Implementing a machine learning microservice for scoring and predicting vehicle driving attributes and their impact on costs

Master's thesis in Computer science and engineering

Johan Blom

Sam Sohrabpour

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Implementing a machine learning microservice for scoring and predicting vehicle driving attributes and their impact on costs

Johan Blom

Sam Sohrabpour

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Implementing a machine learning microservice for scoring and predicting vehicle driving attributes and their impact on operating costs

Supervisor: Carl-Johan Seger, Department of Computer Science and Engineering
Advisor: Alexander Crayvenn, Sigma Embedded Engineering
Examiner: Aarne Ranta, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A stock photo which represents the dashboard of a car[37].
Typeset in LaTeX
Gothenburg, Sweden 2022

Implementing a machine learning microservice for scoring and predicting vehicle driving attributes and their impact on operating costs

Johan Blom
Sam Sohrabpour
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Minimizing automotive insurance costs and other forms of operating costs has become a new priority within the vehicle industry as it is included in the expanding subscription based business model for vehicles. Research has shown that automotive insurance costs can be decreased if the automotive company manages to prove to the insurance company that driving behavior of the vehicles are better than expected. It has also been shown that additional operating costs such as sending a replacement car during service are also expenses which would be reduced by better driving behavior. This study aims to provide an analysis that informs automobile companies how driving behavior attributes (trip data) affects operating cost. Through the use of machine learning models, the question is, if an automobile brand has trip data available, is it possible to create an analysis that can accurately predict service costs and other operating costs for the vehicle?

The analysis was made through the use of a machine learning model using a supervised algorithm called extreme gradient boosting. The machine learning model has been trained using trip data and operating cost data, where the model processes trip data to predict the likelihood of additional operating costs exceeding 10.000kr. In the absence of real data, the data used in the thesis was generated based on car statistics, not from real cars. The work was done with a micro-service structure, meaning multiple small services communicated with each other through API:s. The analysis for the final model demonstrated that it is possible to predict operating costs with a fairly good accuracy according to several evaluation metrics that was used to evaluate the model. The final model resulted with a 57% accuracy in finding vehicles with additional operating costs and a 92% accuracy in finding vehicles without additional operating costs. The results indicate that the dataset is too imbalanced due to that the rarity of requiring additional operating cost. This was handled by using the average trip data per car instead of processing all trip data individually. The machine learning models accuracy significantly increased once the imbalance ratio went above one car that requires operating cost for every 30 cars that does not.

Keywords: Python, Micro-services, Machine Learning, Kubernetes, Azure ML, XGBoost.

Acknowledgements

Throughout the master thesis there have been several people who has advised us on what to look out for and more precisely what is needed.

We would first like to thank our company supervisor Alexander Scott Crayvenn at Sigma Embedded Technology who even though was very busy, provided us with office space and weekly dinner. He also provided us with information about what types of frameworks and software most companies in the industry uses today. This thesis would have been significantly worse if it were not for him.

We would also like to thank our academic supervisor Carl-Johan Seger who has provided us with feedback of the reports structure and grammatical issues on a weekly basis, the report would not have been even close to how it is today if it were not for him. Carl has also provided us with advice that gave us an idea of exactly what was needed for the report to be of good quality. But also warned us of potential problems that typically occurs for master thesis projects within the machine learning field. These warnings have not only saved us from a few heart attacks but also made the progress of the report feel significantly less ambiguous.

At last but not least we would also want to thank our examinator Aarne Ranta who has been very open to talk to and has allowed us to be very flexible with our submission dates.

In addition, we would also like to thank the excessive amount of high quality documentation and information that the Python, Kubernetes and docker libraries provides us with. While technological enhancements from the outsiders perspective might have not expanded as much for the past 3 years, the technological advancements in terms of learning and sharing software definitely have.

Contents

List of Figures

1

Introduction

1.1 Background

As new business models arise, one which has become popular recently is the subscription based business model. Similarly to how this model has become a new standard within the TV industry, it has slowly crawled its way into the automobile industry where additional costs such as insurance and service costs are also included [6]. Other consumer expenses such as gas costs and service costs caused by at-fault accidents are not included in the subscription model. While leasing a car offers similar options, the additional costs are typically not included. This means that the margin of error for subscription based services is significantly higher than the margin of error for leasing models. Due to tight profit margins used in the automobile industry [21], an analysis that could predict these margins would be very beneficial.

To illustrate the basic problem, suppose that a person named Ken is going to rent an Audi using a company's subscription service for $800 every month. Ken consistently drives below the speed limit, barely misuses the breaks, never rapidly accelerates the car and does not have any close collisions with other cars. Meanwhile we have Leo who rapidly accelerates the car, constantly makes hard breaks and very often have close collisions with other cars. Should they really have to pay the same amount of money when they cause different costs to the company? It is true that the driver pays for at-fault accident service costs, but the automobile company who offered the subscription, still has to pay for some hidden operating costs, such as wear and tear, increased insurance costs and sending the driver a replacement vehicle during reparation and maintenance. This could sometimes even go as far as to eat up the whole profit margin that the automobile company initially had expected for the car. This means that a driver with Ken's driving behavior in fact costs the company significantly less than a driver with Leo's driving behavior.

There are many automobile companies who have connected multiple sensors to their vehicles. Some of them also have these sensors connected to the cloud where snapshots of the vehicle's current state is sent to the cloud. This has enabled automobile companies to track driving behavior of the drivers. This data comes in many forms. The data could be anything from a snapshot that is given upon a request, to a list of attributes given for a specific driver within a specific trip (trip data).

If an automobile brand has trip data available, is it possible to create an analysis that can accurately predict service costs and other operating costs for the vehicles?

1.2 Aim

The goal of our work is to provide an analysis that informs automobile companies of how driving behavior affect operating costs. The intention is to gather the results of the analysis, then discuss different measurements the automobile company could take to decrease operating costs, and see if the accuracy of the results is enough to make the approach feasible.

1.3 Scope

The analysis will only be made through the use of machine learning models. Only a limited amount of machine learning algorithms will be tested, such as supervised algorithms. Supervised means that the input data is labeled into categories, then the algorithm will output predictions based on that data. Only the most functional machine learning model will be evaluated for the results. The only form of data that will be used is trip data, meaning any other form of snapshot grabbed by a vehicle sensor will not be considered. Additional bugs in the sensory data that bypasses data filtering may be discussed, but will be disregarded during the analysis.

1.4 Contribution

Currently, the auto-motive industry lacks important metrics when determining the value of a car. Poor evaluations could lead to selling cars for less than what they are truly worth, or make future car prices hard to predict. As of now, many car companies only use mileage and the age of the car when evaluating it, leading to a sub-optimal evaluation. This work would contribute to a more precise evaluation of a car's true value and future value, thus providing an analysis for car companies to utilize when deciding the value of their cars.

1.5 Overview

The theory chapter first begins with an introduction to machine learning which is made for readers who have an academic background but lacks a machine learning background. The second half of the theory chapter goes through all of the important theoretical parts that was used specifically for the thesis.

The method chapter explains the data generation process, the architectural plan for the creation of the machine learning model and how the models are stored on Azures servers through the use of its Kubernetes service.

The results chapter provides an objective overview for the evaluation of the machine learning model that was created.

The conclusion chapter begins with a discussion about different potential measurements the company could take with the results. It also interprets if the results are

accurate enough to be used for those measurements and if there are any improvements that could be made to gain more accurate results.

2
Theory

2.1 Machine Learning

Machine learning (ML) is a way for humans to utilize the fast computations that a machine possesses [8]. A ML algorithm improves the accuracy of its results step by step by rerunning a basic algorithm a large number of times on data. This creates a model, which is trained to recognize specific types of patterns.

2.1.1 Applications of Machine Learning

Analyzing a small amount of data can easily be done by a human. For example, analyzing ten pictures to determine if there is a cat or a dog in the picture is easy for a human. But what if there were million photos to analyze? It would be impossible for one person to analyze that amount of pictures as it would take weeks or even months to complete. A computer with much faster computation capabilities can search through the photos and find patterns to determine the animal in the picture. The accuracy of the predictions of the type of animal depend on the quality of the algorithm. An accuracy as high as 97 percent can be reached [33].

ML is not just about analyzing data and checking the type of animal in a picture. More generally, ML can be used to find patterns and predict trends. This is called pattern recognition and with the help of information acquired from patterns an algorithm groups data into different categories. An example of a pattern could be when an animal has pointy ears, then it would be more likely that the animal is a cat than a dog. Machines are capable of working with larger quantities of data than humans, therefore ML algorithms can potentially find patterns that a human might not find.

At an abstract level, pattern recognition based on ML proceeds as follows. First a large amount of data is collected. A fraction of this data is separated out to be used for testing. The remaining data is used to train some kind of network by gradually adjusting, usually a very large, number of parameters. Once the training has been concluded, the testing data is used to determine how successful the training was.

A study has shown that ML can surpass humans in finding patterns, in, for example, speech recognition or image classification [33]. In the study the humans learned faster than the machine during the first hours of the learning process. But eventually the humans stopped improving. On the other hand, the ML algorithms continued to improve and surpasses the humans when dealing with a large amount of

data. However when the patterns required a deeper understanding, the algorithms struggled to find patterns. The use of pattern recognition could be useful for finding patterns in human driving behaviours and potentially predict how behaviour affects the operating costs.

2.1.2 Data Filtering

When working with ML models it is desirable to have large data sets to achieve a good quality prediction [52]. A large data set is desired as the model generally improves the more data it gets to process. One has to be careful with training too much on a data set however, as it can lead to *overfitting* the model. Overfitting means that the model has been trained so long on a data set that the model has essentially memorized it. Memorizing the data causes the model to struggle with new data and lead to erroneous predictions, which defeats the entire objective of the model.

Data sets can contain data that is unnecessary or irrelevant for the prediction one wants to make. For instance, consider a data set on students in a school with their respective personal info and where the goal is to predict the students' favourite subjects. Data on their grades would probably be relevant for the prediction, but their phone number would most certainly be irrelevant. That is why it is desirable to filter out some of the data so only the relevant part is left. This helps speeding up the process of training the model given the smaller data size. Filtering also helps with visualizing and analyzing the data as it is easier to see what is important when looking at a smaller set.

Data cleaning is a part of data filtering that is important for getting valid predictions [47]. This is when one removes irrelevant data as previously mentioned. But also data that is contrived through mistakes and inaccuracy. This kind of data could be people in a survey that just answers quickly to be done with it. This type of behaviour makes their data useless as they did not pay attention to the questions.

2.1.3 Algorithms

There are three types of ML algorithms: (1) *supervised*, (2) *unsupervised* and (3) *reinforcement* ML algorithms [4]. In supervised algorithms, the data is accompanied with labels denoting the desired result. These labels are used to train the model. In unsupervised algorithms, no such labels exist. Here the algorithms try themselves to partition the data into interesting groups [4]. In reinforced algorithms, the model creates its own data by recurrently simulating an environment. This process is done until the model has learned to take the optimal long-term action in order to obtain a certain goal. The choices made to reach the optimal long-term action are derived from the algorithm [4]. In our work, we will only consider supervised algorithms. This is because the work needs labeled data in order to predict a certain class.

2.1.4 Classification Algorithms vs. Regression Algorithms

To understand which supervised machine learning algorithm would be the optimal, it is necessary to understand the difference between *Regression* and *Classification*. These processes are the two most commonly used with supervised ML.

Regression is the process of predicting a continuous value that can have an infinite amount of possible variations. Examples of a continuous value could be, for instance, price, salary and height [19]. On the other hand, classification is the process of predicting a binary value, such as true or false, or zero or one.

Consider a scenario where one would predict how the weather is going to be by feeding data where the data, month and hourly time is included on an hourly basis. A classification algorithm could predict whether the weather is going to be warm or cold during a given hour, where anything above 15 Celsius is considered warm and anything below 15 Celsius is cold. A Regression model on the other hand is able to predict the temperature in Celsius for a given hour. While the regression model may be providing a more specific number as output, the classifier model may have a higher likelihood of being correct in this case. The classifier model may also provide other useful insight such as predicting the probability of something occurring rather than estimating a specific number. This is highly useful when the attributes of the trained dataset only has a slight implication to the output rather than a strong one.

Algorithms such as the linear regression algorithm only take advantage of one of the regressive processes. But there are numerous of other algorithms that can use both classification and regression processes [19].

2.1.5 Linear Regression and Logistic Regression Algorithms

Regression analysis is a technique which focuses on the relation between a dependant and an independent variable. There are three main uses for regression: find the strength of each predictor property, to recognize which variable relates most to the result value that is desired and forecast future values [51].

Logistic regression uses a function called Sigmoid. The Sigmoid function can take any value and map it to between one and zero, as seen in figure 2.1. Given that the values are between one and zero, the Sigmoid function is useful when dealing with probabilities or binary classification. The function will classify an element into a category or a discrete label based on a threshold [51].

To illustrate two problems where linear regression and logistic regression would be the appropriate algorithm to use, two different examples about brakepads are used. For linear regression, consider trying to predict the cost of brakepad replacements given the number of hardbrakes per month. Here there is hopefully a linear relation between the number of hardbrakes and the wear and tear on the brakepads, as seen in Figure 2.2. Finding the best slope and y-intercept would allow us to quickly estimate the brakepad replacement cost given a particular driver profile.

On the other hand, consider the probability that a car ends up in a major collision requiring a replacement car as a function of average percent below or above the

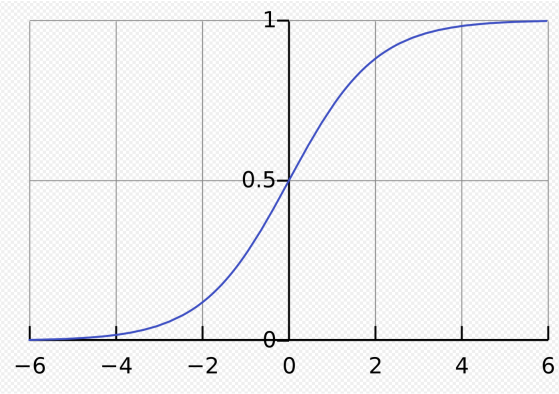

Figure 2.1: Example of a Sigmoid function [50]

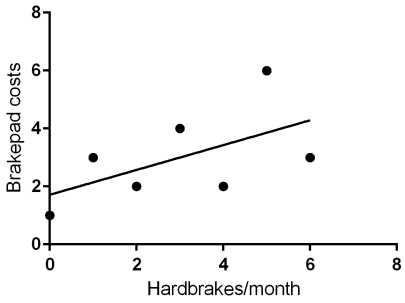

Figure 2.2: Example of a linear regression graph.

speed limit the car is driven. Fitting the Sigmoid function to these data points would give us an estimate for the probability of a serious crash, as seen in Figure 2.3.

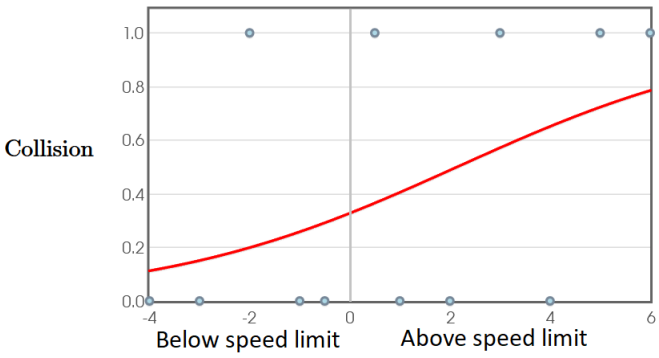

Figure 2.3: Example of a logistic regression graph.

Testing accuracy for linear regression is measured by loss, R^2 and Adjusted R^2 [53]. While logistic regression uses different methods such as accuracy, precision, recall and F1 score.

2.1.6 ARIMA Algorithm

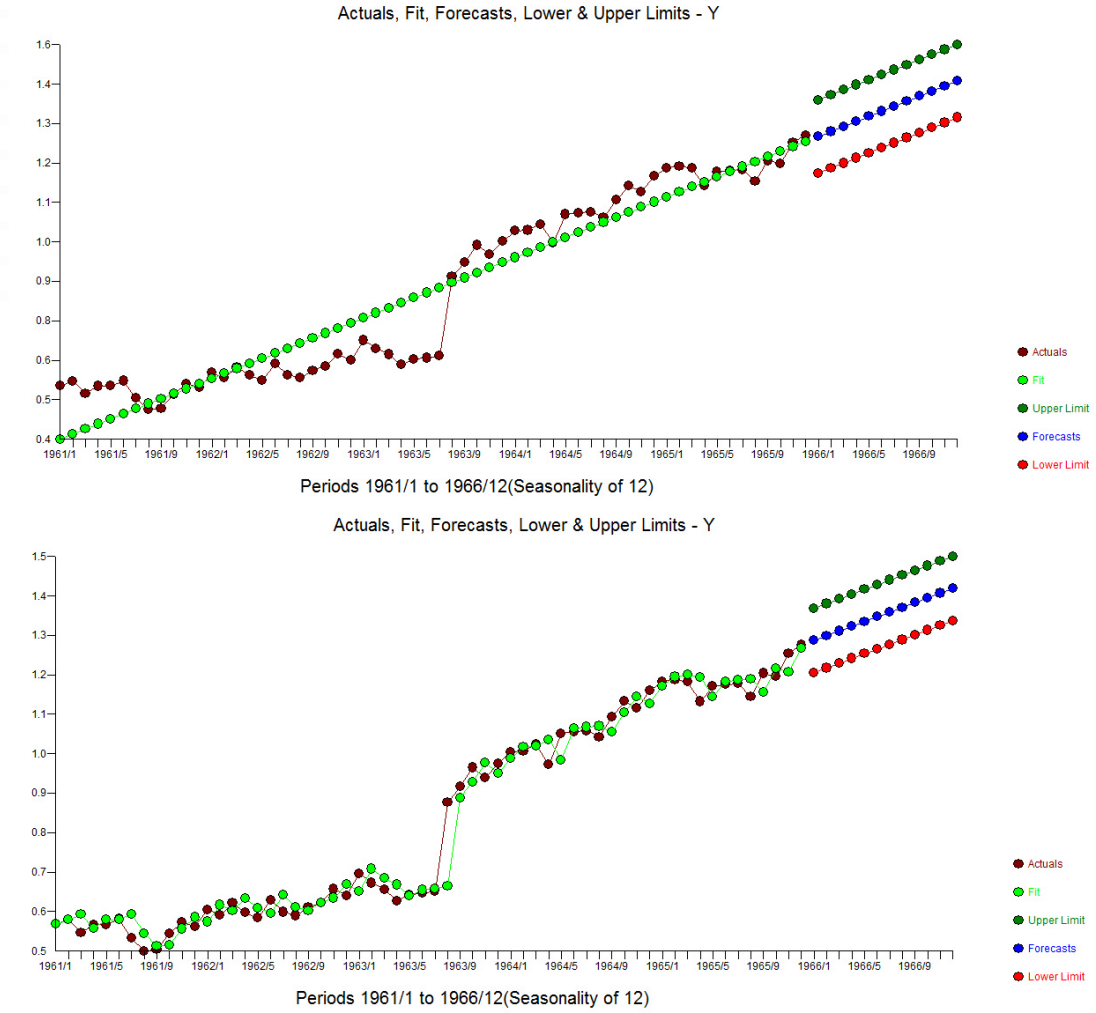

Figure 2.4: An example of how the Auto-regressive model ARIMA forecasts its values compared to normal linear regression model. The first picture represents the results made from the prediction of the linear regression model. The second picture are the results made from the ARIMA model. The brown dots represents the data sets that was used and the green dots represent the predictions that were made.

Linear regression and logistic regression are popular to use as initial machine learning algorithms. The reason behind their popularity is that they are both easy to implement and to understand. However when it comes to ML algorithms that are used in production, more sophisticated algorithms are used [31].

ARIMA is one of the most efficient simple algorithms [38]. The ARIMA algorithm is an amalgamation of two models called auto regressive [39] (AR) and Moving average [36] (MA) models.

2.1.7 Decision-tree

Decision tree is a type of supervised machine learning algorithm. A unique aspect of decision trees is that they do not always yield a single answer. The trees can produce several options, which then a human can decide which answer to use. It is an efficient way of analyzing several alternatives and interpreting them; giving a better understanding of the model.

Trees consist of a root node which is the tree's base. This root can split up into several sub-nodes which themselves can split up to more sub-nodes. Each sub-node is represented by a decision, if that decision is fulfilled then the algorithm will follow that path. Furthest down in the tree are the leaf-nodes which marks the end of the sub-nodes and thus represents the tree's answer(s). An example is shown in figure 2.5 in which an animal is identified.

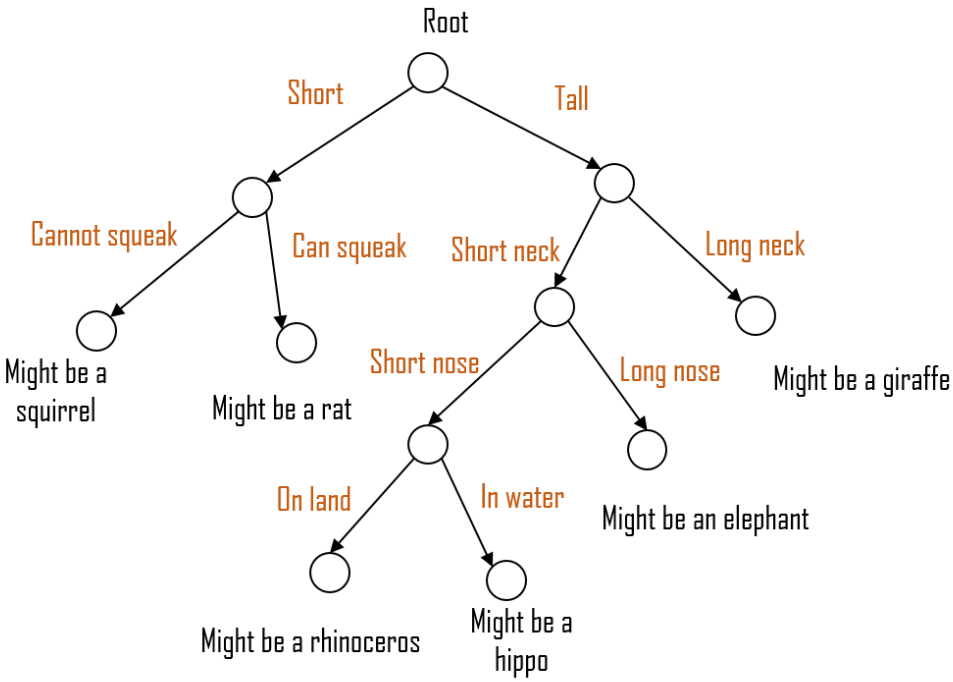

Figure 2.5: An example of a decision tree where it tries to identify an animal

There exist two types of decision trees; *categorical* and *continuous*. In the categorical type the variables are binary in that they are yes or no questions. In continuous trees more variables are utilized than in categorical. It is not just one yes or no question, instead, there are variables that can have an infinite amount of different values, such as numbers. The categorical type is sufficient when dealing with binary questions but is lacking when the questions are more complex. Continuous trees are better to use with regression type problems as they both utilize non-binary values. An example of a continuous tree can be seen in Figure 2.6, where the variables in the decisions are non-binary and can change.

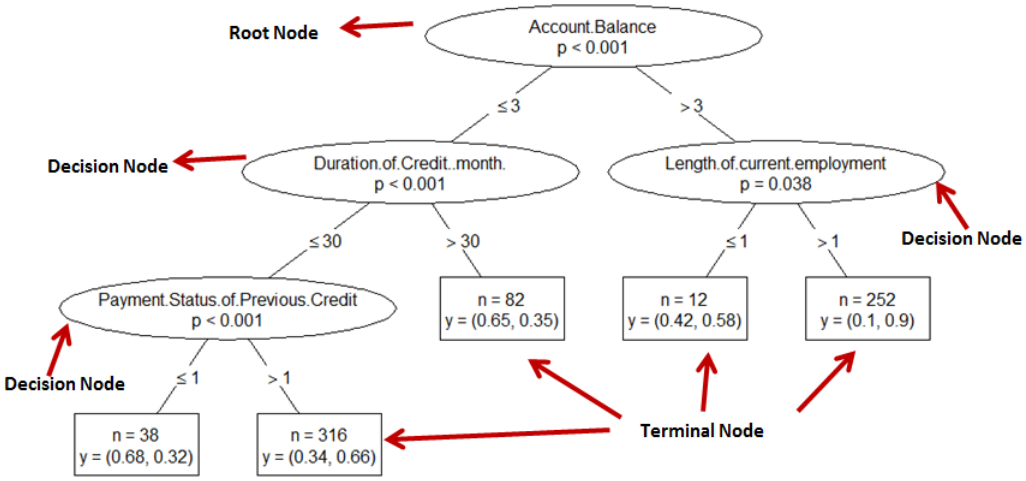

Figure 2.6: An example of a continuous tree [7].

An advantage of decision trees is that they are simple to analyze, thus giving a clear picture on how to improve the model [30]. Another advantage is that trees can deal with both numerical regression types of problems and binary type problems.
Some disadvantages with the trees are that they are not optimal for large data sets. They also often take more time when training a model than other algorithms.

2.1.8 Ensemble Method

An ensemble method is a technique used in machine learning which combines multiple weaker models to produce a single strong model [54]. These multiple models are called weak learners. The weak learners are typically simple ML algorithms, such as linear regression and logistic regression. Weak learners are often used in conjunction with decision trees and take advantage of the simple structure of the trees. Two common types of ensemble methods are:

- *Bagging*: This method combines bootstrapping and aggregating, as seen in Figure 2.7. Bootstrapping is when you generate samples from a data set. The samples are created by randomly picking a subset of the larger data set. These samples are then observed in order to get an estimate of the entire data set. One can, for instance, calculate the mean of a sample and compare it to the mean of other samples, in order to get a clearer picture of the whole data set. Aggregating is when you have several models and combine their predictions. The idea is to use multiple independent models and take the average of their predictions which results in a single model.

  A decision tree is created from each sample and then used with different algorithms for each tree. The output of all the trees is combined into one, which is then used in the final model. The advantage of Bagging is that it enables a collection of poor learners to outperform one single strong learner. A disadvantage however is that interpreting the model is more difficult. This is because it is harder to determine how each individual variable affects the prediction,

which makes it more difficult to identify poor variables [23].

- *Random Forest* (RF): The algorithm consists of several decision trees that work together as an ensemble. RF works by randomly picking a number of data points. It then builds a decision tree from these data points. These two steps get repeated a certain amount of times, depending on how many trees one wants to build. All the trees will then predict values based on new data points. The average of all the trees' predicted values will be the final value for that data point, if it is a regression problem. If it is a classification problem, then the most often selected class will be chosen.

  The key difference between RF and Bagging is that RF only uses a random subset of features, these features include variables or columns. RF can, for instance, choose only columns in its samples, whilst Bagging considers all the features. An advantage with RF is that it can handle both classification and regression problems. It can also reduce overfitting in the decision trees. A disadvantage with both ensemble methods however is that they demand a higher computational power given the amount of trees that have to be built [23].

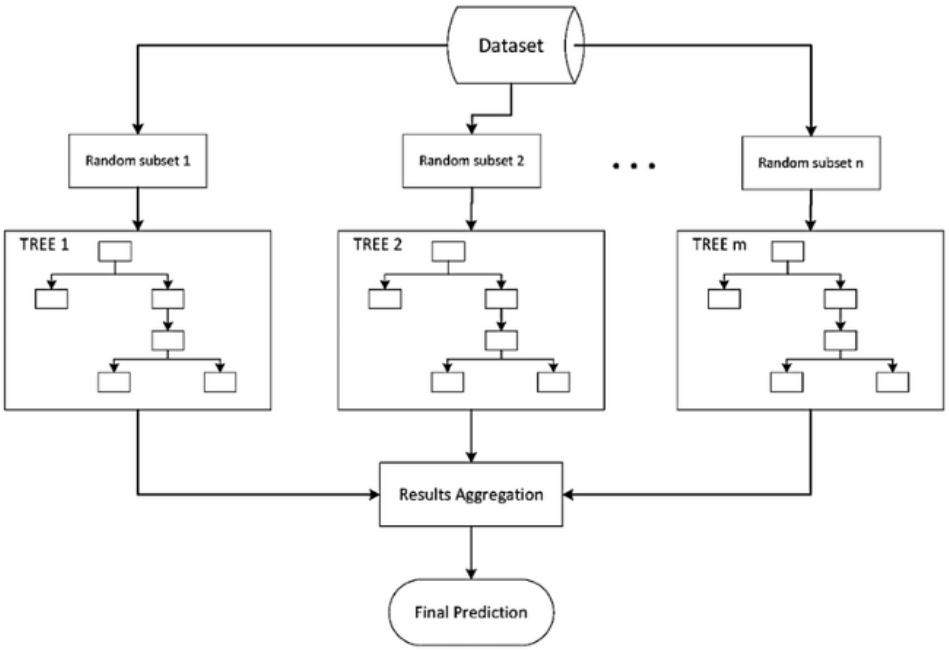

Figure 2.7: An example of the structure of the bagging method [26]

2.1.9 Adaboost

Adaboost, short for adaptive boosting [41], is an algorithm that utilizes boosting and often with the help of decision trees. A common Adaboost technique is to use decision trees with only one level, these are called Decision stumps. Decisions stumps consist just of a root node and its two corresponding leaf nodes. This means stumps only use a single attribute when splitting the root. The Adaboost algorithm

has the advantage that it is less susceptible to overfitting. This is because of the independent parameters that come with the ensemble structure. A problem with Adaboost however is that it requires a quality data set, one without big outliers and junk data. This means that the data have to be cleaned thoroughly before using the algorithm.

2.1.10 Gradient Boosting (GB)

Gradient boosting is one of the most popular machine learning ensemble methods to use in price winning machine learning models [42]. GB takes advantage of weak learners. These learners could, for instance, be a decision tree or a linear regression model. GB takes advantage of weak learners by running them sequentially. It functions the same as Adaboost, in which weak learners learn from the previous learners errors [42].

The reason why gradient boosting is used over other ensemble methods like Adaboost, is its high compatibility with most simple ML algorithms. Since it provides accurate results for most machine learning algorithms it is also practical for most supervised machine learning use cases.

Due to the strength of the gradient boosting algorithm it is typically compared with neural networks rather than other supervised machine learning algorithms. If one were to compare these two algorithms with each other it would be noticed that neural networks may be able to create a higher quality model whilst using large data sets. On the other hand Gradient boosting typically provides more accurate models whilst using a smaller data set.

2.1.11 XGBoost

XGBoost is a decision-tree-based ensemble method that builds upon a customized version of the gradient boosting algorithm. It is known to be one of the best supervised machine learning algorithms due to the low amount of training data one needs to create a model with high accuracy [29].

In the Adaboost algorithm, small decision trees called stumps are used, XGBoost use the gradient boosting algorithm together with stumps as weak learners. This allows these decision trees to learn from each others mistake by comparing the outliers with each other [29].

XGBoost algorithm has a similar flexibility to the gradient boosting algorithm in which any simple classification and regression problem can most likely be efficiently solved by it. On top of that, unlike other boosting algorithms XGBoost also uses regularized boosting methods which helps the model with avoiding overfitting. It can be implemented to automatically handle missing values in data sets which would normally result in errors or non-accurate results in other algorithms. One can also split up the training over a period of time, so one can pause the training of the model and then resume the training later.

Consider a scenario where one would have dataset on how many minutes each worker has interacted with a tab for each day (interact data), and a dataset on how much

work each worker has done for each day (work data). A hypothesis can now be made claiming that the amount of minute spent on the correct process tab has a high implication to the amount of work that has been made. One can now use the interact data as input and work data as output to train a machine learning model to potentially predict how much work each employee will do.

Since the amount of data that is provided is most likely not more than a couple of thousand rows, an accurate machine learning model that is fast, flexible and easy to setup is therefore suggested. While there are better alternatives than extreme gradient boosting, they typically need a high amount of modification to work properly.

The difficult part of XGBoost is to tune its hyper-parameters to fit the data set that is provided. Hyper-parameters are the parameters that control the model's learning process and decides the model's objective. Since gradient boost learners is hard to properly understand, testing a series of assumptions by the use of tuners from certain libraries is necessary to prevent the machine learning algorithm from under- or over-fitting [27].

Some of the most important hyper-parameters in XGBoost:

- Booster: One can choose between SGtree booster or a SGregression booster depending whether one wants the output to be a classifier or a continuous value.
- Objective function: One can choose between softmax, which allows multiple classifications, or softprob, which predicts the probability of a data point belonging to a certain class.
- Eta: Is the learning rate that the machine learning algorithm uses. The default value is at 0.3.
- Max_depth: The maximum depth of a tree in the weak learners. If it is too small the model will end up being inaccurate and if it is too high the model will be overfitted.
- Min_child_weight: Is used to control the overfitting, if it is too high the model will be underfitted and it may end up with inaccurate results.

2.2 Checking Quality of Algorithms

When a machine learning model has been trained and is ready to be used, it is necessary to evaluate how accurate it is. This section discusses useful and common metrics used when evaluating models.

2.2.1 Evaluation Metrics

- Accuracy: Evaluating the accuracy of the predictions by checking how many times they are correct, this is good when dealing with binary predictions. For instance, when checking if a picture contains a cat or dog. However, given the binary nature of this metric it is not useful when measuring the accuracy of continuous quantities. For example, when measuring prices, as a difference of

hundred SEK would be treated the same as a difference of one SEK.

- Root Mean Squared Error (RMSE): A popular evaluation metric when dealing with regression problems. It is the square root of the average of squared errors.

  $\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(predicted_i - actual_i)^2}{N}}$, where N is the sample size.

  If the formula returns zero then the model is perfect as the predictions match exactly the actual values. If it is not zero then a lower number means a better model, as more errors lead to a higher value. This formula is good with regression problems but can be greatly affected by outliers. There are several variations of this formula, one can for example use the log function with RMSE. This version is useful when dealing with large numbers as the larger difference would not be penalized as much. The log function also scales down potential outliers, thus significantly limiting their negative effect.

- R-squared: Another useful regression metric as it measures how good the model fits. R-squared is meant to show how the variances of variables affect other variables. The ideal value is one and the closer to one it is the better the model. It works by comparing the model with a defined baseline which can be a simple model.

  This metric evaluates how good the model is compared to a simple model that just predicts values based on the mean value from the training set. R-squared has some limitations however, as the value does not decrease when adding new variables to the model, even if these variables are redundant and poor. The value can only increase or remain the same, which is not desired if the new variables are of poor quality.

- Adjusted R-squared: This version of R-squared considers the new variables which helps ascertaining if the new variables increases the fit of the model or not.

  $\text{Adjusted } R^2 = 1 - [\frac{(1-R^2)(n-1)}{n-k-1}]$,
  where n is the amount of data points and k is the number of independent values.

  Adjusted r-squared is better when dealing with several variables in the model as the value can actually be affected by new variables, thus being more accurate. If there is an assurance that there is no bias and only two variables, then r-squared would be better, otherwise adjusted is preferred.

- F1 Score: This metric uses recall and precision as it is trying to get the best of them both at the same time. Recall is the amount of correct predictions divided by the amount that should have been correctly predicted, it is the percentage of results relevant to the model. Precision is the percentage of how many of the predictions are correct. For example predicting if a silhouette is of a human, the model identifies five humans out of eleven human silhouettes and some non-human silhouettes. Only three of the model's five predictions were correct while the others were false positives. The recall in this case is three out of eleven, while the precision is three out of five.

F1 Score uses the concepts of precision and recall to its advantage by taking the harmonic mean of the precision and recall.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

This is good if either precision or recall is very small as it will result in a low overall score because it balances the two metrics, hence making it accurate. It also represents many aspects at once because it combines other metrics into just one. F1 is used to evaluate binary classification problems, which classify data into either negative or positive. It is also better to use with imbalanced data than accuracy, as it can measure the accuracy of the positive class and not just the overall accuracy of all predictions.

- Brier score: This score measures the accuracy of a model handling classification problems with probabilistic predictions. It works by calculating the mean squared error between the predicted probability and the actual result. Because of this, the score can only be in the range of 0 to 1, meaning that a lower score means a more accurate model.

- Area under the ROC curve (AUC): This metric combines the area under the curve metric with the receiver operating characteristic curve (ROC). ROC is the performance of a classification model at every threshold. ROC utilizes two parameters; true positive rate and false positive rate. True positive rate is the probability that a class 1 driver will be predicted as class 1. On the other hand, the false positive rate is the probability that a class 0 driver will be predicted as class 0. These parameters keep track on if the predictions are of the correct class.

  AUC is the area under the ROC curve, as seen in Figure 2.8. AUC measures performance of the thresholds in the classification model and shows how good the model is at distinguishing between the classes. A good AUC score indicates that the model does not make many false predictions, for instance, predicting class 1 as class 0. The score ranges from 0 to 1 and a higher score means a better result.

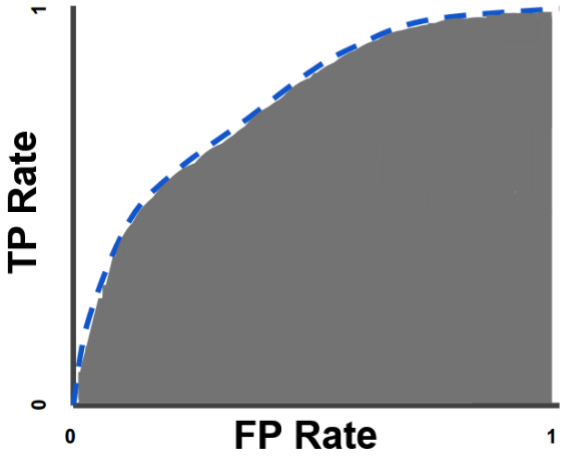

Figure 2.8: An example of a ROC curve with the area under the curve marked in black [20].

2.3 Car Evaluation

2.3.1 Usage-based insurance

Usage based insurance (UBI) is a type of car insurance where cost of the insurance depends on the drivers driving behavior [28]. Similarly to how trip data is measured, the usage based insurance also focuses on measuring driving behavior factors such as mileage, speed, hard braking, acceleration and time. An analysis is made to find the correlation between the driving behavior attributes and the additional service costs which is then used to estimate the price of the insurance cost. The pricing of this type of insurance provided the drivers with a 5% to 10% decrease in costs for just sharing the data and up to an 20% to 30% decrease in costs depending on the driving behavior and the insurance company that was used.

2.4 Deployment of Machine Learning

While machine learning is being discussed and actively researched, deployment of machine learning services in industry is not brought up very often. According to a study conducted at Algorithmia [3], a machine learning operations platform company, out of 750 companies that was developing machine learning solutions, only 51% of them had ever deployed one single machine learning model in practice. That means 49% of them have nothing practical to use and does not utilize machine learning to the fullest extent. There exists a lot of information about different machine learning algorithms, how they work and which one is the most effective but there is not much information about the other steps in deploying a machine learning model. For example, what data is being collected, how the data can be used to predict what it is desired, how the filtration of the data will be made, how the model will update itself after its first deployment, how the model is going to be implemented in the

framework that is planned to be used, etc.

2.5 Micro-services

In order to deploy a ML model in practice, one will be required to consistently train new models and deploy them. Running the ML training on the same hardware as the framework that one want to use it on would not be recommended since the hardware would most likely be slowed down when the ML framework is training a model. This is where a micro-service architecture becomes interesting. In a micro-service architecture all the different frameworks are hosted on separate servers and communicate through a REST-API [32]. A REST-API is an interface that enables interaction between applications and services, the name stands for representational state transfer. Once the REST-API is set up correctly, it can then communicate between the micro-services through so-called HTTP POST and GET requests. One example of this is when the ML framework has finished creating its model, it can then send and retrieve information to the other frameworks using GET and POST requests.

Micro-services are used today for various reasons. For example, when producing a web application that both needs to perform machine learning tasks while still working like a normal website with fast loading. The web application requires a server that has both high bandwidth and uses CPU:s that are capable of running quickly for a very short amount of time. The CPU should be available immediately but may be used for other things for most of its lifetime. The framework that can create the machine learning model requires a lot of CPU and GPU power for a few minutes typically once a day or once a week.

If one were to train a model on the web application server the website will be significantly slower when the server is training the model compared to when it is not doing so. Most cloud services typically offers resources that are designed for specific use cases. A CPU core that is designed for app services could be half the speed and cost one tenth of a CPU core that is designed for machine learning services. Thus, dividing the resources of the services into their own servers may both solve resource management problems and also be less costly.

Other pros of micro-services are that they allow developers to use multiple platforms for the same application. Suppose that a group of developers want to create a web application that can provide ML solutions. If the group want to have a monolithic architecture that uses the same framework for the whole application they may be forced to use a python framework such as Django [22]. Suppose that these developers all have a good background in ReactJS and want to continue using it instead of switching to Django. The micro-service architecture solves this problem since they can code the web-application on the REACT platform and then create the machine learning models with a framework such as python.

With micro-services it is much easier to make future updates for an individual micro-service after the rest of the services are already finished. If there, for instance, have been teams working on individual parts of an application it would be very difficult

for one of the teams to update their part after the other teams have finished their parts. If each part however is split up into its own micro-service the team can easily update one of the parts later since they only interact through outputs and inputs.

While micro-services are very beneficial in many cases, they come at a cost. A downside is that one has to make sure that the output for the microservice will be compatible with all of the other platforms that it is made for. Another downside is that it can easily end up in input and output parameters that are not compatible with the use case that it was made for.

For example, suppose a development team is making an API that is designed to let others gain driving behaviour data. The people that wrote the API did not have machine learning in mind when they wrote that micro-service. They could only give driving behavior data of how one car is behaving exactly when the API request is made, rather than providing the users with historical data of all of the cars directly.

2.5.1 RESTful API

A RESTful API is a programming interface which allows reading, writing, updating and deleting information from a server through the use of web requests and web responses [18]. The RESTful API is typically connected to a web domain to make communication between two sources possible. The standard format that is used to send information is by adding input values to the website that one would be connecting to. Typically these inputs are received in a JSON language format and are then processed by the framework that the API is created with.

Suppose that a person has a fully customizable RGB light at home and that the person has connected and deployed a RESTful API to it. The person can now create a page on a website with the RESTful API framework and update the color of the light through that page.

Since the RESTful API of the lamp already has its values initiated, he must create a page that takes in 'PUT' requests to update the lamp using a format such as:

```
{
    "ID" : Integer
    "RGB1" : Integer
    "RGB2" : Integer
    "RGN3" : Integer
    "isOn" : Boolean

}
```

Suppose that the person knew that the ID for the lamp was 23 and that the person wanted red colour on the lamp, then the API request would have been the following:

```
www.domainName.com/api?ID=23&RGB1=255&RGB2=0&RGB3=0&isOn=True
```

This would have sent the following JSON request to the API:

```
{
    "ID" : 23
    "RGB1" : 255
    "RGB2" : 0
    "RGB3" : 0
    "isOn" : True
}
```

Once the request has been made, the website will automatically run a function with the provided JSON elements as the functions input parameters to update the state of the software's instance. For every GET, POST, PUT, DELETE request that is made, a code made by the creator of the API will process that information in any way the programmer sees fit.

2.5.2 Swagger UI

Since it may be difficult to understand how an API call works for a specific page, most of the RESTful API frameworks have built in addons such as Swagger UI, which enables external users to see all possible API calls in a more clear manner. In Figure 2.9, an example of the Swagger interface can be seen.

This makes it possible for API frameworks to have additional builtin features, such as providing a description of what every API call does. API:s also let the programmer modify the response one gets after a request. The response can prevent showing the user sensitive information whilst at the same time informing the user that everything is going well. Other features which Swagger UI enables is to create many different types of forms which the users can enter, showing what base values that is going to be sent if not entered. Swagger UI can also show the users which type of errors that could occur and can let the developers inform the user of every type of error that is occurring.

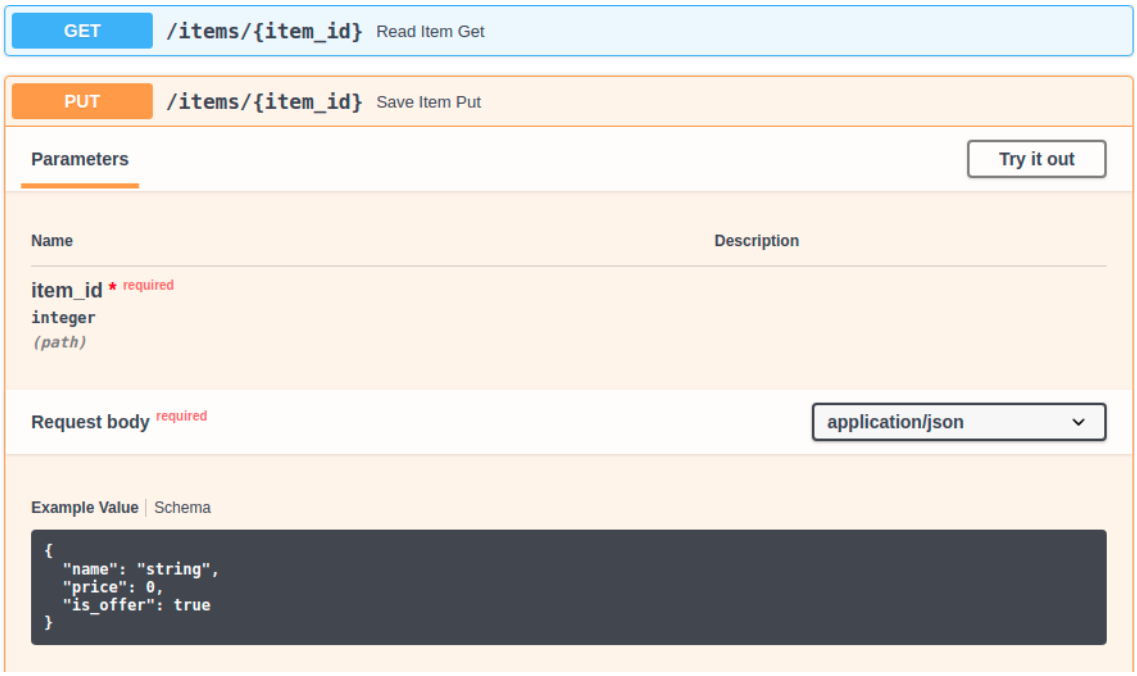

Figure 2.9: Is a screenshot of the API documentation that is automatically generated by Swagger UI. In order to enter this site, one needs to change into <API_Directory>/docs.

2.5.3 Docker

Docker is a platform for packaging applications into containers and managing them safely [48]. When an application is complete it may require multiple libraries in order for the application to work. These libraries must be installed on the computer that needs to use the application. It can be tedious and difficult for the user to find everything and install it in a correct way. The application may demand certain settings in the system as well, further increasing the difficulty of running the application. These problems are what Docker handles with its containers and images.

A Docker image is a small piece of software that contains the code of the application and all its requirements, such as libraries that needs to be installed. This enables the application to run quickly in any environment, without having to adjust the environment's settings. A Docker container is an instance of this image and enables the application to run with all of its requirements ready. There can be multiple running containers of a single image and each one can be managed separately. The advantage with containers is that they are lightweight in size, they are fast to deploy and does not take up much memory. Virtual machines which is the alternative to Docker containers, are much slower to deploy and demands more memory in the system. This is because a single virtual machine requires a full copy of the operating system, whilst containers can share the kernel.

2.5.4 Kubernetes

Kubernetes is a platform for managing containers and workloads [49]. It is meant to simplify the use of multiple containers. Handling containers is not an easy task when they start to increase in numbers. If a container goes down for instance, one would want another to start immediately to guarantee that there is no downtime for the application. Kubernetes can fix this problem by creating clusters of containers and manage the clusters so that they run as expected. If a container goes down; Kubernetes can start another one immediately, without the system owner having to manually start one. Containers that have failed and stopped running can be restarted again by Kubernetes if possible, otherwise they get terminated so that a faulty container cannot disrupt the application. Kubernetes also handles workload balancing, by splitting the traffic between several containers, resulting in them sharing the workload. All of these resources make handling containers efficient and simple.

Since kubernetes may manage other types of units that is not a containers, kubernetes calls these units Pods [12]. These pods run on a virtual or physical machine called a node [13]. This means that the node could either be a part of a computer with its own operating system or the whole computer itself. These nodes can either be rented from a cloud service provider or hosted on a physical machine depending on the needs of the users.

One general issue which occurs in hosting is balancing the amount of resources needed for the use case that it is intended for. One example of this was the back-end servers of the consumer-electronic company Elgiganten, who during a black-friday weekend in the year 2019 had both their webpage servers and their cash register systems overloaded due to too much traffic [25]. This resulted in most consumers not being able to buy anything either from their physical stores or their e-commerce websites. While it is true that having too many servers available at all times might be unnecessary for a company such as Elgiganten, temporarily gaining more servers with a click of a few buttons would have been extremely useful.

What kubernetes framework allows you to do is to easily scale the resources, both through replication of container instances but also changing the amount of resources used by each container on an hourly basis. If Elgiganten had their servers on Azure kubernetes they could have just ordered more computing resources for a few days and then disabled them afterwards.

2.5.5 Kubernetes Services

In order to make a pod service available externally to other people it is necessary to connect a set of pods into a Kubernetes network service [14]. There are three types of network services: (1) ClusterIP which is a service that offers a single internal IP-address that redirects to the all pods connected to it, (2) NodeIP service which offers an external IP address to a set of pods which can then be used by online by other people, and (3) a Loadbalancer service which offers both types. The service therefore enables connecting a set of pods into a single IP-address and then the pods that the service chooses is dependant on a policy that is set by the person

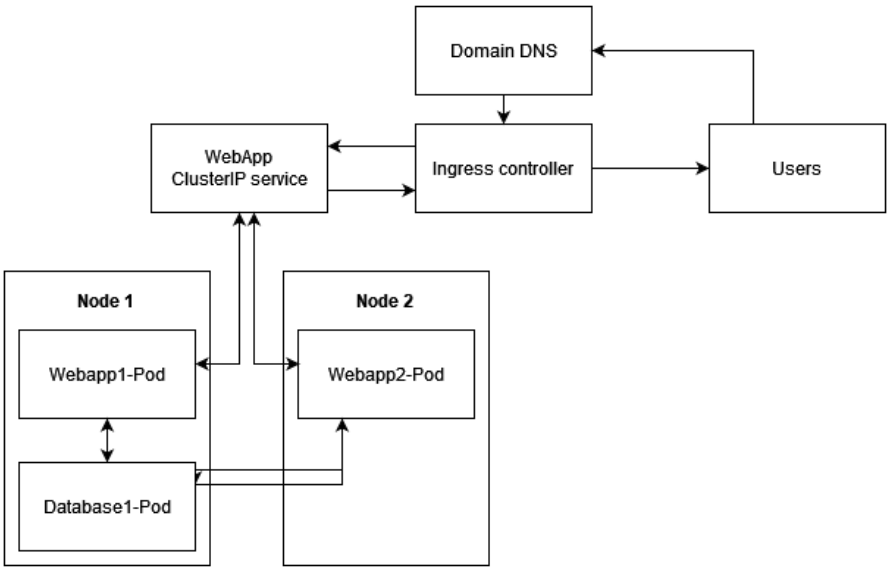

Figure 2.10: Example of a kubernetes architecture with a webpplication service which uses two replica pods connected to a database pod.

who created the pod. An example of such policy is called limit ranges. This policy limits the computing resources a certain container can have. The policy prevents a container from taking all of the resources available and starving other containers in the process.

2.5.6 Kubernetes Ingress

While it may be possible to host a website by using an external IP address given by a Loadbalancer or a NodeIP service, it should only be done for testing purposes. To prevent unnecessary security vulnerabilities on a kubernetes cluster, another feature called Ingress exists which is an API object that enhances the external communication between a server and a domain [10]. When entering most webpages it is typically expected to have TLS or SSL security when connecting to the domain, this is something which Ingress helps to solve. TLS and SSL stands for Transport Layer Security and Secure Sockets Layer respectively. Once an Ingress object is set up, it can then run on an Ingress controller to make communication between a pod and a webdomain possible.

3
Methods

3.1 Machine Learning Model Training Platform

The goal with this machine learning model is to create a micro-service which can, through the use of API requests, predict operating costs of a vehicle. The FastAPI platform is used both for the training of the model and for the creation of the API.

FastAPI is hosted on Azure's Kubernetes service which is Azure's most popular platform to host applications on [11]. Kubernetes also offers APIs connected to the model which they label as endpoints.

The ML algorithm has trained its model using driving behavior data (trip data) which is meant to measure how the driver has behaved for each trip. Initially the company had promised a dataset filled with actual trip data to be used for the thesis. Since the company that created the tools that gathers the trip data had delayed their release date of their product by a year, the data that was received in its current state lacked most of its core features such as hard braking count, over-speed count, rapid acceleration count, collision warning count and aggressive turn count. The trip data that is used for the ML model is therefore generated data that is designed to mimic the trip data used in the final release of the companies data gathering tool.

3.1.1 Generating Data

Generating data that covers driving behaviour needs to be accurate enough so that it is feasible in real life scenarios. A driver can not drive a million kilometers per hour as that would not make sense, therefore there had to be certain ranges decided beforehand on what values the data could be. The data was generated with the programming language Python. The data parameters were given by the company and were the real parameters they use in their cars. These parameters stems from the measurements that usage based car insurance companies are using [28]. They are meant to reflect important factors when driving that can showcase the risk of accidents that can cause additional costs to the vehicle. The parameters generated were:

- VIN : This is the identification number for each car and stands for Vehicle Identification Number. This was generated in the same format as real VIN:s but even if the fake VIN matches a real one, they have nothing in common.

- Date_(days) : This is the data that covers the date in days and is meant to show which day in the timeframe the car has made a trip or several trips. Each day could have several trips and the amount of trips was decided by a random distribution based on a statistic on 2.24 average trips per day [5]. The data was simulated on three months or 90 days, with random trips during each day for each car.
- TimeZoneLowRisk, TimezoneMediumRisk, TimeZoneHighRisk : These were binary attributes indicating if a certain trip is driven in a timezone with a low, medium and high risk of accident. If one of these attributes are 1, the others must be 0, which means the data does not cover two timezones in one trip. The risk was based on the most dangerous times of the day to drive and was received from the company. For instance 00:00 to 03:00 was considered to be most dangerous and the high risk zone, medium risk at 04:00 to 08:00 and low risk at 09:00 to 23:00.
- TripMeterKm : Covers how far the car drives in kilometers during a trip. The parameter is randomly generated from a normal distribution with hard coded numbers that is based on the type of driver the car has. The distribution was based on the European average trip length of 20 kilometers [16]. Since every other attribute is correlated with the amount of kilometers driven, all attributes is multiplied by the amount of kilometers driven.
- OverSpeedCount : This attribute covers how many times during a trip a car has driven faster than the allowed speeding limit. The speeding limit on the car's current road can be detected by the car and the car's sensors keep track of every time the driver is overspeeding. The acceptable limit that was decided was that less than three was acceptable, as most people drive faster than the limit at least a few times. If the amount was between three to five, then it would be a medium risk driver with a slightly higher risk of accidents. Everything over five times would be considered dangerous driving and is labeled as an aggressive driver with a higher risk of accident.
- HardbrakingCount : Tells how many times the car performs a hard braking maneuver, for instance, when braking fast in order to avoid a collision. The acceptable amount was set to a maximum of two times during an average trip of twenty kilometer, if the attribute exceeds 2 then it would be considered aggressive driving. The number two is arbitrary and could be changed, but it was decided that it was a reasonable number as a good driver should not have to brake hard often.
- RapidAcceleration : Shows how many times a car accelerate rapidly, with rapidly meaning almost full throttle. Sometimes a driver have to rapidly accelerate, for instance on ramps to a highway, it is therefore acceptable with a few of them. With ramps in mind the acceptable amount is three, everything over is considered aggressive, as during an average trip there are not that many ramps to a highway.
- CollisionWarnings : This is an attribute that covers how many times the car have to brake on its own, in order to avoid collision. The car has a sensor that

detects if collision is imminent and if true, the car will brake on its own. One time is acceptable during an average trip, this is because sometimes the car in front can brake suddenly and the sensor has better reflexes then the driver and will brake to avoid collision. If it happens more than once, it may indicate that the driver is often too close to cars in front which will be classified as aggressive driving.

- AggresiveTurnCount : The amount of times a driver turns too fast, which can be measured in angular velocity by the car. This can be very dangerous behaviour so only one time is acceptable per trip, as fast turning can lead to driving off the road and much more, it shows that the driver does not plan ahead when driving. One time is acceptable, as every driver could have to turn fast in order to avoid an unexpected event in some cases, for instance, a child running out on the road. A good driver could just make a mistake as well once, but more than one time could indicate aggressive behaviour.

3.1.2 Data Generation Algorithm

To gain a proper understanding of the code generation, an abbreviated snippet of the data generation code is shown below. Where for each car, a VIN ID is first generated, then the program randomly generates a number which represents which type of driver the car has. Each car will then loop for 90 days and then use a trip counter function that randomly generates how many trips the car will have that day.

The trip will then generate normalized values which are set based on the driver type that the car has. This means that passive drivers have good driving behavior, average drivers have average driving behavior and aggressive drivers have bad driving behavior.

```
carList = new Dataframe()
For each car:
    Vin = vinGenerator()
    Profile = randomNumber(min = 1, max = 10)
    If (Profile > 1 && Profile < 9) # Average
        While(date < 365)
        For in range (genTripAmount())
            Trip = New trip(
                "TripMeterKm" : max(0, numpy.random.normal(20,10)),
                #...
                )
            carList.insert(trip)
        Date += 1
    If (Profile == 1) # Agressive
        # ...
    If (Profile == 10) # Passive
        # ...
```

3.2 Architecture Plan

This section explains the architecture of the machine learning process that was built with Azure Kubernetes Service. The architecture, as seen in Figure 3.7, outlines the different steps in the process in the form of boxes, with each box representing pipelines starting from the creation to the deployment of a machine learning model.

3.2.1 Step 1, Data Processing

To create a model that can produce good results one would need to filter out as much false data as possible. Example of false data could be that the automatically assigned value when a value is NULL may be unreasonably high or low.

Other factors that may give unreasonable values are caused by bugs with the sensors that gathers the data. So data that has some missing values in some rows or columns may be removed as well.

Once the data has been filtered, the data will then be split up into two categories; one for training the model so it can learn, and the other for testing the model. The testing is done after training is completed and is meant to give an unbiased evaluation of the final model, by giving it new data and evaluating the predictions derived from it. The ratio of splitting the dataset can vary and depends on the size and type of the dataset.

If the dataset is medium sized, for example less than one million, then a 70%-30% split is recommended [1]. In smaller datasets you want to have around thirty percent for testing to get an accurate picture of the data as you want the testing to be representative of the whole dataset. A smaller number than thirty increases the risk of many outliers that skews the results, which one wants to avoid for accurate results. The testing percentage cannot be too high however as that could lead to underfitting. Underfitting means that the model cannot learn the patterns correctly due to the short amount of training, and therefore lead to poor predictions. For datasets larger than a million, the split can be 99%-1% , given that one percent of a million should be enough to accurately evaluate the model.

3.2.2 Step 2.1, Choosing Algorithm and Training the Model

Once the training datasets is filtered and split up, one can use the data together with a machine learning algorithm to train a model.

A supervised machine learning algorithm is used since this project is taking advantage of the labeled data of the driving behavior.

The algorithm that is chosen is typically dependant on the input and the output types of the data. If one for instance would want the output to be an object it may need a classification algorithm while if one wants the output type to be a continuous value it may need a regression algorithm [4]. Another key factor when trying to choose an algorithm is the problem that the machine learning model is

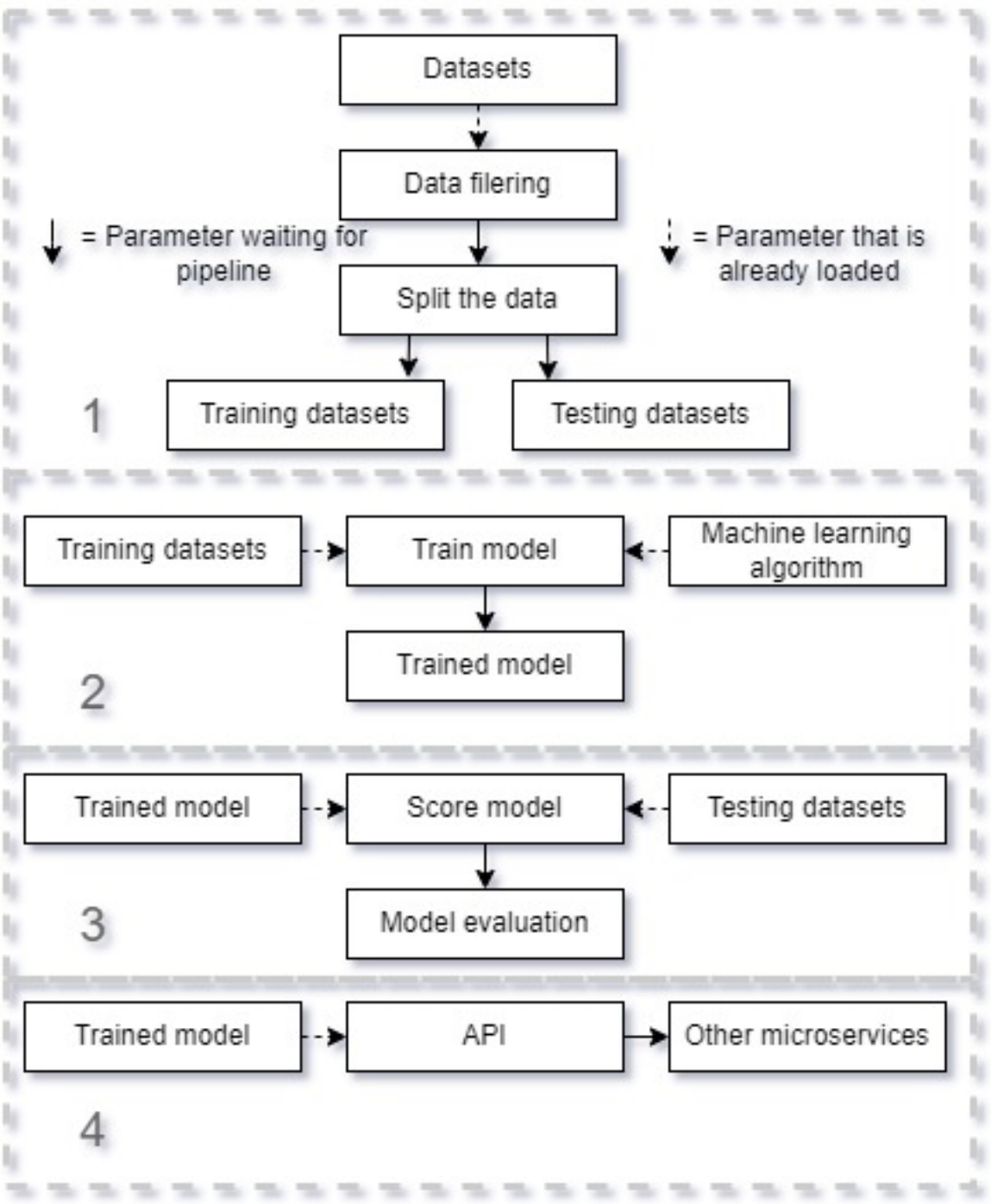

Figure 3.1: Is a sketch of the Architectural plan for the machine learning process along with its process to deploy the machine learning model in the API so that it then can be used for other micro services. Each container either represents an object value or a Pipeline and the arrows represent the values required in order to make the pipelines to start.

going to solve. That solution is typically heavily tied to a methodology which only a set number of algorithms may use [4].

3.2.3 Step 2.2, Algorithms to Choose from

Since the chance of a vehicle requiring additional service cost is slim, predicting the likelihood of something occurring would be significantly more understandable than predicting an expected service cost score. The output is therefore a binary classifier, returning a zero if total service cost for a car is below 10.000 SEK during a three month period and a one if it is above 10.000 SEK.

XGBoost is the machine learning algorithm that is used for this thesis because of its library documentation and accurate results without requiring large datasets [9]. The dataset used to create the machine learning model was highly imbalanced. Handling imbalanced data is a process that is typically highly demanding, but it is less demanding with XGBoost than many other algorithms due to its highly flexible library. If enough time is available to properly configure a machine learning model that supports an highly imbalanced dataset, then there may be a possibility for improvement in ROC and AUC scoring accuracy [20].

3.2.4 Step 3, Testing the Model

When the model has been trained and is complete, one must first test it so that it is assured that it works properly. Usually goals of testing include ensuring that a software behaves as expected and finding if there are any bugs, but with machine learning more steps are required. Given that the model learns a certain logic depending on the input data one must ensure that this logic stays consistent given new data, whilst guaranteeing that the logic is accurate. Therefore it is not enough to just test the model to find any faults, one also needs to evaluate it in order to make sure the model's predictions make sense.

3.2.4.1 Testing Parameters

The XGBoost classifier has a lot parameters that can have many different values, it can therefore be difficult to know which combination of these parameters gives the best result. We found the best parameters by testing a number of combinations of the parameter values and used the parameters that gave the best prediction in the final version. The combinations can be seen in Figure 3.2.

In order to save time; only the uncertain parameters max depth and minimum child weight were tested, as seen in Figure 3.3. The ones we could logically deduce were optimal did not have to be tested. For instance, the parameter called objective, which decides the learning objective of the classifier and affects the predicted output. The desired output was a probability and only the objective "binary:logistic" fulfilled this, therefore only one acceptable setting for that parameter existed and no others needed to be tested.

```
param_test1 = {
    'max_depth':range(3,10,2),
    'min_child_weight':range(1,6,2)
}
```

Figure 3.2: Depicts an example of testing that was made using different combinations for the maximum depth and minimum child weight of the trees.

```
gsearch1.fit(X_train,y_train)
gsearch1.best_params_, gsearch1.best_score_
```

```
({'max_depth': 5, 'min_child_weight': 5}, 0.7201497386510186)
```

Figure 3.3: The combination that gave the best prediction is returned by the function and is thus used in the final classifier.

3.2.5 Step 4.1, Deploying the Model to API

When the model has been trained and evaluated it is ready for use, given the microservice structure of the project it is recommended to have it deployed on Kubernetes.

Kubernetes normally allows internal communication between microservices by having API:s that communicate within Kubernetes (endpoint). This makes communication possible between the microservices since they can through these endpoints make API calls to run, configure and edit other microservices within its system.

If a developer would have a microservice architecture where we have a webapplication microservice made with the Framework react, a form can then be created where one uses the forms values as input. The webapplication microservice may then send an API call to a ML microservice where the inputs of the form is the input of the API call. The ML framework may then respond with an output which is then sent and processed by the webapplication in the end. To get a more clear picture of this explanation, see Figure 3.4.

Since the goal of this project is to only build an API and to let other people learn from our API, there are two versions of the project. The first where the project is used as a External API uploaded to a website available to everyone, as seen in Figure 3.6. The second where the machine learning microservice is designed to be used internally for a website that may be public, an example of this is seen in 3.5. This allows readers to both test and see what types of use cases there are for an machine learning micro-service so that the readers themselves may be encouraged to try it out in the future.

3.2.6 Step 4.2, Setting up the Kubernetes Deployment

Since online applications leveraged to users typically requires different resource demands depending on the time of day, it is typically recommended to use Kubernetes

as an deployment alternative. Kubernetes has different types of units, the smallest type of units being a pod.

These units are either ran on an external proxy server from a cloud provider or ran on a virtual server from a cloud provider called a node. All Kubernetes units can either be deployed from the command-line or through the use of YAML configuration files. These deployment files create Kubernetes pods by either importing a docker image available on the computer that the command-line is running on or by importing it from a container repository such as Dockerhub [15].

Here is an example of how the configuration of the master thesis pod deployment file looks like:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cost-estimation
  labels:
    app: cost-estimation
spec:
  selector:
    matchLabels:
      app: cost-estimation
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: cost-estimation
    spec:
      containers:
      - name: cost-estimation
        image: dockerName/cost-estimation:0.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8000
          name: http
          protocol: TCP
        livenessProbe:
          httpGet:
            path: /
            port: 8000
          initialDelaySeconds: 5
          timeoutSeconds: 1
```

```
          periodSeconds: 600
          failureThreshold: 3
        resources:
          requests:
            memory: "64Mi"
            cpu: "50m"
          limits:
            memory: "1000Mi"
            cpu: "1000m"
```

As seen in the snippet above the configuration file starts off by declaring which type of file it is. Then there is metadata where the instance is named and a label is added to that instance which may later be used to identify the instance of the deployment. The first set of specifications of the configuration file mainly revolves around how many pods are needed (replicas), how to handle a pod when it crashes or updates (strategy) and what the pod is going to be (spec). The spec configuration is mainly focused on configuration of the pod itself. It includes the location and port connection of the container image that it is going to import from, the expected resources that it is going to use and how to handle timeouts if they were to occur.

3.2.7 Step 4.3, Setting up the Docker Container

When any form of server is running whether it is a webapplication or a database, it is typically running on a specific operating system with a specific platform and typically requires a few command prompt commands to run. Docker automates this by running any type of application on smaller versions of MacOS, Linux or Windows inside of a container. It then autoruns a set of commands which the developer chooses through the use of a Dockerfile and then based on the configurations of the Dockerfile creates an image which can be used to run a new instance of the server. These images can then be used, deployed and autorun using Kubernetes either by importing the image from a repository such as DockerHub or by uploading it directly to the source of the Kubernetes instance.

Here is how the Dockerfile for the service cost estimation microservice looks like:

```
# Importing python version 3.10
FROM python:3.10

# Copying the current directory over to the image folder
# /user/src/app
COPY . /usr/src/app

# Changing base directory of the image to usr/src/app
WORKDIR /usr/src/app

# Installing all python libraries for FastAPI application
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Making the port 8000 visible for kubernetes.
EXPOSE 8000

# Running a command to create a FastAPI instance on port 8000.
CMD uvicorn working:app --host 0.0.0.0 --port 8000
```

3.2.8 Step 4.4, Connecting the Domain to an External IP

To connect a pod into an address that is available on the internet, the pods must be connected to a Kubernetes Network Service. The pod that is used in the thesis for instance, is called LoadBalancer which both offers an internal IP address which other Kubernetes services can connect to but also an External IP address that is accessible to the web as seen in Figure 3.8. An external IP address may also be connected to a domain by connecting the domains to a DNS A record, then to the external IP address as seen in Figure 3.9.

3.3 Tools

This section showcases the tools used in the project.

3.3.1 Scrum

The organization of the project was done according to the agile framework Scrum. Scrum is a framework which is built for complex adaptive problems. Instead of the old way of organizing projects where you only showcase a final product at the end of the project, the core principle of Scrum is that the work is divided into small time fragments. Each fragment should result in something that can be delivered to a customer instead of only delivering the final product at the end of the project. Each time fragment is called a sprint, the time period of a sprint can vary but in this project a length of two weeks was chosen. Two weeks is long enough for a substantial amount of work to be done, whilst short enough for being able to adapt in case something changes in the project. Before each sprint, the work to be done during the sprint is planned. At the end of a sprint, a sprint review takes place in which the project members and company supervisor reviews the work done.

In order to display the planned work a Scrum board was used on an application called Trello, which shows the tasks that need to be done and displays the progress done on each task. Scrum was used mainly because of its flexibility to adapt, which can occur during the frequent meetings. It also enables transparency for the company supervisor, as he could follow the process clearly and provide feedback during these meetings, as well as checking the progress on Trello during a sprint.

3.3.2 Database

The large amount of data that the machine learning model processed needed to be stored somewhere, therefore a database was used. The database was built in Visual Studio Code with a database management system called SQLite [34]. Lite in this case means that it is lightweight in the way it is setup, as it is simple to setup and require only a few resources. A good thing about SQLite is that it does not require a server to run, which helps keeping the access of data from being complex. Given that no server is required, installation and configuration of SQLite is not needed for the starting process to work. It is also self-contained in the sense that it does not need much support from the operating system, this makes it flexible as it can work in most environments.

In the database you can add, delete and update objects that are needed. The data was stored in tables with rows and columns. SQLite utilizes something called dynamic types for its tables, which makes it so that you can store any type in a column, no matter what the data type is. This is useful when dealing with several different data types in your database. Swagger UI could utilize the functions from Visual Studio Code that affects the database in real-time, which made it simple to update the data.

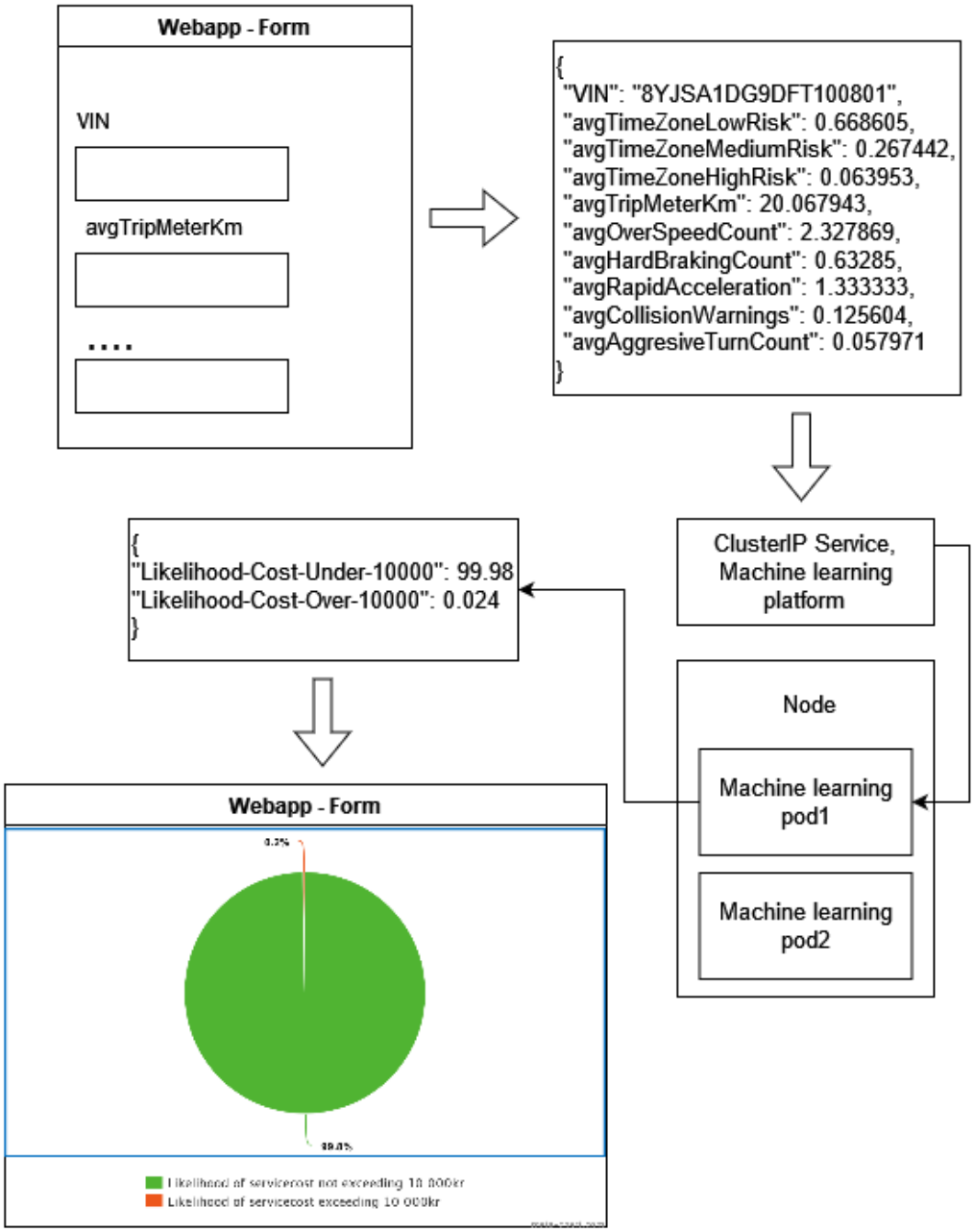

Figure 3.4: A picture which represents an example of how a webapplication can communicate internally to a machine learning API by through GET and POST API calls through the use of the JSON language via a ClusterIP service.

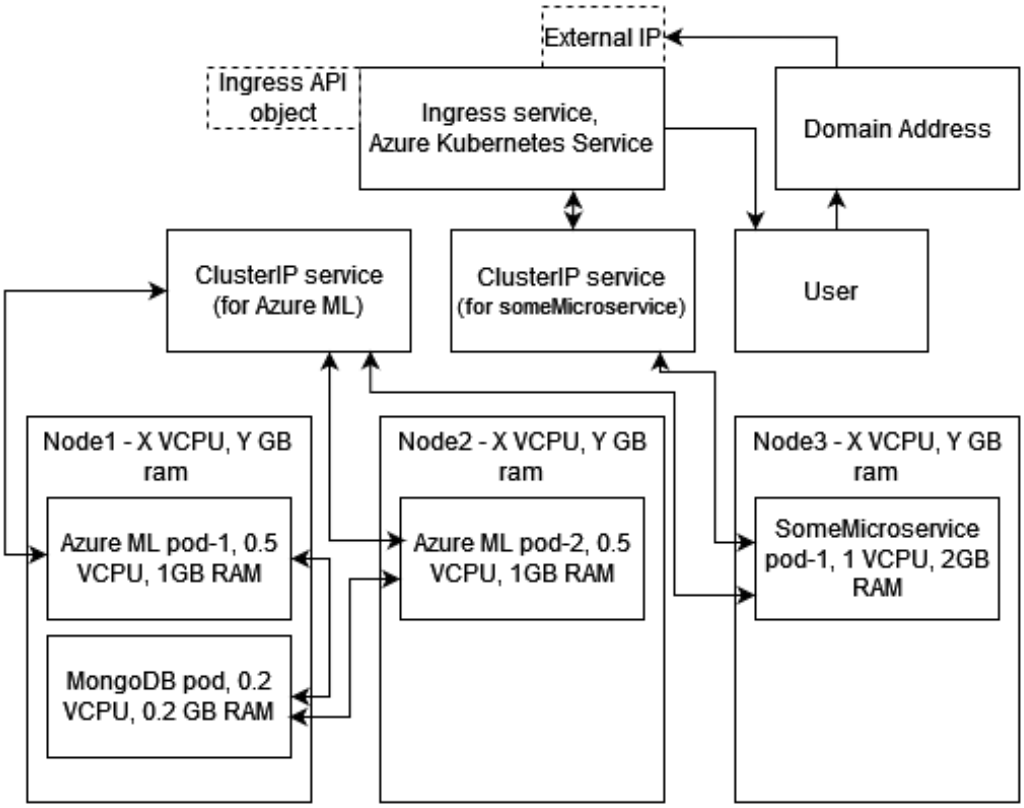

Figure 3.5: The ideal Kubernetes architecture that is suggested to be used in a real world production case. Where the machine learning platform is made on Azure ML instead of FastAPI and where other microservices use the machine learning framework internally. To then afterward forward that internal connection into an external one with the added security comming from the Ingress service.

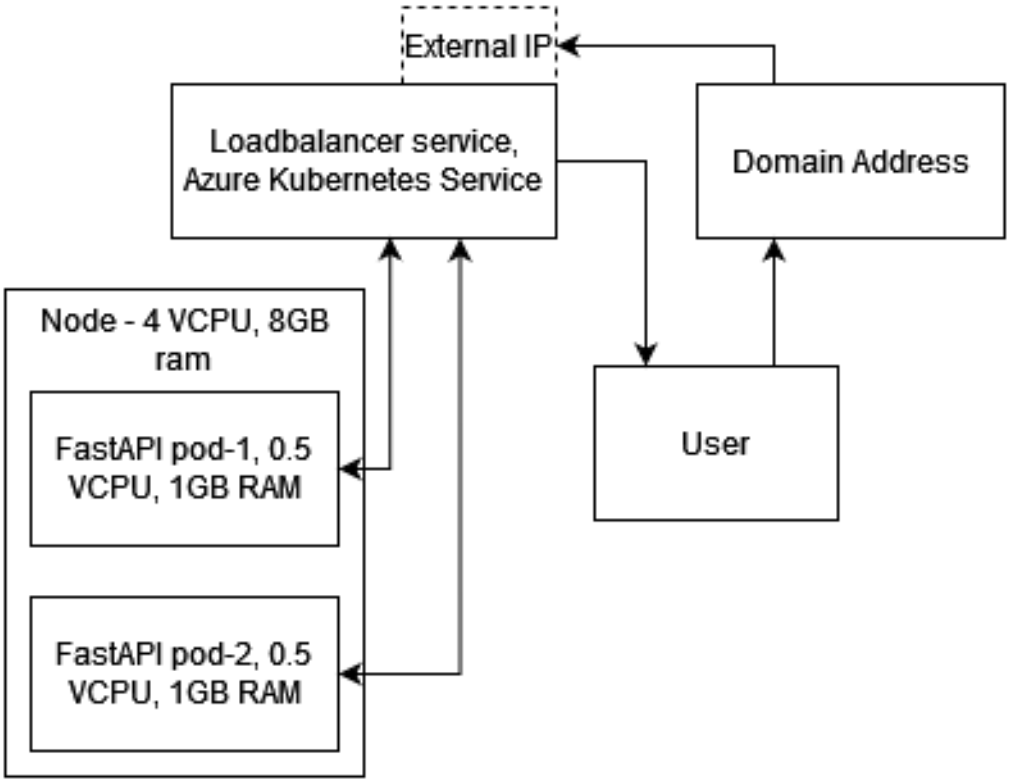

Figure 3.6: The Microservice architecture that is deployed for the master thesis.

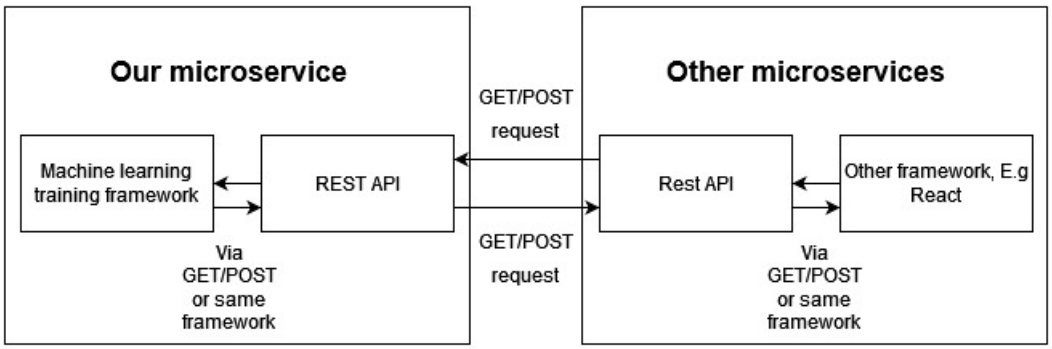

Figure 3.7: Is the architecture of the microservice depicting the connection with the company's REST API and its front end web application.

```
/work # kubectl get services -n cost-estimation
NAME                        TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)
cost-estimation-service     LoadBalancer   10.0.51.225    20.240.9.35    80:30013/TCP
```

Figure 3.8: Running a terminal command to get all the services which exposes all of the IP address and ports that are open.

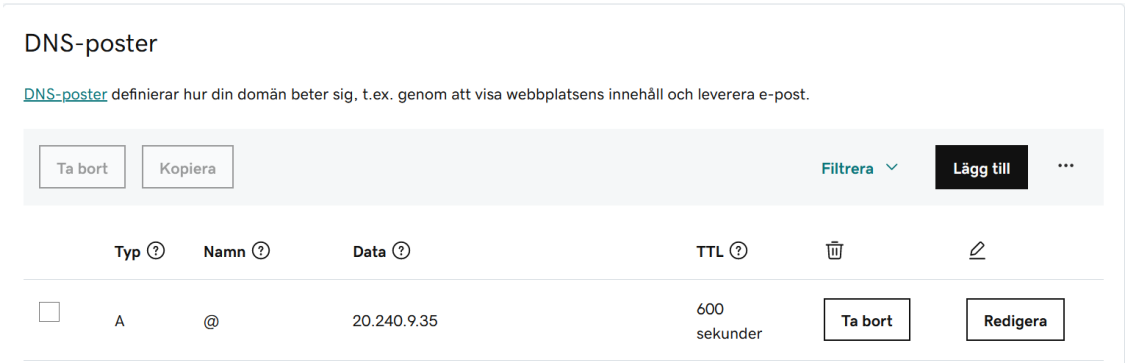

Figure 3.9: Connecting the external IP-address of the LoadBalancer used in the master thesis to the A record in the DNS settings of a domain.

4

Results

As mentioned in the introduction, the goal with this thesis is to make an analysis that can protect assets within the automobile industry from operating costs caused by driving. This is done through the use of a machine learning micro-service which uses a model that can predict operating cost based upon a vehicle drivers driving behavior.

In order to ensure that the predictions of the driving behavior is valid, an evaluation of the machine learning model will be shown.

Once the predictions are available, car companies can analyse how customers drive their cars. How this analysing can be done will be discussed in chapter 5.

4.1 Machine Learning Model Prediction results

The evaluated results of the model is shown in Table 4.1, containing six different generated datasets with different amount of trips per car, different amount of cars and different likelihoods of causing service cost.

To balance and to find the best choice for the priority between having false positives and true positives, a ROC curve is used, as seen in Figure 2.8, where each value shows the accuracy for false positives and true positives for each type of balancing that has been made. The AUC score is a scoring of the flexibility of prioritizing true positive and false positives without making the F1 score worse. See Section 2.2.1 for more information about these metrics.

The imbalance ratio in Table 4.1 represents the amount of cars that does not require additional operating cost for every car that requires it. An imbalance ratio of 1:30 would therefore be 30 cars that does not require additional operating cost for every car that does require additional operating cost. The imbalance ratio is dependant on the amount of trips each car has driven since the risk of needing additional service cost increases for every trip that is made. Since the dataset is generated rather than being real, different likelihoods of causing operating cost were also tested to see if that would potentially increase the evaluation scoring of the dataset. The imbalance ratio is also dependant on the likelihood of actually needing operating cost, if the likelihood of operating cost factor was to go from 1.0 to 3.0, the imbalance ratio would, for instance, go from 1:27 to 1:9.

Time driven per car (days)	Amount of cars	Likelihood of operating cost	Imbalance ratio	precision	f1 score	AUC
730	10000	1.0	1:27	0.91	0.71	0.77
730	10000	3.0	1:9	0.93	0.64	0.72
365	10000	1.0	1:40	0.91	0.41	0.58
365	20000	1.0	1:40	0.90	0.60	0.60
180	10000	1.0	1:75	0.96	0.20	0.57
90	10000	1.0	1:122	0.99	0.00	0.53

Table 4.1: Illustrates the evaluation of six different imbalanced datasets trained on the same extreme gradient boosting algorithm machine learning algorithm.

In Table 4.2, the best true/false ratio of the predictions is shown. It shows the model's accuracy in predicting the correct class for both classes.

	Predicted Negative	Predicted Positive
Actual Negative	2666	237
Actual Positive	41	56

Table 4.2: A Confusion matrix showing the true/false ratio for both the positive and negative classifier for the XGBoost model that was made. The generated dataset had 10000 cars with an average runtime of 2 years for each car.

In Tables 4.3 to 4.8, the other generated datasets' true/false ratios are shown. The implication of these tables is discussed in chapter 5. An illustration of the true/false ratio in the form of a bar can be seen in Figure 4.1

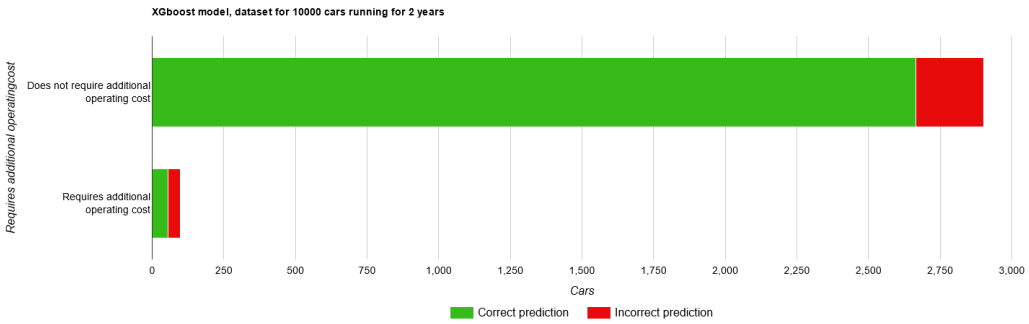

Figure 4.1: A stacked bar illustration of true/false ratio for the same dataset of 10000 cars running for 2 years. (Same dataset used in table 4.2)

	Predicted Negative	Predicted Positive
Actual Negative	2666	237
Actual Positive	41	56

Table 4.3: A confusion matrix that illustrates the true/false ratio for both the negative and the positive classifier for the XGBoost model. The generated dataset had an average run-time of 2 years, used 10000 cars and had 1.0 times the normal crash risk

	Predicted Negative	Predicted Positive
Actual Negative	2516	176
Actual Positive	159	149

Table 4.4: A confusion matrix that illustrates the true/false ratio for both the negative and the positive classifier for the XGBoost model. The generated dataset had an average run-time of 2 years, used 10000 cars and had 3.0 times the normal crash risk

	Predicted Negative	Predicted Positive
Actual Negative	2690	242
Actual Positive	50	18

Table 4.5: A confusion matrix that illustrates the true/false ratio for both the negative and the positive classifier for the XGBoost model. The generated dataset had an average run-time of 1 year, used 10000 cars and had 1.0 times the normal crash risk

	Predicted Negative	Predicted Positive
Actual Negative	5272	579
Actual Positive	82	67

Table 4.6: A confusion matrix that illustrates the true/false ratio for both the negative and the positive classifier for the XGBoost model. The generated dataset had an average run-time of 1 year, used 20000 cars and had 1.0 times the normal crash risk

	Predicted Negative	Predicted Positive
Actual Negative	2846	109
Actual Positive	40	5

Table 4.7: A confusion matrix that illustrates the true/false ratio for both the negative and the positive classifier for the XGBoost model. The generated dataset had an average run-time of 6 months, used 10000 cars and had 1.0 times the normal crash risk

	Predicted Negative	Predicted Positive
Actual Negative	2952	26
Actual Positive	22	0

Table 4.8: A confusion matrix that illustrates the true/false ratio for both the negative and the positive classifier for the XGBoost model. The generated dataset had an average run-time of 3 months, used 10000 cars and had 1.0 times the normal crash risk

4.1.1 Kubernetes resource usage

The Kubernetes cluster used for the thesis was running on Azures Standard_E4s_v3 node server. The node included 4 VCPUs which is equivavelent to 4 threads/2 cores of an AMD EPYC 7763v-processor, 32 GB of ram and 150 GB of hard drive space. It was however discovered that significantly less resources were required than previously thought for the Kubernetes cluster to work as seen in Table 4.9. This also means that more pods would be deployed once the API pods become overloaded.

Service	VCPU	Size	RAM	Cost (USD/m)
FastAPI Pod 1	0.650	2.86GB	1GB	31.04
FastAPI Pod 2	0.650	2.86GB	1GB	31.04
ClusterIP	0.080	<0.1GB	<0.1GB	2
Loadbalancer	0.100	<0.1GB	<0.1GB	2

Table 4.9: A table illustrating the cost of Azures resource use with the most suitable type of virtual machine.

5
Conclusion

5.1 Discussion

In the hypothesis of the thesis, we asked ourselves if it was possible to make an analysis that could predict service cost for vehicles if trip data was available. This hypothesis was attempted to be proved through an evaluation of a machine learning model using data that is designed to emulate trip data.

5.1.1 Data Imbalance

The likelihood of an average person crashing is not very high during an average trip of 20 kilometers. Because of this only a few driving profiles in the data will result in operating costs. This means that there is a substantial imbalance in the data. More than 99.95% of the data belongs to class 0, which is the no-cost class. This imbalance led to problems in the model training, as it is more difficult for the model to learn the class 1 given the low numbers. Another problem with such a large imbalance is that the model can become biased towards the majority class. If, for instance, the model only predicts the majority class it will have a very high accuracy, but will fail to identify any costs, which is the purpose of the model.

To alleviate this issue, we combined multiple trips into one trip that took the average out of all those trips for a driver. We also increased the number of days to increase the range where possible accidents can occur. These two steps was done to increase the number of operating costs in the data without altering the odds of crashing to unrealistic proportions. This in turn led to a higher percentage of costs in the data than before, which meant the model improved its accuracy for class 1.

The data imbalance went from a tiny percentage (less than 0.05%) to approximately 0.5% to 8% depending on the amount of trips each car had. This means that a significant amount of data was sacrificed to increase the datasets imbalance issue. But a data imbalance, while more acceptable now, still exists and must be considered when evaluating the model's performance. If one only test the model's accuracy for all of the data it would result in a very high percentage of accuracy. But this exceptional accuracy would be misleading as the accuracy is this high because most of the data is no-cost and therefore logical that the model predicts such. To truly test the model's accuracy, one wants to see how it performs when predicting the class with operating costs and evaluate that accuracy. That is why the main part of the testing was done with the metrics AUC ROC and F1 score, as they measure the accuracy of the class 1 predictions.

5.1.2 ML Result Interpretation

From all of the tests gathered and from the results in the model evaluation (Table 4.1) we can see that the prediction accuracy showed better results once the imbalance ratio was greater than 1 in 30 cars needing additional operating cost. We can also see that more data resulted in a significant increase in accuracy. This means with this dataset structure that we have, once the imbalance ratio of the data reaches below 30, the only way to increase the accuracy of the results would be to increase the amount cars being used for the dataset.

We can see that the best test case was where we generated 10000 cars that had been running an average time of two years. The results showed an AUC score of approximately 0.77 and a F1 score of 0.71. The best model resulted in a 57% accuracy in finding vehicles with additional operating costs and a 92% accuracy in finding vehicles without additional operating costs (as seen in Figure 4.1). This is very good results given that the generated data is not designed to be directly correlated to the output of the result.

Since our driving behavior data covers the likelihood of additional operating cost and does not cause the operating cost itself, the likelihood of operating cost is what driving behavior changes rather than having a formula which can find exactly when additional operating cost will occur. The likelihood for operating cost used for the thesis is based on driver profile times the driving mile-age.

The aggressive driver profile represents 10% of the dataset and the total amount of predicted operating cost drivers was 9.6%. Having an accuracy that is below 10% requires the operating cost likelihood to be correlated with other factors other than driver profile as well, which as of now only is the amount of kilometers that has been driven.

In response to the hypothesis of the thesis, it is possible to create an analysis that can predict service cost of vehicles with the use of trip data, as long as the trip data has any form of implication to additional service cost risk. According to our earlier sources where we gathered the likelihood from the generated dataset, the empirical evidence claims that bad driving behavior causes more accidents. This means that as long as the sensors gathered from the trip data can accurately gather driving behavior, the machine learning model in this thesis should be able to predict operating cost with a good accuracy.

5.1.3 Importance of result

The results of this thesis show that it is possible to make a mostly accurate ML model in regards to predicting operating costs. With the model returning the odds of operating costs exceeding 10000 SEK, the main value of the model is for analysis purposes. It can, for instance, be used by car companies who want to protect the value of their cars by identifying potential bad drivers. The companies can analyse the percentages and do as they wish with the corresponding driver.

Another value with this work is that it has built a foundation for further work. If real data ever becomes available, the model can then easily be improved. If in the

future a better algorithm than XGBoost is developed, then that algorithm could simply replace XGBoost and potentially improve the accuracy.

We can with the results see that the machine learning algorithm extreme gradient boosting can provide accurate predictions with datasets that contains attributes which depends on other attributes. We can also see that we can predict how a set of data values can predict the likelihood of a certain condition being met.

While previous products such as Tesla's safety score[40] has focused on making scoring systems, these results demonstrate that correlating scoring systems with other factors such as vehicle service cost is also possible.

5.1.4 Limitation of results

Due to the lack of real data collected from sensors, the results cannot confirm if having bad trip data results gathered from said sensors will in fact cause higher operating costs. This is an essential factor since the machine learning model may be useless in its current state if the hypothesis is not true.

The choice of making generated datasets instead of using real datasets were constrained by not being able to know if there is were any quality assurance of the data gathered from the sensors. Other constraints of the analysis was that we did not know whether the data will be stored in a database or if it would to be collected in real-time through some form network.

It is beyond the scope of this study to compare other machine learning models to see which model provides the best accuracy.

5.1.5 Use cases

While usage-based insurance for single cars exists where insurance discounts could reach as far as 30%, this is still not common when purchasing car insurance in bulk. It is instead used as a bargaining tool in contracts to decrease insurance costs by a few percentage points instead.

This means that lower service costs for a subscription based automotive company provide more bargaining power for insurance discounts and allows the company to avoid the additional costs such as paying for a replacement cars to a driver. In this subsection, some measurements that would allow a subscription based car company to decrease its service cost will be mentioned.

5.1.5.1 Safety score

The model could be used to calculate a safety score, instead of just a percentage of cost occurring. This score could be a number ranging from 0-100 and be displayed in the car for the driver to see. Examples of companies that use safety score are Tesla [45] and Toyota [46]. The performance of drivers can improve if they see a score on how well they drive. There are many people who are competitive and would want to get a higher score than their peers. In order for them to get a higher score they need to improve their driving, this is an advantage with displaying the score

to the drivers. Good scores could also result in small rewards, for instance, Tesla provides access to Full Self-Driving beta for those with a safety-score above 95 [2]. This kind of positive encouragement would encourage drivers to perform well and maintain that level.

There have been some issues with this scoring method however, in regards to displaying the score to the drivers. If the drivers can see their score all the time and discover what driving maneuvers result in a good score, they can exploit it. This happened with Tesla where some drivers found and posted an exploit that resulted in a perfect score [44]. The drivers could just restart the system and all their latest errors in traffic would be erased. They could also manipulate how the scoring was calculated by driving in a specific way, in order to increase the scoring. These exploits led to dishonest scores that showed people driving better than they were doing in reality. This problem could be helped by only showing a score of the latest month, instead of real-time, making it harder for the drivers to find the exploits that affect the score. The score could also be removed and be hidden from the drivers completely, but would also remove the positive encouragement from seeing their score.

5.1.5.2 Hidden scores with positive affirmation

It is not only leasing companies that can take advantage of the model, but insurance companies as well. Since insurance companies need assurance that good trip data really correlates with lower service costs, it could be optimal to even hide driver scores from the users to prevent users from abusing any form of bugs that may bring up their scores. The idea is that instead of giving users live feedback of how they behaved, the company can give them temporary rewards if their driving behavior has become better for a specific period and then permanent rewards once they go above a specific threshold.

The leasing companies can convince the insurance companies to have the same usage-based cost reductions that single drivers currently have (up to 30%). If we assume that the subscription car companies pays 1000$ for the insurance on a yearly basis, up to around an additional 300$ each year per car could potentially be saved. If one were to assume that the average gas consumption of a car is 1500$ each year, it would mean that a 10% gas discount could be an example of an award that could be given temporarily for drivers with an upward trajectory in driving behavior and permanently to drivers with perfect driving behavior. If this positive affirmation tactic would work, it would win the company 150$ per year for every perfect driver just from the insurance company and most likely more money saved from the replacement cars that has to be sent during service time as well.

Since changing current drivers driving behaviour is a difficult task, marketing to people with good driving behavior is also an option. One example of this is to find in which specific areas good driving behavior is more common and then target all of the ads in those specific areas to maximize the amount of drivers with good behavior.

5.2 Conclusion

This research aimed to give analytical strategies for the automotive companies to decrease operating costs. These analytical strategies are done through the use of ML models which were created to predict operating costs through the use of trip data. Based on an evaluation designed for ML models with imbalanced data, it can be concluded that it is possible to create a ML model that can predict operating costs with a reasonably good accuracy. The results indicate that the accuracy may be significant enough to be used for production as long as the imbalance ratio of the data does not exceed a 30 to 1 ratio for the ML model. The results also indicated that using more vehicles in the dataset would significantly increase the accuracy of the results as well.

5.2.1 Potential Improvements

Machine learning is often used and is meant to be used with real data, as data is more realistic when it comes from real life situations. However in our case there was no data available and data had to be generated by ourselves. Even though the data was generated with the help of statistics on driving and reasonable numbers, it is not as good as real data would be. The realism of the predictions would most certainly improve if real data was used instead. If real data ever becomes available in the future, further work could be done to improve the project with that data.

One of the reasons we used XGBoost was because of the good accuracy with its predictions whilst being simple to implement. However in machine learning there are a multitude of algorithms to choose from and we did not have the time to try them all. If there were more time, a more complex algorithm could be used, which could have led to a longer computation time but potentially more accurate predictions. One example of a system of algorithms we could have used is fully connected neural networks, which could take advantage of the trip data that had to be rounded down to average trip data to match XGBoost's input format. One big issue with neural networks however is that the trip data will still remain extremely imbalanced. This means that additional measurements need to be taken to make the model compatible with an imbalanced dataset.

5.2.2 Future Work

There is a lot of potential in the machine learning industry that can be expanded upon from this work. One potential development which would be useful to study would be to take advantage of trip data sequential time order for each car. This can possibly be done through the use of fully connected neural networks (FCNN) and could potentially increase the accuracy of the predictions by a significant amount.

Another potential development in the future could be utilized within the car insurance industry. The insurance company could track their customers' driving behaviour and then use a ML model to review their performance. If a customer drives well in traffic, they could get rewarded with a lower fee on their insurance. On the other hand, if a customer performs poorly in traffic, then that person could get a

warning or even a higher fee on the insurance. This would encourage good driving behaviour and could also lead to less traffic incidents, which would help both the insurance company and the customer.

A third potential development is within eco-driving, which means driving in an environment-friendly way. The focus of this thesis however was on factors that can affect the operating costs of a vehicle, for instance, overspeeding. But in the future, the factors could be changed to focus on behaviour that affects the fuel consumption instead. These factors could be, for example, how often a driver changes gear, keeping a steady speed or how often the driver is accelerating. A ML model could then be used to predict how environment-friendly a person drives. This could help individuals and companies analyse their effect on the environment, which may lead to people lowering their fuel consumption and helping the environment.

Bibliography

[1] Afshin Gholamy, Vladik Kreinovich, Olga Kosheleva. Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation, 2018.

[2] Aguirre, J. (2022, March 15). Tesla is getting ready to expand the FSD beta to owners with a safety score of 95+. Not a Tesla App. Retrieved May 23, 2022, from https://www.notateslaapp.com/news/722/tesla-is-getting-ready-to-expand-the-fsd-beta-to-owners-with-a-safety-score-of-95-plus

[3] Algorithmia. (2020). 2020 state of enterprise machine learning. Retrieved February 22, 2022, from https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf

[4] Almaliki, Z. A. (2022, January 5). Do you know how to choose the right machine learning algorithm among 7 different types? Medium. Retrieved February 26, 2022, from https://towardsdatascience.com/do-you-know-how-to-choose-the-right-machine-learning-algorithm-among-7-different-types-295d0b0c7f60

[5] "American Driving Survey, 2015-2016" Foundation for Traffic Safety. (n.d.). Methods - AAA Foundation for Traffic Safety. Retrieved March 29, 2022, from https://aaafoundation.org/wp-content/uploads/2018/02/18-0019_AAAFTS-ADS-Research-Brief.pdf

[6] Atiyeh, C. (2021, November 8). Car Subscription Services: The Complete Guide to getting the car you want. Forbes. Retrieved May 4, 2022, from https://www.forbes.com/wheels/advice/car-subscription-services/

[7] Bhalla, D. (2015, April). Decision tree in R : Step by step guide. ListenData. Retrieved May 13, 2022, from https://www.listendata.com/2015/04/decision-tree-in-r.html

[8] Brown, S. (2021, April 21). Machine Learning, explained. MIT Sloan. Retrieved May 2, 2022, from https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[9] Chen, T., amp; Guestrin, C. (n.d.). XGBoost: A scalable tree boosting system,(2016) - arxiv. XGBoost: A Scalable Tree Boosting System. Retrieved April 28, 2022, from https://arxiv.org/pdf/1603.02754

[10] Cloud Native Computing Foundation. (2022, April 18). Ingress. Kubernetes. Retrieved April 28, 2022, from https://kubernetes.io/docs/concepts/services-networking/ingress/

[11] Cloud Native Computing Foundation. (2022, April 18). Ingress. Kubernetes. Retrieved April 28, 2022, from https://kubernetes.io/docs/concepts/services-networking/ingress/

[12] Cloud Native Computing Foundation. (2022, April 18). Ingress. Kubernetes. Retrieved April 28, 2022, from https://kubernetes.io/docs/concepts/services-networking/ingress/

[13] Cloud Native Computing Foundation. (2022, April 23). Nodes. Kubernetes. Retrieved April 28, 2022, from https://kubernetes.io/docs/concepts/architecture/nodes/

[14] Cloud Native Computing Foundation. (2022, April 23). Nodes. Kubernetes. Retrieved April 28, 2022, from https://kubernetes.io/docs/concepts/architecture/nodes/

[15] Docker, Inc. (n.d.). Docker Hub. Retrieved April 29, 2022, from https://hub.docker.com/

[16] Fiorello, D., Martino, A., Zani, L., Christidis, P., amp; Navajas-Cawood, E. (2016). Mobility data across the EU 28 member states: Results from an extensive Cawi Survey. Transportation Research Procedia, 14, 1104–1113. https://doi.org/10.1016/j.trpro.2016.05.181

[17] Fortune business insights. (n.d.). Car leasing market size, share amp; covid-19 impact analysis, by application type (business use and personal use), by lease type (open-end, and close-end) and regional forecast, 2022-2029. Car Leasing Market Size, Share, Analysis | Industry Trends, 2029. Retrieved April 26, 2022, from https://www.fortunebusinessinsights.com/car-leasing-market-105417

[18] Gillis, A. S. (2020, September 22). What is Rest Api (restful API)? SearchAppArchitecture. Retrieved June 29, 2022, from https://www.techtarget.com/searchapparchitecture/definition/RESTful-API

[19] Gong, D. (2022, February 25). Top 6 machine learning algorithms for classification. Medium. Retrieved February 27, 2022, from https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501

[20] Google. (n.d.). Classification: Roc curve and AUC | machine learning crash course | google developers. Google. Retrieved May 10, 2022, from https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[21] Hirani, A. (n.d.). Automotive industry profitability for 2021 and Beyond. ZT Corporate. Retrieved May 4, 2022, from https://ztcorporate.com/blog/determining-automotive-industry-profitability-for-2021-and-beyond/

[22] IBM Cloud Education. (n.d.). Django-explained. IBM. Retrieved June 29, 2022, from https://www.ibm.com/cloud/learn/django-explained

[23] Kapalko, R. (2019, November 7). Predictive model ensembles: Pros and cons. Perficient Blogs. Retrieved June 10, 2022, from

https://blogs.perficient.com/2019/11/07/predictive-model-ensembles-pros-and-cons/

[24] Kopestinsky, A. (2022, March 5). Global and US Auto Sales Statistics for 2021: Policy advice. PolicyAdvice. Retrieved April 26, 2022, from https://policyadvice.net/insurance/insights/us-auto-sales-statistics/

[25] Lund, N. (2019, December 2). Elgiganten Efter Helgens Haveri: "Påverkat E-handeln NEGATIVT". Ehandel.se. Retrieved April 28, 2022, from https://www.ehandel.se/elgiganten-efter-helgens-haveri-paverkat-e-handeln-negativt

[26] Lutins, E. (2017, August 2). Ensemble Methods in Machine Learning: What are they and why use them? Medium. Retrieved March 4, 2022, from https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f

[27] Martins, D. (2021, December 14). XGBoost: A complete guide to fine-tune and optimize your model. Medium. Retrieved March 10, 2022, from https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663

[28] Metz, J. (2022, May 18). How usage-based car insurance works. Forbes. Retrieved May 23, 2022, from https://www.forbes.com/advisor/car-insurance/usage-based-insurance/

[29] Morde, V. (2019, April 8). XGBoost algorithm: Long may she reign! Medium. Retrieved March 10, 2022, from https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

[30] Myles, A. J., Feudale, R. N., Liu, Y., Woody, N. A., amp; Brown, S. D. (2004, September 7). An introduction to decision tree modeling - myles - 2004 . Retrieved March 4, 2022, from https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/abs/10.1002/cem.873

[31] Narasimhan, K. A. (2021, February 5). Why linear regression is not suitable for classification? Medium. Retrieved May 2, 2022, from https://medium.com/analytics-vidhya/why-linear-regression-is-not-suitable-for-classification-cd724dd61cb8

[32] Networks, A. (n.d.). What is microservices? microservices definition and ... - Avi Networks. Retrieved June 29, 2022, from https://avinetworks.com/glossary/microservice/

[33] Niklas Kuhl, Marc Goutier, Lucas Baier, Clemens Wolff, Dominik Martin, 2020. Human vs. supervised machine learning: Who learns patterns faster?

[34] Owens, Michael. The Definitive Guide to SQLite, 2006.

[35] Pafka, S. (2015, November 22). Benchmarking random forest implementations: Data Science Los Angeles. DataScience.LA. Retrieved April 28, 2022, from http://datascience.la/benchmarking-random-forest-implementations/

[36] Peixeiro, M. (2021, December 6). Defining the moving average model for time series forecasting in Python. Medium. Retrieved May 15, 2022,

from https://towardsdatascience.com/defining-the-moving-average-model-for-time-series-forecasting-in-python-626781db2502

[37] Qimono. (n.d.). Speedometer-1249610_1280. Pixabay. Retrieved May 27, 2022, from https://cdn.pixabay.com/photo/2016/03/11/02/08/speedometer-1249610_1280.jpg.

[38] Rajbhoj, A. (2022). ARIMA simplified.. towardsdatascience.com. Retrieved 1 March 2022, from https://towardsdatascience.com/arima-simplified-b63315f27cbc

[39] Rasmussen, J. T. (2022, January 9). How to use an autoregressive (AR) model for time series analysis. Medium. Retrieved May 15, 2022, from https://towardsdatascience.com/how-to-use-an-autoregressive-ar-model-for-time-series-analysis-bb12b7831024

[40] Safety score beta. Tesla. (2022, February 25). Retrieved May 17, 2022, from https://www.tesla.com/support/safety-score

[41] Saini, A. (2021, September 15). AdaBoost algorithm - A complete guide for beginners. Analytics Vidhya. Retrieved June 29, 2022, from https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/

[42] Singh, H. (2018, November 4). Understanding gradient boosting machines. Medium. Retrieved March 6, 2022, from https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab

[43] Storchmann, K. On the Depreciation of Automobiles: An International Comparison. Transportation 31, 371–408 (2004). https://doi.org/10.1023/B:PORT.0000037087.10954.72

[44] Szymkowski, S. (2021, October 21). It's easy to Game Tesla's safety score system, owners say. CNET. Retrieved May 23, 2022, from https://www.cnet.com/roadshow/news/its-easy-to-game-teslas-safety-score-system-owners-say/

[45] Tesla. (2022, February 25). Safety score beta. Tesla. Retrieved May 23, 2022, from https://www.tesla.com/support/safety-score

[46] Toyota. (n.d.). Search. What is Driver Score? Retrieved May 23, 2022, from http://toyota.custhelp.com/app/answers/detail/a_id/10536/ /what-is-driver-score

[47] Waterloo, X. C. U. of, Chu, X., Waterloo, U. of, Ilyas, I. F., Berkeley, S. K. U. C., Krishnan, S., Berkeley, U. C., University, J. W. S. F., Wang, J., University, S. F., Research, I. B. M., Labs, H. P., & Technology, M. I. of. (2016, June 1). Data Cleaning: Proceedings of the 2016 International Conference on Management of Data. ACM Conferences. Retrieved May 2, 2022, from https://dl.acm.org/doi/pdf/10.1145/2882903.2912574?casa_token=NnEg849L_nEAAAAA%3A
R0aXpHCwTDROOUnstBY7xEarONuYacsQHwTgmxHzpBvZExaA3sKD6

[48] What is Docker? Opensource.com. (n.d.). Retrieved June 29, 2022, from https://opensource.com/resources/what-docker

[49] What is kubernetes? Kubernetes. (2022, April 3). Retrieved June 29, 2022, from https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[50] Wikimedia Foundation. (2022, April 25). Sigmoid function. Wikipedia. Retrieved May 2, 2022, from https://en.wikipedia.org/wiki/Sigmoid_function

[51] Yildirim, S. (2020, July 27). 11 most common machine learning algorithms explained in a Nutshell. Medium. Retrieved March 1, 2022, from https://towardsdatascience.com/11-most-common-machine-learning-algorithms-explained-in-a-nutshell-cc6e98df93be

[52] Zamboni, J. (2019, March 2). What is the meaning of sample size? Sciencing. Retrieved May 2, 2022, from https://sciencing.com/meaning-sample-size-5988804.html

[53] Zavarella, L. (2019, February 5). How to better evaluate the goodness-of-fit of regressions. Medium. Retrieved March 1, 2022, from https://medium.com/microsoftazure/how-to-better-evaluate-the-goodness-of-fit-of-regressions-990dbf1c0091

[54] Zhou, Z.-H. (2012). Ensemble methods: Foundations and algorithms. Taylor amp; Francis.