Tampere University

Samu Jussila

WORKSITE DATA ANALYSIS USING CLOUD SERVICES FOR MACHINE LEARNING

Faculty of Information Technology and Communication Sciences
Master's thesis
October 2019

ABSTRACT

Samu Jussila: Worksite data analysis using cloud services for machine learning
Master's thesis
Tampere University
Software Engineering
October 2019

This thesis studies utilizing machine learning in cloud services. The aim of the thesis was to create a machine learning pipeline in a cloud service platform and use the pipeline to test how well worksite data could be used for machine learning. The data was operational data already existing in a database, and the test done was predicting a worksite's time of completion based on its state and history using different machine learning models.

The pipeline was built in Amazon Web Services, and consisted mainly of the services Glue and SageMaker. Glue was used to transform the data into a format readable by machine learning models, and SageMaker was used to implement the machine learning models. Amazon Web Services had a ways of creating all the needed parts of the pipeline, and the amount of boilerplate code was minimal. The implementation was sufficiently easy with no large setbacks.

The machine learning models implemented were linear regression, random forest, XGBoost, and artificial neural network. The model performance results show that the best model for this problem is a two-hidden-layer artificial neural network, but the results vary drastically by the method the whole data set is split into training and test data.

Keywords: Machine learning, cloud services, Amazon Web Services, regression, comparison, pipeline

TIIVISTELMÄ

Samu Jussila: Työmaadatan analysointi käyttäen pilvipalvelun tarjoamaa koneoppimista
Diplomityö
Tampereen yliopisto
Ohjelmistotuotanto
Lokakuu 2019

Tässä työssä tutkitaan koneoppimispalvelujen käyttöä pilvipalvelualustalla. Työn tavoitteena oli rakentaa pilvipalvelualustalle dataputki, jolla voidaan rakentaa käyttökelpoisia koneoppimismalleja käyttäen operationaalista työmaadataa olemassaolevasta tietokannasta. Datan soveltuvuutta koneoppimistarkoituksiin mitattiin ennustamalla työmaan valmistumisaikaa sen nykytilan ja historian perusteella käyttäen erilaisia koneoppimismalleja.

Dataputki rakennettiin Amazon Web Services -pilvipalvelussa, ja koostui pääosin Glue- ja SageMaker-palveluista. Glue:a käytettiin operationaalisen datan muuntamiseen koneoppimismallien lukemaan muotoon, ja SageMaker:ia rakentamaan mallit. Amazon Web Services -alusta sisälsi tarvittavat palvelut putken rakentamiseen, eikä ylimääräistä itse putkeen liittyvä koodia tarvinnut kirjoittaa juurikaan. Putki oli helppo toteuttaa, eikä suuria vaikeuksia tullut vastaan.

Koneoppimismallit, joilla dataa testattiin olivat lineaarinen regressio, random forest, XGBoost sekä neuroverkko. Tulokset näyttävät, että paras malli tähän ongelmaan on kahden piilotetun kerroksen neuroverkko, mutta tarkkuuteen vaikuttaa huomattavasti tapa, jolla koko data puolitetaan opetus- ja testidataan.

Avainsanat: Koneoppiminen, pilvipalvelut, Amazon Web Services, regressio, vertailu, dataputki

PREFACE

This thesis was written in Tampere, Finland between February and October 2019. I learned a lot while writing this thesis, and would like to thank my colleague Hannu Hytönen for providing me with this interesting topic, and my employer Bitwise Oy for granting the time to work on it. I would also like to thank Professor Hannu-Matti Järvinen and Associate Professor Heikki Huttunen for supervising this work and giving detailed feedback on it. Lastly, thanks go to Inari for providing insights into scientific writing and supporting me.

Tampere, 2nd October 2019

Samu Jussila

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ANN	Artificial Neural Network
AWS	Amazon Web Services
CART	Classification And Regression Tree
CSV	Comma-separated Values
Epoch	An iteration of all the training data passing through an artificial neural network
HTTP	Hypertext Transfer Protocol. The underlying protocol of communications through the Internet
Loss	An artificial neural network's error function to minimize
MAE	Mean Absolute Error
MLP	Multilayer Perceptron
MSE	Mean Squared Error
S3	Simple Storage Service. The object storage service in Amazon Web Services
SGD	Stochastic Gradient Descent
SQL	Structured Query Language. A common way of accessing data in a database.
SSH	Secure Shell. A protocol for secure remote access
UI	User interface
URL	Uniform resource locator
XGBoost	eXtreme Gradient Boosting

1 INTRODUCTION

Machine learning is a popular topic in the field of technology, and is being utilized by companies for an ever increasing amount of tasks which would previously require a human, or an inferior hard-coded solution. Machine learning solutions can be used to for example to identify tumors in brain images [43], detect malicious users in computer networks [40], play the game of go better than humans [39], and predicting the durations of traffic incidents [45]. A correctly functioning machine learning application requires a lot of data, and many companies have proactively gathered data and are looking for ways to utilize it with machine learning.

Cloud services are a relatively recent alternative to on-premise hosting of services. Cloud services free their users of the burdens of managing physical infrastructure. In the cloud, infrastructure can be easily scaled up or down based on demand, even automatically, and the information can be accessed from anywhere in the internet [20]. An increasing number of companies are starting to utilize cloud services [35]. Cloud service providers have also started offering machine learning services on their cloud platforms. Other applications and the data needed for a machine learning process might already be stored in the cloud platform, which makes starting utilizing machine learning easier. Difficult machine learning tasks also require powerful hardware, which are easily accessible in the cloud.

This thesis aims to perform data analysis on existing operational data in a cloud service database, using machine learning. The goal is to first choose a cloud service platform, and then study how the data applies to a meaningful machine learning task, and how well the platform supports building a usable application utilizing the machine learning results. The machine learning task for this thesis has been chosen to be predicting a worksite's time of completion, using its state and history as input.

The goals are met using the design science methodology [19]. In this work, a proof-of-concept application is designed, consisting of two artifacts constructed using cloud services. The first artifact is a data pipeline from an operational database to using the machine learning results. The second artifact are the machine learning models. The pipeline's performance is evaluated based on its ease of construction and its functionality for actual use. The machine learning models are all evaluated with well-known popular accuracy metrics, and the best one is chosen for possible future use.

In Chapter 2, machine learning will be explained in depth. First, its general principles

will be presented, and then the mathematical background of each model used will be explained in more detail. Chapter 3 will give a more detailed explanation of what the cloud is, and what kind of services are offered in the cloud. How machine learning services are implemented in the cloud is also discussed. Chapter 4 will first explain the choice of the cloud platform used. Then the architecture for the machine learning applications is presented. The results of both utility of the data, and the suitability of the cloud platform, will be presented and discussed in Chapter 5. Finally, in Chapter 6, the results are briefly summed up.

2 MACHINE LEARNING

Machine learning is a field of research which studies computer programs that automatically learn from examples [30]. There are numerous different domains and applications where machine learning is used, including analysing images for medical purposes, security heuristics for protecting ports of networks, object recognition for self-driving cars and pattern recognition for detecting weaknesses in code [25].

Machine learning in general uses the principle of *Concept learning*: learning broad concepts through sets of specific examples [30]. What this means is that a computer is taught to perform the wanted task by giving it examples of how it should work, and hoping it learns the actual concept which the examples try to illustrate. Once the computer is taught the principles through the examples, later called *training data*, it can use its acquired knowledge to correctly perform even with data outside of the set of examples it learned with. For example, a simple image recognition task could have the computer learn to recognize whether there is a cat or a dog in an image. The computer is shown images with either a cat or a dog in them, and told what it should output. Eventually, after having seen enough images, the computer should learn to correctly identify the animal in any image it is shown, even ones it has not been shown before and told the right answer to.

Machine learning applications are implemented using mathematical constructs, called *models*. A trained model is the output of a machine learning algorithm using the provided training data [14]. The model can be used to infer the trained attributes from new data. A model's performance depends on both the structure of the model itself, and the amount and quality of the training data. Some of the most common reasons for models performing poorly are *over-fitting*, *under-fitting* [14].

Over-fitting means the model performs excellently with the examples it was trained with, but poorly with any data it was not trained with. Over-fitting can occur when the model is too complex to represent the principle it is trying to learn, or when there is not enough training data to model the principle in enough detail [2]. An over-fit model in effect, or even literally, just memorizes the training data and does not learn the correct principle [14]. Generally, to overcome over-fitting, either the model complexity has to be reduced, or the amount of training data has to be increased [2]. Under-fitting is the opposite of over-fitting [14]. Under-fitting can happen if the model is not complex enough to represent the taught principle, and gives poor results. Under-fitting can be overcome by increasing model complexity to match the complexity of the learned principle.

This chapter will first further describe types of machine learning problems. Secondly, the machine learning models used in this thesis will be discussed in more detail.

2.1 Types of machine learning

Machine learning algorithms can be roughly split into three categories: supervised learning, unsupervised learning and reinforcement learning [42]. Supervised learning means teaching the computer to mimic the training data. Training data in supervised learning consists of pairs of inputs and desired outputs. The *is there a cat or a dog in the image* problem described earlier is a supervised learning problem, where the input is an image, and the output is the boolean value "yes" or "no" corresponding to the image. [25].

The previous example is a classification problem. Classification means training the computer to choose the most probably correct one of predefined labels, classes, for each input data. The other sub category of supervised learning is regression. A regression problem means training the computer to infer or predict a continuous value from the input data. An example of a regression task would be predicting salary of a job based on its description. [17]

Unsupervised learning algorithms, unlike supervised learning, do not try to output a value based on input values, as there are no known output values for the input data to train the model with. Instead unsupervised learning algorithms perform analysis on and recognize patterns in the data itself [13]. Some examples of usages for unsupervised learning algorithms are clustering, dimensionality reduction [25] and anomaly detection [16]. Clustering algorithms divide the data into categories inferred by the algorithms themselves [25]. Dimensionality reduction algorithms take high-dimensional data sets, and transform them into fewer dimensions, trying to take only the most valuable features into the reduced set [25]. Anomaly detection algorithms look for anomalies or outliers in data, for example to detect credit card frauds or a system breakdown [6].

Reinforcement learning is training a computer to choose a correct action in its observable environment state. In reinforcement learning the computer has to find which action ultimately yields the largest numerical reward [42]. The reward may not be immediate, which makes reinforcement learning different from supervised learning: In supervised learning the training data would include correct actions for the data set's environment states, but in reinforcement learning, they are not known, and the only thing which matters is the final reward. The computer has to then, by trial-and-error, find a policy which leads to the best outcome from any environment state.

A well-known example of a reinforcement learning algorithm is AlphaGo, the program which has won some of the world's best human players in go [39]. A reinforcement learning algorithm learning to play go could be given a positive reward for winning a game, and a negative reward for losing a game. The ultimate reward then reinforces or diminishes the choices the algorithm made during the game.

2.2 Models

The data used in this thesis has known output values for the training data, and the output values are continuous numbers, which means that the algorithms used in the thesis belong to supervised learning and regression. This section introduces the machine learning models used in the thesis.

2.2.1 Linear regression

This section will explain the mathematical background used in linear regression models for machine learning.

Simple linear regression

Simple linear regression is a model which has a single input variable x, and a response y. The model represents a straight line, and can be represented as

$$y = \beta_0 + \beta_1 x + \epsilon \tag{2.1}$$

where β_0 and β_1 are unknown constants, usually also called regression coefficients. They denote slope and intercept respectively. ϵ is a random error component. The errors are assumed to be uncorrelated random variables with zero mean and variance σ^2. The result y can be seen as a probability distribution with mean

$$E(y|x) = \beta_0 + \beta_1 x \tag{2.2}$$

and variance

$$Var(y|x) = \sigma^2 \tag{2.3}$$

as the mean of the error is zero, and the unknowns are constants with no error. [31]

The regression coefficients can be estimated from training data ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n)) using the method of least squares, which finds values with which the sum of squared residuals, differences between the wanted output and the model output, is the smallest. Given $\hat{\beta}_0$ and $\hat{\beta}_1$, which estimate β_0 and β_1, difference between the training data y_i and the model output $\hat{y}_i$ for input x_i is

$$e_i = y_i - \hat{y}_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i). \tag{2.4}$$

The squared sum of residuals, $RSS(\beta_0, \beta_1)$ is then

$$RSS(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2. \tag{2.5}$$

The minimum can be found by partially differentiating the sum function with respect to both β_0 and β_1, setting the derivatives equal to 0, and solving the equation pair. This ultimately yields

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}((x_i - \overline{x})(y_i - \overline{y}))}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{2.6}$$

and

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x} \tag{2.7}$$

where $\overline{x}$ and $\overline{y}$ are the averages [31],[46]

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i. \tag{2.8}$$

Multiple linear regression

Simple linear regression can only have one dimension in its input variable x_i, which does not fulfill the general case, where there can be any number of dimensions. The equations of 2.2.1 can however be extended to consider any number of dimensions.

The multiple linear regression model with k input dimensions is defined as

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + ... + \beta_k x_{i,k} + \epsilon \tag{2.9}$$

where $\beta_{1..k}$ are the regression coefficients, $x_{n,1..k}$ are the input values, y the response, and ϵ the random residual. [31],[46]

The regression coefficients can, similar to 2.2.1, be estimated using the method of least squares. Given training data (n sets of parameters and results: $(x_{1,1}, x_{1,2}, ..., x_{1,k}, y_1)$, $(x_{2,1}, x_{2,2}, ..., x_{2,k}, y_2)$, $(x_{n,1}, x_{n,2}, ..., x_{n,k}, y_n)$), and parameters $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, ..., \hat{\beta}_k$, the ith response $\hat{y_i}$ by the model is [31],[46]

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \hat{\beta}_2 x_{i,2} + ... + \hat{\beta}_k x_{i,k} + \epsilon_i. \tag{2.10}$$

Least squares method minimises the sum of squared residuals, which can be written analogous to 2.5 as [31],[46]

$$RSS(\beta_0, \beta_1, \beta_2, ..., \beta_k) = \sum_{i=1}^{n}\epsilon_i^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \sum_{j=1}^{k}(\hat{\beta}_j x_{i,j}))^2. \tag{2.11}$$

Minimising the sum can be done by differentiating with respect to each $\beta_j, j = 1..k$, setting the derivative to zero, and solving the system of equations. [31],[46]

Equation 2.11 can be conveniently expressed in matrix notation as [31],[46]

$$RSS(\boldsymbol{\beta}) = \sum_{i=1}^{n} \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \tag{2.12}$$

where

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,k} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \dots & x_{n,k} \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix}, \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_k \end{bmatrix}. \tag{2.13}$$

From equation 2.12, the estimate $\hat{\boldsymbol{\beta}}$ can be solved as [18, p. 45]

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}, \tag{2.14}$$

provided that the inverse $(\boldsymbol{X}^T \boldsymbol{X})^{-1}$ exists. The inverse will exist if the regressors are linearly independent.

The estimator only depends on the statistics $\boldsymbol{X}^T \boldsymbol{X}$ and $\boldsymbol{X}^T \boldsymbol{y}$, which are the uncorrected sums of squares and cross products between the columns of $\boldsymbol{X}$, and uncorrected sums of cross products between columns of $\boldsymbol{X}$ and $\boldsymbol{y}$. [31],[46] The least squares estimate $\hat{\boldsymbol{\beta}}$ should not be calculated using 2.14, because the uncorrected sums are susceptible to large rounding errors, which may make the result inaccurate. Instead the calculations should be done on corrected sums instead. [46] The matrix $\boldsymbol{U}$ is defined as the training data parameters matrix $\boldsymbol{X}$ without the first column of ones, and the column means subtracted from each column

$$\boldsymbol{U} = \begin{bmatrix} (x_{1,1} - \overline{\boldsymbol{x}}_1) & \dots & (x_{1,k} - \overline{\boldsymbol{x}}_k) \\ (x_{2,1} - \overline{\boldsymbol{x}}_1) & \dots & (x_{2,k} - \overline{\boldsymbol{x}}_k) \\ \vdots & \ddots & \vdots \\ (x_{n,1} - \overline{\boldsymbol{x}}_1) & \dots & (x_{n,k} - \overline{\boldsymbol{x}}_k) \end{bmatrix} \tag{2.15}$$

and the vector $\boldsymbol{V}$ is defined similarly as the training data result vector $\boldsymbol{Y}$ with their mean

$\overline{y}$ subtracted from each element

$$\boldsymbol{V} = \begin{bmatrix} y_1 - \overline{\boldsymbol{y}} \\ y_2 - \overline{\boldsymbol{y}} \\ \vdots \\ y_n - \overline{\boldsymbol{y}} \end{bmatrix}. \tag{2.16}$$

The matrices $\boldsymbol{U}^T\boldsymbol{U}$ and $\boldsymbol{U}^T\boldsymbol{V}$ have the same needed statistical properties as the matrices $\boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{X}^T\boldsymbol{y}$, and can be used to calculate $\hat{\boldsymbol{\beta}}$ more accurately, very similarly to 2.14 as

$$\hat{\boldsymbol{\beta}}^* = (\boldsymbol{U}^T\boldsymbol{U})^{-1}\boldsymbol{U}^T\boldsymbol{V}, \quad \hat{\boldsymbol{\beta}}_{\mathbf{0}} = \overline{y} - \hat{\boldsymbol{\beta}}^{*T}\overline{\boldsymbol{x}} \tag{2.17}$$

where $\hat{\boldsymbol{\beta}}^*$ is the vector containing all values of $\hat{\boldsymbol{\beta}}$ except for the intercept, $\hat{\boldsymbol{\beta}}_{\mathbf{0}}$. [46]

The vector $\hat{\boldsymbol{\beta}}$ can be used to predict an unknown value $\hat{\boldsymbol{y}}_u$ by multiplying a parameter vector $\boldsymbol{x^T}$ ([1, $x_{u,1}$, $x_{u,2}$, ..., $x_{u,k}$]) with it:

$$\hat{y} = \boldsymbol{x^T}\hat{\boldsymbol{\beta}} = \hat{\beta}_0 + \sum_{j=1}^{k} \hat{\beta}_j x_j \tag{2.18}$$

or multiple predictions at once by [31],[46]

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}. \tag{2.19}$$

2.2.2 Decision trees ensembles

Decision tree learning is a machine learning algorithm which constructs a decision tree to learn the wanted principle from training data. Decision trees can be used as a model alone, but multiple trees can also be aggregated as a single ensemble, a forest. The two most popular ensemble methods for decision trees are randomization and *boosting*. This section will first explain decision trees themselves, and present an algorithm for creating them. Then, the algorithms used to create the ensemble models used in this thesis are explained.

Decision trees

A decision tree is a model which consists of a root node, intermediate nodes, and leaf nodes. Each node contains either a test on one of the parameters of the input data, or the final answer (a leaf node). When using a decision tree to find the output value from input data, the root node is accessed first. By following the branches of the tree by

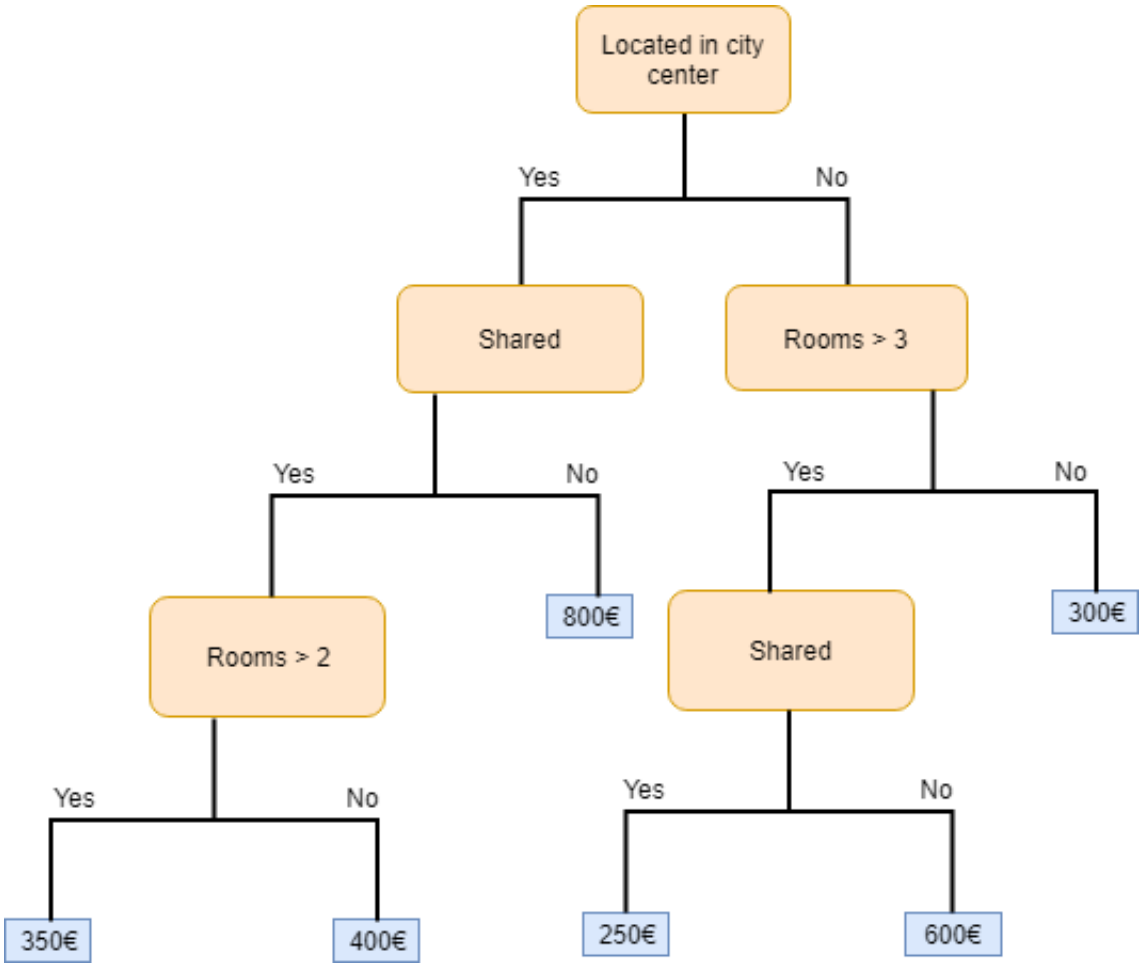

Figure 2.1. *Decision tree for inferring monthly rent based on apartment location, amount of rooms, and whether the apartment is shared with others*

answering the tests in the nodes with the input data eventually a leaf node containing the final value is reached. [30] For example, a simple decision tree in Figure 2.1 could be used to give an estimate on apartment's monthly rent using three parameters as input data. The parameters in this case would be a boolean value of whether the apartment is located in the center of the city, a number of how many rooms there are in the apartment, and a boolean value of whether the apartment is shared with other people. The leaf nodes give an estimate based on the answers given to get to that leaf.

Decision tree learning is an efficient and simple learning model, especially so with large amounts of dimensions and training data. They are trained with greedy divide-and-conquer algorithms, which recursively split the data into smaller and smaller subsets. [44] There are many specific algorithms for constructing a decision tree, such as ID3, ASSISTANT, C4.5 [30] and CART [48]. The CART algorithm will be explained in detail, as very similar algorithms to it are used by both the random forest algorithm [4] and the XGBoost algorithm [33, p. 50], which are the tree ensemble methods used in this thesis.

In regression problems decision tree models' weaknesses include instability [44], non-smoothness [44][2, p. 666], many algorithms only doing a split on one variable in a

node [2, p. 666]. A decision tree is unstable, because each node splits the data, and the following branches only have part of the data to use, and depend on their parent nodes' decisions. If there are two training data sets with only minimal differences, tree models trained from them could have completely different structures, if the root node's decision parameter changes even a little. Non-smoothness is problem only in regression models. It means that the tree can only give a finite amount of discrete final answers for any parameters, as a leaf node only contains one value. Typically regression models are trying to model smooth functions instead of a choice out of discrete values [2, p. 666]. Doing a split on only one variable can lead to much more complex trees than would be needed if many variables were considered. For example, if a decision boundary between two values could be easily expressed as the sum of two variables, single variable splitting cannot represent that in a simple way. A sum of two variables requires a decision boundary diagonal to both variable axes. Instead a single variable splitting tree model uses numerous splits of both variables to emulate the diagonal line. [2, p. 666] Decision trees also very susceptible to over-fitting if they are not sufficiently constrained [44].

CART algorithm

The CART algorithm, introduced by Breiman et al. in 1984, produces binary trees which split the data in two by one variable in each node. The algorithm provides tools for creating both classification and regression trees, but this explanation only focuses on regression. CART trees are also capable of using both least squares and the least absolute deviation as error measures, but this explanation will only focus on the least squares method. The algorithm consists of the following steps [48]:

1. Grow a large tree
2. Iteratively prune the tree back to its root
3. Choose the best tree acquired during pruning

The first step is growing a large tree T_{max}. This is done by recursively splitting the training data in nodes until a stopping condition in each branch is satisfied and the leaf nodes are grown. The recursive split is done by choosing the best split from the set of possible splits on any one variable. The variables can be either categorical or numerical. A split on a numerical value x_n in CART is always of the form $x_n \leq A$, where A is a constant number. A split on a categorical variable x_n is of the form $x_n \in B$ where B is a subset of all possible values of x_n. [3]

The best split for a regression tree in CART is defined as the split which most decreases error measure $R(t)$, where t is the node in question. The best split $\mathfrak{s}*$ out of all possible splits $\mathcal{S}$ is defined as

$$\Delta R(\mathfrak{s}*, t) = \max_{\mathfrak{s} \in \mathcal{S}} \Delta R(\mathfrak{s}, t), \tag{2.20}$$

where

$$\Delta R(\mathfrak{s}, t) = R(t) - R(t_L) - R(t_R). \tag{2.21}$$

The variables t_L and t_R are the left and right branches of the node as the result of the split $\mathfrak{s}$.

Define $\boldsymbol{x}_n$ as the nth training data instance in node t, N as the number of training data instances in node t, y_n as the nth response, $\overline{y}(t)$ as the average of all the response values in node t, and $p(t)$ as the proportion of training data instances in node t ($N(t)/N_{total}$). The $R(t)$ itself can be then defined as

$$R(t) = s^2(t)p(t) \tag{2.22}$$

where

$$s^2(t) = \frac{1}{N}\sum_{\boldsymbol{x}_n \in t}(y_n - \overline{y}(t))^2. \tag{2.23}$$

It is worth noting that $s^2(t)$ is the sample variance of the response variables in node t, and the best split is then the one which reduces the weighted sum of variances the most.

The splitting is done in each node until a stopping condition is met. CART trees are grown deliberately large, so the only stopping rule usually used is $N(t) \leq N_{min}$, where N_{min} is usually 5 [3, p. 233]. Other possible stopping rules include the tree's maximum depth limit, and all the response variables being exactly the same. When a stopping condition is met, the node becomes a leaf node, and it is assigned a single response value. The response value $y(t)$ for node t is

$$y(t) = \frac{1}{N(t)}\sum_{\boldsymbol{x}_n \in t} y_n, \tag{2.24}$$

which is the arithmetic average of the response variables of training data in the node. [3]

Least squares regression error for model β in general is defined as [44, p. 63]

$$Err(\beta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - r(\beta, \boldsymbol{x}_n))^2, \tag{2.25}$$

where $r(\beta, \boldsymbol{x}_n)$ is the response value given by the model for the input data. A tree model's total error $Err(T)$ is defined as the weighted average of its leaves' errors [44]

$$Err(T) = \sum_{l \in \widetilde{T}} p(l)Err(l), \tag{2.26}$$

and a leaf node's error as the mean squared error in that node

$$Err(l) = \frac{1}{N_l}\sum_{\boldsymbol{x}_n \in l}(y_n - \overline{y}_l)^2. \tag{2.27}$$

A CART tree minimizes $R(t)$ of its leaf nodes, which is the weighted least squared error criterion. Thus, a CART tree minimizes least squares error criterion.

The second step in CART algorithm is pruning the tree back to its root. Pruning means picking a node, and deleting all its descendant nodes, aggregating all the training data in its descendants to itself. The algorithm uses pruning instead of more elaborate stopping conditions, because no stopping rule gives generally as good results as pruning post-growing does [3, p. 61-62]. Pruning a large tree T_{max} back to its root means pruning the tree until only its root exists. During the process each intermediate step of pruning is saved, and a sequence of trees $T_{max} \to T_1 \to T_2 \to \cdots \to T_{root}$ is acquired.

The sequence of pruned trees is acquired through error-complexity pruning. In error-complexity pruning the tree is assigned an error-complexity measure $R_\alpha(T)$, which is the linear combination of the error measure $R(T) = Err(T)$ and the amount of leaf nodes in the tree $|\widetilde{T}|$

$$R_\alpha(T) = R(T) + \alpha|\widetilde{T}|. \tag{2.28}$$

For each value of the cost complexity α there is a subtree T_α which minimizes $R_\alpha(T_\alpha)$. By increasing α, different minimizing subtrees of the sequence are found, and eventually the cost for having any number of leaf nodes at all is so large that the minimizing tree is just the root node.

In practice this is done in reverse, by cutting weakest links in the trees, and finding the corresponding α values. Define $\{t\}$ as node t made into a leaf node by deleting all its descendants, and T_t as the subtree starting from node t. Now, for each node

$$R_\alpha(\{t\}) = R(t) + \alpha, \tag{2.29}$$

and

$$R_\alpha(T_t) = R(T_t) + \alpha|\widetilde{T}_t|, \tag{2.30}$$

where $R(T_t) = Err(T_t)$, the weighted sum of the leaf nodes below T_t. As long as $R_\alpha(T_t) < R_\alpha(\{t\})$, the complexity of the branch is acceptable to lower the error measure, but for every node t, there is some value of α, where deleting the descendants starts producing lower overall error-complexity. For every node, the value $g(t)$, the lowest α where the nodes's descendants are pruned can be calculated as

$$g(t) = \begin{cases} \frac{R(\{t\}) - R(T_t)}{|\widetilde{T}_t| - 1}, & \text{if } t \notin \widetilde{T} \\ \infty, & \text{if } t \in \widetilde{T} \end{cases}. \tag{2.31}$$

Weakest link is the node $\bar{t}$ where

$$g(\bar{t}) = \min_{t \in T} g(t). \tag{2.32}$$

The weakest link is cut, and the next tree in the pruning sequence is acquired. [3]

The final step in CART is to choose the best tree in the pruning sequence. This is done by using either cross validation or test set. In test set method, some number of training data

is taken out of the data set and not used to construct the tree. Then the taken-out test set is used to assess the accuracy of each tree in the pruning sequence and the best tree size is picked. Cross validation makes v different pruning sequences where each $T_{v,max}$ is grown with $\frac{v-1}{v}N$ training data samples and tested with the remaining $\frac{1}{v}N$. When v is large, each sequence's trees' accuracy is very similar to the trees grown with all the data, and cross validation gives a good estimate of the right size of a tree. [3]

Random forest

Random forest is a machine learning model, which consists of an ensemble of individual trees. Random forests grow the individual trees using bootstrap aggregation (bagging). Using bagging, each individual tree is trained with a randomly selected subset $\boldsymbol{X}_n$ of the whole training data $\boldsymbol{X}$. The selection is made by taking a random sample from the training data with replacement, and a random sample from the dimensions without replacement. The individual trees are then grown with the CART algorithm, but modified in two ways: The trees are not pruned, and the best split at each node is found in a small random subset of features instead of all the features. Finally, when making predictions with new data, the whole regression model's output $F(\boldsymbol{X})$ for input parameters is the average of the set of all the individual trees' predictions $\{T_k(\boldsymbol{X})\}$ [4]

$$F(\boldsymbol{X}) = \frac{1}{K}\sum_{k=1}^{K} T_k(\boldsymbol{X}). \tag{2.33}$$

The random forest method improves on single trees by creating multiple strong predictors, which do not under-fit, but do over-fit severely. The effect of over-fitting is lessened by having the individual trees be independent of each other, and the result is, in theory, a model which does not under-fit and over-fits very little. As the number of trees in the forest grows towards infinity, the mean squared error of the forest approaches a limit, and makes the error of the forest F

$$PE^*(F) \leq \overline{\rho} PE*(T), \tag{2.34}$$

where $PE^*(T)$ is the average error of a single tree in the forest, and $\overline{\rho}$ is the weighted correlation between the errors of the individual trees. This shows that in order for a random forest to be any better than a single tree, the randomization has to aim to as independent subsets of the data and the input variables as possible. [4]

Random forests are very resistant to noise because of the averaging effect of the ensemble. Random forests deal with the weaknesses of single decision trees too. They have much smoother result variables, as their output does not have to be a single value from the training set. Instability is not a property of random forests, as they are averages of many unstable trees. Random forests are very fast to generate compared to their size, as all the individual trees being independent of each other, they can be constructed in

parallel.

Gradient Boosting

Boosting models create ensembles consisting of multiple single models, but the approach is very different from randomization. Boosting models instead add new, weak single models, later referred to as *base learners*, to the ensemble sequentially and incrementally. The new base learner added is trained to correct the errors of the whole ensemble. [32]

Boosting uses weak base learners to battle the over-fitting and under-fitting dilemma. The idea is reverse from the randomization method used by random forests. A weak leaner does not over-fit at all, but under-fits the data severely. New models are sequentially added to correct the errors, which can only be due to under-fitting. When adding new base learners, under-fitting is decreased, while over-fitting increases only a little.

Gradient boosting algorithms minimize the expected value of some loss function $L(y, F(\boldsymbol{x}))$, which can be for example the squared error between the real output value y and the output given by the model $F(\boldsymbol{x})$. The model F is a weighted sum of K weak base learners $\phi(\boldsymbol{x})$

$$F(\boldsymbol{x}) = \sum_{k=0}^{K} \beta_k \phi_k(\boldsymbol{x}), \tag{2.35}$$

and is constructed by first taking some initial guess $F_0(\boldsymbol{x})$, which is just a constant value minimizing the loss function most as

$$F_0(\boldsymbol{x}) = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, \rho). \tag{2.36}$$

Then, each next addition, or *boost* to the ensemble is calculated as the function which most decreases the loss function if summed with the rest of the ensemble

$$(\beta_k, \phi_k) = \arg\min_{\beta, h} \sum_{i=1}^{N} L(y_i, F_{k-1}(\boldsymbol{x}_i) + \beta\phi(\boldsymbol{x}_i)), \tag{2.37}$$

with the new model then being [11][32][10]

$$F_k = F_{k-1} + \beta_k \phi_k. \tag{2.38}$$

As the optimal parameters (β_k, ϕ_k) can be difficult to calculate in practice [32][10], an easier procedure is used. The procedure is to instead find the base learner most parallel to the negative gradient $\tilde{y}_i$ of the ensemble, and then find the weight for the found learner. The negative gradient is the derivative of the loss function

$$\tilde{y}_i = -\frac{\partial L(y_i, F_{k-1}(\boldsymbol{x}_i))}{\partial F_{k-1}(\boldsymbol{x}_i)}. \tag{2.39}$$

The base learner ϕ_k is then found by a simpler least squares minimization

$$\phi_k = \arg\min_{\rho,h} \sum_{i=1}^{N} (\tilde{y}_i - \rho\phi(\boldsymbol{x}_i))^2. \tag{2.40}$$

The weight β_k is then found using the original minimization equation 2.37 but ϕ_k fixed, as [10][11]

$$\beta_k = \arg\min_{\beta} \sum_{i=1}^{N} L(y_i, F_{k-1}(\boldsymbol{x}_i) + \beta\phi_k(\boldsymbol{x}_i)). \tag{2.41}$$

Gradient boosting with least squares error regression makes calculating the negative gradient $\tilde{y}_i$ very simple in practice. The gradient reduces to

$$\tilde{y}_i = y_i - F_{k-1}(\boldsymbol{x}_i), \tag{2.42}$$

which means that using least squares as the loss function causes each boosting base learner to be fit to the literal errors of the ensemble before it. [10]

Gradient tree boosting is a case of gradient boosting, where the weak base learner $\phi(\boldsymbol{x})$ is a decision tree, and in this thesis's context, a regression CART tree. A tree's output is just the constant value in the leaf which the input data belongs to. It can be expressed as the sum over all J leaf nodes, but only if the input belongs to the parameter space R_j which leads to the jth leaf

$$\phi(\boldsymbol{x}) = \sum_{j=1}^{J} b_j I(\boldsymbol{x} \in R_j), \tag{2.43}$$

where

$$I(a) = \begin{cases} 1, & \text{if } a = true \\ 0, & \text{if } a = false \end{cases}. \tag{2.44}$$

A gradient tree boosting model's equation 2.38 becomes [10]

$$F_k = F_{k-1} + \sum_{j=1}^{J} \gamma_{jk} I(\boldsymbol{x} \in R_{jk}), \tag{2.45}$$

where

$$\gamma_{jk} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{jk}} L(y_i, F_{k-1}(\boldsymbol{x}_i) + \gamma). \tag{2.46}$$

XGBoost (eXtreme Gradient Boosting) is an algorithm which extends the basic gradient tree boosting algorithm. XGBoost tries to find the next tree by minimizing an objective function $\mathcal{L}$

$$\mathcal{L} = \sum_{i=1}^{N} [L(y_i, F_{k-1}(\boldsymbol{x}_i)) + g_i\phi_k(\boldsymbol{x}_i) + \frac{1}{2}h_i\phi_k^2(\boldsymbol{x}_i)] + \Omega(\phi_k), \tag{2.47}$$

where $-g_i$ is the negative gradient $\tilde{y}_i$ from equation 2.39 and h_i is the second derivative

of the loss

$$h_i = \frac{\partial^2 L(y_i, F_{k-1}(\boldsymbol{x}_i))}{\partial^2 F_{k-1}(\boldsymbol{x}_i)}. \tag{2.48}$$

Ω is a model complexity value, defined as

$$\Omega(F_k) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} w_j^2, \tag{2.49}$$

where J is the number of leaves, and w_j is the jth leaf node's response value. λ and γ are complexity costs for the values and number of the leaves. By removing the constant parts from the objective function, it simplifies into [8]

$$\tilde{\mathcal{L}} = \sum_{i=1}^{N} [g_i \phi_k(\boldsymbol{x}_i) + \frac{1}{2} h_i \phi_k^2(\boldsymbol{x}_i)] + \Omega(\phi_k) \tag{2.50}$$

XGBoost constructs its single trees using the objective function. The optimal value for a leaf j is given by

$$w_j^* = -\frac{G_j}{H_j + \lambda}, \tag{2.51}$$

where

$$G_j = \sum_{i \in I_j} g_i, H_j = \sum_{i \in I_j} h_i, \tag{2.52}$$

I_j being the set of instances in the jth leaf node.

The quality of a tree structure q is given by

$$\tilde{\mathcal{L}}(q) = -\frac{1}{2} \sum_{j=1}^{J} \frac{G^2}{H + \lambda} + \gamma J, \tag{2.53}$$

and the best split s^* at each node is the one which most increases the quality of the structure, as

$$s^* = \arg\max_s \frac{1}{2} (\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} + \frac{G_j^2}{H_j + \lambda}) - \gamma, \tag{2.54}$$

where G_L, G_R, and H_L, H_R are the values in the left and right nodes in the proposed split [8]. If no gain is positive, no split is done at the node, and it will stay as a leaf node [47].

XGBoost has achieved recognition as being fast and scalable [8] as well as winning numerous machine learning competitions [33]. Tree boosting in general deals with some of the problems of single regression trees. Boosted models can yield smoother responses than single trees, but a boosted models are still unstable. Boosted models are slower to generate than random forests, because they depend on the previous trees and thus cannot be created in parallel. Boosted trees can also over-fit, if too many trees are grown into the sequence.

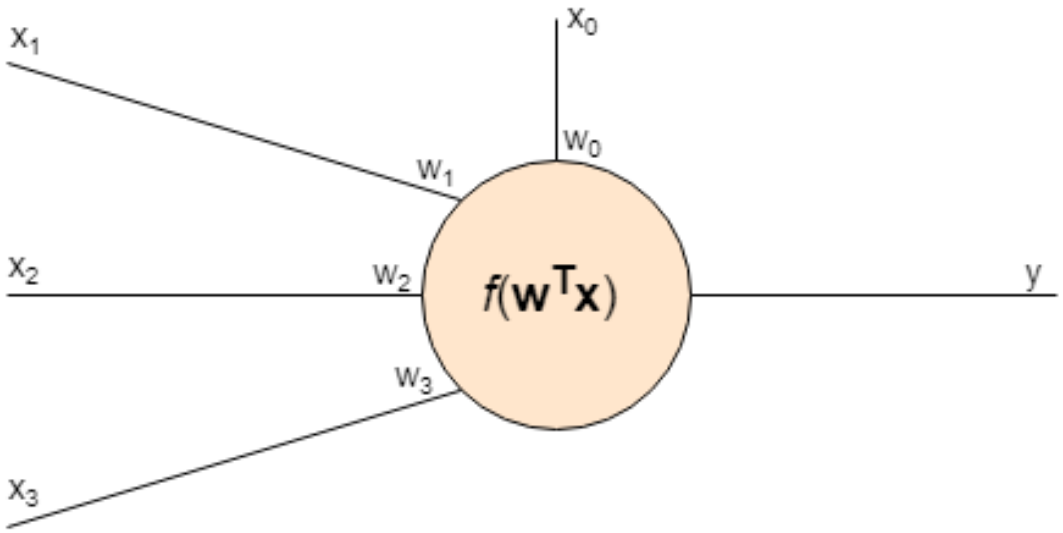

Figure 2.2. *Neuron in an artificial neural network*

2.2.3 Artificial neural network

Artificial neural network (ANN) is a machine learning model which comprises multiple *neurons* that interconnect in some way. The background for artificial neural networks is in neural activity in the human brain. The neurons in a brain send electrical signals to each other with chemical rules to process information, while the neurons in artificial neural networks output numbers to other neurons to calculate a result value for input values. Although ANNs are inspired by biological neural networks, they do not aim to exactly model the networks in a human brain [30].

This thesis will only consider one neural network model, the *multilayer perceptron*, which is the most widely used neural network model [2]. In this section, the principles of a single neuron will be first explained, followed by constructing a multilayer perceptron out of the neurons. Finally, the mathematics of training the network with training data will be introduced.

Neuron

A neuron in an ANN consists of inputs, weights, and an activation function. The basic structure is illustrated in Figure 2.2. A neuron's input variables can consist of raw input values or output values of other neurons. The input vectors are padded with the value 1 to allow neurons to add a constant value, a *bias* to the inputs. If there are k input variables to the neuron, the input vector $\boldsymbol{x}$ is then

$$\boldsymbol{x} = [1, x_1, x_2, \ldots, x_k]^T. \tag{2.55}$$

Each input value, and the bias, are multiplied with their respective weights from a $k+1$-

Table 2.1. *Common activation functions and their derivatives.*

Name	$f(x)$	$f'(x)$
Rectified Linear Unit (ReLU)	$f(x) = max(0, x)$	$f'(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x > 0 \end{cases}$
Logistic sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Hyperbolic tangent (TanH)	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f^2(x)$
Linear	$f(x) = x$	$f'(x) = 1$

length weight vector and summed to produce value a [1, p. 272]

$$a = \boldsymbol{w}^T \boldsymbol{x}. \tag{2.56}$$

The value a is very analogous to a linear regression model output described in section 2.2.1.

A neuron's output value y is obtained by using an activation function f. An activation function has to be differentiable, as the differentials are used when training the network. The activation functions in general are also nonlinear, as if all activation functions in a network were linear, the network could be reduced to linear regression [2]. Some common activation functions [23] are shown in table 2.1. Using a chosen activation function f [2],

$$z = f(a). \tag{2.57}$$

Multilayer perceptron

A multilayer perceptron (MLP) is a type neural network which consists of successive layers of parallel neurons. An MLP is a *feed-forward* network, meaning that the neurons only feed their outputs forwards, not creating any cycles in the network. This means the output of the a network is a deterministic function of only the input values.

Figure 2.3 depicts an MLP which takes three-dimensional inputs, and outputs one value. It has one hidden layer of neurons in between the inputs and outputs. The layers are all densely connected. It means that each hidden layer unit takes all three dimensions of the input layer as their inputs and the output layer neuron takes all four dimensions output by the hidden layer as its input. Define I as the number of input dimensions, and H as the number of hidden layers. With the activation function of the output layer σ and the activation functions in the hidden layer h, the output of the network can be expressed as [2]

$$y = \sigma(\sum_{j=0}^{H} w_j^{(2)} h(\sum_{i=0}^{I} w_{ji}^{(1)} x_i)). \tag{2.58}$$

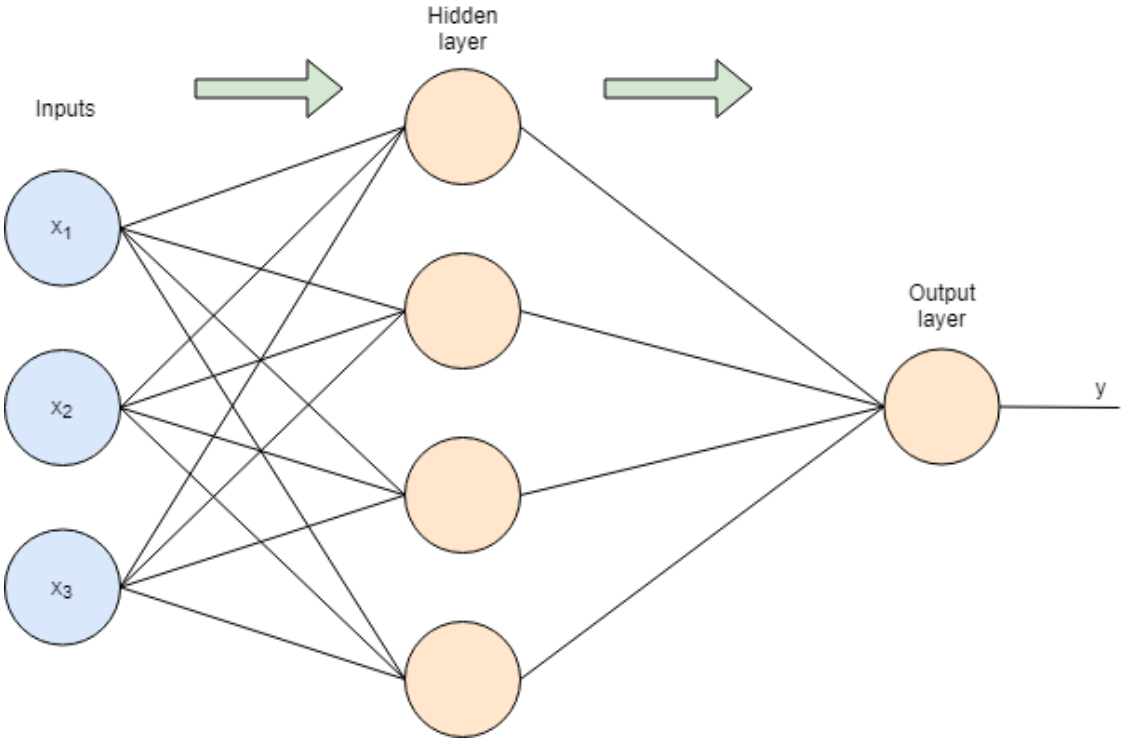

Figure 2.3. *A multilayer perceptron with three input dimensions, four hidden neurons, and one output value*

The superscripts denote the layer of the neuron containing the weight.

Gradient descent

The topology of an MLP does not change once it is constructed. Thus the only thing to change in the network structure are the weights inside the neurons. The weights are updated using a weight adjustment algorithm, which uses the gradients of the errors of output values as inputs. The error gradients at each neuron are acquired through the *backpropagation* algorithm explained later in this section. [2]

The traditionally most popular weight adjustment algorithm is *stochastic gradient descent* (SGD), which is an approximation of the *gradient descent* algorithm. The gradient descent algorithm is a simple formula to update the weight vector $\boldsymbol{w}$ of a neuron. Denoting superscript (τ) as the current weights and $(\tau + 1)$ as the new weights, gradient descent rule is

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E(\boldsymbol{w}^{(\tau)}), \tag{2.59}$$

where $\nabla E(\boldsymbol{w})$ is the sum gradient of all the training data rows' errors, and η is learning rate constant. Learning rate is a small number which denotes how fast the weights move towards the gradient. The normal gradient descent can be very slow in practice [30, p. 92][24], and the stochastic approximation is usually used instead. SGD instead updates

weights one training data sample at a time [2]

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n(\boldsymbol{w}^{(\tau)}). \tag{2.60}$$

SGD, while faster than normal gradient descent, can still be quite slow to converge [36][24], and many variations and extensions of it have been created. *Momentum* is a technique which makes the weights accelerate towards the negative gradient if the gradient direction stays same. Each weight has its own momentum, and the momentum instead of the weight is updated towards the gradient

$$\begin{aligned} \boldsymbol{v}^{(\tau+1)} &= \gamma \boldsymbol{v}^{(\tau)} + \eta \nabla E(\boldsymbol{w}^{(\tau)}) \\ \boldsymbol{w}^{(\tau+1)} &= \boldsymbol{w}^{(\tau)} - \boldsymbol{v}^{(\tau+1)} \end{aligned}. \tag{2.61}$$

The variable γ is a decay term, which gradually slows the speed of change. Another variation to SGD is adjusting learning rate dynamically. The *Adagrad* algorithm does this by dividing the learning rate at each iteration and weight by the square root of $G_{i,i}^{(\tau)}$, where G is a diagonal matrix containing the sum of the squares of each of the neurons's weights gradients up to time τ. The formula is

$$\boldsymbol{w_i}^{(\tau+1)} = \boldsymbol{w_i}^{(\tau)} - \frac{\eta}{\sqrt{G_{i,i}^{(\tau)} + \epsilon}} \nabla E_n(\boldsymbol{w_i}^{(\tau)}), \tag{2.62}$$

where ϵ is a small positive constant to avoid division by zero. The *Adam* algorithm stores a decaying momentum $m^{(\tau)}$ much like the Momentum algorithm, but also a decaying average of the squared past gradients $v^{(\tau)}$, which are updated by the formulas

$$\begin{aligned} m_i^{(\tau+1)} &= \beta_1 m_i^{(\tau)} + (1-\beta_1) \nabla E_n(\boldsymbol{w_i}^{(\tau)}) \\ v_i^{(\tau+1)} &= \beta_2 v_i^{(\tau)} + (1-\beta_2) \nabla E_n^2(\boldsymbol{w_i}^{(\tau)}) \end{aligned}, \tag{2.63}$$

where β_1 and β_2 are decay rates close to 1. As m and v tend to be biased towards zero, especially when the decay rates are very close to one, the biases are corrected with

$$\begin{aligned} \hat{m_i}^{(\tau+1)} &= \frac{m_i^{(\tau)}}{1-\beta_1} \\ \hat{v_i}^{(\tau+1)} &= \frac{v_i}{1-\beta_2} \end{aligned}. \tag{2.64}$$

Finally, the weights are updated with the formula [36]

$$\boldsymbol{w_i}^{(\tau+1)} = \boldsymbol{w_i}^{(\tau)} - \frac{\eta}{\sqrt{\hat{v_i}^{(\tau+1)}} + \epsilon} \hat{m_i}^{(\tau+1)}. \tag{2.65}$$

Backpropagation

Backpropagation is the algorithm used to calculate the error gradient vector $\nabla E_n(\boldsymbol{w}^{(\tau)})$ for each of a jth neuron's ith weight value $\boldsymbol{w}_{ij}$. This is achieved by finding the derivative of a loss function of an output value with respect to each of the weights. The loss $L(y,t)$ of an output value y_n for input vector x_n is some function for network output y and the target value t known in the training data. Usually the loss function is the sum of squared errors between network outputs and the target values

$$L(\boldsymbol{y},\boldsymbol{t}) = \frac{1}{2}\sum_{k\in outputs}(y_k - t_k)^2. \tag{2.66}$$

For simplicity, define $E_d = L(y_d, t_d)$, where d is the index of the training data sample. [30] For a simple single-output regression problem like this thesis is considering, the sum in the equation can be omitted, and the error is just half the squared difference between the network output and the target value.

Finding the partial derivative $\frac{\partial E_d}{\partial w_{ij}}$ involves realizing that a weight w_{ij} can only influence the network output y through the neuron output z_j and therefore through the intermediate sum a_j, introduced in equations 2.56 and 2.57 respectively. This enables the use of the derivative chain rule, and writing

$$\frac{\partial E_d}{\partial w_{ij}} = \frac{\partial E_d}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}}. \tag{2.67}$$

The partial derivative $\frac{\partial a_j}{\partial w_{ij}}$, being just a sum of multiplications, is just x_{ij}, the ith input value to the neuron. The partial $\frac{\partial z_j}{\partial a_j}$ is the derivative of the chosen activation function f in the neuron j. [30]

If the neuron j is the output neuron, $z_j = y_d$, and the error $\frac{\partial E_d}{\partial z_j}$ is simply the derivative of the loss function. For the squared error loss, this is simply

$$\frac{\partial E_d}{\partial z_j} = -(t_d - y_d). \tag{2.68}$$

The the jth neuron is not an output neuron, $\frac{\partial E_d}{\partial z_j}$ is not as straightforward to acquire. In that case, it is realized that z_j can only influence the network output through the other neurons $k \in K$ which its output is sent as input is used. This fact permits the use of the chain rule again, to write the partial derivative as the total derivative

$$\frac{\partial E_d}{\partial z_j} = \sum_{k\in K}(\frac{\partial E_d}{\partial z_k}\frac{\partial z_k}{\partial a_k}\frac{\partial a_k}{\partial z_j}). \tag{2.69}$$

The partial $\frac{\partial a_k}{\partial z_j}$ is simply the input weight w_{jk}. The rest of the partials are calculated in equation 2.67 for the output neuron, which can be used for the previous layer from the output layer, completing 2.69 for the layer. Those values can then be used for the

previous layer, and similarly the gradients can be calculated for any number of layers by propagating the calculated values backwards in the network. [30]

The gradient $\nabla E_n(\boldsymbol{w}^{(\tau)})$ used in gradient descent can now be defined as

$$\nabla E_n(\boldsymbol{w_{ij}}^{(\tau)}) = \frac{\partial E_d}{\partial w_{ij}}. \tag{2.70}$$

Training a network

When the network is trained, training data is fed one-by-one, or in batches, through the network. Each sample's output value is recorded, and the error gradient is propagated backwards with the backpropagation algorithm. The gradients are used by the gradient descent, or a similar algorithm, to update the neurons' weights. This is repeated until a termination condition is met. A termination condition can be for example some number of iterations (epochs) reached, or when the loss function over a training set or a separate validation set falls below a wanted value. [30]

The choice of a termination criteria is important in ANNs, because they are relatively complex models with potentially a huge number of weight parameters. This means that the network can easily overfit the data. Termination criteria helps with overfitting in ANNs because they tend to overfit in later iterations rather than in the early iterations. This is because the gradient descent algorithms use a small learning rate to slowly nudge the weights towards the "correct" values. The weights are usually initialized to small random values, and underfit the data at first, but as training continues the weights start to represent the training data more and more. One of the most successful methods is to use a separate validation set, or cross validation, to find the number of epochs which produces a network with lowest loss values for unseen data. [30] Another way to minimize overfitting is by utilizing a technique called dropout [41], which randomly temporarily removes neurons from the network during its training with a probability p.

3 CLOUD COMPUTING

Cloud computing has many definitions [9], but one commonly agreed on and used [9, 49] definition is given by the United States National Institute of Standards and Technology (NIST), which defines cloud computing [28] as follows: *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

Cloud computing is often [5, 9, 22, 49] compared to technologies like grid computing and cluster computing. Grid computing is an infrastructure traditionally built for intensive scientific calculations, which uses distributed autonomous resources to compute a wanted result faster than a single machine could [49][9]. Cluster computing is very similar to grid computing, coupling multiple independent machines to form a more powerful computation unit. Cluster differs from grid in that its machines are not autonomous and are tightly linked with each other both programmatically and geographically [22]. Grid and cloud computing both aim to achieve providing "invisible" resources to the user, but they differ in the way that grids aim for maximum performance for large computation tasks, while cloud aims to offer a more generic computation solution for all kinds of on-demand tasks [9].

Cloud computing is the most recent paradigm to promise of delivering *utility computing*, which means providing computation as a utility, providing and charging consumers resources based on actual usage [49]. Cloud computing has been seen capable of delivering computing as "the 5th utility", alongside with water, electricity, gas, and telephony [5]. According to RightScale report in 2018 [35], 96 percent of organizations already use some form of cloud computing, and the numbers are only growing.

The NIST definition of cloud computing, in addition to the quotation at the beginning of the chapter, also lists five essential characteristics cloud computing services have, their three main service models, and four deployment models, shown in Figure 3.1, to support the definition. This chapter describes the NIST-listed essential characteristics of cloud computing in 3.1, then the deployment models in 3.2, and service models in 3.3.

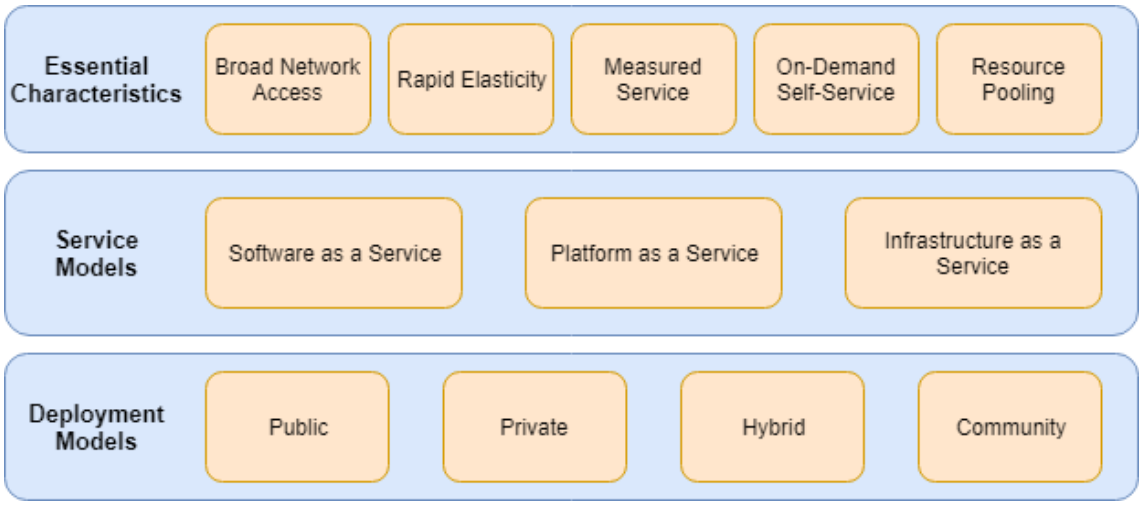

Figure 3.1. *NIST definition of cloud computing*

3.1 Essential characteristics

The essential characteristic *Broad Network Access* means, firstly that the resources provided by the cloud are accessed through the internet. Secondly, it means that the mechanics of accessing the resources are standard and promote use of any physical devices. [28] [9]

Rapid Elasticity means that the resources can be provisioned and upscaled or downscaled quickly, and even automatically. The consumer does not see any physical resource limits of the provider, but rather is offered seemingly unlimited resources. [28] [9]

Measured Service means that the cloud-provided resources can be, and are measured and monitored actively. [28] The monitoring can be used by both the provider, for example to gather usage statistics, or by the consumer, for example to detect possible errors or to optimize provisioned resource amounts.

On-Demand Self-Service means that the consumer of a cloud service can provision resources independently, without needing any human interaction with the cloud service provider. [9][28]

Resource Pooling means pooling resources together, using a multi-tenant model or virtualization, to serve multiple consumers using any number and types of actual physical machines. [9] This essentially makes physical resources completely invisible to the consumers, making them not exactly or at all know where their computation or data storage is. [9] It also allows the provider to optimize the physical resources to match customer demands.

3.2 Deployment models

Private cloud is a cloud deployment model where the cloud infrastructure is used by a single organization. A private cloud might be hosted by the organization itself on-premise, or by a third-part provider. [28][9] Private cloud might be used because of security concerns or because the organization wants to be in full control of the cloud backend [9][21].

Community cloud is like private cloud in that it is used by an exclusive group of users, but unlike private cloud in that the users are from several different organizations. A community cloud provides services which fulfill similar consumer needs between the organizations. It may be hosted by any of the user organizations or by a third party provider. [28][9]

Public cloud provices cloud services for open use by anyone [28], and is the most popular form of the deployment models [9]. A public cloud is managed and completely controlled by the cloud provider, which sets their own policies and charging models [9].

A *hybrid cloud* solution tries to achieve the positives of both private and public clouds by combining two or more distinct cloud solutions. The combination is made by creating a solution between the individual clouds to make the data and applications in them portable between each other. [21][28]

In addition to the types listed by NIST, literature [49][9] often lists another solution, similar to hybrid cloud, which tries to combine public and private clouds' good sides, called *Virtual private cloud*. Virtual private cloud is a solution built on top of a public cloud which, using virtual private networks (VPN), makes it essentially private to consumers.

3.3 Service models

This section will explain the three main service models defined by NIST [28], shown in Figure 3.2, as well as a less-used term *Machine learning as a service*. The three service models are all different abstractions of resources that could be used as services of an own datacenter or software installed on own computer. Although Machine learning as a service is not a distinct abstraction level of resources provided, it is a distinct set of services that are often provided by cloud service providers, and that are much discussed and used in this thesis.

3.3.1 Infrastructure as a service

Infrastructure as a service (IaaS) is the lowest abstraction level of the service models. It aims to provide consumers the capabilities of a physical data center, without the customer

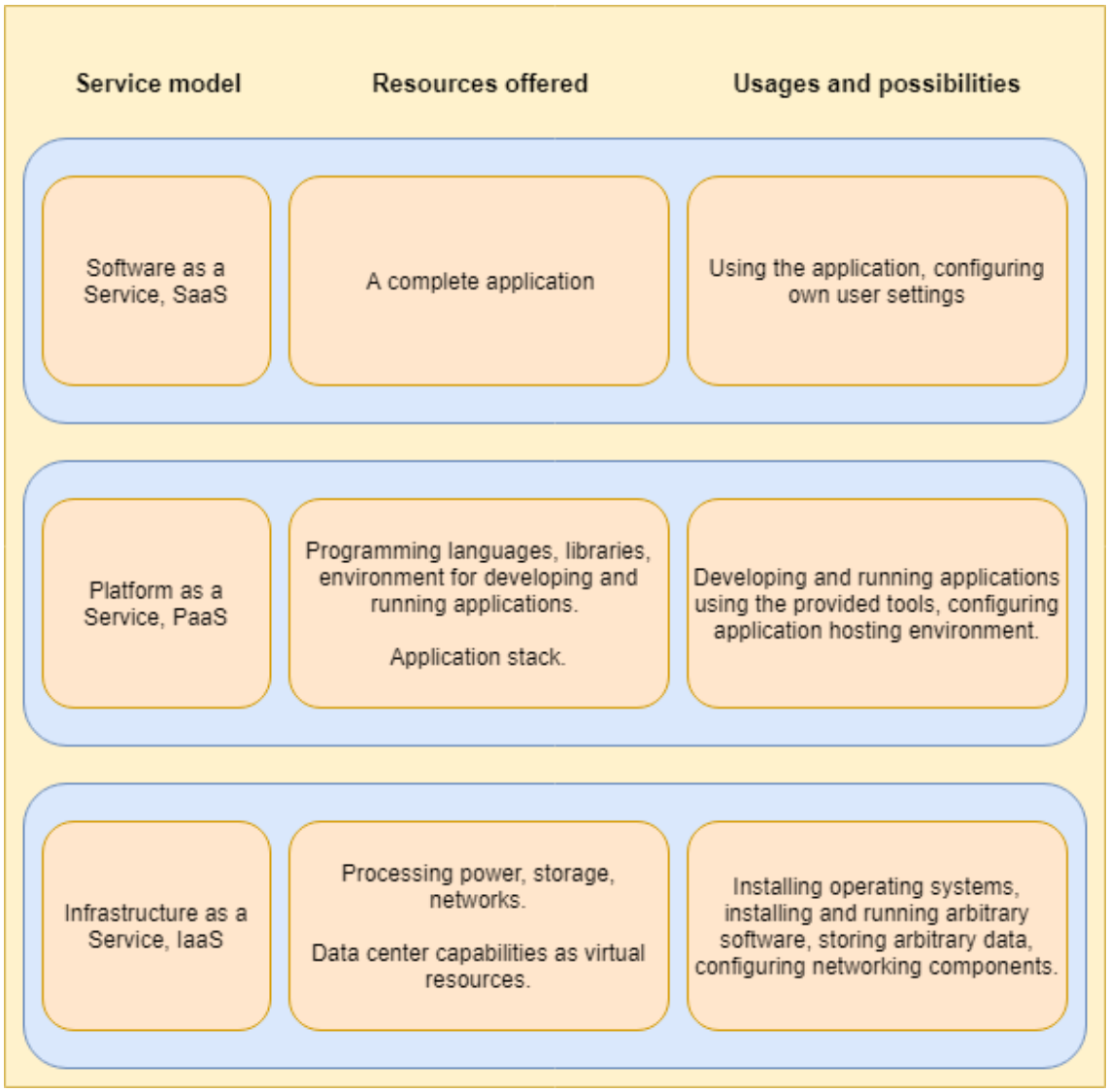

Figure 3.2. *Cloud service model levels*

needing to maintain a physical data center [21]. It provides the consumers with the fundamental infrastructural resources like processing power, networks and storage, which the consumer can use as they see fit [28]. The consumer can use the provided resources to for example build a standard web application consisting of a database, server and a UI by using the provided storage to store the database and the server executable, using the provided processing power to run the server executable, and configuring the network so that users can access the UI from the Internet.

IaaS is typically implemented by heavily using virtualization to quickly provision or destroy resources according to consumer needs [9]. A common example [9, 21, 35] of an IaaS platform is Amazon Web Services EC2, a service which provides, as Amazon's documentation [38] says, "virtual computing environments, known as instances". The instances can be configured from operating system upwards, and when created, can be used very similar to any normal remote computer by connecting to it using for example SSH [38].

3.3.2 Platform as a service

Platform as a service (PaaS) is an abstraction level on top of IaaS. A PaaS solution provides its users with an environment for developing and running applications. The environment contains support for a programming language and an assortment of libraries to use with it. In other words, PaaS offers consumers an application stack tailored for a purpose. [9][21][28]

A PaaS solution might for example provide a typical web server setup by offering a predefined database solution, an environment for running a server application in a specific programming language and framework, and serving the UI to users over the Internet in a configurable URL. An example of a PaaS which does just that is Heroku, which offers a variety of different programming languages and frameworks it knows how to run. One way of using Heroku is setting up a git repository which contains the web application code in a format Heroku knows, and git pushing the code to Heroku. [37]

3.3.3 Software as a service

Software as a service (SaaS) is the ultimate abstraction level of the service models. A SaaS solution provides the consumers with a complete, running application which can be accessed through the Internet. The only thing the consumer can configure are application-specific, with all the details of how the application is running hidden from them. [28]

SaaS solutions are plentiful on the Internet. Some of the common examples for them include enterprise resource planning (ERP) systems [21], and any web email client, for example Google Mail [9].

3.3.4 Machine learning as a service

Machine learning as a service (MLaaS) is not an abstraction level like the service models in the definition, but is a distinct set of services across previously mentioned service levels. Cloud providers [38][15][29] often provide a wide selection of SaaS-level services made possible by machine learning, like face detection from images, or turning speech audio into text. These services do not offer machine learning itself, but already trained models for common machine learning tasks.

The machine learning services themselves are usually PaaS-level solutions, which provide interfaces with machine learning libraries. The model training process is abstracted, letting the users to focus on the data itself [34]. Cloud services often also provide services for storing the data itself, and transferring and transforming it between services. The machine learning services are also often supplemented by services for using the created

model for predictions after training. This can be done for example by creating a HTTP endpoint which returns the model prediction for a request containing input data.

4 IMPLEMENTATION

This chapter will discuss the implementation of the machine learning architecture outlined in Chapter 1. First, the data and how it is used will be briefly discussed. Then, the requirements for the cloud service platform used are defined and listed. Then, the three biggest [35] cloud platforms are evaluated based on the requirements. Finally, one platform is chosen, and an architecture for implementing the machine learning applications will be presented.

4.1 Data

The data is stored in a relational database provided by Amazon Web Services. The data is operational in nature, and contains a history of work done in each worksite. The input data for the algorithm has to represent the state of the worksites in a static set of variables, which are not found in the database, and transformations have to be done first.

To represent a state and history of a worksite in a set number of variables, a number of metrics are calculated from the operational data. Each worksite's data is sampled with one hour intervals, removing hours of no work done to avoid duplicates. The metrics calculated from each of the hours will be an input data vector. So, if a worksite took ten working hours to complete, there will be ten rows in the final training data representing the different states of progression in that worksite.

A worksite's completion time is modeled as hours of work needed before completion, which means a machine learning model's output values is the number of working hours needed before the worksite is complete. This is much simpler to model than actual calendar completion time, and removes the need of handling actual dates and times. It also means that the algorithm will not be able to differentiate between worksites worked for example in different seasons. The hours of work needed before completion can only be calculated for already completed worksites, so the training data will only contain worksites completed before the extraction of the data. It is not a large setback however, as there are a lot more completed worksites than ones in progress.

4.2 Evaluation

The models are evaluated on two metrics: Mean squared error (MSE), and mean absolute error (MAE). All the models are trained minimizing mean squared error, as it is supported by all the model implementations used. Mean squared error tells straight how well the model minimized the loss value. Mean absolute error instead is a better estimate of how good the model is for its purpose, since mean squared error responds very strongly to outliers [44, p. 23], and eliminating a low amount of outliers is not as important as getting as good a prediction as possible for most of the worksites. Thus, mean absolute error is the metric used to choose the best model.

The data is split into 70% training data, and 30% test data. As there are multiple rows for each worksite in the whole data, there are two distinct ways of doing a random split. The first way, referred to as the *worksite* split, is made so that each worksite has all its data rows in either one set. The test set consists of roughly 30% of the worksites, and 30% of the total amount of the data rows. The second split, referred to as the *blind* split is made fully randomly, the test data consisting of 30% of the total amount of rows in the data. In the blind split, a worksite is likely to have its data both in the test set and the training set, which makes the task more of an interpolation than an extrapolation problem. The actual usage situation is an extrapolation problem, using past worksites' data to predict an unseen worksite's time of completion, so the worksite split is used for deciding the best machine learning model. The models are however tested with both splits, because the differences in their accuracy results can give insight into the quality of the parameter set extracted from the database.

4.3 Requirements

The first requirement for the cloud service platform is that it must be possible to implement all the machine learning models presented in Chapter 2 (Linear regression, Random forest, XGBoost and Multi-layer perceptron). All the models should be tested on a single platform to avoid having to create an architecture on multiple platforms.

The second requirement is that there must be a way to transform the data according to Section 4.1. The transformation cannot be done in the database, to keep the database only for operations. An integrated service built for transforming data is preferred to a general IaaS solution in which one would write a whole application for transforming the data.

The third requirement is that there must be a way to use the trained model for predictions after training. Although the architecture in this thesis is just a proof of concept, the same architecture should work if taken into production use. In its simplest, this could be just saving the model somewhere where another application can fetch the model and use it for predictions. However, an integrated solution is again preferred.

4.4 Platforms

This section will review the three biggest [35] cloud platforms from the viewpoint of fulfilling the requirements given in Section 4.3.

4.4.1 Amazon Web Services

Amazon Web Services (AWS) offers general machine learning services through a service called Amazon SageMaker. SageMaker enables building machine learning models, training them, and hosting them through HTTP endpoints. SageMaker offers multiple popular machine learning models built in. Of the models chosen for this thesis, XGBoost is offered built in. In addition, SageMaker offers library support for multiple machine learning frameworks. The additional frameworks include TensorFlow and PyTorch for creating and training neural networks, and Scikit-learn, which is capable of creating random forest and linear regression models [12]. Using the frameworks requires the user to write the training code themselves. For this purpose SageMaker offers Jupyter notebooks, where one may write the code in browser, and interactively run and test it. SageMaker is not designed to read data straight from a relational database, and needs something to create a file in AWS's object storage service S3 (Simple Storage Service) first. [38]

For transforming data and moving it between services, AWS offers two services: Glue and Data Pipeline. Glue is an ETL (extract, transform, load) service, which is specifically made for moving data between different data stores. Glue works by reading data in data source chosen from different options, transforming it with a script written in either Spark (using Python or Scala as programming language) or pure Python, and saving it to a data target. The process can be automatically run on a schedule. Glue offers notebook instances in the same way as SageMaker for developing the scripts, and has crawler functionality for automatically determining the schema of the data source. Data Pipeline is a service for automating movement of data between services. Data Pipeline provides connections to multiple data sources like Glue, and offers simple scheduling tools for running the transfers. Data Pipeline is however not capable of doing complex transformations to the data by itself, but is instead more designed to be used together with dedicated analysis services, making the data available for them to use. [38]

4.4.2 Google Cloud

Google cloud offers general machine learning services as AI platform. AI platform is very similar to AWS SageMaker in that it offers Jupyter notebooks for writing scripts, and several frameworks as well as built-in algorithms for creating models. The built-in models include XGBoost, and the other required models are covered by the frameworks, including TensorFlow and Scikit-learn. AI Platform also offers hosting the trained models

for predictions. AI Platform does not provide a managed way to prepare data for training, and another service is needed for that. [15]

For data transformation and transfer, there is one main service provided in Google Cloud: Cloud Dataflow. Dataflow is a service for doing stream or batch transformations to data using Apache Beam, a data stream processing framework. Dataflow does not seem to have a managed interface for accessing a relational database though, and would require a dedicated program to fetch wanted data from the database and save it into the object storage service Cloud Storage first.

4.4.3 Microsoft Azure

Microsoft Azure has their general machine learning services in a service called Azure Machine Learning. Azure Machine Learning works similarly to the other cloud platforms' corresponding services; it offers Jupyter notebooks for developing the machine learning scripts, with multiple machine learning frameworks. Azure Machine Learning software development kit supports Scikit-learn and TensorFlow, but XGBoost is not supported out-of-the-box. The service has possibility to use any framework of choice by using a general Estimator class, for which Python packages to install can be specified by name. [29]

Data transformation from SQL database in Azure can be done via a service called Data Factory. Data Factory can create a connection between two data stores, and supported datastores include Azure's object storage service, all Azure databases, multiple generic database and file system connections, and other cloud service providers' services. Data Factory supports multiple data transformation frameworks, including Spark and MapReduce. [29]

4.5 Architecture

This section will describe the architecture of the machine learning application created for testing the different models. First, the choice of the implementation platform will be justified, and then the data transformation achitecture and the machine learning architectures are separately presented.

4.5.1 Choice of platform

All of the cloud platforms listed in Section 4.4 are capable of the machine learning tasks required, although Azure requires a little more overhead, without the native support for XGBoost. The services in the platforms are in fact very similar to each other.

All the platforms offered data transformation, but Google Cloud documentation does not

outline a clear way of extracting data from a SQL database. Azure and AWS both offer very similar ways of transforming data in SQL database into a CSV file accessible for the machine learning services. The machine learning architecture is implemented in AWS, as it offered the most managed and suitable services for the requirements. The operational data is also already stored in AWS, removing the step of copying it to somewhere else.

4.5.2 Data transformation

Data transformation is done using the AWS Glue service. AWS Glue consists of connections, databases, crawlers, jobs, and triggers. A connection is simply the location of and credentials to the data store containing the input data. A database in Glue is a set of table definitions, which consist of the table schema containing column names and data types much like in SQL. All the table definitions are linked to a connection, to specify from where the data to the table should be fetched. Crawlers are a utility in Glue to automatically create table definitions for a database. A crawler is given a connection and some simple filters, and when run, it will create the table definitions for that connection.

Jobs are the main part of Glue. A job is a script written in either Spark or pure Python. Spark can be written using Python or Scala as the underlying language. A job reads data from Glue databases' tables, does transformations on them, and writes the data somewhere. In this thesis, the job reads data from the SQL database, does the hourly sampling described in Section 4.1, finds the features and the target values, and writes them as a single CSV file in AWS S3. A job can be automatically run with trigger functionality in Glue. A trigger can be configured to execute on a schedule, on a set of job events like failure or success, or on demand. In a production application, the job would be run on a scheduled trigger, but in this proof of concept architecture, they are not used, and the job is run by hand. The whole Glue process is illustrated in Figure 4.1.

4.5.3 Machine learning

Machine learning is implemented in AWS SageMaker. SageMaker is a managed machine learning service which provides tools to develop, train and deploy a machine learning model. For this thesis's purposes, SageMaker consists of training jobs, models, endpoints, and notebooks.

A training job contains all the needed instructions for creating and training a model. The information is the algorithm definition, input data configuration, output data configuration, and hyperparameters. An algorithm definition defines which machine learning model will be created when the job is run. A definition is linked to a training image in AWS Elastic Container Registry (ECR), which contains a docker image with all the needed programming languages and libraries for the algorithm. Algorithm definition also defined which AWS virtual machine (instance) type to use for training. There are multiple to

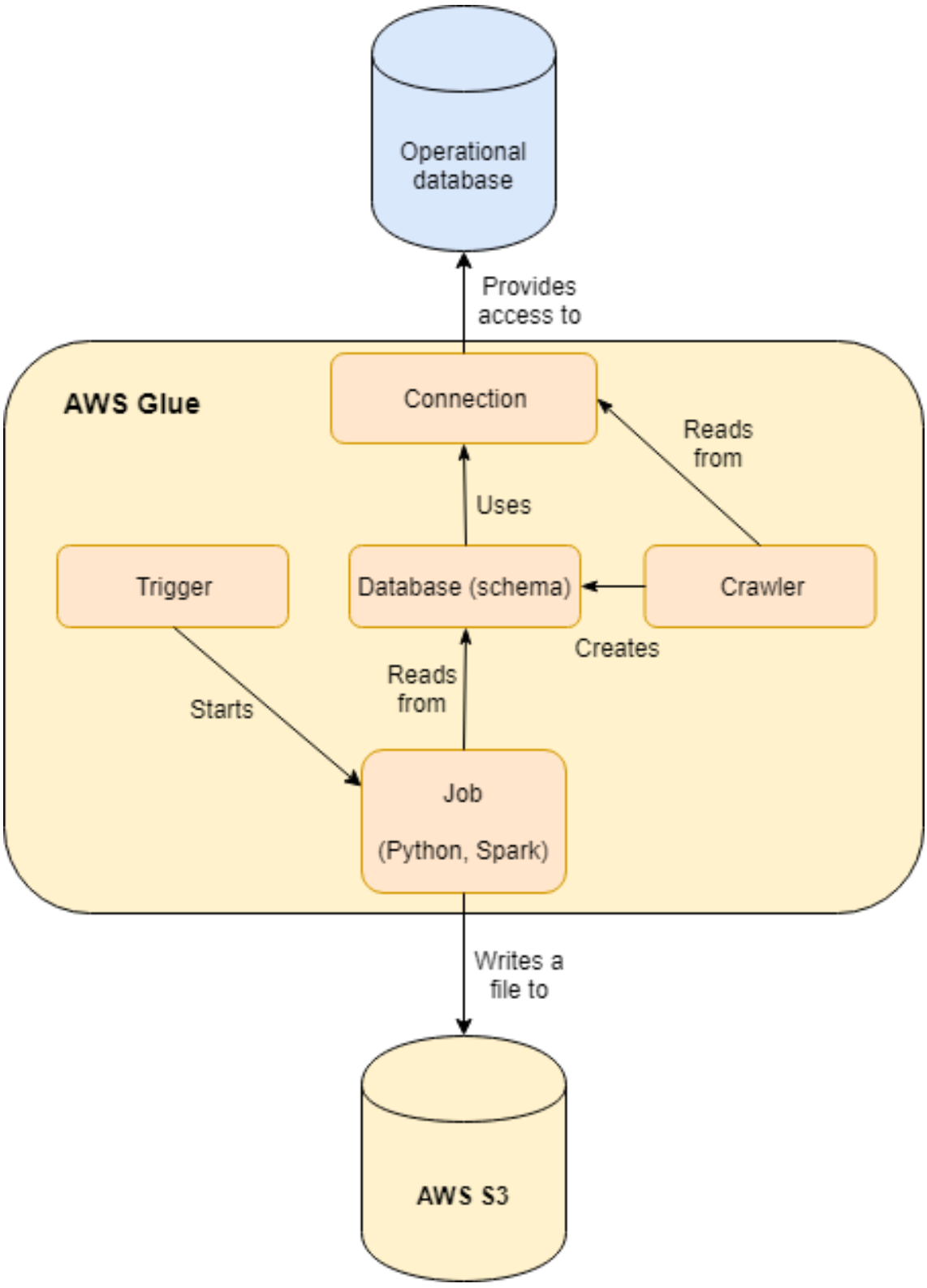

Figure 4.1. *AWS Glue for transforming data in a database into a CSV file.*

choose from, with capabilities optimized for different areas (CPU, memory, storage). Input and output data configuration contain the AWS S3 links where training data, test data and other possible data are located, and where the model will be saved. Hyperparameters contain the parameters for modifying the model structure, training process, etc. [38]

Models in SageMaker are the machine learning models created by training jobs. Each model is saved into S3, and can be used by SageMaker endpoints or downloaded to be used elsewhere. Endpoints use the models and create instances, which act as wrappers around the models. Endpoints specify a HTTP interface for getting predictions for input data with the model.

SageMaker notebooks can be used to interactively develop training jobs, run then, and deploy endpoints for using the model. Notebooks make creating the training jobs and endpoints easier, but they are not required for creating the machine learning pipeline in this thesis, and will not be discussed in more detail. Training job and endpoint creation can be automated using AWS Lambda, which can be used for scheduling and running tasks across the AWS platform. The machine learning pipeline is not fully automated in this thesis, as it is just a proof of concept. It is however possible to do using Lambda, should the pipeline be taken into production use.

Three algorithm definitions are used in this thesis: Scikit-learn, TensorFlow, and XGBoost. The scikit-learn image contains the scikit-learn library which can be used for both random forest and linear regression. The Scikit-learn algorithm takes an additional script as a hyperparameter. This script defines which actual Scikit-learn model will be created, and with which hyperparameters. Each of the Scikit-learn models is created their own script file, but otherwise they can use the same training image, which is provided by AWS [38]. Scikit-learn is also able to create a simple Multi-layer perceptron model, but offers nothing more than the simple perceptrons, and is missing for example dropout-functionality discussed in Section 2.2.3. The TensorFlow algorithm provides the library, and requires an additional script which constructs the neural network model using the library, much like with Scikit-learn. XGBoost is a managed algorithm in SageMaker, and can be used off-the-shelf. The whole process is illustrated in Figure 4.2.

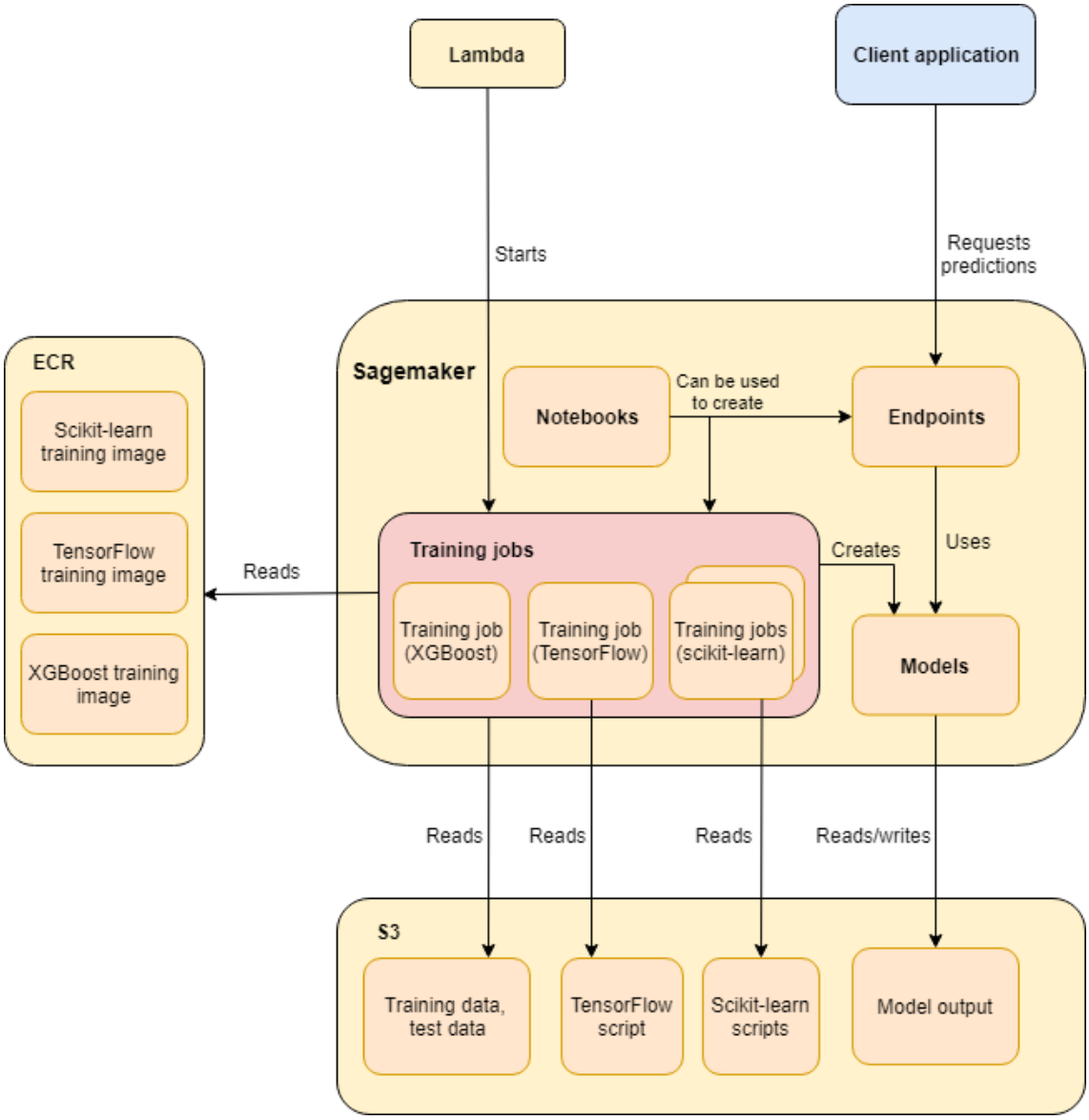

Figure 4.2. *AWS Sagemaker infrastructure.*

5 RESULTS AND DISCUSSION

In this chapter, the results of the machine learning pipeline are presented and discussed. First, the machine learning results are presented and explained, and second, the usability of the cloud platform for the pipeline created is evaluated.

5.1 Models

The results of the different models will be discussed in this chapter. For each model, its chosen hyperparameters are shown, and its performance in both chosen metrics and data splits described in Section 4.2 will be discussed. All hyperparameter optimization was done by hand, primarly aiming for as good a mean absolute error as possible with the dataset split by worksites.

5.1.1 Linear regression

Linear regression model, discussed in Section 2.2.1, is the simplest of the models used in the thesis. It requires no hyperparameters, so it was very straightforward to use. The results are illustrated in Figure 5.1. MAEs of the two splits are very similar, but MSE is, surprisingly, better for the worksite split. The reason for this is not clear. It is possible that because linear regression is a very simple model, it is already at the limit of its expressional power at the worksite split, and the added info of the blind split does nothing for the model, and by chance happens to be worse modeled by linear regression in this case.

Linear regression can only model linear relationships between variables, and could possibly have been made more accurate by introducing new, possibly more meaningful variables which are derived from the original parameters. Those parameters could be powers of the original parameters, or different parameters multiplied or divided by each other. Derivative parameter optimization was left outside the scope of this thesis.

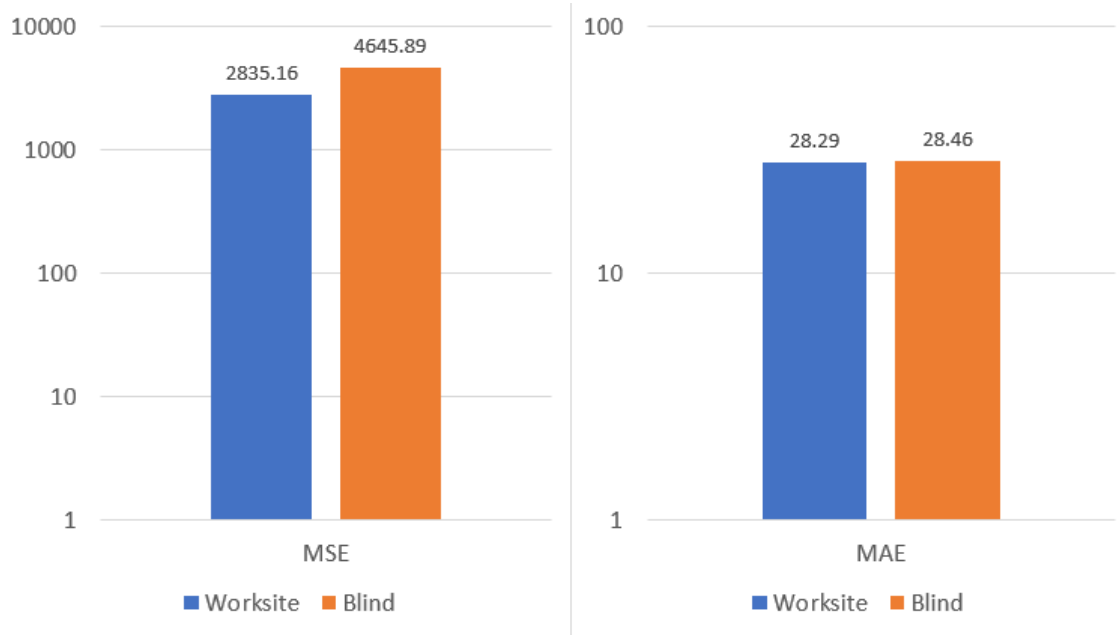

Figure 5.1. *MSE and MAE results for the linear regression model. Logarithmic scale.*

5.1.2 Decision tree ensembles

Random forest

Random forest, the model discussed at Section 2.2.2 is another easily configured model, where the only parameter to change from the Scikit-learn defaults was the number of trees. The number of trees was set to 64, as no significant improvement to accuracy was gained with any more trees. Even less trees are able to model the problem with relatively little accuracy loss, so the number of trees is rather arbitrary, as the accuracy only starts to noticeably drop when the number of trees goes below 10.

Random forest's results are shown in Figure 5.1. Random forest performs drastically better on the blind split in both MSE and MAE. Random forest does not need much configuration, and has an inherent resistance to over-fitting due to its random nature. This is why it does not generally need limiting parameters to counter over-fitting, and can fit very precisely to the data set in the case of the blind split.

Gradient boosting

Gradient boosting trees were implemented with the XGBoost algorithm, discussed in Section 2.2.2. XGBoost generally requires some parameter tuning to work efficiently. Multiple parameters were tried to tune, including min_child_weight, gamma, and eta, as well as the number and depth of individual trees [47, Notes on parameter tuning], but only a few parameters offered any noticeable difference to the performance. The parameters finally used for these results were the default values of the XGBoost library

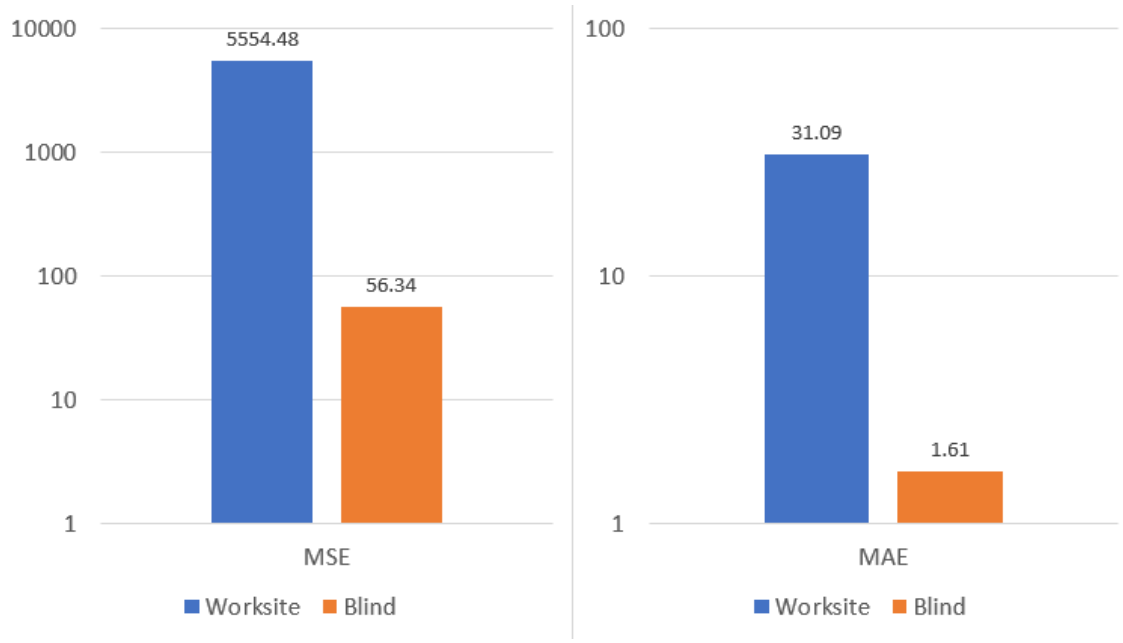

Figure 5.2. *MSE and MAE results for the random forest model. Logarithmic scale.*

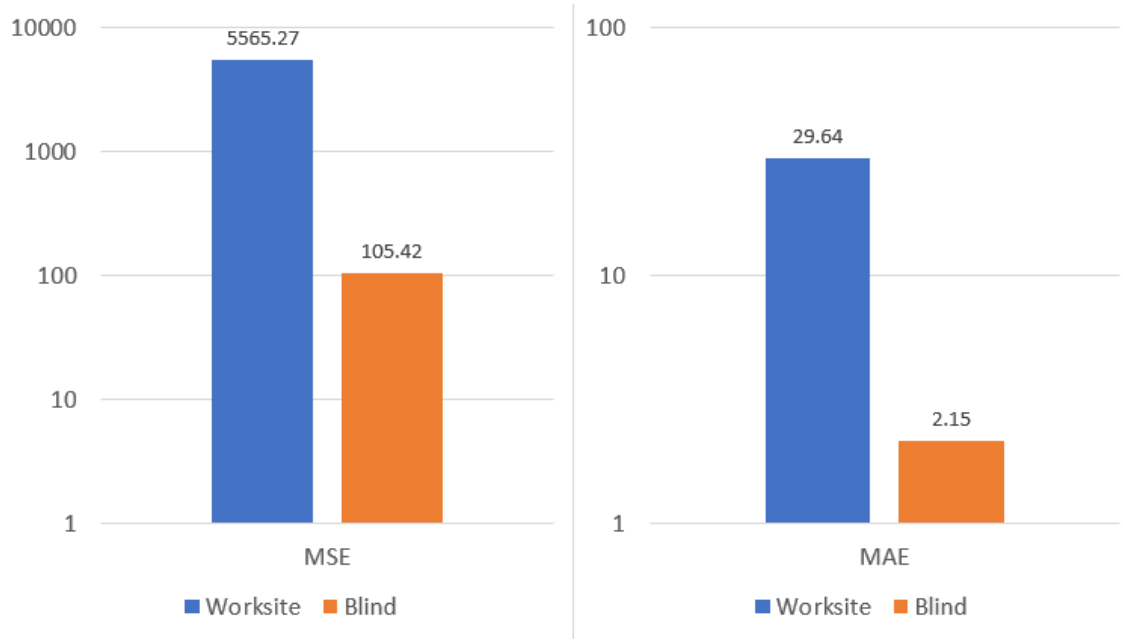

Figure 5.3. *MSE and MAE results for the XGBoost model. Logarithmic scale.*

used in SageMaker except for the number of trees, a tree's maximum depth, and eta (learning rate) which were set to 500, 8, and 0.1 respectively.

The XGBoost model's results are shown in Figure 5.3. XGBoost performs much better on the blindly split dataset than the one split by worksites. XGBoost's accuracy on the blind split could be further increased to the same level as random forest's by increasing the maximum depth and the number of trees, but that leads to smaller accuracy on the worksite split.

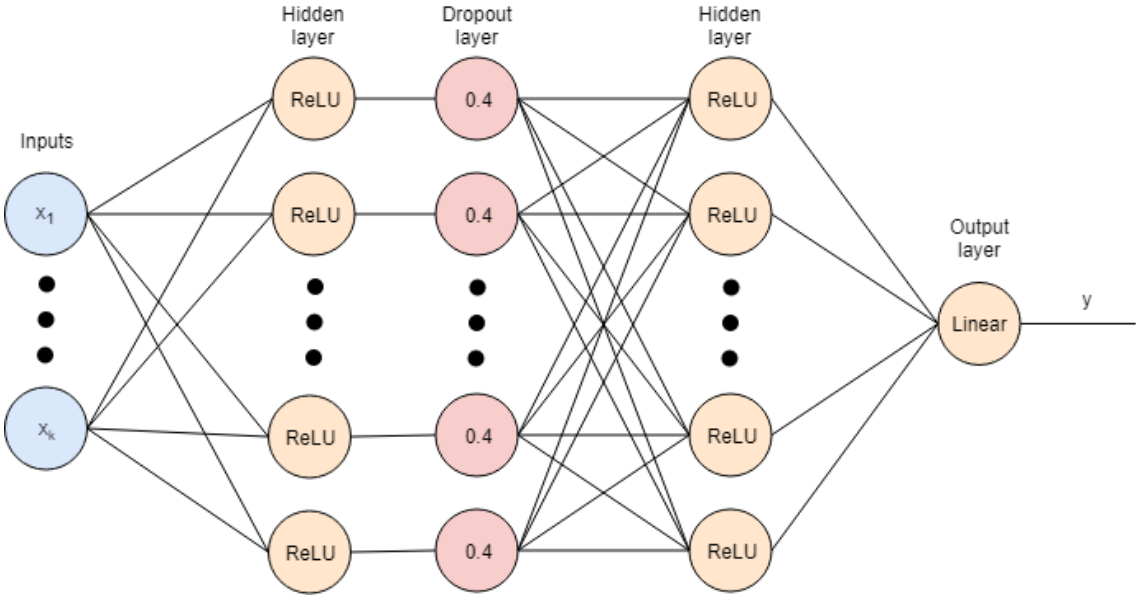

Figure 5.4. *Topology of the MLP network used.*

5.1.3 Artificial neural network

Artificial neural network was implemented as a multilayer perceptron, a model discussed at 2.2.3. MLP is the model requiring the most amount of fine tuning out of the ones used in this thesis. The best model found was a network with two hidden layers and a dropout layer between them. The hidden layers both had 1024 neurons, and the dropout layer in between had a dropout rate of 0.4. The hidden layer neurons had the rectified linear unit as their activation functions, and the output layer was a single neuron with a linear activation function. The network topology is illustrated in Figure 5.4. The gradient descent function used was *Adam*. For other hyperparameters, 2500 training steps was used, with the default batch size of 128. Learning rate was set to 0.001. A neural network model works best if the features are normalized [12, p. 214], so each feature was scaled using the Scikit-learn StandardScaler-function, which linearly transforms the inputs to have a mean of zero and a variance of one.

The MLP model' results are illustrated in Figure 5.5. MLP performed expectedly better on the blind split, but not as substantially as the tree-based models. The model's performance did not change drastically with other topology. Very similar results could be attained with the number of layers ranging from one to four, and decreasing the dropout rate did not make a large difference either. The choice of the gradient descent function made a larger difference. The very popular SGD did not work at all with the dataset and gave abysmal results. The RMSProp algorithm gave more acceptable results with MAEs in the 30s with the worksite split, but the Adam algorithm outperformed others in this dataset.

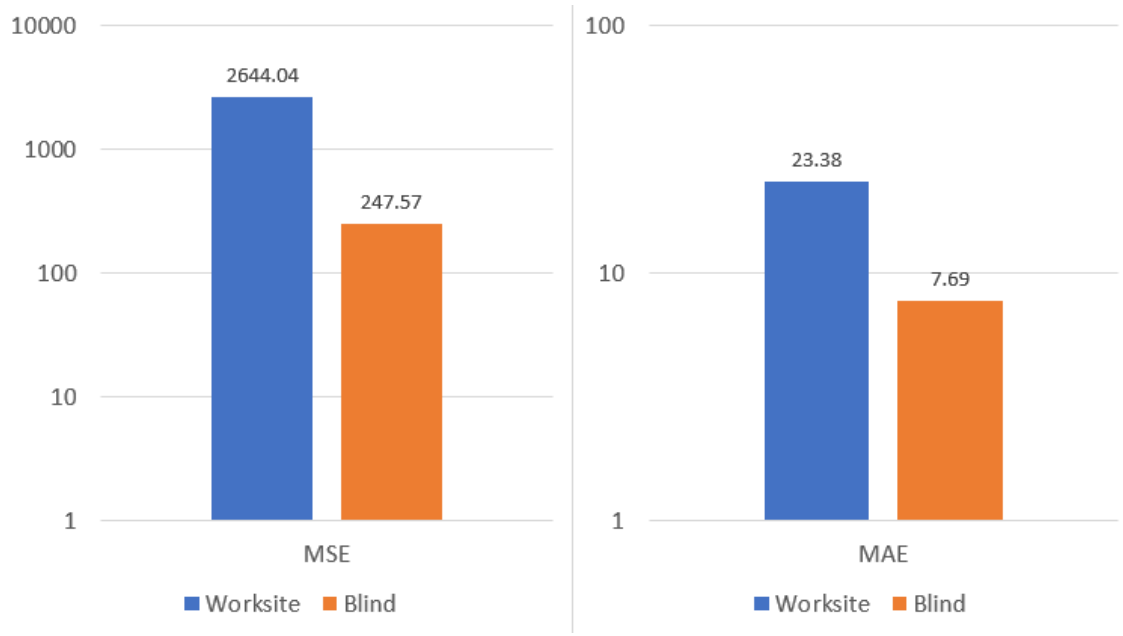

Figure 5.5. *MSE and MAE results for the multilayer perceptron model. Logarithmic scale.*

5.1.4 Comparison

Out of the machine models tested, the multilayer perceptron model performed the best on the primary evaluation metrics, which are MAE with the data split by worksites. The difference to the others was not substantial, but the network model performed noticeably better than all the others, which were very close to each other in score. The results with the primary evaluation metrics are shown in Figure 5.6. With the blindly split dataset, the tree-based ensemble models far outperformed the others.

All models except for linear regression performed better on the blind split data set. Especially the tree models performed vastly better on the interpolation task than the extrapolation task. This is somewhat expected, as trees in their core remember the input-output pairs and map new inputs to similar trained inputs' outputs. Thus, they work exceptionally well when the test data is very close to the training data. In general the models performing better on interpolation than extrapolation ultimately tells that worksites in the dataset are very different from each other, and the differences are not adequately captured in the feature set extracted from the operational database.

MLP being the best model on the primary evaluation metrics, although not profoundly better than the others, is unexpected. A study on incident duration prediction [45] did not find a neural network model to stand out from other models, except with an edge case with low number of training data. Recent comparisons on regression methods [27][26][7] have found tree ensembles, especially gradient boosted trees, to be the best performing models. In particular XGBoost has been recognized as a very suitable tool for many problems [33].

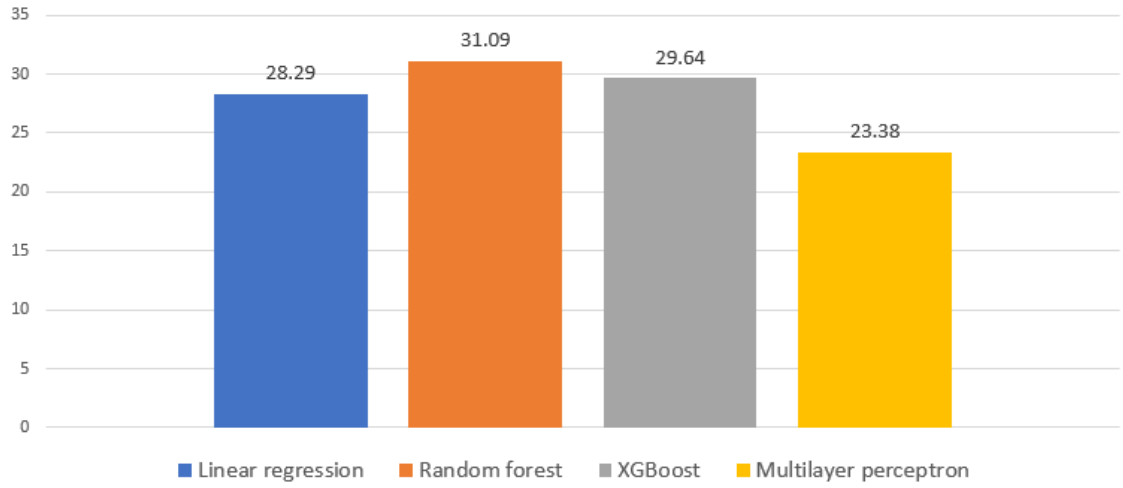

Figure 5.6. *Model MAE results with the worksite data split. Linear scale.*

5.1.5 Future development

Improvements to results could be made in both feature selection, and more hyperparameter tuning. The data did not have much information on the circumstances of the worksites, and the data was mostly statistics on the progress of the worksites. More circumstance data could improve capturing the evident differences between worksites. Another way feature selection could be improved is to introduce meaningful derivative variables, as explained in Section 5.1.1. The new variables would be especially useful for linear regression, but could improve other models' performances too.

In hyperparameter tuning a very obvious improvement which could not be done in the testing in this thesis is to change the error to minimize from MSE to MAE. A model could get better MAE results when minimizing MAE than when minimizing MSE. Last, a profound way of improving machine learning results is to add more data. The data set was not overly large, and could be much larger before model training time starts to be a problem.

5.2 Cloud platform

Amazon Web Services was chosen as the cloud platform to implement the machine learning algorithms. Using AWS for the task was mostly straightforward, helped by my own previous experience with the platform, and that the database was already located there. There are a lot of good AWS-provided tutorials and examples of almost everything possible to do with the SageMaker and Glue services. Most of the code written is heavily based on the official examples. Sagemaker and Glue both utilize Jupyter notebooks which made writing the code much more straightforward than it could have been. While developing in a notebook the developer seamlessly uses the other features of a service, which would

otherwise be much more difficult to use. For example, in SageMaker, a notebook script creates the training jobs and endpoints, of which configurations can be copied for later use without the manual interaction in the notebook.

The good examples are almost necessary though, as the API documentation can be hard to find and non-explanatory. If one wants to do something which has no example provided for, it can be difficult. Another occassional hardship with AWS was its permission system, which does not proactively warn about services missing permissions for using other services. When a permission is missing, the operation simply fails, and one has to look through the error logs to find out that a permission is missing and set that permission in another service. Luckily the error reports were descriptive.

6 CONCLUSIONS

The aim of this thesis was to implement a machine learning pipeline from operational database to a usable machine learning model in a cloud service platform. The pipeline built is a proof-of-concept case study, but should be transferable to a production environment as well. The implementation served two goals. The first was to try how the data applies to machine learning algorithms by testing multiple different models with it. The second was how the cloud platform applies to building the required pipeline.

The pipeline was built in Amazon Web Services and consists of Glue, a service meant for transforming and transferring data in AWS, and SageMaker, a machine learning service with multiple frameworks and models supported. Glue reads data from the operational database, extracts features from it, and saves it as CSV-files which SageMaker can read and use with its machine learning models.

The usability of the data was measured with the case study of predicting a worksite's completion time based on its current state and history. Four different models were implemented in SageMaker with two different ways of splitting the data to training and test set. The first set was to have all of single worksite's rows in either one of the files, and the second was to blindly split all rows between the files. This means the first split is an extrapolation problem, and the second is an interpolation problem. The first split was used as the primary split for evaluation, with mean absolute error as the error metric. The models tested were linear regression, random forest, XGBoost and multilayer perceptron. The best model was the multilayer perceptron, with a MAE of 23.38 hours. Tree-based models performed the best by a large margin with the second split's interpolation problem. In general models performed expectedly better on the data split blindly between files, which means worksites are very different from each other, and the features extracted do not adequately model the differences between them.

The AWS platform was very well-suited for building the machine learning pipeline. It provided the necessary managed services for the whole pipeline, so there was not any overhead code managing the pipeline self at all. Some difficulties were encountered with the platform's documentation, which can be difficult to search through, and the permission system, which can be laborious to use. The difficulties were overcome with excellent official examples for all the services, and good error reporting in case of missing permissions.

REFERENCES

[1] E. Alpaydin and I. Books24x7. *Introduction to machine learning*. English. Third; 3rd. Cambridge, Massachusetts; London, England: The MIT Press, 2014. ISBN: 0262325748.

[2] C. M. Bishop. *Pattern recognition and machine learning*. English. New York: Springer, 2006. ISBN: 0387310738.

[3] L. Breiman. *Classification and regression trees*. English. Belmont, Calif: Wadsworth, 1984. ISBN: 9780534980535.

[4] L. Breiman. Random Forests. English. *Machine Learning* 45.1 (2001), 5–32. DOI: 1010933404324.

[5] Buyya, Yeo, Venugopal, Broberg and Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* (2009). DOI: doi:10.1016/j.future.2008.12.00.

[6] V. Chandola, A. Banerjee and V. Kumar. Anomaly detection: A survey. English. *ACM Computing Surveys (CSUR)* 41.3 (2009), 1–58. DOI: 10.1145/1541880.1541882.

[7] J. Chen, K. de Hoogh, J. Gulliver, B. Hoffmann, O. Hertel, M. Ketzel, M. Bauwelinck, A. van Donkelaar, U. A. Hvidtfeldt, K. Katsouyanni, N. A. H. Janssen, R. V. Martin, E. Samoli, P. E. Schwartz, M. Stafoggia, T. Bellander, M. Strak, K. Wolf, D. Vienneau, R. Vermeulen, B. Brunekreef and G. Hoek. A comparison of linear regression, regularization, and machine learning algorithms to develop Europe-wide spatial models of fine particles and nitrogen dioxide. English. *Environment international* 130 (2019), 104934. DOI: 10.1016/j.envint.2019.104934.

[8] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. English. Vol. 13-17-. ACM, 2016, 785–794. ISBN: 1450-342329. DOI: 10.1145/2939672.2939785.

[9] T. Dillon, C. Wu and E. Chang. Cloud Computing: Issues and Challenges. English. IEEE, 2010, 27–33. ISBN: 1550-445X. DOI: 10.1109/AINA.2010.187.

[10] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. English. *The Annals of Statistics* 29.5 (2001), 1189–1232. DOI: 10.1214/aos/1013203451.

[11] J. H. Friedman. Stochastic gradient boosting. English. *Computational Statistics and Data Analysis* 38.4 (2002), 367–378. DOI: 10.1016/S0167-9473(01)00065-2.

[12] R. Garreta and G. Moncecchi. *Learning scikit-learn: machine learning in Python*. English. 1st ed. Birmingham: Packt Publishing, 2013. ISBN: 9781783281947.

[13] Z. Ghahramani. Unsupervised learning. English. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3176 (2004), 72–112.

[14] S. Gollapudi and V. Laxmikanth. *Practical machine learning: tackle the real-world complexities of modern machine learning with innovative and cutting-edge techniques*. English. 1st. Birmingham, UK: Packt Publishing, 2016. ISBN: 1784394017.

[15] Google. *Google Cloud Platform Documentation*. URL: https://cloud.google.com/docs/ (visited on 06/08/2019).

[16] S. Guha, N. Mishra, G. Roy and O. Schrijvers. Robust random cut forest based anomaly detection on streams. English. Vol. 6. 2016, 3987–3999. ISBN: 9781-510829008.

[17] G. Hackeling. *Mastering Machine Learning with scikit-learn*. English. 1st ed. Birmingham: Packt Publishing, 2014. ISBN: 1783988371.

[18] T. Hastie, R. Tibshirani and J. Friedman. *The elements of statistical learning: Data mining, inference, and prediction. 2nd ed., corrected at 11th printing*. English. New York: Springer, 2016. ISBN: 0172-7397. DOI: 10.1007/978-0-387-84858-7.

[19] A. R. Hevner, S. T. March, J. Park and S. Ram. Design Science in Information Systems Research. English. *MIS Quarterly* 28.1 (2004), 75–105. DOI: 10.2307/25148625.

[20] T. Kajiyama, M. Jennex and T. Addo. To cloud or not to cloud: how risks and threats are affecting cloud adoption decisions. English. *Information and Computer Security* 25.5 (2017), 634–659. DOI: 10.1108/ICS-07-2016-0051.

[21] M. Kavis and I. Books24x7. *Architecting the cloud: design decisions for cloud computing service models (SaaS, PaaS, and IaaS)*. English. Hoboken, New Jersey: Wiley, 2014. ISBN: 9781118826270.

[22] R. Kumar and S. Charu. Comparison between Cloud Computing, Grid Computing, Cluster Computing and Virtualization. Jan. 2015. DOI: 10.13140/2.1.1759.7765.

[23] Y. Lecun, Y. Bengio and G. Hinton. Deep learning. English. *Nature* 521.7553 (2015). Cited By :8286, 436–444. DOI: 10.1038/nature14539.

[24] X. Li. Preconditioned Stochastic Gradient Descent. *IEEE Transactions on Neural Networks and Learning Systems* 29.5 (May 2018), 1454–1466. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2017.2672978.

[25] P. Louridas and C. Ebert. Machine Learning. English. *IEEE Software* 33.5 (2016), 110–115. DOI: 10.1109/MS.2016.114.

[26] A. Lucas, R. Barranco and N. Refa. EV idle time estimation on charging infrastructure, comparing supervised machine learning regressions. English. *Energies* 12.2 (2019), 269. DOI: 10.3390/en12020269.

[27] H. Mao, J. Meng, F. Ji, Q. Zhang and H. Fang. Comparison of Machine Learning Regression Algorithms for Cotton Leaf Area Index Retrieval Using Sentinel-2 Spectral Bands. English. *Applied Sciences* 9.7 (2019), 1459. DOI: 10.3390/app9071459.

[28] P. M. Mell and T. Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Tech. rep. Gaithersburg, MD, United States, 2011.

[29] Microsoft. *Microsoft Azure Documentation*. URL: https://docs.microsoft.com/en-us/azure/ (visited on 06/08/2019).

[30] T. M. Mitchell. *Machine learning*. English. New York: McGraw-Hill, 1997. ISBN: 0070428077.

[31] D. C. Montgomery. *Introduction to linear regression analysis*. English. Fifth; 5th. Vol. 821. Hoboken, New Jersey: Wiley, 2013; 2012; 2015. ISBN: 1118640802.

[32] A. Natekin and A. Knoll. Gradient boosting machines, a tutorial. English. *Frontiers in Neurorobotics* 7 (2013), 21. DOI: 10.3389/fnbot.2013.00021.

[33] D. Nielsen. Tree Boosting With XGBoost - Why Does XGBoost Win "Every" Machine Learning Competition? MA thesis. Norwegian University of Science and Technology, 2016.

[34] M. Ribeiro, K. Grolinger and M. A. M. Capretz. MLaaS: Machine Learning as a Service. *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. Dec. 2015, 896–902. DOI: 10.1109/ICMLA.2015.152.

[35] Rightscale. *RightScale 2018 State of the Cloud Report*. Tech. rep. 2018. URL: https://assets.rightscale.com/uploads/pdfs/RightScale-2018-State-of-the-Cloud-Report.pdf.

[36] S. Ruder. *An overview of gradient descent optimization algorithms*. Jan. 2016. URL: http://ruder.io/optimizing-gradient-descent/ (visited on 05/24/2019).

[37] Salesforce. *Heroku Dev Center*. URL: https://devcenter.heroku.com/ (visited on 04/06/2019).

[38] A. W. Services. *AWS Documentation*. URL: https://docs.aws.amazon.com/index.html (visited on 04/06/2019).

[39] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. English. *Nature* 529.7587 (2016), 484–489. DOI: 10.1038/nature16961.

[40] R. Smith, A. Bivens, M. Embrechts, C. Palagiri and B. Szymanski. Clustering approaches for anomaly based intrusion detection. *Proceedings of intelligent engineering systems through artificial neural networks* (2002), 579–584.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. English. *Journal of Machine Learning Research* 15 (2014), 1929–1958.

[42] R. S. Sutton, A. G. Barto and F. Bach. *Reinforcement learning: an introduction*. English. Second. London, England; Cambridge, MA: The MIT Press, 2018. ISBN: 0262039249.

[43] K. Suzuki, L. Zhou and Q. Wang. Machine learning in medical imaging. English. *Pattern Recognition* 63 (2017), 465–467. DOI: 10.1016/j.patcog.2016.10.020.

[44] L. Torgo. Inductive learning of tree-based regression models. English. *AI COMMUNICATIONS* 13.2 (2000), 137–138.

[45] G. Valenti, M. Lelli and D. Cucina. A comparative study of models for the incident duration prediction. *European Transport Research Review* 2.2 (2010). ID: Valenti2010, 103–111. DOI: 10.1007/s12544-010-0031-4.

[46] S. Weisberg. *Applied linear regression*. English. Fourth. Hoboken, New Jersey: Wiley, 2014; 2013. ISBN: 9781118789551.

[47] *XGBoost documentation*. 2016. URL: https://xgboost.readthedocs.io/ (visited on 05/04/2019).

[48] V. K. Xindong Wu. *The Top Ten Algorithms in Data Mining*. English. CRC Press, 2009, 179–201. ISBN: 9781420089646.

[49] Q. Zhang, L. Cheng and R. Boutaba. Cloud computing: state-of-the-art and research challenges. English. *Journal of Internet Services and Applications* 1.1 (2010), 7–18. DOI: 10.1007/s13174-010-0007-6.