based on error gradient
5. Repeat steps 2, 3 & 4 until convergence or iteration limit is reached

After the training phase, the network can be used for price predictions and the weights can be saved for future use.

Working with Neural Networks has both advantages and disadvantages. First of all, given appropriate data, Neural Networks can usually find a trends invisible to data scientists. Secondly, the network finds relationships autonomously, limiting the need for human intervention. On the other hand, the architecture of a Neural Network can be challenging to get right, especially due to its black box nature which prevents data scientists from understanding which changes to the architecture could be beneficial. However, because of the known power of finding relations in data, the Feedforward Neural Network could yield significant results for this application.

Recurrent Neural Networks In many aspects, the Recurrent Neural Network (RNN) works similarly to a Feedforward Neural Network, however RNNs contain cycles between their hidden layers; the output of the previous time step is taken into account when predicting for the next time step. This allows RNNs to be able to predict time series as well as capture long and short term effects. This can be useful to capture changing user interests in for example movie or music recommendation. A common problem in RNN systems is the vanishing gradient problem, where the gradient used to correct the weights of the model during training becomes very small, preventing the weights from updating. Numerous solutions to this problem have been proposed, of which the most popular ones are Gated Recurrent Units (GRUs) [10] and Long Short Term Memory models (LSTMs) [17]. GRUs are simplified and memory-less versions of the LSTM and have been found to often yield similar accuracy as LSTMs, while they reduce training times [12]. Just like the Feedforward networks, RNNs predict an output value by learning a function from input to output. However, where the input in Feedforward networks is a single input vector, in RNNs a sequence of input items is fed into the network. This has as an advantage that the order of data is taken into account, which is not possible in Feedforward Neural Networks. Another advantage of RNNs over Feedforward NNs is that they are able to evolve over time. Disadvantages of RNNs are that they, just like Feedforward NNs, often require large amounts of data to be trained correctly, which also causes these models to have high training times. Also, the inner workings of such a model are hardly visualizable, which causes that it is hard to understand which factors are highly influencing the output of the model. Lastly, as an RNN uses a more complex structure than normal Feedforward models, they contain more free parameters which leads to a harder and longer optimisation of the model. While it might be worth trying the Recurrent Neural Network, we believe that since Sjauf's data does not contain sequences, there won't be much of an improvement over the Feedforward Neural Networks to compensate for the increased complexity.

Support Vector Regression
Like Neural Networks, Support Vector Machines (SVMs) work by dividing an n-dimensional sample space by a hyper plane. However, unlike Neural Networks that try to find any hyper-plane that satisfies the label assigned to most of the datasets, a SVM will find a hyper-plane that separates the sample space with the maximum spacing between each separate cluster[18]. A SVM can be used in a classification sense or regressions sense just like Neural Networks; if a SVM is used for the purpose of regression it is called a Support Vector Regressor.

The main advantage of a SVM over a Neural Network for this application is that if data is scarce it will find the most optimal solution for the general case, making the predictions more accurate for a wider variety of predictions compared to a Neural Network, which needs a lot more data. However this advantage also comes at a cost as if there is an outlier in the data that lies in the wrong cluster, a SVM will never converge as it needs all of the values to be separable. This means that if we want to use a SVM in the product, the data has to be thoroughly scanned for any outliers. In the case that there is no simple hyper-plane that bisects the sample space, a mapping function Γ can be used in order to transform the data into a more easily separable dataset without loosing the characteristics of the dataset[26].

In the past SVRs have been successfully used in areas such as the prediction of house prices in the housing market as well as predicting future stock prices in the stock market[25, 28]. Because Sjauf also needs price-predictions we will definitely try to implement a SVR.

Ensemble learning
As has been seen previously, many different algorithms for price prediction are possible. Each of these methods have their own strengths and benefits. In the world of Machine Learning, Ensemble learning can be used to combine the strengths of different models; it is comparable to consulting different experts when trying to solve a problem. The way Ensemble learning works is that it consults different Machine Learning algorithms which all have their own speciality, and through the combination of these algorithm the ensemble (a combination of algorithms) comes up with a more precise prediction than it would have, given that only one of the algorithms was consulted[6].

Conclusion
There are many techniques possible to generate a price, as described above. From all discussed approaches, we will not be considering User-User Collaborative filtering, because of the requirement that price recommendations are independent between users. Item-Item Collaborative filtering will not be applied because there is a limited amount of users making multiple rides, resulting in a sparse interaction matrix. Recurrent Neural Networks will only be implemented when there is extra time, as we do not believe they will improve the accuracy enough to warrant the development time involved in creating one. Ensemble learning techniques may not be used depending on whether they can deliver a prediction within the allotted 200ms. All other methods will be considered and their outputs evaluated when deciding on the most optimal model for predictions.

2.4. Generating price
Since the price suggestion algorithm for Sjauf directly influences the amount of profit the company makes, it is important that the suggestions are correct and within the permissible range. It is hard to speculate about an upper bound on how high the suggested price can be, since a lot of variables might have influence on this price. We can however decide on a lower bound. We will call this lower bound the minimum price. After generating a price by using one of the above mentioned algorithms, we can check whether this price lies above the computed minimum price. The minimum price can be calculated by taking the time that the trip will take plus the time it will take for the driver to get back to the start point, and multiply it with some standard wage for the driver. These times can be calculated using the Google maps API. The standard wage for the driver will be determined by Sjauf. Therefore, for each trip it is possible to check if the recommendation by our system is below the minimum price and action can be taken accordingly.

2.5. Language Choice

When choosing a language for this project we took a few factors into account. In order to comply with the requirements set in Subsection 2.1.3 we were not able to use Java, and an efficient language would increase the chance of staying within the set time constraints.

On top of the requirements for the language we had from the project requirements, we also had our own requirements. These were that the language had to have a lot of support in the Machine Learning community as we would be relying on libraries in order to implement most of the different models, since the time frame of the project was too short to write everything ourselves. Furthermore, the language should be easy to use on different operating systems as far as downloading dependencies goes, as we already expect to be using quite a few libraries.

Keeping all these requirements in mind, we agreed on using Python, more specifically Python 2.7. This is because of the extensive Machine Learning library support, such as MLLib[4] and Tensorflow[5]. On top of this, Elixir has an easy to use way of calling code from Python by using the native Python library. Lastly, as Python is an interpreted language, it allows us to use an interpreter like CPython that converts the Python code to C. This gives us the rapid execution time of programming in C with only a small amount of overhead whilst developing. During deployment, we can use the compiled C code for faster train and prediction times.

2.5.1. Interface

In order to comply with the final requirement set out by the client, our code needs to be callable from an Elixir environment. Python is also a good choice to fulfil this requirement as Elixir natively contains the Erlport[6] library, which allows the calling of Python modules. We will create an Elixir interface, making it possible to call Python functionality from the Elixir environment used by the client.

2.6. Conclusion

Analysis of the problem showed that Sjauf wants a price suggestion system build in to their service to improve the rate of fulfilled requested rides and to inform the customer beforehand on the cost of a trip. To make this possible multiple requirements have been noted and different approaches to deal with this problem have been investigated.

There are multiple steps to be taken when preprocessing the data into a usable dataset. First the data has to be filtered, which includes the detection and removal of outliers and filling in missing data. Secondly, the data from the different databases needs to be merged into a uniform scheme. Third, the data needs to be transformed to feature vectors. Lastly, the dimensions need to be reduced as much as possible, without losing critical information.

To be able to generate a price for a requested trip it is necessary to predict prices on past data. This can be done with multiple different models, such as Content-Based filtering and several Machine Learning approaches. After these models have predicted a price for the requested trip, it is needed to verify whether the models predicted a realistic price. Therefore, a minimum pricing component will be added to ensure a fair wage for the driver. Once this has been ensured, the recommendation will be accessible through the created API and can be presented to the drivers and customers.

There may be a future problem when using these methods, because they are incompatible with the planned second phase envisioned by Sjauf. If an exact price is generated and used then all new data is generated by the model itself. Therefore, it would be necessary to change the type of feedback that the model is trained on (from explicit to implicit), to prevent the model from reinforcing itself. This would mean that the created model needs to be adjusted to continuously improve itself, instead of converging to a set of prices and never change again. Another option would be to stop training the model and just use it as is. This would however require an alternative solution in the long run.

[4] https://spark.apache.org/mllib/
[5] https://www.tensorflow.org/api_docs/python/
[6] http://erlport.org/, Elixir is ran in an Erlang VM

3

Architecture

The main goal of this project was to create a module that can be used with Sjauf's application. As a consequence of our language choice it was decided to create a standalone API in Python which would be called by an interface written in Elixir, the language used by Sjauf's back-end. This chapter is dedicated to illustrating the setup of the project and the architecture of the final product. It elaborates on the relations between various components of the Python module. In section 3.1 the layout of the standalone API is explained together with the interface that enforces consistency. To ensure that Sjauf can use this API, an Elixir interface is created, which is explained in Section 3.2.

3.1. Interface for models

To predict the prices for trips many models have been created and tested. To ensure that these different models are consistent with each other, and to make sure that these models can be trained, optimised and used in a unified manner an interface was created. This interface enforces each model to implement the same methods. These methods are the following:

train(features, targets) Trains the model using features of the training data with their corresponding targets

generate_price(features) Applies the trained model to the features of a new trip to predict its price

export_model(location) Saves the trained model to the specified location

import_model(location) Reads the trained model from the specified location

random_initialization() Used in the genetic algorithm to create random instances with random variables of the model

crossover(dad_genome) Used in the genetic algorithm to crossover the parameters two models

mutate() Used in the genetic algorithm to mutate the parameters of a model

get_params() Returns all the parameters from a specific model

set_params(params) Sets the parameters for a model

It is important to note that the *random_initialization*, *crossover* and *mutate* methods in the model interface are used for the genetic algorithm. Every model has many parameters that decide the quality of the result of the model. To find the optimal combinations of the parameters, a genetic algorithm has been implemented that can be applied on every model. This genetic algorithm works by generating random instances of a model with random parameters and training them. Using crossovers and mutations of the best parameters, an optimal combination of parameters can be found. The genetic algorithm will be further explained and discussed in Chapter 5. Figure 3.1 shows how the model interface is implemented in the price suggestion algorithm.

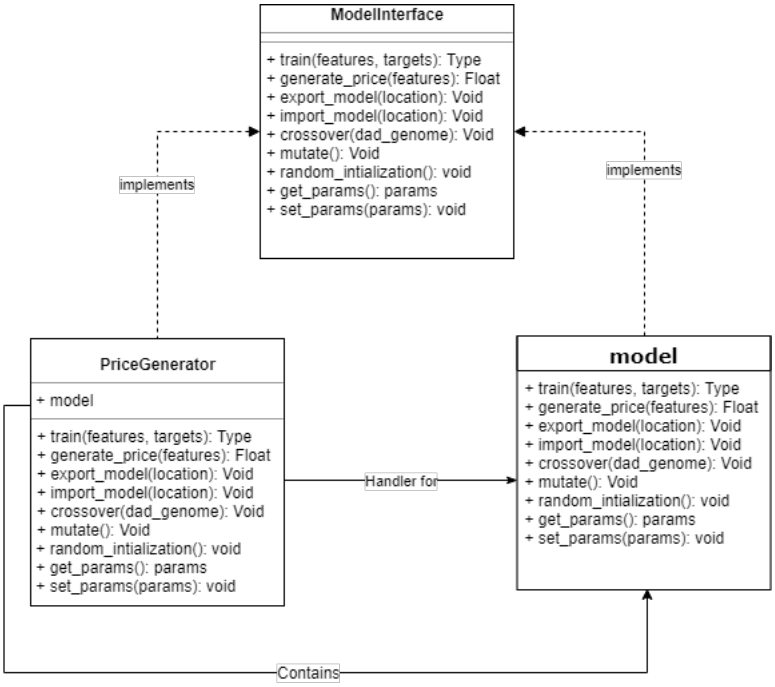


Figure 3.1: A representation of the Classes involved with the models

The *ModelInterface* enforces the methods described previously. The interface is implemented by both the *PriceGenerator* and the *model*. The *PriceGenerator* is a class that is able to generate the price for every model. Since many models are implemented and tested, this *PriceGenerator* only requires a model as input and is able to handle the training, testing, optimisation and verification of the model.

3.2. Elixir Interface

Sjauf's back-end is written in Elixir[1], which is a functional programming language based on the Erlang Virtual machine BEAM. As Erlang, or more specifically Elixir, is a functional programming language, it is not intended to be used for Machine Learning applications, and so there are few to no Machine Learning libraries available. To fix this gap between required functionality and what Elixir can offer, a bridge was built that let an Elixir program call functions from the Python scripts that were used to build the Machine Learning models. This bridge is managed by the supervisor (Pricing.Api.Application).
This supervisor manages both the interface (Pricing.Api.Interface) and the Python instance. The interface is managed through the standard supervisor implementation of Elixir[2], this means that on exceptions or crashes the supervisor terminates the failed instance and relaunches another instance to take its place. The ErlPort library used to connect Python and Elixir is not compatible with the native supervisor method. Instead, the Python instance is separately launched in the start function, and a monitor ensures the Python instance is relaunched if it crashes. This setup is shown in Figure 3.2. The process ID of the Python process is stored separately in *python_process_name* for easy access from both supervisor and interface.

[1] https://elixir-lang.org/
[2] https://hexdocs.pm/elixir/Supervisor.html

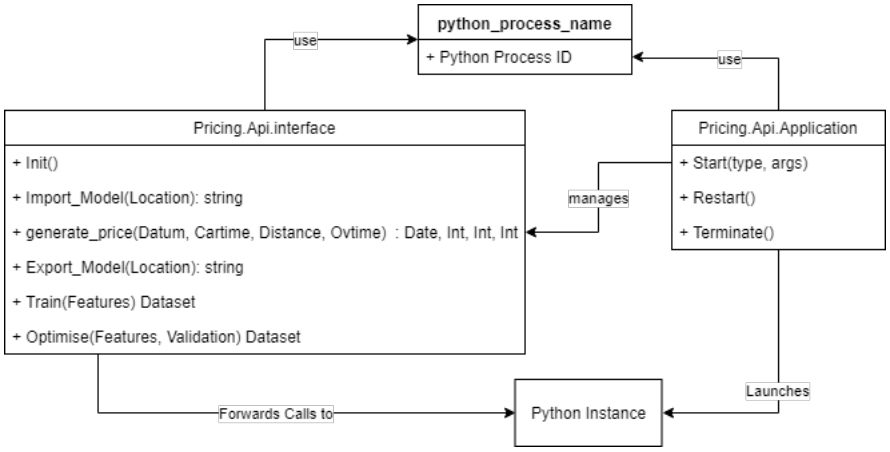


Figure 3.2: Relations of the Elixir-Python Bridge

The main part of the bridge is the interface as seen in Figure 3.3, this forwards the calls made to the Python instance and transforms some of the Elixir data formats into an usable format for Python. The Python instance is reachable through its process ID, which is available through an agent which spans both supervisor and interface.

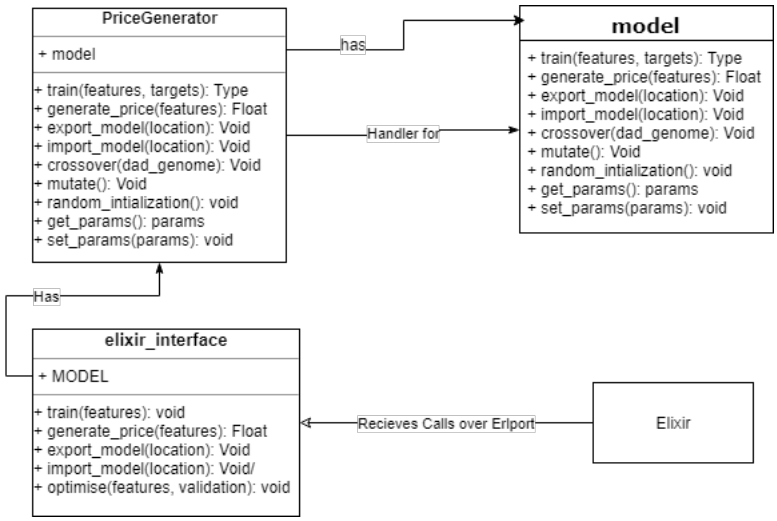


Figure 3.3: The structure of the Elixir Interface

Should the client decide on a different implementation in the future, all that would need to be changed is the bridge that can call the functions from the Python module.

4

Data Processing

The focus of this chapter is the preprocessing of the raw data received from the database. Section 4.1 discusses how the trips are reconstructed from the received data. The following section (Section 4.2) shows how the data is transformed into an usable format for Machine Learning. Section 4.3 explains how more information about the trips is gathered. In Section 4.4, useless data points are dealt with by removing them from the dataset. The final section, Section 4.5, describes how the dataset can be balanced so that every class is evenly represented.

4.1. Creating trips

As the data that we need for predicting a price is divided into two databases (an older and newer one), the first step to unify the data is to agree on a common database schema to which both databases will be transformed. Figure 4.1 shows the database schema that contains every column needed to generate good predictions. After the data from the two datasets has been merged, it has to be transformed. As both the old and new databased stored offers instead of trips, all entries that were not accepted had to be removed in order to find the price of trips.

offer_id	price	status	trip_id	request_location_id	latitude	longitude	street	house	zip	city	country	requested_trip_date
1657	55.0	accepted	950	1967	51.9353964	4.4608784	Lisztrode	5	2717 EW	Zoetermeer	Netherlands	2017-05-17 11:13:00+02

Figure 4.1: A simplified example dataframe of unprocessed unified data from both databases after being merged together

As an offer is made on a trip, but the schema only contains one latitude and longitude, we have to group the data from the merged database on the 'offer_id' field and decide which latitude and longitude correspond to the start-, via- and end-location. To do this, we sort the grouped data on the 'request_locations_id' and assume that the start location has a lower value for this field than the end- and possible via-locations. As a maximum of two via-locations is allowed in the app from Sjauf, the created group size is never bigger than 4 (start-, two via- and an end-location), therefore to transform the offers into a trip we create a new database schema. This schema contains four latitude and 4 longitude fields and is visualised in Figure 4.2. The latitude and longitude values from the grouped offers are filled in accordingly. After this has been done, the other column fields from the grouped offers (which are the same for every trip) are copied to the created schema. The only exception is the status value, that says whether a trip is accepted or not. This field will first be transformed from a 't/f' character to a '0/1' integer value to create a numeric representation of this field. After this is done, the trips have been created from offers and can be used in consequent preprocessing steps.

offer id	offer price	status	request id	request loc_id	confirmed tripdate	requested tripdate	accepted	start latitude	start longitude	via1_lat	via1_long	via2_lat	via2_long	end latitude	end longitude
1657	55.0	1	23	1967	2017-05-17 11:13:00+02	2017-05-17 11:13:00+02	1	51.9353964	4.4608784	null	null	null	null	51.0752405	4.7980408

Figure 4.2: An example dataframe with a trip going from start to end

4.2. Transforming data

To be able to compare different trips with each other it is desirable to transform all information to numeric values. There are two main items from the trips that need transformation: the date and the time of the day. The date by itself does not say much in this case because the data from Sjauf is only from the past 18 months. However, the day of the week on which the trip will take place can say a lot. One could imagine that a trip during the weekend is priced differently than a trip during the week. Therefore, the dates are transformed to days of the week, represented by numbers (0 for Monday, 1 for Tuesday etc.). On the other hand, the time of the day can also say a lot about the price of the trip. A trip at midnight might be priced different from a trip around noon. Therefore it was decided to divide the day into time buckets. Another intuitive option would be to just transform the time of day into hours. However, the problem with this approach is that a difference is introduced between for instance 10PM and 11PM while these trips are both in the evening and around the same time. Therefore, bigger time buckets were chosen to capture these times as being the same. Currently time buckets of six hours are used, which means that each trip is assigned a value between 0 and 3. A 0 is assigned between 00:00 - 06:00, a 1 indicates a time between 06:00 - 12:00, and so on. The length of these time buckets is variable, which means that they can easily be changed, for instance to buckets of four hours. Figure 4.3 shows an example of how this transformation works.

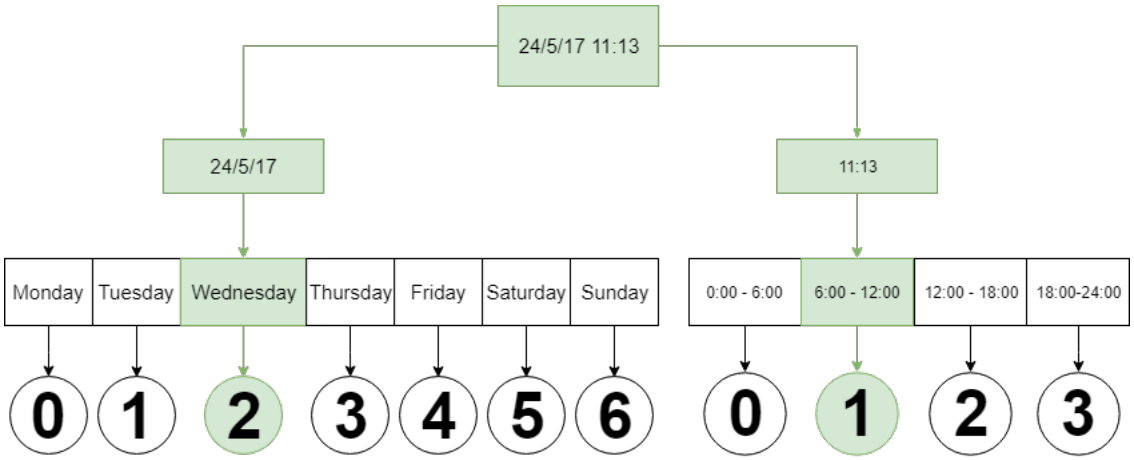


Figure 4.3: A high-level explanation of transforming the time

4.3. Creating more information

At this point the data is transformed to trips and completely represented by numeric values. This data could already be used to in Machine Learning, but in that case, a lot of useful information might be forgotten which is implicitly embedded in the data. The distance of a trip or the required travel time might for instance have influence on the trip. However, this information is not yet contained in the trip data. To create this data the Google Maps Distance Matrix API[1] was used. This API takes two coordinates and returns trip information in a JSON format. It also calculates the travel times for trips by car, as well by public transport. This is useful information because the public transport time might influence the price, since the driver will need to use public transport to return from the destination to their home. From this information the trip distance, travel time and the public transport travel time were extracted and included in the trip data.

Since the majority of the trips in the data are direct trips, the fields for the via-locations are often empty. As empty fields cannot be handled by the majority of the recommendation models, these empty fields need to be filled. To avoid introducing unnecessary inaccuracies by using dummy values, total distance and total travel times were used instead of intermediate destinations. An example data entry with new information is shown in Figure 4.4.

[1] `https://developers.google.com/maps/documentation/distance-matrix/intro`

offer id	offer price	status	request id	request locations id	confirmed tripdate	requested tripdate	accepted	start latitude	start longitude	end latitude	end longitude	cartime	distance	ovtime	day of week	time period
1657	55.0	1	23	1967	2017-05-17 11:13:00+02	2017-05-17 11:13:00+02	1	51.9353964	4.4608784	51.0752405	4.7980408	1342	2304	7653	2	1

Figure 4.4: An example dataframe after more information is gathered

4.4. Removing data points

To be able to train on clean data, anomalies and entries with missing values have to be removed in order to predict reasonable prices for trips. To do this, we first remove every trip which is not accepted as these trips could be cancelled for various reasons unknown to us (price too high/ride not needed anymore, etc.). From the remaining trips, we remove any entry which contains an empty value in one of its columns, as every of the remaining columns do contain critical values needed for prediction. Correcting this missing data by hand is not feasible in the amount of time available for this project. This part of the process is done before anything else, to reduce the load on following steps.

After the processes described in above paragraphs are finished, we combine the 'confirmed_trip_date' and 'requested_trip_date' fields, where the value of 'confirmed_trip_date' is used whenever this is available. This is done as most of the data entries do not have a 'confirmed_trip_date' value, however this value is important as it denotes the agreed upon departure time. We also remove the columns containing IDs used to construct trips, as these are no longer useful and should not be trained on.

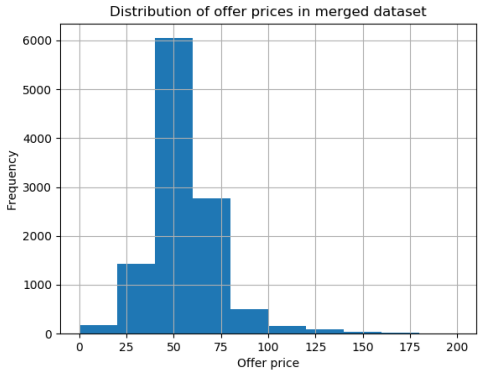
After this has been done, we use an Isolation Forest [23] to find anomalies. The Isolation Forest uses an ensemble of Isolation Trees, which isolate every data point in the dataset. Isolation Forests are based on the assumption that anomalies are far more likely to be isolated early. This means that the paths of anomalies to the root of the tree are shorter than the paths of normal entries, and therefore it is possible to detect anomalies based on the length of their average path. The anomalies that are found are removed from the dataset which results in the cleaned dataset that will be used during training. The fields present in the finalised dataset are described in Table 4.1.

confirmed_trip_date	The agreed upon departure time.
time_period	The time period of the day, set to 4 buckets of 6 hours each.
day_of_week	The day of the week as a number, starting with Monday as 0.
offer_price	The price of the trip.
cartime	The expected time taken to drive from start to end in seconds.
distance	The distance to drive from start to end in meters.
ovtime	The expected length of the return trip using public transport, in seconds.
start_latitude	The latitude of the origin location.
end_latitude	The latitude of the destination.
start_longitude	The longitude of the origin location.
end_longitude	The longitude of the destination.

Table 4.1: Description of columns present in the dataset after preprocessing

4.5. Balancing the dataset

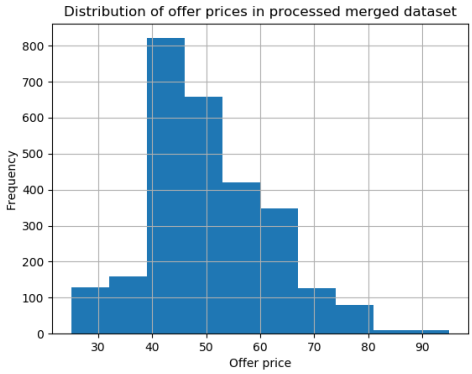
Two databases have been provided by the client, which are both processed to contain the fields described in Table 4.1. The first dataset, referred to as the 'old' dataset contains the trips up to December 12, 2017, while the other 'new' database contains the trips from December 13, 2017 up to April 24, 2018. Models are evaluated on both the new dataset, as well as on the 'merged' dataset which is a concatenation of the old and new dataset. Upon inspection of both of these datasets it becomes evident that the distribution of the prices for trips is heavily skewed towards €50,-. Figures 4.5a and 4.5b show the price distributions of the datasets before data is removed.



(a) Histogram showing the offer price distribution in the merged database

(b) Histogram showing the offer price distribution in the new database

Figure 4.5: Visualisation of the price distribution in the 'merged' and 'new' datasets



(a) Histogram showing the offer price distribution in the processed 'merged' database

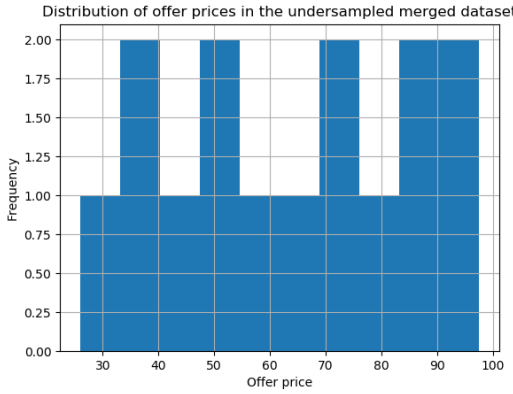
(b) Histogram showing the offer price distribution in the processed 'new' database

Figure 4.6: Visualisation of the price distribution in the processed 'merged' and 'new' datasets

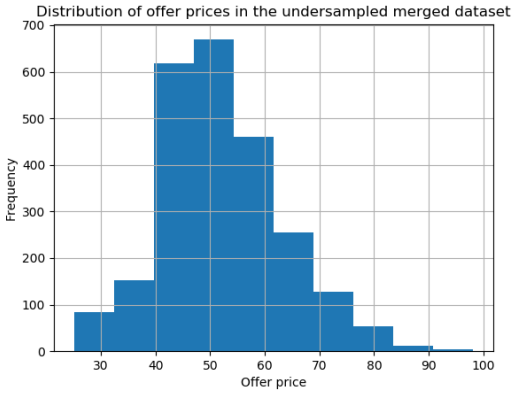
After filtering the data and using the IsolationForest to remove outliers from both the new and merged datasets, it would be ideal if the resulting datasets had been balanced by the filtering methods. However, as can be concluded from Figures 4.6a and 4.6b this is not the case, as the data is still centred around €50,-. To ensure that the models learn the patterns in the data rather than the dataset, it is beneficial to have an equal amount of entries for each price. Due to the distribution the data currently has, a model could perform decently by simple predicting €50 for each trip. In order to prevent this, under- or oversampling techniques can be used, which balance out under- or over represented prices.

4.5.1. Undersampling

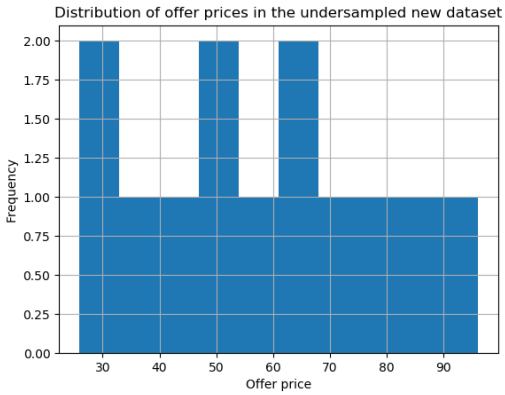
Undersampling techniques try to find the bias in the data and use this bias to remove entries of the majority class. By removing data points from over represented classes, an evenly distributed dataset is generated which contains fewer items than the original dataset. Examples of undersampling techniques are Cluster Centroids, which for each majority class removes a complete cluster except its centroid, and Tomek links [31] which removes all entries from the majority classes that have a nearest neighbour from another class. For our experiments with under- and oversampling we have used the imbalanced-learn Python library [21], from which we ran both the Cluster Centroids and Tomek-links undersampling methods. The resulting distribution of the datasets after applying these techniques can be found in Figure 4.7.



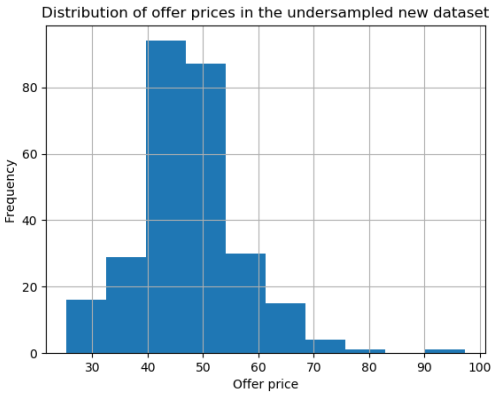
(a) Price distribution of the merged database after applying Cluster Centroids undersampling



(b) Price distribution of the merged database after applying Tomek-links undersampling



(c) Price distribution of the new database after applying Cluster Centroids undersampling



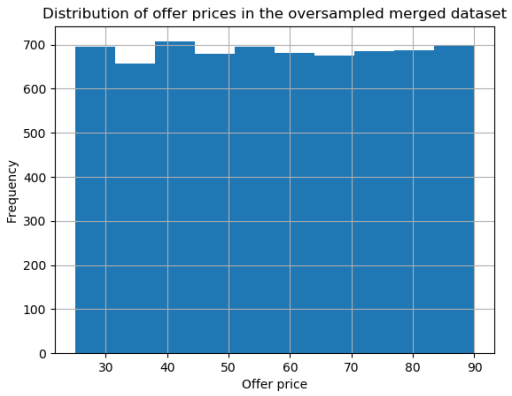
(d) Price distribution of the new database after applying Tomek-links undersampling

Figure 4.7: Visualisation of the price distribution of the datasets after applying undersampling techniques

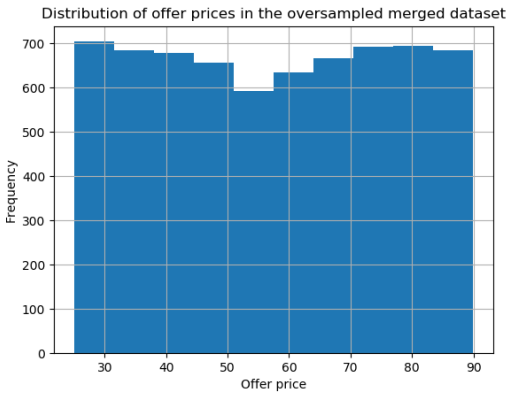
As becomes apparent from Figures 4.7b and 4.7d, there are no Tomek-links present in our datasets. Therefore, this technique does not remove any samples from majority classes and is consequently ineffective. The Cluster Centroids technique on the other hand, removes too many data points, resulting in a far too small dataset, as is depicted in Figures 4.7a and 4.7c. As both the Cluster Centroids and Tomek-links undersampling techniques do not result in the desired result, undersampling has not been applied to the dataset.

4.5.2. Oversampling

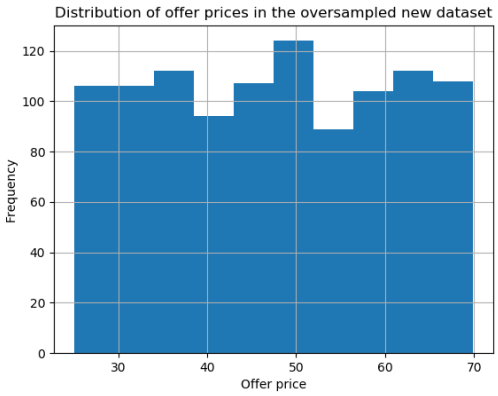
Oversampling techniques attempt to generate more data points for minority classes to balance the dataset. By creating new artificial data points for all minority classes an evenly distributed dataset is created with more entries than the original dataset. Popular oversampling techniques are Synthetic Minority Over-sampling Technique (SMOTE) [9] and Adaptive Synthetic Sampling (ADASYN) [16]. SMOTE creates new samples for a minority class by taking a sample from this class and find the k nearest neighbours of this sample. A new data point is then created by interpolating the values of these k nearest neighbours and multiplying this result with a random number (0,1]. ADASYN follows the same methodology as SMOTE, however additionally it uses a weighted distribution for minority classes according to their level of difficulty to learn. This results in more datasamples in minority classes that are hard to learn than in minority classes that are easier to learn. In our experiments we have used both techniques to create a new balanced dataset, whose results can be found in Figure 4.8



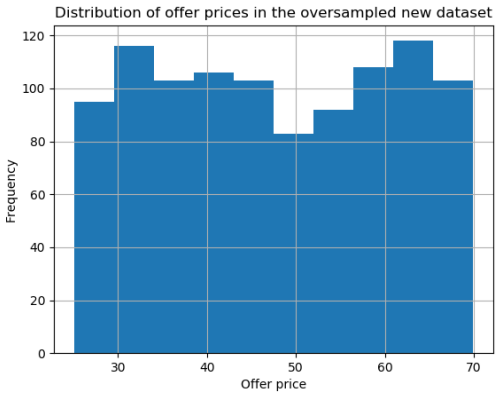
(a) Price distribution of the merged database after applying SMOTE oversampling



(b) Price distribution of the merged database after applying ADASYN oversampling



(c) Price distribution of the new database after applying SMOTE oversampling



(d) Price distribution of the new database after applying ADASYN oversampling

Figure 4.8: Visualisation of the price distribution of the datasets after applying oversampling techniques

Upon first inspection, these techniques seem to have created a balanced dataset in which every class contains roughly the same amount of elements. However, in reality these algorithms have created a dataset in which the elements from the minority classes are heavily duplicated. An explanation for this could be that there are too few distinct data points in the data's minority classes, which all lie closely to each other, causing the synthetic interpolated data points to lie closely to these points as well, forming almost exact duplicates. This causes the algorithm to learn a mapping of said duplicate values, rather then learning the effect each separate entry has on the output. As a result of this, oversampling has not been applied to the dataset, but may become viable in the future once more data has been gathered.

5

Genetic Algorithm

When working with Machine Learning models, especially in ensembles, there are many different variables that need to be optimised in order to reach a good result. This can vary from the type of kernel functions being used in the Support Vector Regressor, to the layout of the network in the different Neural Networks. In order to find an optimal solution, a naive method can be used such as a grid search, where many different parameters are tested and the best ones are selected. This can require a lot of time to run all combinations, as an improvement on this Genetic Algorithms can be used.

5.1. Inner workings

A genetic algorithm as the name suggests is an algorithm that is based on the idea that all things have a certain set of genes that determine what that thing will look like, similar to how it works in humans and animals. This idea was mimicked by Computer Scientists when they designed the Genetic algorithms [32].

A genetic algorithm consists of a number of phases. In the first phase a population of genomes is created, where each genome represents a set of values for the parameters of the model. During the second phase, all genomes are evaluated using a fitness function which defines how well a genome has performed. In the third phase a portion of genomes is selected for the creation of a new population, based on the value of the fitness function computed in the previous step. After this has been done, the selected genomes are combined with each other using a so called crossover function to form new genomes. In the final step, values of genomes can be mutated with a certain probability, to avoid getting stuck in local maxima/minima.

After all these steps have been taken, a new population is created and the aforementioned steps will repeat themselves until a certain exit criteria has been reached. This exit criteria can be a maximum amount of epochs in which it may evolve, convergence due to a too small increase of fitness over consecutive generations, or the fitness has reached a set threshold. As during every iteration only the best genomes are selected for crossover and mutation, this algorithm can be used to search a (semi)optimal value for the parameters without having to perform a full grid search, which would be far more time-consuming.

5.2. Implementation

The genetic algorithm is implemented as a static method which can be run on every model that adheres to the *ModelInterface*. As described above, the algorithm performs three key steps in order to optimise the variables of a model: creating an initial population, selecting the best performing genomes and use crossover and mutation functions to create a new population. The implementation of each step will be discussed below.

5.2.1. Creating initial population

Creating the initial population is done by another static method which randomly creates models and adds them to the population until a certain population size has been reached. However, as each model has different parameters and different possible values for these parameters, each model has to have a method to randomly initialise it. This ensures that the parameters of each model are set within an acceptable range.

The generic method will call this function and add the result to the population. The sequence diagram of this method can be found in Figure 5.1, where a population of the Content-based filtering model is created.

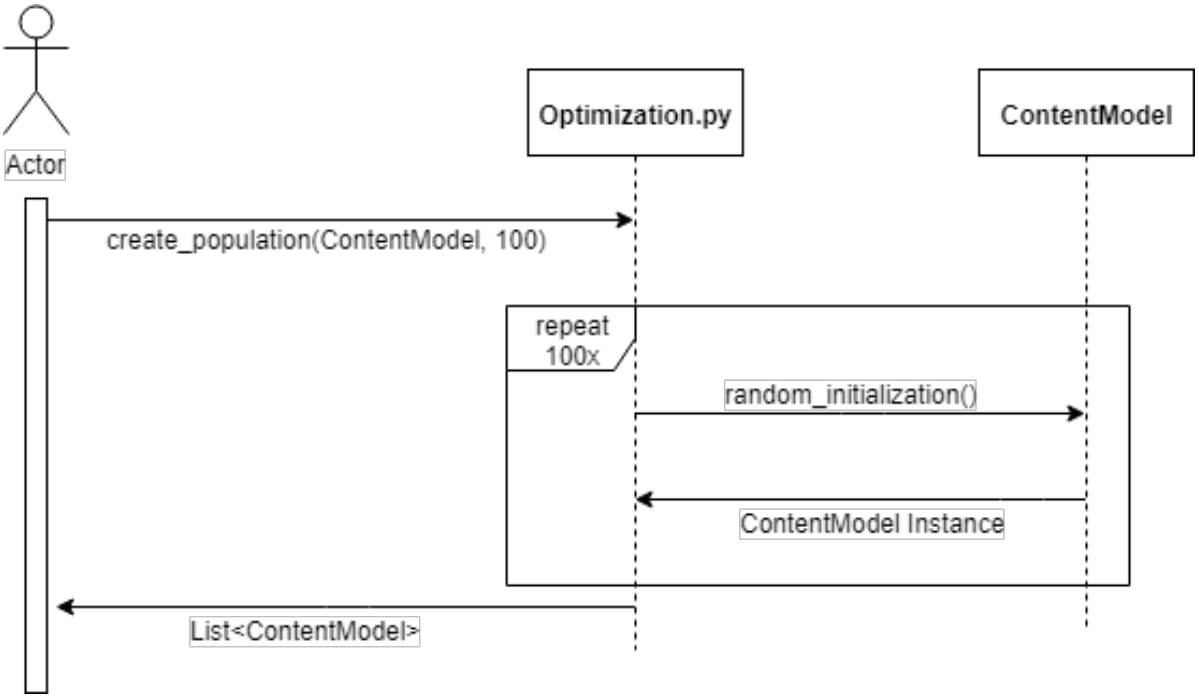


Figure 5.1: Example flow diagram of the creation of a population

5.2.2. Selecting fittest genomes

In order to approach an optimal solution, the genetic algorithm needs a way in which it can measure how well a model predicts the price of a trip. For this purpose there are many different fitness functions, we use the Root Mean Squared Error, which as the name hints takes the root of the average squared error. A deviation to either side of the price is cancelled out with the squaring (all values will be positive) and then the square root is taken in order to return to the same order of magnitude as which the errors are.

Once the top n genomes with the lowest RMSE have been selected, they can be passed along to the next function. This function uses these genomes to create a new population of better algorithms based on the top selection of the current population. The flow of this method is visualised in Figure 5.2.

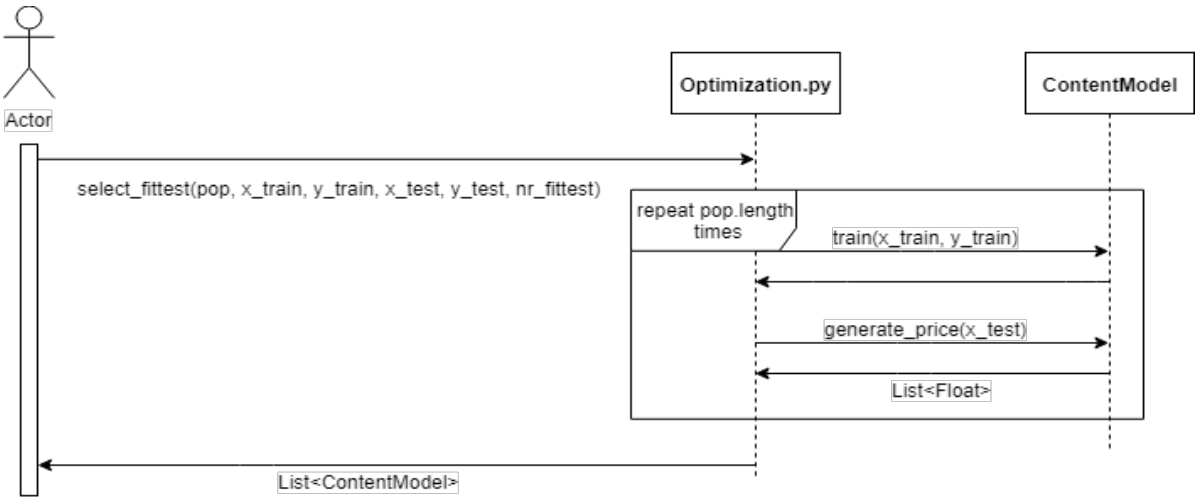


Figure 5.2: Example flow diagram of the selection of the fittest genomes from a population

5.2.3. Creating a new population

To repopulate the population it is required that new genomes can be created from the selected genomes. As every model uses different parameters and has different values for these parameters, a crossover and mutate method is defined in each model. Both of these methods return a new genome with modified values, which will be added to the new population. This is repeated until the maximum population size has been reached.

In the crossover phase of the new population 2 genomes are taken from the population that will play the role of mom and dad. From these 2 genomes, with an equal probability, a trait is taken either from the mom or the dad. This will create an offspring genome that can be used in the next population which will result in a more optimal solution over multiple generations[7]. The crossover of the genome is more clearly depicted in Figure 5.3.

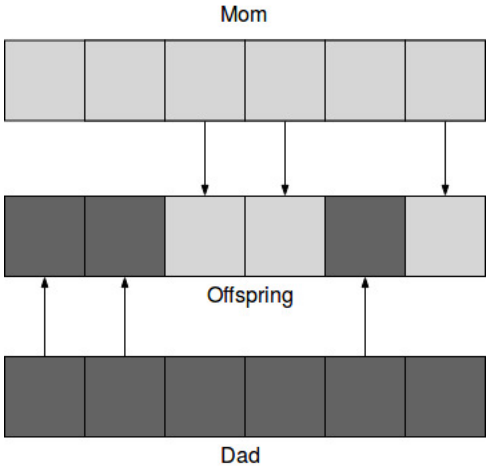


Figure 5.3: Graphical representation of the crossover of a genome

One of the problems that arises when using the genome of the mom and dad is that depending on the available options for the genome there may be some settings that are not represented at all in the population. Therefore, these settings will never be evaluated whilst searching for a solution. To fix this, the mutate function is introduced which for each feature in the genome has a certain low probability of randomly switching one of the features in the genome for a random other applicable one. This helps to make sure all possible settings get a chance to be represented as well as making it possible for the algorithm to escape from a local maxima or minima in which it would otherwise get stuck [7]. The mutation of genes is shown more clearly in Figure 5.4, while in Figure 5.5 the complete flow of creating a new population can be found.

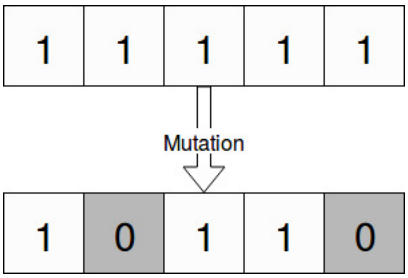


Figure 5.4: Graphical representation of the mutation of a genome. Mutated genes are highlighted

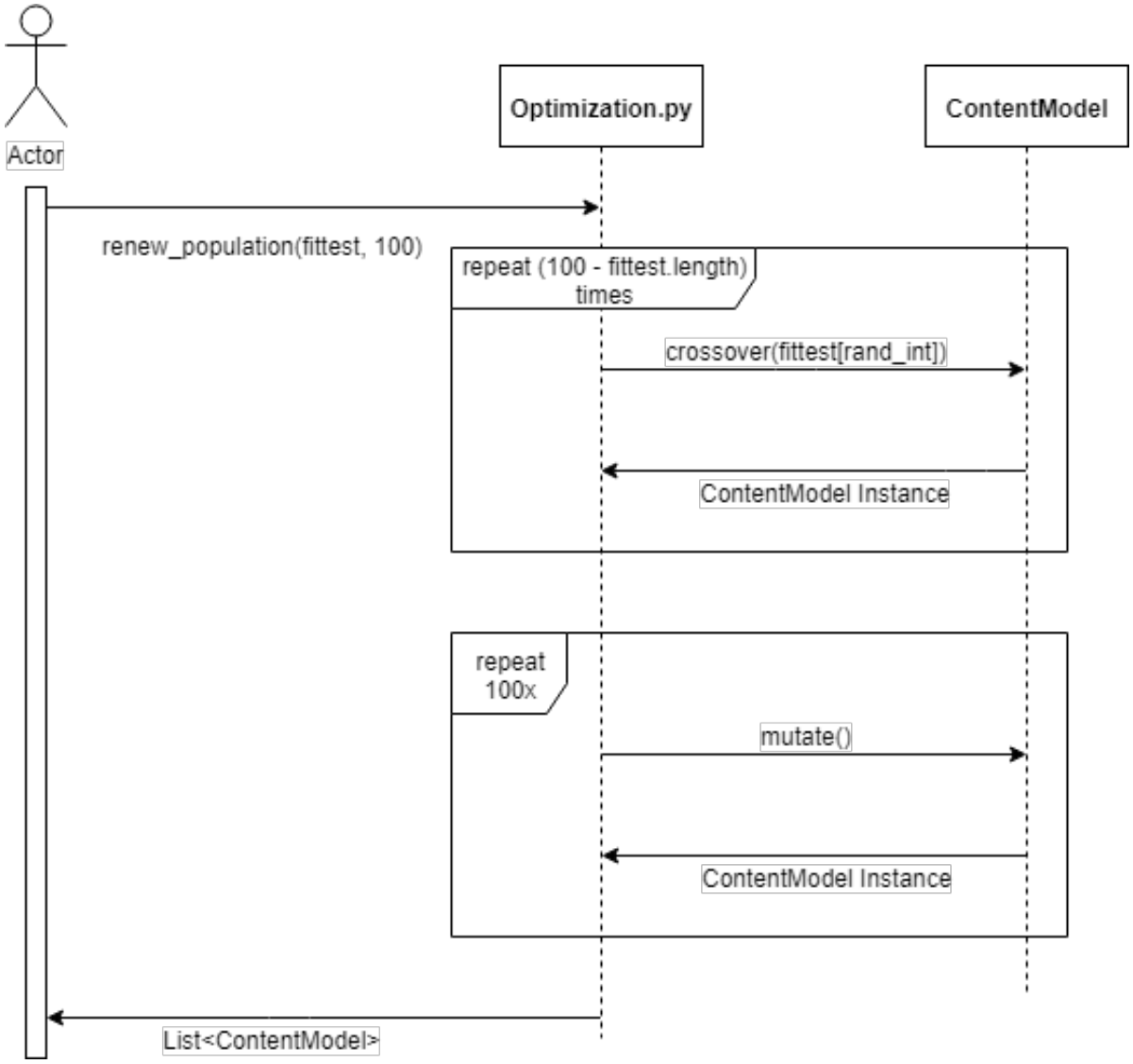


Figure 5.5: Example flow diagram of the renewal of a population

6

Price generation

In this chapter the most important part of the project is explored; the price generation for a new trip based on past trip data. Used models and their underlying architecture are discussed, as well as how the performance of each model is evaluated. Based on these results, a conclusion is drawn about which model will be used for price prediction. Section 6.1 looks at the created machines one by one and explain how they function. The following section (Section 6.2) describes how a comparison is made between the various models and how the results are used to select the best model. The final section, Section 6.3, focuses on the exact flow of how a new incoming trip is processed to generate a price for that trip.

6.1. Models

Various models were created over the course of this project. Additional information about the inner workings of the models are laid out in the following section. Additional changes that have been made to the Genetic Algorithm on a case by case basis are discussed when applicable.

6.1.1. Baseline

The first model that was implemented is a linear function. This linear function is a function of the drivers' travel time and a standard entry cost. This function will be used as baseline for every other model such that the predicted prices by the model is never lower than the baseline. This is implemented because Sjauf wants to ensure that the drivers are paid a minimum wage of €11.20 an hour. Sjauf defined the formula for the baseline as follows:

$$price = 15 + 11.20 * hours - min(15,\ 3 * hours) \tag{6.1}$$

This means that the baseline for the price of a trip equals to €11.20 per travel hour together with a flat rate of €15 which gets decreased by €3 for every travel hour. For example, a trip of 2 hours has a baseline of

$$15 + 11.20 * 2 - min(15,\ 3 * 2) = 15 + 22.40 - 6 = 31.40 \tag{6.2}$$

6.1.2. Content based

The Content based Recommender System predicts the price of new trips by looking at known trips with similar attributes. The attributes in this application are the distance, travel time, the day of the week and the time of the day. It would be very time consuming to have to calculate the similarity of attributes between the new trip and all the trips in the database. To reduce the amount of trips in the database that need comparing, only the trips that have approximately the same travel distance are checked. The similarities are calculated between the unknown trip and the database trips with about the same travel distance. After this, a weighted average of the prices of the database trips with the highest similarities is calculated and returned as prediction for the price of the new trip. The similarity between two vectors X and Y is calculated using the Bray-Curtis distance, and is defined as:

$$\text{similarity} = 1 - \frac{\sum |\ x_i - y_i\ |}{\sum |\ x_i + y_i\ |} \tag{6.3}$$

This results in a value [0, 1], where 1 denotes a complete similarity and 0 denotes a complete dissimilarity. This similarity metric is selected using empirical comparison using the same test dataset.

6.1.3. Feedforward Neural Network

The Feedforward Neural Network that has been created for this application consists of multiple layers with multiple neurons. To find an optimal architecture the genetic algorithm was used. The shape of the Feedforward Neural Network consists of a variable amount of layers. Each layer has a random activation function[1] and a random initializer[2]. The network also has a randomly generated loss function, batch size, optimizer and amount of epochs. The implementation is done using the Keras library[11], because Keras has the feature of creating layers as a whole. The optimal combination of all the variables is noted in Appendix C.

6.1.4. Recurrent Neural Network

The shape of the Recurrent Neural Network is different from the Feedforward Neural Network. The Recurrent Neural Network consists of a variable amount of the same GRUs (Gated Recurrent Units) with a random (recurrent) activation function, (recurrent) dropout, kernel initializer and recurrent initializer. These random layers are followed by a dropout which randomly sets a fraction of the input units to 0; this counters overfitting. The layer after the dropout is a normal dense layer with a random amount of nodes and a random initializer and activation functions. This dense layer is followed by another dropout to prevent overfitting before it goes into the output layer which outputs a suggested price. The loss function, optimizer, batch size and amount of epochs are random again. This Neural Network has also been implemented using the the Keras library[11].

6.1.5. Convolutional Neural Network

The Convolutional Neural Network is very similar to the Feedforward networks, but it typically uses different hidden layers between the input and output layer. Similarly it also does not allow loops in the network. Four types of layers are used in the Convolutional Neural Network:

- Convolutional Layers
- Activation Layers
- Pooling Layers
- Fully-Connected layers

A Convolutional layer generates a value by looking at the input from the previous layer and the neighbouring inputs. This is done by using a filter matrix which decides how many neighbours are used and with which weights they are used. This only effects the output corresponding with the original input and not the neighbouring values. The goal of this layer is to extract features from the original data. Because the Convolutional layer uses a linear function, activation layers are used to make sure the Neural Network is not simply a linear function. The Convolutional network in this project uses leaky rectified linear unit (leaky ReLU), which divides any negative value by ten.

Pooling layers reduce the size of the network by applying an operation over a group of values. An example of this is outputting only the max of every 3 consecutive inputs.

Fully-Connected layers work as described in the Feedforward network. These layers are the part of a Convolutional network that facilitate learning, and are necessary because the other layers are deterministic.

The genetic algorithm is used to find the optimal variables and shape for the model. The general shape is constant, but the exact amount of layers and what they do can vary greatly. The Convolutional in this project uses the mean absolute error as loss function, and the optimiser function is decided by the genetic algorithm.

[1] https://keras.io/activations/
[2] https://keras.io/initializers/

First the network has a generated amount of groups containing a variable amount of Conv2D layers. Each Conv2D layer in a group has the same variable filter matrices, but each group uses different matrices. Each group is followed by a leaky ReLU layer. After these layers follows a single pooling layer with variable size followed by a flatten layer to reduce the values to a one-dimensional array. Then there is a single Fully-Connected layer with variable amount of nodes followed by another leaky ReLU layer. Finally a value is decided with a Fully-Connected layer with size 1 and a linear activation function. The Keras library[11] was used for the implementation of this model.

6.1.6. Support vector regressor

The Support Vector Regressor (SVR) used in this project is simply the standard SVR provided by the sklearn library[3]. To optimise this model we use the Genetic Algorithm to find the best fitting kernel and size of the epsilon tube. This will ensure that we have a converging model with the highest fitness.

6.1.7. k-Nearest Neighbour model

For the *k*-Nearest Neighbour model the *KNeighborsRegressor* of the sklearn library is used[4]. This model has a few variables that could be optimised by the Genetic Algorithm. First of all the number of neighbours says a lot about how the algorithm performs. The algorithm used to compute the nearest neighbours and the initialisation of the weights have an influence on the results. Lastly the leaf size is an important variable that needs optimising. In Appendix C the optimised values for this model are given.

6.1.8. Ensemble model

Two variations of ensembles are used, a bagging ensemble and a stacking ensemble. The bagging ensemble simply takes the weighted average of the output of the models to determine a final price (further referred to as weighted ensemble). The stacking ensemble works by feeding the output of the models into a final Feedforward Neural Network. In order to optimise both of these ensembles, a Genetic Algorithm was used as described in Section 5. The models used in the ensembles are not changed by the ensemble, and should be optimised themselves prior to being added to the model list of an ensemble. Besides changing the list of used models, the weighted ensemble does also change the weights used on each models' result. The stacking ensemble mutates the Feedforward model used to combine the values during optimisation.

6.2. Comparisons

Some models might be superior to others with respect to price suggestions. The models described in Sections 2.3 and 6.1 have been implemented and optimised for the application of generating prices for Sjauf. To make sure the best models are chosen to predict the prices the different models have to be compared to each other. This process of comparing models is complex as each of these models work differently and use different parameters. To be able to compare these methods in a legitimate manner, unified metrics and validation methods have to be chosen that work for each of these methods. For validation the *k*-fold cross validation method is used and to compare the models the Root Mean Squared Error (RMSE), variance and R2-score metrics are used.

k-fold cross validation

To be able to compare different models it is required that the same validation method is used. A commonly used method for validation is called *k*-fold cross validation. This method works through splitting the available data into *k* randomly chosen different sets of the same size, also called folds. Whenever the data is split into, for example, 5 distinct sets, one would speak of 5-fold cross validation. After splitting the data into *k* folds, the same steps are taken. First, a fold is chosen as validation set, and the model is trained on the other *k*-1 folds. The result of this validation (expressed by some metric, for example an RMSE value) is saved and this process is repeated for all *k* folds. In the case of 5-fold cross validation, five results are obtained from every fold which are averaged to receive a final result for the model. Models that are evaluated on the same dataset can now be compared using this averaged result.

[3] http://scikit-learn.org/stable/modules/svm.html
[4] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

Metrics

Now that a validation method is chosen, metrics need to be chosen. For this application the RMSE, R2-score and variance were used as metrics as well as the average time taken per prediction. The RMSE, Root Mean Squared Error, gives the root of the average squared error of the test dataset between the predicted price and the real price. Let $\hat{y}$ be the predicted price, y the real price and n the size of the test set the RMSE is calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}} \tag{6.4}$$

Besides the RMSE, another frequently used metric in regression is the R2-score, also known as the coefficient of determination. This metric describes how well future samples are likely to be predicted correctly by the model. The function for this metric is defined as follows:

$$R2 = 1 - \frac{\sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{samples}-1} (y_i - \bar{y}_i)^2} \tag{6.5}$$

where $\bar{y}_i$ is defined as:

$$\bar{y}_i = \frac{\sum_{i=0}^{n_{samples}-1} y_i}{n_{samples}} \tag{6.6}$$

In the case that the model perfectly predicts the price of every trip, the R2-score would be 1. The score decreases as the accuracy decreases and can even get become negative.

The last metric in terms of prediction accuracy is the variance. The variance tells the consistency of the results. Two models might end up with the same mean error, but one could be consistently predicting €5 too high, while the other could predict correctly in most cases but sometimes predict €40 too high. The first case would have a low variance, while the latter case would have a high variance. In this application, it is desirable to have both a low error and a low variance, because predicting €40 too high or too low would be catastrophic for either the customer or Sjauf.

6.2.1. Comparisons

After optimising and testing each model and comparing them with the RMSE, variance and R2-score, the following table was created. To be able to compare the models on accuracy and execution speed, the time to train the model and the time to generate a price for a new trip is also included. The 'Train-Time' is computed on a dataset with 3120 entries and the 'Generate-Time' is calculated on 1 input. Both times are calculated on a machine with an Intel Core i5 Dual-core 7200U 2500 MHz with Intel HD Graphics 62.

Table 6.1: RMSE, variance and R2-score of the different models using k-fold validation. Bold values indicate the best value of the column. The best model is highlighted.

Model	**RMSE**	**Variance**	**R2-score**	**Train-Time**	**Generate-Time**
Baseline	16.5612	85.9120	-4.0639	0s	<1ms
Content-based	10.1382	102.3666	-1.2731	0.002s	688ms
Convolutional NN	8.1926	**62.6237**	**0.2386**	460s	3ms
Feedforward NN	**7.9461**	63.2648	0.2384	4s	4ms
Recurrent NN	8.7340	76.1419	-0.6782	82s	2ms
k-Nearest Neighbour	9.6783	92.8077	-1.6808	0.008s	140ms
Support Vector Regressor	8.1991	67.5085	0.1596	1.5s	6ms

Most importantly, what can can be seen from Table 6.1 is that the created models outperform the baseline model, which was put forward by Sjauf. Another thing that can be concluded from the table is that three out of six models that were created had more accurate predictions than the others. These three models are the Feedforward Neural Network, the Convolutional Neural Network and the Support Vector Regressor. Although the error is a little higher than both of the Neural Networks, the Support Vector Regression model has a RMSE, variance and R2-score that is better than the other models. A big advantage of the Support Vector Regression model is that the model has a significantly faster train time than the Neural Networks and the ensembles.

However, the allowed train time is 8 hours as stated in the requirements and every model has a train time that is within the 8 hours. Therefore, the train time is not a deciding factor, which eliminates the Support Vector Regressor as best overall model.

The Convolutional Neural Network and the Feedforward Neural Network both outperform all other individual models, and have a similar variance and R2-score. However, the Feedforward Neural Network has a significantly lower RMSE out of the two models. Due to this, we select the Feedforward Neural Network as the best individual model.

Table 6.2: RMSE, variance and R2-score of the different ensembles using k-fold validation. Bold values indicate the best value of the column. The best ensemble is highlighted.

Model	**RMSE**	**Variance**	**R2-score**	**Train-Time**	**Generate-Time**
Weighed Ensemble	**7.8950**	**62.6664**	**0.2731**	129s	10ms
Stacking Ensemble	8.2642	68.6632	-0.0081	249s	11ms

From Table 6.2 it can be seen that the weighted ensemble performs best. It has an RMSE and R2-score that is better than any of the individual models, and a variance that is almost as good as the best variance in both lists. This optimal weighted ensemble consists of two Feedforward Neural Networks and one Convolutional network. See Appendix C for the exact networks and weights of the optimal network. The fact that these networks are present in the ensembles is not surprising since these networks individually also performed well. Since this ensemble of networks conforms to the requirements of both train time and the time to generate a price, this is the ensemble that we recommend Sjauf implements in their app.

6.3. Generating price for new data

Now that the models have been evaluated the algorithms are ready for deployment. This means that the created algorithms can be used for price suggestion in the Sjauf app. The algorithm can be retrained regularly by using the train function. This means that the model is trained using the latest and updated data after which the model can be saved. When a trip is selected by the customer the trip information can be sent to the algorithm using the generate_price method (see Section 3.1). This method uses the features of the new trip and runs them through the saved model to predict a price for the new trip. This price can then be returned as a suggested price to the customer.

7 Ethics

When developing a software program, special care must be taken to ensure that the implementation follows ethical standards, especially in the case of Machine Learning. There have been many situations when even multi-billion dollar companies such as Google and Microsoft have made blunders when training Neural Networks meant for end users. These Neural Networks made mistakes ranging from misidentifying black people as gorillas[19] to being turned racist by malicious users[29]. In addition to wanting to avoid blatant racism, Sjauf also had some guidelines that they felt were important when recommending a price. In order to demonstrate that the program being delivered handles ethically to the best of our knowledge we will discuss the nature of the training data used to train the models, and which models are not used as they could have been in conflict with the ethical requirements set forward by Sjauf.

7.1. Nature of the data

In the aforementioned cases of Neural Network failures, the failures could be attributed to either the input data during runtime, or the data that the network is being trained on. For Tay the Twitter bot, Microsoft did not check what kind of data was being given to the network by the user, and thus some people manipulated the outcome. In the case of the misidentification by Google the image being used just happened to have the traits the network was checking for, which is a known flaw for all Neural Networks. For the network that was developed for this project these errors should not be exploitable as the input being given to the network is validated in the app, before being sent to the database.

Furthermore, the Neural Networks do not receive any data about individuals. Special attention was given to this problem, because Sjauf has noticed that non-Caucasian drivers get paid less by users. In order to comply with the wish of Sjauf that all users will be treated equally, any information about the users or drivers is not taken into account. This removes any chance that the network will start profiling users, and giving different prices depending on looks, or names. Even if using user or driver information could have lead to a better prediction, the risk of it going wrong, and the damage caused, far outweighed the benefit it could have had.

7.2. Rejected Predictors

In Machine Learning there are more than just Neural Networks that can be used to predict prices. Another class of models are Collaborative Filtering models. These models can be split into 2 groups, user-user and item-item Collaborative Filtering. Just like with Neural Networks these models also have the chance to discriminate between different users.

When using Collaborative Filtering models to get a price recommendation, we had to make a choice between using a Item-Item or User-User model. As the name already hints Item-Item collaborative filtering models only take the trips into account, predicting the price on the similarity to other trips. However, price recommendations can also be made by looking only at the type of users that is requesting a price recommendation without looking at the trip itself. This is what happens in a User-User collaborative filtering model. This obviously is not a smart choice to use when trying to generate a price that is the same for every user.

7.3. Fair wages

One of the main dangers with using Machine Learning to decide prices, and thus also wages of drivers, is that the model can optimise results by exploiting the drivers. Sjauf wanted to make sure that their drivers are always paid a minimum of €11,20 per hour. In order to ensure that this price is always offered, an extra check is done on the output of the model. If the price generated is lower than the hourly rate paid out to the driver, the baseline model is used which is based purely on the time taken to drive the trip. This ensures that in the worst case the drivers will be paid €11,20 per hour and in the best case they will make more.

Looking at the design decisions that have been made whilst developing the price prediction models, it should be clear that the guideline set forth by Sjauf regarding the equal treatment of customers, is adhered to, to the best of our ability. This has been done by omitting models that in this situation would have guaranteed differing treatment between different customers. It is however still possible for the Neural Networks to have implicit bias through the provided data.

8

Conclusion

This project was started with the goal of predicting prices for Sjauf's chauffeur service. This would allow customers to have an indication of the price of a trip without requiring the drivers to make a bid on trips. Machine Learning approaches are well-known for their capability to find correlations in multidimensional data without having to define a similarity function manually and are for this reason a good fit for this project.

After ten weeks of research and implementation, the system is fully developed and tested and is ready for implementation. All set requirements for this project have been met in the final product. The best performing models all adhere to the timing constraints on training (8 hours) and prediction times (200ms). They can therefore be deployed in a real-time environment. Moreover, a performance check has been built in, which assures that a predicted price for a trip is never lower than a set hourly wage, multiplied by the duration of said trip, to avoid advertising trips for a price that is too low. Additionally, all of the functionality can be accessed through a stand-alone Elixir API, which offers train, optimise and price generation capabilities. As required the whole project can be run isolated in a docker environment, The project requirements of using Machine Learning or Artificial Intelligence and testing sufficiently have also been fulfilled.

Performance-wise, all developed models greatly outperform the originally used baseline formula in terms of accuracy and serve well as a proof of concept. The weighted ensemble is currently the most accurate model. For the models to work at the best of their capabilities however, more data is needed, for example to balance the train dataset. Therefore, we expect the models to improve over time, as more data is added. The generic train and optimise methods allow the models to be trained on future data, to allow the models to adapt to possible price fluctuations over time. Changes in the data will result in different accuracy scores for all models and could cause a different model to be the most accurate.

In conclusion, we have developed a stand-alone future-proof API, which allows Sjauf to generate prices for requested trips. The API can easily be extended, by implementing all interface methods for a new model. The delivered product proves that generating prices for trips is possible using Machine Learning techniques, however we advise Sjauf to wait with a full implementation of the system until more data has been gathered to allow for higher model robustness.

9

Future work and Recommendations

A big danger when developing any kind of project is feature creep. This is when developers continually keep adding functionalities to a project, and thus never finish a product. As this project has a hard deadline some possible features and improvements have intentionally been left out as they would have taken too long to implement. In the following sections these improvements will be laid out, and their benefits explained.

9.1. Feedback for the models

The nature of Machine Learning models requires that some form of feedback is provided about the data that they are being trained on. This means that when using the current models only the offers that have been accepted can be used when training models. The consequence of this is that when a trip was accepted but has to be cancelled by the customer for reasons other than pricing, a data point is lost that could have been used for training. A solution to this is to ask the customer for a reason when they decide to cancel a trip.

Furthermore, using a different type of model, it may have been beneficial to use data about trips that were not accepted, to improve the accuracy of predictions. Improvements in Recommender Systems using negative feedback have been shown to be possible as Zhao et al. have shown in their paper on 'Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning'[34]. This can increase the amount of usable training points in the data greatly, as each requested trip only has one accepted price but can have many rejected prices.

9.2. Using the generated price as set price

In the future, it could be viable to return a single price instead of a price range. Currently, this has the downside that there is not a lot of data, meaning that the predictions are not accurate enough. When more data becomes available, accuracy can be improved greatly. Returning a single price however also means that the data regarding offer prices that are created by the model can not be used while training the models, as this would create a self-reinforcing loop.

9.3. Using locations for price generation

In the current input for the models the location of where the trip comes from has been omitted fully. This is partially due to the fact that this is a sure way to prevent discrimination against certain areas from an ethical point of view. However, another problem is that there are simply not enough trips in the dataset for the models to find an accurate correlation between the departure location and how it influences the price. Once more data has been collected from different locations the accuracy of price prediction can most likely be improved by checking the location of the trips.

9.4. Under/Over Sampling

A big problem in Machine Learning is that when certain entries are over or under represented in the dataset the model will become biased against them. This can be fixed as has been laid out earlier through the use of over- or under-sampling. However, as at this moment that would result in having either barely any data points left (in the case of undersampling) or having many of the exact same data points (in the case of oversampling). We suggest that the client should wait until there is more data in the database, especially with outlier trips before implementing over- or under-sampling.

9.5. General Recommendations using Machine Learning

Due to the fact that the client is not well versed in the use of Machine Learning, we deem it important to give some general advice about the use of data and retraining the models in the future.

Discarding old data

As the accuracy of the model depends on the spending habits of the people requesting the trip, caution has to be taken on the age of the data used when training the model. Over time the spending habits of people could change, although this does not happen from one day to the next there is a definite trend that has been observed in for example the relation between the expectation for inflation and consumption [14]. In order to maintain the accuracy of the model despite having stale data, at some point in time the older data has to be removed from the dataset. Unfortunately there is no guaranteed time period that decides how often this has to be done and Sjauf will need to use their own judgement for this. The most important aspect that they have to remember is that removing a lot of data will reduce the accuracy of the model because there is not enough data, but keeping too much old data will decrease the accuracy of predictions as the user's spending habits will have changed.

Garbage in, Garbage out

An important saying in the Machine Learning community is 'Garbage in garbage out'. this means that if the models are trained using bad data, then they will give bad predictions. The source of bad data can range from test entries that were entered into the database being used as training points by accident, to users inputting illogical values.

Currently there is a filter in place that removes as many 'bad' data points from the data as possible, however the definition of 'bad' data points will change over time. Due to this, we recommend to not put test entries in the production database, or at least remove the test entries before they enter the models.

Retraining and re-optimising the model

The more data that is available, the better the models can be trained. When enough data is collected over the lifetime of the company it is useful to retrain the models using this extra data. Especially when there are only small amounts of data available, adding new data will have a larger effect than in the future when there is a larger dataset available. Therefore, we recommend that the model is retrained more frequently at the start to be able to incorporate large effects quickly. Once more data is collected and used for training, the model can be trained less frequently. As re-optimising takes far longer than retraining the model, it is advised to do this less often than retraining, but still as often as possible when enough new data is gathered. Optimising is similarly dependent on the amount of available data.

9.5.1. Evaluation

When evaluating a Machine Learning model, it is important that the data on which the model is evaluated is not part of the training dataset. This is due to the fact that the models train to adapt as well as possible to the training data. So if one were to take the same data in the training data as validation data (to evaluate the model) it would not give a good representation of how well the model will work with predicting prices with data it has never seen before.

We recommend to the client to maintain 2 sets of data. One set of data contains the train data which has all data from the new database up to 3 weeks prior, and a secondary dataset, the validation dataset, that we use for evaluating the models that contains all the data from the last 3 weeks. This has the benefit that the evaluation gives a better idea of how well the models fit to the current spending habits of the customers, rather than taking random samples from the data.

Bibliography

[1] Zillow prize. URL https://www.zillow.com/promo/zillow-prize.

[2] Blue book for bulldozers, Jan 2013. URL https://www.kaggle.com/c/bluebook-for-bulldozers.

[3] Frank J Anscombe. Rejection of outliers. *Technometrics*, 2(2):123–146, 1960.

[4] L Douglas Baker, Thomas Hofmann, Andrew McCallum, and Yiming Yang. A hierarchical probabilistic model for novelty detection in text. In *Proceedings of International Conference on Machine Learning*, 1999.

[5] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, Nov 2002. ISSN 1573-1391. doi: 10.1023/A:1021240730564. URL https://doi.org/10.1023/A:1021240730564.

[6] Yunqian Ma Cha Zhang. *Ensemble Machine Learning*. Springer, 2012. ISBN 978-1-4419-9325-0.

[7] Lance Chambers, editor. *The practical handbook of Genetic Algorithms*. Chapman & Hall, 2001. ISBN 1-58488-2409-9.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[9] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[10] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[11] François Chollet et al. Keras. https://keras.io, 2015.

[12] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[13] Claudio De Stefano, Carlo Sansone, and Mario Vento. To reject or not to reject: that is the question-an answer in case of neural classifiers. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(1):84–94, 2000.

[14] Ioana A Duca, Geoff Kenny, and Andreas Reuter. Inflation expectations, consumption and the lower bound: Empirical evidence from a large micro panel. 2017.

[15] Laurene Fausett, editor. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994. ISBN 0-13-334186-0.

[16] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE, 2008.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Andreas Christmann Ingo Steinwart. *Support Vector Machines*. Springer, 2008. ISBN 9780387772424.

[19] Jana Kasperkevic. Google says sorry for racist auto-tag in photo app. Jul 2015.

[20] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81:1–10, 2009.

[21] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL `http://jmlr.org/papers/v18/16-365.html`.

[22] Yihua Liao and V Rao Vemuri. Use of k-nearest neighbor classifier for intrusion detection1. *Computers & security*, 21(5):439–448, 2002.

[23] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):3, 2012.

[24] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[25] Phayung Meesad and Risul Islam Rasel. Predicting stock market price using support vector regression. *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6, 2013.

[26] Rashmi Raghava M.N. Murty. *Support Vector Machines and Perceptrons*. Springer, 2016. ISBN 978-3-319-41062-3.

[27] Lucas Parra, Gustavo Deco, and Stefan Miesbach. Statistical independence and novelty detection with information preserving nonlinear maps. *Neural Computation*, 8(2):260–269, 1996.

[28] Abirami R. and Vijaya M.S. Stock price prediction using support vector regression. In P. Venkata Krishna, M. Rajasekhara Babu, and Ezendu Ariwa, editors, *Global Trends in Computing and Communication Systems*, pages 588–597, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29219-4.

[29] Hope Reese. Why microsoft's 'tay' ai bot went wrong, Mar 2016. URL `https://www.techrepublic.com/article/why-microsofts-tay-ai-bot-went-wrong/`.

[30] Winter School. Data Mining Techniques and Tools for Knowledge Discovery in Agricultural Datasets. pages 139–144, 2011. URL `http://iasri.res.in/ebook/win_school_aa/notes/Data_Preprocessing.pdf`.

[31] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on systems, Man, and Cybernetics*, (6):448–452, 1976.

[32] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[33] Ji Zhang and Hai Wang. Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. *Knowledge and information systems*, 10(3):333–355, 2006.

[34] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. Recommendations with negative feedback via pairwise deep reinforcement learning. *CoRR*, abs/1802.06501, 2018. URL `http://arxiv.org/abs/1802.06501`.

A

Process

This appendix describes the steps that are taken to guarantee a successful project. This has not been put in the main body of the report because it does not influence the end product directly.

A.1. Quality Control & Testing

To ensure we deliver high quality code, we used various techniques and processes. Most importantly we used peer reviews to check code before any merge, to ensure that the most obvious errors and oversights are removed before moving code to the main branch. We decided to require two reviews before a merge which occasionally led to complications because our team consists of four people. To maintain consistent style across all group members, Pylint[1] was used.

It is important to the client that the code we delivered conforms to their standard. To ensure this, multiple code reviews with the CTO of Sjauf have been realised. As a result of this, we have chosen to use continue statements instead of elif statements to improve the clarity of the code. Luckily there were no huge discrepancies between the way we wrote code and what Sjauf's preferred style is and thus we did not need to make drastic changes.

We have decided not to use continuous integration (CI) because many Machine Learning models run for a long time. Moreover, our unittests run the models to make sure the models function as intended. From experience we know that (free) CI provides a rather slow execution of the tests, which would result in an ineffective use of time.

After consulting Sjauf, we have chosen to aim for tests with a 80% line coverage and have mostly maintained this throughout the project. This did mean that we, at some points, did have to write additional tests to keep the percentage high, but overall it has provided a great indicator when things are changed in a unsuspected way.

A.2. Communication

In view of the fact that clear communication with the client is necessary for a good product we decided that we would be at Sjauf's office in Amsterdam every Monday and Friday so that we could start and end our sprint with the CTO. Barring the few times this was not possible or unnecessary this plan was executed. To communicate with Sjauf whilst working in Delft we were added to the Slack channel of the client. Moreover, we also used a WhatsApp group for more immediate communication with both Sjauf and our coach.

Communication with the coach was done through weekly meetings and by sending the notes of every meeting. Setting dates and scheduling was done with the secretary of our coach.

[1] https://www.pylint.org/

A.3. Scrum

As with any project, division of labour is an important part of the process. In order to do this in an efficient and clear way, the scrum methodology was used throughout the project. During the scrum process there always has to be a development team, a scrum leader and a client. In this case the client was always Stefan from Sjauf. However, in order for all group members to learn something about the Scrum method, Stefan wanted each member to lead at least one sprint, ensuring that everyone got experience in being a scrum leader.

B

Use-Case Diagram

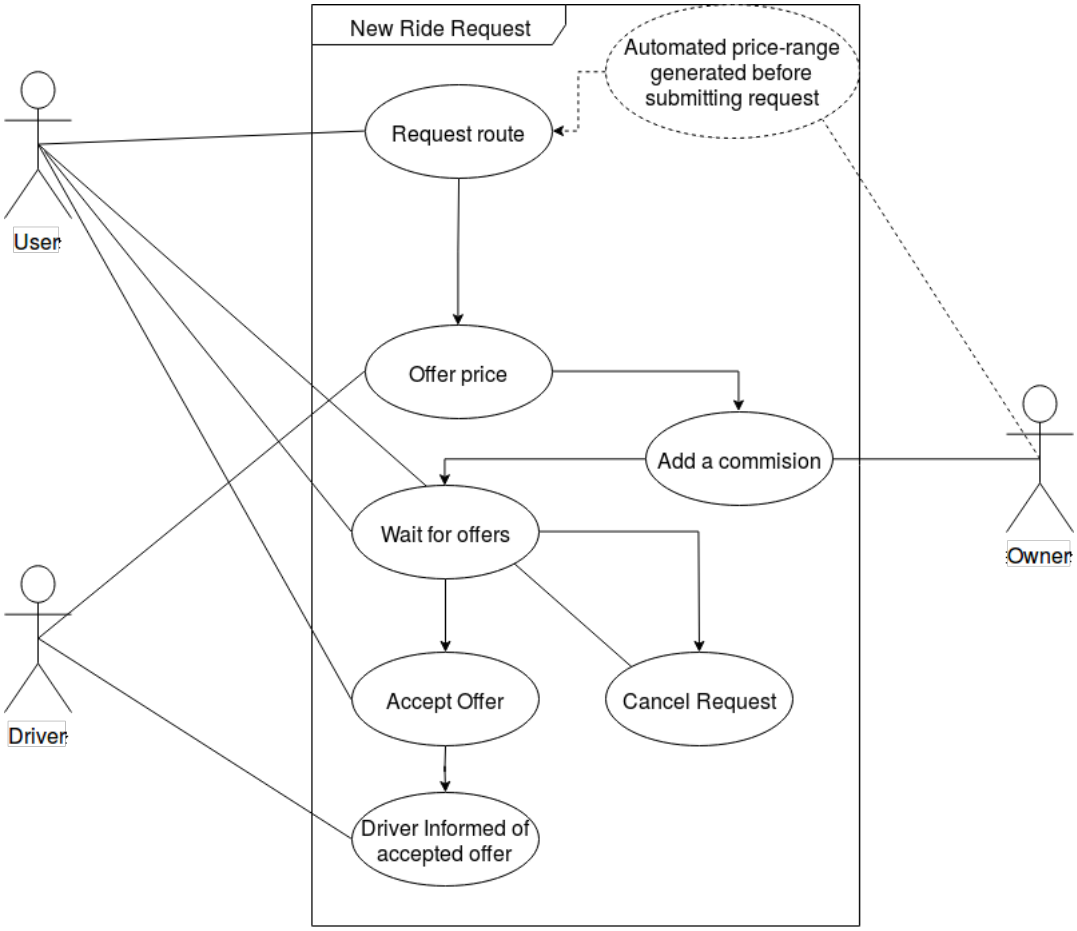


Figure B.1: Use-Case Diagram for handling of trip requests. The dotted elements indicate parts that are implemented by this project.

C

Optimal values

This appendix shows the best values found by the genetic algorithm for each model. In the Neural Networks, lists of values are given for some parameters. The position in the list refers to the layer in the model it applies to, i.e. if the 'nodes' parameter would have the value [64, 32, 2], this would indicate a model with layers of 64, 32 and 2 nodes respectively.

Content Based

distance:	0.16905940518063323
top_n:	23
return_val:	46

Feedforward Neural Network

optimizer:	'RMSprop'
batch_size:	580
activation_functions:	['linear', 'linear', 'linear']
initializers:	['glorot_uniform', 'glorot_uniform', 'VarianceScaling']
epochs:	322
loss:	'logcosh'
nodes:	[118, 55, 1]

Recurrent Neural Network

optimizer:	'Adamax'
epochs:	429
kernel_initializers:	['Zeros', 'RandomUniform', 'TruncatedNormal']
activations:	['elu', 'linear']
layer_nodes:	268
recurrent_nodes:	99
dropouts:	[0.761150252623648, 0.6820661847377243, 0.6433832619310824]
loss:	'mean_absolute_percentage_error'
batch_size:	1065
recurrent_activation:	'softsign'
recurrent_dropout:	0.9874462438136483
recurrent_initializer:	'lecun_normal'

Convolutional Neural Network

nr_filters:	[78, 108, 150, 237, 255]
batch_size:	128
loss:	'mean_absolute_error'
epochs:	240
optimizer:	'Adamax'
dense_size:	151
filter_size	[(6, 1), (4, 1), (3, 1), (1, 1), (6, 1)]

Support Vector Regressor

gamma:	434
C:	1
kernel:	'linear'
degree:	609
max_iter:	8953
shrinking:	1
epsilon:	0.001
tol:	5e-05

k-Nearest Neighbour model

algorithm:	'brute'
n_neighbor:	22
leaf_size:	202
initialization:	'distance'

Weighted Ensemble

model_pool:	List of the 5 best models by optimisation found per class
weights:	[0.2992895161850744, 0.4502198213144899, 0.25049066250043583]
models:	[Feedforward1, Convolutional, Feedforward2]

Feedforward Neural Network 1

optimizer:	'Adadelta'
batch_size:	451
activation_functions:	['softplus', 'linear']
initializers:	['RandomNormal', 'VarianceScaling']
epochs:	123
loss:	'logcosh'
nodes:	[376, 1]

Convolutional Neural Network

nr_filters:	[150, 164, 11, 194]
batch_size	186
loss	'mean_absolute_percentage_error'
epochs	: 244

optimizer	'Adam'
dense_size	580
filter_size	[(5,1), (4,1), (5,1), (1,1)]

Feedforward Neural Network 2

optimizer:	'Adam'
batch_size:	641
activation_functions:	['softplus', 'linear']
initializers:	['RandomNormal', 'VarianceScaling']
epochs:	110
loss:	'logcosh'
nodes:	[376, 1]

Stacking Ensemble

model_pool:	List of the 5 best models by optimisation found per class
models:	[Convolutional1, Feedforward, Convolutional2]
combiner_model:	Feedforward2

Convolutional Neural Network 1

nr_filters:	[150, 164, 11, 194]
batch_size	186
loss	'mean_absolute_percentage_error'
epochs	: 244
optimizer	'Adam'
dense_size	580
filter_size	[(5,1), (4,1), (5,1), (1,1)]

Feedforward Neural Network 1

optimizer:	'Adamax'
batch_size:	431
activation_functions:	['softplus', 'linear']
initializers:	['RandomNormal', 'lecun_normal']
epochs:	123
loss:	'logcosh'
nodes:	[376, 1]

Convolutional Neural Network 2

nr_filters:	[181, 93, 177, 16, 255]
batch_size	128
loss	'mean_absolute_error'
epochs	: 149
optimizer	'Adam'
dense_size	179
filter_size	[(1, 1), (3, 1), (6, 1), (4, 1), (6, 1)]

Feedforward Neural Network 2 (Combiner)

optimizer:	'Nadam'
batch_size:	326
activation_functions:	['tanh', 'linear']
initializers:	['TruncatedNormal', 'RandomNormal']
epochs:	493
loss:	'logcosh'
nodes:	[396, 1]

D

First SIG feedback

Feedback:
De code van het systeem scoort 3.7 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Interfacing en Unit Size vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden.

In jullie project heeft de constructor van support_vector_regressor.py een groot aantal parameters nodig, waarvan de meeste default-waardes hebben. Het is beter om dit soort constructies in een parameter-object onder te brengen, dat houdt zowel de constructor zelf als de aanroepen van de constructor beter leesbaar op het moment dat het aantal opties in de toekomst gaat toenemen.

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.

store_distance_matrix_api_results.py bevat nu drie verschillende strategieën, die allemaal in-line worden afgehandeld. Door daar aparte methodes van te maken hou je de code leesbaar, en het wordt zo ook makkelijker om unit tests te schrijven.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

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

Commentary: While we are happy with receiving the feedback from SIG, we also had some questions regarding the feedback. Most importantly we are fairly sure that the tests where included in our submission, so we have mailed for a double-check for this part.

On top of that we are not sure how we can reduce the amount of parameters without losing functionality or introducing unnecessary complexity and have asked for clarification. This is because moving the model's constructor parameters to a parameter object would require a separate parameter object for each model as each model requires completely different parameters. This would mean that the problem would be moved from the model's constructors to the parameter object constructors. Reading parameters from a file is no option either as it removes the ability to have models of the same type with different parameters in for example an ensemble. Lastly, setter methods have been considered, but found as bad practice as creating a model with set values and immediately overwriting these would not only lead to confusing model initialisation, but also to unnecessary overhead. Regarding the constructor parameters, these purposely have default values, which allows for easy creation of a model with the current best found parameter values.

We have split our in-line functions in store_distance_matrix_api_results.py as advised in the feedback. Regretfully we have not received a response to our email asking for further clarification. Because of this, no better option was provided for the other indicated problems. After a discussion with the Client about the best possible solution to this problem, we agreed that no further changes had to be made as we feel the best solution was already implemented.

E

Second SIG feedback

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.

In de eerste upload werden Unit Size en Unit Interfacing als verbeterpunten genoemd. We zien bij Unit Size een kleine verbetering, die echter weer teniet gedaan is omdat in de nieuwe code ook weer nieuwe lange methodes zijn geïntroduceerd (de constructor van RNN in ecurrent.py [*sic*] is hier een voorbeeld van). Bij Unit Interfacing zien we dat de genoemde voorbeelden zelfs erger zijn geworden. Dat is een logische ontwikkeling als je de onderhoudbaarheid van de code niet expliciet in de gaten houdt. In vergelijking met andere Python-systemen hebben jullie wel erg veel parameters nodig, en dat probleem zal ook steeds erger worden als er steeds meer functionaliteit aan het systeem tegevoegd wordt.

Zoals jullie per email hebben aangegeven hadden jullie in de eerste upload wel degelijk testcode, dus die opmerking uit de feedback op de eerste upload komt te vervallen. Qua testcode is het goed dat jullie naast nieuwe productiecode ook nieuwe tests hebben geschreven, maar de hoeveelheid nieuwe tests is wel een stuk lager dan bij de eerste upload. Probeer ook hier scherp op te zijn, zelfs als de deadline van het project in zicht komt.

Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback op de eerste upload deels zijn meegenomen tijdens het ontwikkeltraject.

Commentary: We disagree with using the amount of tests as a measurement of how good we tested, the test coverage is far more important, and we have kept the test coverage far above 80%. Therefore we feel that the complaint about the amount of tests is invalid.

The complaint regarding Unit Interfacing has been evaluated as described in Appendix D and has, because of a lack of better alternatives, not been changed. Furthermore, our client has looked at the code, and agrees that the amount of parameters has been kept to a minimum. Unfortunately however, this many parameters are required by the various Machine Learning libraries. We also do not see a clear connection between functionality and the amount of parameters.

We agree that the Unit size is sometimes on the long side. We assume this is mostly aimed at constructors of various models. In these constructors, we simply set a large amount of parameters which are, again, required by the various Machine Learning libraries. An example of this is our Convolutional Neural Network, which is implemented using the Convolutional Neural Network from the Keras library [1], in which 16 arguments can be passed to construct it. To be able to mutate these variables, we need to store them in the object, hence a big constructor is needed.

[1] https://github.com/keras-team/keras/blob/master/keras/layers/convolutional.py

F

Standalone platform for automated price suggestions using AI

Name of client: Sjauf
Date of final presentation: 5 Juli 2018

Sjauf is a start-up which allows customers to book a driver for their own car. Currently, ordering a trip with Sjauf is done without prior indication of the cost of the trip; drivers bid on incoming offers which can then be accepted by the customers. To remedy this, Sjauf wishes to have an automated price recommendation algorithm that can indicate a price for both the customer and driver. To be able to adjust to the changing spending habits of users over time, a Machine Learning algorithm should be used. The challenge of this project is to create an Machine Learning implementation that is based on a limited dataset. It is therefore key to find a good balance between current accuracy and the ability to improve based on future data. These models should also allow for real-time price generation.

The main focus during the research phase was to find state-of-the-art regression models, but also on ways to preprocess (i.e. filter and transform) our data. During development, we used Scrum to be able to quickly adapt to wishes of the client and used unit tests to verify our code base. At the end, we have developed a stand-alone price prediction API, where multiple models are available for price prediction. All of the developed models outperform the initial implementation, however it should be noted that accuracy and robustness could vastly improve when more data is available. Therefore, it is recommended that Sjauf waits until more data is available before switching to this approach completely.

Team Members:

Jonathan Katzy
Interests: Real life application of Machine Learning techniques
Contribution: Lead AI Developer
Tim Rietvield
Interests: Pattern Recognition, Data Visualisation and Data Filtering
Contribution: Head of Communications, Developer
Jaap-Jan van der Steeg
Interests: Data Mining and Machine Learning
Contribution: Quality control, Developer
Erik Wiegel
Interests: Machine Learning and Economics
Contribution: API designer, Developer

Coach:	Dr. M. Birna van Riemsdijk	Dept. of Intelligent Systems, Interactive Intelligence Group
	Prof. Dr. Catholijn M. Jonker	Dept. of Intelligent Systems, Interactive Intelligence Group
Client:	Stefan Dorresteijn	CTO Sjauf
	Roel Bloo	CFO Sjauf
Contact:	Erik Wiegel	erik@jwiegel.com

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

G

Project Description

Sjauf Technologies

`https://sjauf.nl/`
Sjauf is the country's only private driver app & platform. Using our services you can book a driver who will take you home, wherever you might be.

Our backend tech stack is completely modular, using GraphQL to use multiple APIs using a singular gateway. Most of our backend applications are written using Elixir and our frontend apps are written entirely using React Native.

Our platform allows customers to book trips in advance as well as on the spot. Our drivers make offers on the trips made and customers can accept or decline these offers. Once an offer is accepted, the trip is booked.

We're hoping to move towards an automated pricing system (instead of the offer-based system) and would like to have a group of students build a standalone system/application for this, using AI to improve itself continuously. This application can be written using any language as long as the API on top of it is written in Ruby, Elixir or JS.

Other information

Interns will be compensated for their work as well as their travel and food/drinks. We're a pretty small, young team who are extremely motivated to come up with innovative solutions for this market's biggest problems.