Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt

2017•2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

# Masterthesis

Using SVM and Deep Learning to facilitate orthognathic surgery planning

PROMOTOR :
Prof. dr. ir. Luc CLAESEN

PROMOTOR :
dr. ing. Yi SUN
ing. Wout SWINKELS

Danilo Peeters
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

2017•2018
Faculteit Industriële ingenieurswetenschappen
**master in de industriële wetenschappen: elektronica-ICT**

# Masterthesis

Using SVM and Deep Learning to facilitate orthognathic surgery planning

**PROMOTOR :**
Prof. dr. ir. Luc CLAESEN

**PROMOTOR :**
dr. ing. Yi SUN
ing. Wout SWINKELS

**Danilo Peeters**

# Acknowledgements

This Master's Thesis was performed during the entire academic year of 2017-2018. Even though the journey has been full of ups and downs, the experience and knowledge I have attained during this period have been invaluable. I have discovered my interest and passion for machine learning, which opened up new perspectives regarding my future career. During this time I have learned how to put my engineering skills to practice but, more importantly, I have had the pleasure to work with amazing people whom are experts in their field.

First and foremost, I would like to thank dr. ing. Yi Sun for allowing me to work on this interesting project and for providing his insightful advice. He taught me a lot about orthognathic surgery planning and how to interpret results from a medical standpoint.

Secondly, I would like to thank prof. dr. ir. Luc Claesen for allowing me to take this opportunity and for always believing in the good outcome of this project. I believe the results of this Master Thesis might have differed greatly without his counseling.

Thirdly I would like to extend my gratitude to ing. Wout Swinkels for his guidance during the project and for his company when meeting in Leuven.

Finally I would like to thank my friends and family. I firmly believe this thesis would have looked very differently were it not for their support.

# Abstract

Using machine learning, surgeons can be aided in performing pre-surgical planning for orthognathic surgery. Before every surgery, a clinical study is performed to gather insight into the patient's situation. This clinical study consists of clinical measurements and a cephalometrical analysis which account for a total of 120 parameters per patient. This amount of data complicates decision-making for young, inexperienced surgeons. An attempt has been made to use machine learning to predict multiple surgical decisions, including predictions with regard to the patient's maxilla, mandible and chin. These models have been built within a local Python environment, and have been deployed afterwards on a web server to be used in conjunction with a user-friendly web interface. For the analogue values, multiple neural networks are constructed using ReLU activations and a linear output activation. These networks are constructed for each surgery decision separately using 4 hidden layers with each 20 nodes. Each network uses the 20 most correlated input features as input nodes. The discrete values are determined using two SVM implementations using a linear and RBF kernel, each determined by a personalized grid search. The regression output values are lowly correlated with the input data, and should be improved upon by acquiring more data ($R^2_{test} \approx 0.5$). The classification SVMs both predict their output values with an accuracies of 83% and 86% and similar F1-scores.

# Abstract (Dutch)

Machine learning kan de pre-operatieve planning binnen de mond-, kaak- en aangezichtschirurgie vergemakkelijken. Vóór elke operatie wordt een klinische studie uitgevoerd om inzicht te verkrijgen in de situatie van de patiënt. Deze klinische studie bestaat uit klinische metingen en een cefalometrische analyse welke instaan voor een totaal van 120 parameters per patiënt. Deze hoeveelheid informatie compliceert de besluitvorming voor jonge, onervaren chirurgen. Er werd een poging ondernomen om machine learning te integreren om meerdere operatieve ingrepen te voorspellen met betrekking tot bovenkaakbeen, onderkaakbeen en kin. Deze modellen werden eerst gebouwd in een lokale Python-omgeving, en werden later geïmplementeerd op een webserver om gebruikt te worden met een gebruiksvriendelijke interface. Voor de analoge waarden werden meerdere neurale netwerken gebouwd gebruik makende van ReLU activeringen en een lineaire outputactivatie. Deze netwerken werden apart gebouwd voor elke operatieve beslissing gebruik makende van 4 verborgen lagen met elk 20 eenheden. Elk netwerk gebruikt de 20 meest gecorreleerde ingangskenmerken als ingangsknopen. De discrete waarden werden bepaald door twee SVM implementaties met een lineaire en RBF kern, elk bepaald door een gepersonaliseerde grid search. De uitgangswaarden voor regressie zijn slecht gecorreleerd met de uitgangsresultaten, en zouden verbeterd kunnen worden door meer gegevens te verzamelen ($R^2_{test} \approx 0.5$). De classificatie SVMs voorspellen beiden uitgangsgegevens met een nauwkeurigheid van 83% en 86% en gelijkaardige F1-scores.

# Contents

# Glossary

**activation function** Function which is applied to the hidden units in a neural network. 9, 10, 23

**backpropagation** The act of calculating the error of each node with regard to the actual output values, starting from the output layer. 10

**bias unit** Nodes containing the value +1 which are only connected to the nodes in the next layer. They are used for implementing the constant value in the cost function for neural networks. 9

**Bimax** Surgery where both the maxilla and mandible are operated on. 16

**BSSO** Surgery where only the mandible is operated on. 16

**classification based algorithm** Supervised learning algorithm which outputs the probability of the combination of one or several input features $x_n^{(i)}$ being in a discrete class $y_k^{(i)}$. 4, 12, 15, 20, 24, 29

**confusion matrix** Schematic overview of (in)correctly classified examples in classification problems. 12

**cost function** Function which describes the error between the hypothesis and the actual values. 5, 6, 9, 10, 14, 20

**feature** A feature of a dataset is a type of data that is used in a machine learning algorithm. 4, 8, 11, 14, 15, 17–19, 21, 23, 29

**forward propagation** The act of calculating the values of all the nodes in the neural network, starting from the input layer. 9

**gradient descent** Algorithm for determining the minimum value of a function. 5, 8

**grid search** Act of defining the best hyperparameters for machine learning algorithms by iterating through different combinations of these parameters. 14, 15, 19, 20, 24, 26

**hidden layer** The layers of a neural network which are situated between input and output layer. 8, 19, 29

**hyperparameter** Parameters used to optimize machine learning algorithms. 14, 15, 20, 24, 29

**input layer** One layer which contains all the input features $x_n^{(i)}$ from one or multiple training example(s). 8

**kernel** A kernel function provides a measure of the distance between feature $x^{(i)}$ and landmark $l^{(i)}$. 11, 14, 19, 20, 24, 29

**Le Fort I** Surgery where only the maxilla is operated on. 16

**node** Calculation unit which takes inputs from all nodes in the previous layer and activates this value. 8, 9, 14, 19, 23

**output layer** One layer with one or more nodes representing the output class(es) of the neural network. The output of these nodes is the probability of input features $x_n(i)$ belonging to the class that node represents. 8–10, 19

**regression based algorithm** Supervised learning algorithm which outputs an analogue value $y^{(i)}$ based on one or several input features $x_n^{(i)}$. 4, 11, 15, 20, 27

**supervised learning algorithm** Algorithm with one or several input features $x^{(i)}$ with a descriptive output $y^{(i)}$. 3, 4, 29

**training example** One row of values from the dataset, containing all the input features and output value(s) used for training the algorithm and determining the cost function. 4–6, 9, 10, 13, 14, 28

**unsupervised learning algorithm** Algorithm with one or several input features $x^{(i)}$ without a descriptive output $y^{(i)}$. 3

# Acronyms

# List of Figures

# List of Tables

Chapter 1

# Introduction

Due to the significant increase in computing power over the past decade, machine learning has become more prevalent in a variety of sectors. Within the dentistry sector, attempts have already been made to predict the necessity of extracting teeth during orthodontic treatments [4]. This research focuses on pre-operative orthognathic surgery decisions and is performed in cooperation with the Computation Sensor Systems (CoSenS) research group of Hasselt University and the University Hospitals Leuven. The University Hospitals Leuven is a reputable hospital with four campuses located in Leuven. Campus Sint-Rafaël, which is one of the four campuses, has a department that is specialized in oral and maxillofacial surgery.

## 1.1 Problem statement

Before performing orthognathic surgery, clinical data has to be collected in order to make surgical decisions. The acquisition of this data is done through clinical studies. These studies include acquiring images of the patient in order to virtually reconstruct their countenance. This part is necessary in order to gain perspective on the patient's situation and to perform pre-surgical planning. Images are taken using Cone-Beam CT (CBCT) (figure 1.1), but often additional image acquisition is required in order to accurately represent the patient's head. Computer programs can then derive the distances and angles between certain points on the patient's head. This data is mostly saved as numerical data, both as absolute value and as deviation from the clinical norm [1].

The clinical studies present a substantial amount of data to the surgeons. Based on these data, surgeons make decisions on which kind of surgeries to perform and on how to perform them. It seems evident that surgeon's experience pertains a vital role in the difficulty and success of the surgery, which unexperienced surgeons usually do not possess. It is therefore useful to create a system which helps unexperienced surgeons with their decisions. Currently, no such system exists within the department which translates this significant amount of data into an effective decision pattern. This thesis proposes several methods of machine learning which aid the surgeon in this process.

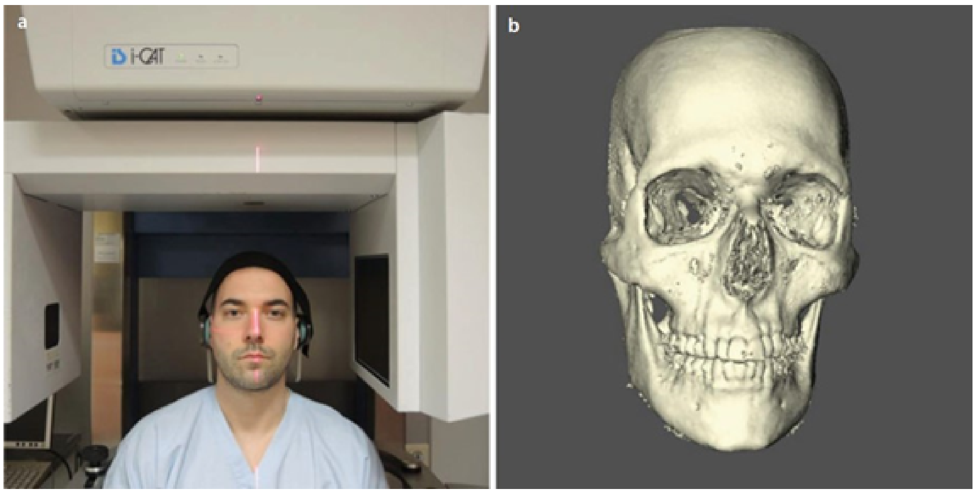

**Figure 1.1:** CBCT imaging of patient's head and 3D representation [1, p. 22]

Chapter 2

# Literature study

This chapter covers the fundamentals of machine learning which are necessary to understand the concepts behind the algorithms used in this thesis. This chapter will first explain the different concepts of machine learning. Secondly, it will cover the basics of the used algorithms from a mathematical perspective. It will then explain different performance metrics which are used to evaluate algorithms. These performance metrics will usually highlight certain issues with regard to under- and overfitting, which will be addressed afterwards. This information can then be used to find the optimal model as discussed in section 2.5. Finally, the data used in this project will be explained more in detail.

## 2.1 The concepts of machine learning

According to [5], Machine Learning is defined as follows:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [5, p. 14].

For example, a robot trained to play a game of checkers would become better at playing checkers (task T) by measuring the amount of wins (performance P). The amount of wins should increase over time as the robot plays more games and learns to distinguish good moves from bad moves (experience E).

Task T refers to a problem which is too difficult or time-consuming for humans. This becomes evident when the amount of input data increases. Due to the amount of moves available in checkers at any point in time, it becomes nearly impossible for the human brain to predict which move would result in the best outcome.

The performance measure P heavily depends on the desired outcome of the algorithm. For the example explained above, the performance measure could be a value which correlates to the amount of games won. More iterations of the algorithm should result in a higher value for P.

The experience E depends heavily on the type of data set. Generally, it suffices to classify machine learning algorithms into two categories:

- Unsupervised learning algorithms;
- Supervised learning algorithms.

Unsupervised learning consists of an unstructured data set. Most often, this means that there are several input features $x^{(i)}$ without a descriptive output $y^{(i)}$ which corresponds to these inputs. The unsupervised learning algorithm attempts to classify these data into groups. This is not the case for supervised learning algorithms. Supervised learning algorithms contain one or several input features $x^{(i)}$ which are directly linked to an output $y^{(i)}$ [6].

## 2.2 Supervised learning algorithms

Due to the nature of the problem presented in this thesis, it suffices to explain supervised learning algorithms only. Supervised learning algorithms are classified into two main categories:

- Regression based algorithms;
- Classification based algorithms.

This thesis will make use of both types of algorithms, and so both types will be explained more in detail.

### 2.2.1 Linear regression

Regression based supervised learning algorithms work on data sets which plot an output value $y^{(i)}$ as a function of several input features $x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}$. In this notation, $n$ stands for the index of the features and $i$ represents the $i^{th}$ training example. An example of a (small) data set is represented in table 2.1.

**Table 2.1:** Example of a small data set with 5 training examples and 2 features.

| $i$ | $x_1^{(i)}$ | $x_2^{(i)}$ | $y^{(i)}$ |
|---|---|---|---|
| 1 | 10 | 10 | 100 |
| 2 | 10 | 50 | 200 |
| 3 | 50 | 10 | 120 |
| 4 | 50 | 50 | 250 |
| 5 | 30 | 50 | 225 |

In this data set, every set of input features corresponds to one output value. In other words, $y^{(i)} = f(x_1^{(i)}, x_2^{(i)})$. From this data set, however, it is impossible to determine the output value for say $f(10, 40)$ as there is no data point with these values in the dataset. To determine this value, it is necessary to form a hypothesis which best describes the behaviour of this data set. For the sake of simplicity, multivariate linear regression is chosen as a candidate. This hypothesis is described in equation 2.1 and outputs real, continuous values.

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} \tag{2.1}$$

If $\theta$ and $x$ are described as column vectors it is possible to rewrite the hypothesis as shown in equation 2.2.

$$h_\theta(x) = \theta^T X \tag{2.2}$$

Given that $x_1$ and $x_2$ are known, it is necessary to find the optimal values for $\theta$ in order for the hypothesis to best describe the original data set. The error of the hypothesis is defined as the squared distance between the values of the hypothesis and the original

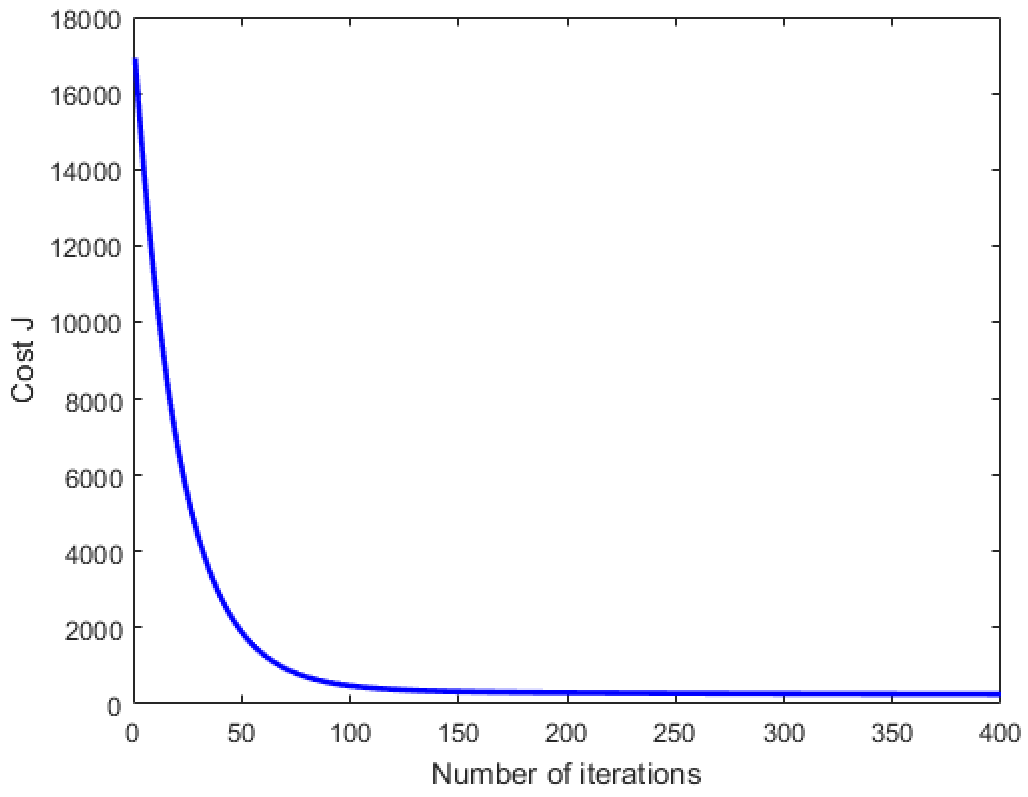

**Figure 2.1:** Gradient descent for 400 iterations [2]

output values. This error is divided by the amount of training examples multiplied by two. This function is known as the cost function and is described in equation 2.3. It is then further broken down in equation 2.4. The meaning of the term $\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ will be explained in section 2.4.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{2.3}$$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left((\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{2.4}$$

Given that this function represents the error of the hypothesis, it seems evident that the best possible values for $\theta$ are determined by minimizing this function. One possible method is to use gradient descent, in which an iteration can be formulated as in equation 2.5

$$\theta_n = \theta_{n-1} - \gamma \times \nabla J(\theta) \tag{2.5}$$

The value of $\gamma$ has to be chosen so that the algorithm does not diverge or take a considerable time to compute. When implementing equation 2.5, it is important to update every value of $\theta$ simultaneously as to make sure that the same original function is used for gradient descent. In this example, a value of $\gamma = 0.00001$ is chosen. For this dataset, the cost function is plotted as a function of the number of gradient descent iterations, which is shown in figure 2.1. The optimal values for $\theta_n$ are shown in table 2.2. Equation 2.1 can now be applied with these values of $\theta$. For $x_1 = 10$ and $x_2 = 40$, this prediction is shown in equation 2.6.

**Table 2.2:** Optimal values for theta after 400 iterations

| $\theta_0$ | $\theta_1$ | $\theta_2$ |
|---|---|---|
| 0.101682 | 1.835737 | 3.410662 |

$$h_\theta(x^{(i)}) = 0.10 + 1.84 \times 10 + 3.41 \times 40 = 154.9 \tag{2.6}$$

### 2.2.2 Logistic Regression

The hypothesis of a linear regression function always outputs a real and continuous value. Logistic regression instead classifies input data into discrete output classes. One example includes the classification of spam e-mail, where an algorithm is trained on pre-determined data to classify new e-mails. The precedent for this algorithm is shown in equation 2.7 and is summarized in equation 2.8

$$h_\theta(x) = \begin{cases} 1, & \text{if } h_\theta(x) > k \ | k \in [0,1] \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

$$h_\theta(x) = P(y = 1 | x; \theta) \tag{2.8}$$

The hypothesis $h_\theta(x)$ now outputs the probability of the training example belonging to discrete class 1 or discrete class 0. Threshold value $k$ determines the minimum probability of a new training example in order to be classified as a discrete 1. For a proportional distribution, a value of $k = 0.5$ is used.

In this hypothesis, linear regression from section 2.2.1 cannot be used as doing so would result in values outside the range of $[0, 1]$. Instead, the new hypothesis is defined as in equation 2.9.

$$h_\theta(x) = g(\theta^T X) \tag{2.9}$$

The function $g(x)$ is the sigmoid function defined in equation 2.10. Applying the sigmoid function to the hypothesis converts all the values of the hypothesis to values in the range of $[0, 1]$. This allows the binary classification represented in equation 2.7.

$$g(x) = \frac{1}{1 + \exp(-z)} \tag{2.10}$$

The cost function for logistic regression looks similar to the cost function in section 2.2.1 with the exception of new logarithmic terms surrounding the hypothesis (2.11).

$$cost(h_\theta(x^{(i)}, y)) = \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{otherwise.} \end{cases} \tag{2.11}$$

The reason for the addition of the logarithmic terms can be explained by examining figures 2.2 and 2.3. For $y = 1$, the cost function from figure 2.2 will be used. The cost for the input value x decreases as the hypothesis approaches value 1. For $y = 0$, the cost function shown in figure 2.3 will be used instead. An erroneously predicted value results in a high cost, whereas correctly predicted values result in a low cost. This behaviour can be summarized by equation 2.12.

$$J(\theta) = -\frac{1}{m} \times \sum_{i=1}^{m} \left( y^{(i)} \times \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \times \log(1 - h_\theta(x^{(i)})) \right) \tag{2.12}$$

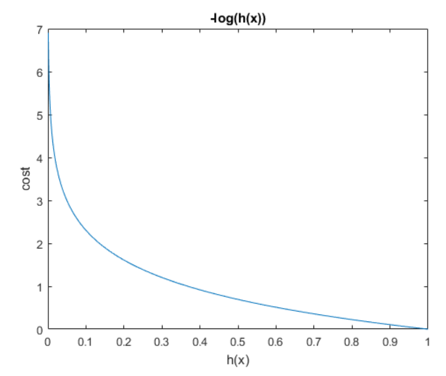

**Figure 2.2:** Logistic regression: $cost(h_\theta)$ for $y = 1$ [2]

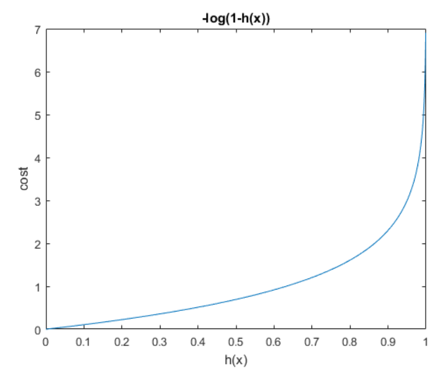

**Figure 2.3:** Logistic regression: $cost(h_\theta)$ for $y = 0$ [2]

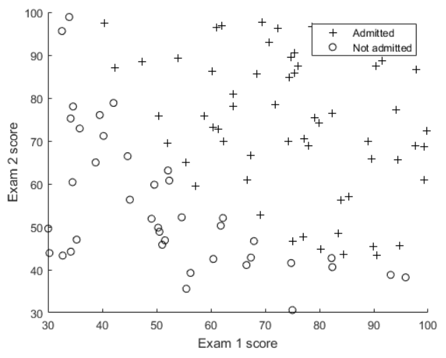

**Figure 2.4:** Logistic regression: Original dataset [2]

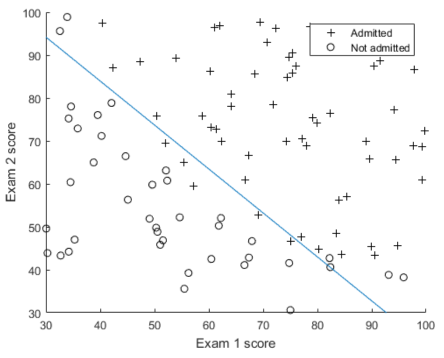

**Figure 2.5:** Logistic regression: Original dataset with decision boundary [2]

**Table 2.3:** Optimal values for $\theta$ for logistic regression (values rounded to 2 decimals)

| $\theta_0$ | $\theta_1$ | $\theta_2$ |
|---|---|---|
| -24.93 | 0.20 | 0.20 |

The values for $h_\theta(x)$ can also be interpreted as the parameters for the decision boundary of the dataset. Given that the dataset only contains two features, the decision boundary can be calculated using the same hypothesis as in equation 2.9 but without applying the sigmoid function. The values which lie on this line are the values for which $P(y = 1|x; \theta) = 0.5$, which signifies the boundary. For the dataset from figure 2.4, the $\theta$ values from table 2.3 were found using gradient descent. This equation can be verified by examining figure 2.5 (Note the interval of the axes).

$$y_{\text{decision}} : x_1, x_2 \mapsto -24.93 + 0.20 \times x_1 + 0.20 \times x_2 \tag{2.13}$$

$$x_2 = -x_1 + 124.65 \tag{2.14}$$

### 2.2.3 Neural networks

Neural networks mimic the operation of the human brain. In machine learning, they are mostly used to classify data sets which contain many features. It is possible, however, to also use neural networks for regression problems. Neural networks are an extension of the logistic regression algorithm described in section 2.2.2.

#### Model Representation

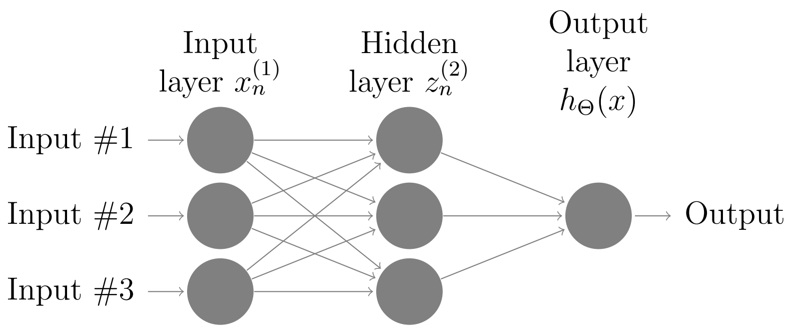

**Figure 2.6:** Neural network representation (one hidden layer, three layers in total)

Figure 2.6 depicts a neural network. It generally consists of one input layer, one output layer and $N$ hidden layers (in total $L$ layers). Each circle represents a *unit* which is connected to the previous and next layer. In case of the input layer, the units are initialized with the values of the chosen features. Each connection between the units represents a calculation where the result is passed on to the next layer. In the following example, the calculation used is the multiplication of the node's value with $\theta$. Given that the units of the input layer represent the features $x_1, x_2, x_3$, and the units of the hidden layer are represented by $z_1, z_2, z_3$, equation 2.15 applies.

$$\begin{cases} z_1^{(2)} = \Theta_{1,1}^{(1)} \times x_1 + \Theta_{1,2}^{(1)} \times x_2 + \Theta_{1,3}^{(1)} \times x_3 \\ z_2^{(2)} = \Theta_{2,1}^{(1)} \times x_1 + \Theta_{2,2}^{(1)} \times x_2 + \Theta_{2,3}^{(1)} \times x_3 \\ z_3^{(2)} = \Theta_{3,1}^{(1)} \times x_1 + \Theta_{3,2}^{(1)} \times x_2 + \Theta_{3,3}^{(1)} \times x_3 \end{cases} \tag{2.15}$$

$\Theta^{k}_{i,j}$ represents the value of $\theta$ applied to unit $j$ in layer $k$, which results in a new value for unit $i$ in layer $k+1$. It is important to note that in equation 2.15 and in figure 2.6 the constant value $x_0 = 1$ to account for $\theta_0$ is not represented yet. To correctly account for this term, 2.15 is replaced by equation 2.16.

$$\begin{cases} z^{(2)}_1 = \Theta_{1,0} \times x_0 + \Theta^{(1)}_{1,1} \times x_1 + \Theta^{(1)}_{1,2} \times x_2 + \Theta^{(1)}_{1,3} \times x_3 \\ z^{(2)}_2 = \Theta_{2,0} \times x_0 + \Theta^{(1)}_{2,1} \times x_1 + \Theta^{(1)}_{2,2} \times x_2 + \Theta^{(1)}_{2,3} \times x_3 \\ z^{(2)}_3 = \Theta_{3,0} \times x_0 + \Theta^{(1)}_{3,1} \times x_1 + \Theta^{(1)}_{3,2} \times x_2 + \Theta^{(1)}_{3,3} \times x_3 \end{cases} \tag{2.16}$$

It is important to mention that these values are still continuous values. To convert them to values in the range of $[0, 1]$, the sigmoid function from 2.10 is applied. It is possible to use other so-called activation functions as well.

$$\begin{cases} a^{(2)}_1 = g(z^{(2)}_1) \\ a^{(2)}_2 = g(z^{(2)}_2) \\ a^{(2)}_3 = g(z^{(2)}_3) \end{cases} \tag{2.17}$$

The values for $x_0$, $a^2_0, \dots, a^{L-1}_0$ are not drawn by convention. They are called *bias units* and are equal to the value $+1$. They require no inputs from previous layers and are only used to calculate the constant value of the linear expression in the next layer. The output of this neural network can then be calculated according to equation 2.18.

$$\text{output} = h_\Theta(x^{(i)}) = \Theta^{(2)}_{1,0} \times a_0 + \Theta^{(2)}_{1,1} \times a_1 + \Theta^{(2)}_{1,2} \times a_2 + \Theta^{(2)}_{1,3} \times a_3 \tag{2.18}$$

For neural networks with one output, $h_\Theta(x)$ represents the probability of the $i^{th}$ training example belonging to class 1. Calculating the values for $a^{(i)}_n$ and finally for $h_\Theta(x)$ is called forward propagation.

The cost function for a neural network is similar to logistic regression, with the exception of a new summation term to cover the output layer with multiple nodes.

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{K=1}^{K} \left( y^{(i)}_K \log\left(h_\Theta(x^{(i)})_K\right) + (1 - y^{(i)}_K) \log\left(1 - (h_\Theta(x^{(i)})_K\right) \right) \tag{2.19}$$

To minimize this cost function, it is necessary to provide all the current values for $\Theta$ that were used to calculate $J(\Theta)$. These values, in addition to the cost function, are used in optimized minimization functions provided by MATLAB or by other relevant programming languages.

## Backpropagation algorithm

To calculate the derived values for $\Theta$, all current values for $a^{(2)}_n, \dots, a^{(N+1)}_n$ are calculated in addition to the output $h_\Theta(x)$ for every training example. Next, the error between all the current values for $a^{(L)}$ and $y^{(i)}$ are calculated.

$$\delta^{(L)} = a^{(L)} - y^{(i)} = h_\Theta(x)^{(i)} - y^{(i)} \tag{2.20}$$

These error values are then backwards propagated through the network. If the values for $\Theta$ and $a$ are represented by matrices, equation 2.21 applies. The operator $.*$ stands for the element-wise multiplication between matrices.

$$\delta^{(l)} = ((\Theta^{(l)})^T \times \delta^{(l+1)}). * (a^{(l)}. * (1 - a^{(l)})) \tag{2.21}$$

It is unnecessary to calculate the error value for $a_0^{(l)}$, as the partial derivative of a constant value is zero. The error value for the inputs is not calculated either because these values are fixed. It can then be proven that the partial derivative is equal to equation 2.22.

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}} = \frac{1}{m} \times a_j^{(l)} \times \delta_{(i)}^{(l+1)} \tag{2.22}$$

### Linear regression

It is possible for a neural network to output continuous values. This is possible by simply removing the activation function from the output layer. This does, however, require the construction of a new cost function and backpropagation algorithm.

## 2.2.4 Support Vector Machines (SVM)

Support Vector Machines (SVMs) are based on the idea of logistic regression. They do, however, perform better computationally and implement new classification options which may be useful in some classification problems.

### SVM and Logistic Regression

The cost function for SVMs is given by equation 2.23.

$$J(\theta) = C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) + \sum_{j=1}^{n} \theta_j^2 \tag{2.23}$$

As opposed to logarithmic terms, $cost_1$ and $cost_0$ are defined by equation 2.24. The values for the slope and intersection can be chosen freely, as long as the curve approaches the original logarithmic function (see figure 2.7). This approximation results in faster calculations due to the need for calculating a linear term instead of a logarithmic term. The parameters $C$ and $\sum_{j=1}^{n} \theta_j^2$ will be explained in section 2.4

$$\begin{aligned} cost_1 &= \begin{cases} -\frac{4}{11}x + \frac{4}{11}, & \text{if } \theta^T X < 1 \\ 0, & \text{otherwise.} \end{cases} \\ cost_0 &= \begin{cases} \frac{4}{11}x + \frac{4}{11}, & \text{if } \theta^T X > -1 \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{2.24}$$

Given a proportional distribution ($k = 0.5$), equation 2.7 from logistic regression shows that the output class of a training example is defined by $h_\theta(x) = g(\theta^T X)$. Given the behaviour of the sigmoid function, this equation can be transformed into equation 2.25.

$$h_\theta(x) = \begin{cases} 1, & \text{if } \theta^T X \geq 0 \\ 0, & \text{if } \theta^T X < 0 \end{cases} \tag{2.25}$$

Support Vector Machines implement a margin between the decision boundary and the training examples. This generally allows for better classification. The output of a training example is now given by equation 2.26.

$$h_\theta(x) = \begin{cases} 1, & \text{if } \theta^T X \geq 1 \\ 0, & \text{if } \theta^T X < -1 \end{cases} \tag{2.26}$$

### Kernels

SVM implement a different type of feature vector for training the algorithm. Whereas logistic regression uses a linear combination between the input features and their respective weights, SVM replaces the input features with specific functions (equation 2.27). One possible kernel function is the Radial Basis Function (RBF) kernel [7].

$$y = 1 \text{ if } \theta_0 + \theta_1 f_1 + \cdots + \theta_n f_n \geq 1 \tag{2.27}$$

The RBF kernel defines $f$ as the similarity between an input feature $x^{(i)}$ and a certain landmark $l^{(i)}$. This is represented by equation 2.28. The value of $\gamma$ has to be tuned according to the dataset. The landmarks are simply defined as the location of the actual input features.

$$f = \exp(-\gamma \| x - l^{(i)} \|) \tag{2.28}$$

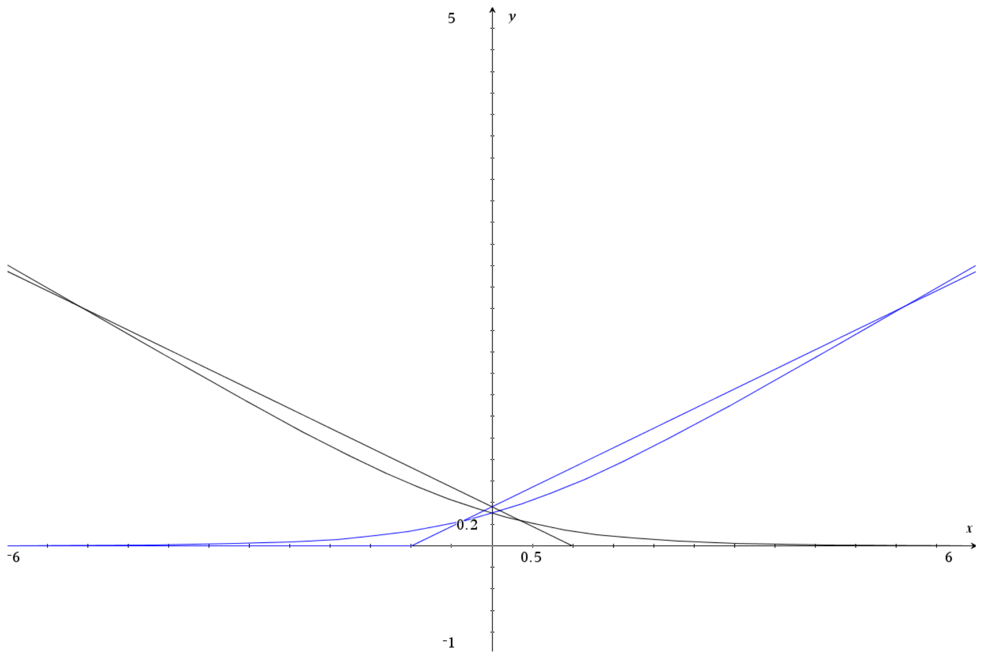

**Figure 2.7:** Logistic regression activated cost function function vs SVM activated cost function for $y = 1$ (black) and $y = 0$ (blue)

## 2.3 Performance metrics

Different types of algorithms can be evaluated on their performance by checking a combination of different parameters. These performance metrics are based on the type of algorithm. An algorithm should always be checked based on how well it reflects its data, and how well it predicts new values. A low value for either of these criteria will result in bad predictions, which will be addressed in section 2.4.

### 2.3.1 Regression based algorithms

For regression based algorithms, this thesis uses the coefficient of determination and mean squared error for measuring its accuracy.

### Coefficient of determination

The coefficient of determination is a value which represents the ability of a model to predict a new example accurately [8]. The value for $R^2$ is given by equation 2.29 [9].

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum\limits_m y_{true} - y_{pred}}{\sum\limits_m y_{true} - \bar{y}} \tag{2.29}$$

This formula normally outputs a value between 0.0 and 1.0. It is possible, however, that exceptionally bad models output a negative value [10].

### Mean Squared Error

The Mean Squared Error (MSE) is given by equation 2.30. It is a value which represents the deviation of the predicted samples with respect to their real values. A higher deviation results in a higher value for the MSE due to the squared term. The Mean Absolute Error (MAE) is also included in the performance metrics for the used algorithms. It is, however, not used to calculate its performance. The MAE is given by equation 2.31.

$$MSE = \frac{1}{m} \sum_m (y_{true} - y_{pred})^2 \tag{2.30}$$

$$MAE = \frac{1}{m} \sum_m \mid y_{true} - y_{pred} \mid \tag{2.31}$$

## 2.3.2 Classification based algorithms

Classification based algorithms can be evaluated based on how many examples are predicted correctly. This thesis proposes the use of the F1-score and the sum of correctly classified examples to evaluate a classification model.

### F1-score

Given a multiclass classification problem with three classes, table 2.4 represents the confusion matrix. The confusion matrix describes the performance of an algorithm by measuring how many examples are (in)correctly classified. For multiclass classification problems, samples can either be a True Positive (TP), False Positive (FP), False Negative (FN) or True Negative (TN). A wrongly identified example can either be a FP, FN depending on which class is used as reference.

**Table 2.4:** Confusion matrix

| | | Predicted Class | | |
|---|---|---|---|---|
| | | 0 | 1 | 2 |
| Real Class | 0 | TP | FP/FN | FP/FN |
| | 1 | FP/FN | TP | FP/FN |
| | 2 | FP/FN | FP/FN | TP |

The F1 score is calculated by implementing equation 2.32. Precision is a metric which explains the ratio between true positive examples with respect to the total positive predicted examples. Recall explains the ratio between true positive examples with respect to the total number of examples in the actual class. The F1-score is a combination of both precision and recall.

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.32}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{2.33}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{2.34}$$

For a certain class $x$, the amount of true positives is the value of the diagonal position $(x, x)$. The amount of false positives and false negatives can be calculated by taking the sum of the column and row of $x$ respectively, with the exception of the amount of true positives.

#### Ratio of Correctly Classified Examples (RCCE)

The algorithm's accuracy can be measured by simply implementing equation 2.35. This equation outputs the ratio of correctly classified training examples.

$$\text{RCCE} = \frac{1}{m} \sum_{m} y_{true} == y_{pred} \tag{2.35}$$

## 2.4 The problem of under- and overfitting

The performance metrics discussed in section 2.3 give a good view of how well an algorithm performs on its training data. However, it does not necessarily evaluate how well the algorithm would perform on new samples. For example, a dataset which is solely used for training might accurately predict similar training examples, but might provide a random result for new samples. To account for this issue, these performance metrics are evaluated on a different data set.

### 2.4.1 Dividing the data set

Every data set is subdivided into three smaller data sets (percentages depend on the data set):

- Training set (60%);
- Testing set (20%);
- Cross-validation set (20%).

The training set is solely used for training the algorithm and is not used for calculating the performance metrics from section 2.3. During training, the most optimal model is found by calculating the performance metrics using the cross-validation set. This will be explained more in detail in section 2.5. When an optimal model is found, the final performance result will be calculated using the testing set.

### 2.4.2 Bias-variance trade-off

It is possible that after training, the model seems to perform poorly on the testing set, but very well on the training set. In this case the model is most likely overfitting or suffering from high variance. There are three main ways of solving high variance:

- Increase the amount of training examples;
- Decrease the amount of features;
- Increase the regularization parameter.

Regularization can be added to a cost function to decrease the importance of the features with respect to the cost. For linear and logistic regression, the parameter $\lambda \sum_{j=1}^{n} \theta_j^2$ can be added to the cost function. An increasing value of $\lambda$ will lower the importance of the selected features, and thus reduces the chance of overfitting the model. For Support Vector Machines, the parameter $\sum_{j=1}^{n} \theta_j^2$ is added to the cost function as well as a parameter $C$ as multiplication term for the first term. The parameter $C$ is effectively the inverse of parameter $\lambda$, as increasing its value will increase the importance of the selected features.

For neural networks, a similar implementation can be used for preventing high variance. When testing different models of the neural network during this thesis, however, a different implementation of regularization is used. A technique called dropout removes some nodes and connections from a deep neural network at random during training, stopping nodes from co-adapting too much. This technique seems to perform better than other, more standard regularization techniques [11].

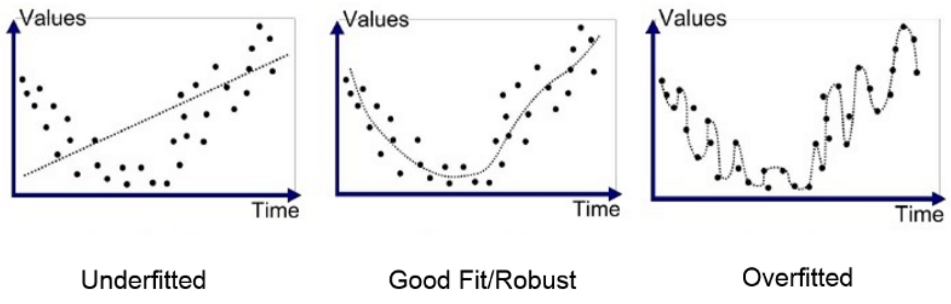

**Figure 2.8:** High bias (underfitting), good fit and high variance (overfitting) [3]

When an algorithm performs poorly both on the training and test set, the model may be suffering from high bias. This is solved by inverting the solutions provided above. It is important to find a balance between high bias and high variance.

## 2.5 Finding the optimal model

The choice of regularization parameter $\lambda$ depends on the data set. Most often though, more parameters have to be chosen to optimize the model. Recall equation 2.28, which represents the RBF kernel. This kernel implements a parameter $\gamma$ which directly influences the cost function. A good balance between $\lambda$ and $\gamma$ has to be chosen to make sure that the model does not under- or overfit. The values for these so-called hyperparameters can be found by performing a grid search.

### Grid search

When performing a grid search, an interval of values for the hyperparameters is chosen. For example, the grid search performed in figure 2.9 uses an interval of $[0, 1000]$ for $C$ and $[0, 1]$ for $\gamma$. The chosen step width for each interval is $0.025\times$maxvalue for a total of 1600 values. For every possible combination of these values, an algorithm is trained for a set number of iterations and is evaluated based on its performance. In figure 2.9, the F1-score is used as evaluation metric on the z-axis. This performance evaluation is based on the cross-validation set instead of the testing set. It is important to use a different data set, as using the testing set would introduce a certain unwanted dependency between the testing set and the model.

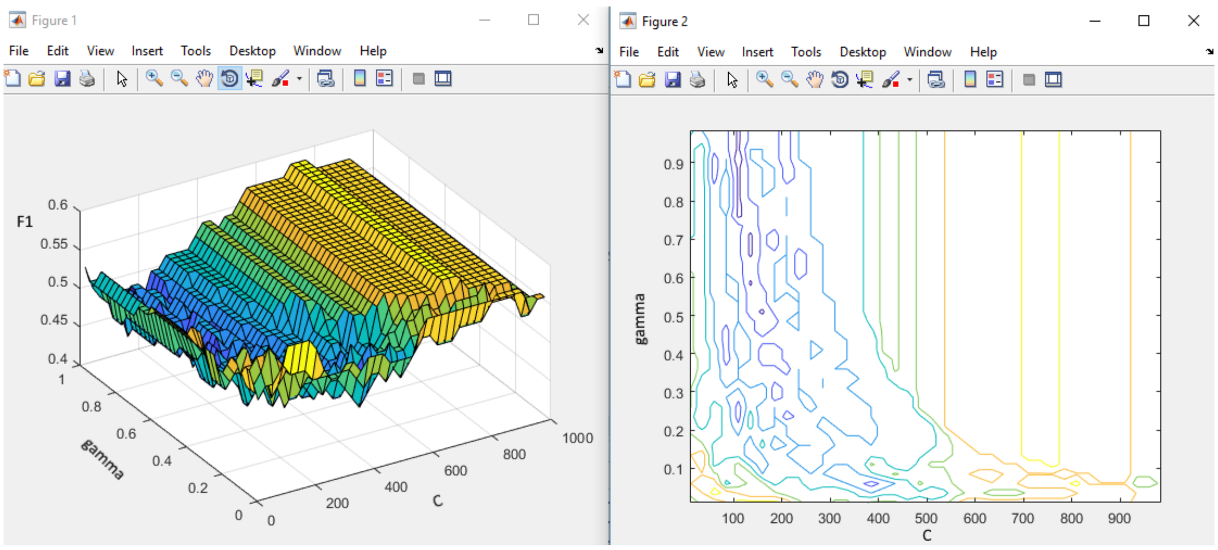

**Figure 2.9:** Grid search example for SVM with RBF kernel

## 2.6 Project background

The final goal of this thesis is to produce and implement one or more machine learning algorithms to predict certain surgical decisions. To achieve this goal, data from 344 patients (of which 320 have been selected as valid) has been received in two separate Excel files. One file contains clinical measurements with general information about the patient and consists of 40 features per patient. The other file contains cephalometrical data based on a 3D image from the patient using a Cone-Beam CT scan and consists of 80 features. A patient's data is declared invalid when either the cephalometrical data is not provided, or one of the two files misses crucial information.

The clinical measurements include information that has been measured by the surgeons. Apart from general information (eg. gender, age) and the measurements, the surgeon's decision is included per patient. Table 2.5 shows the different output types which were included in the file. Outputs followed by $(C)$ are discrete values which require a classification based algorithm. Likewise, outputs followed by $(R)$ are analogue values which require a regression based algorithm. For each of these values it is possible that the patient does not need to undergo that kind of operation. In that case, the output value is either zero or 'no'.

Each surgery type operates a certain body part. Le Fort I operates only on the maxilla. BSSO operates only on the mandible. Bimax is a combination of an operation on the maxilla and mandible. After this surgery is performed, it is up to the surgeons and the patient to decide whether they would like to operate on the chin. The data contains a relatively low amount of chin operations, as this operation has a long recovery time for the patient.

**Table 2.5:** Surgical decision output per type. (C = classification, R = regression)

| **Surgery type** | **Maxilla** | **Mandible** | **Chin** |
|---|---|---|---|
| Le Fort I (C) | Advancement (R) | Advancement / Setback (C) | Advancement (R) |
| Le Fort I & Chin (C) | 1 / 3 pieces (C) | | Intrusion / Extrusion (R) |
| BSSO (C) | Anterior (R) | | |
| BSSO & Chin (C) | Posterior (R) | | |
| Bimax (C) | Midline Rotation (R) | | |
| Bimax & Chin (C) | | | |

Chapter 3

# Method

This chapter will discuss all necessary steps which have been performed in order to construct and implement the chosen algorithms. It will first discuss how data is obtained and transformed into usable, numerical data. It will then discuss the different types of algorithms used during the research including their architectures. Lastly, it will discuss the project structure of the final Python script.

## 3.1 Importing data

The first goal of this thesis has been to import the data from the patient into arrays readable by MATLAB and Python. The directory structure is shown in figure 3.1. General information of all patients is contained within `clinical_measurements.xlsx`. The cephalometrical data is given per patient in the `ceph_tracing` directory and is sorted by name (eg. `Peeters Danilo.xlsx`). It is necessary for the script to construct a matrix containing all patient features per patient, as well as a matrix containing all output decisions per patient.

```
Script
├─ data
│  ├─ ceph_tracing
│  │  ├─ patient_one.xlsx
│  │  ├─ patient_two.xlsx
│  │  └─ ...
│  └─ clinical_measurements.xlsx
└─ bin
   ├─ script_data
   │  └─ algorithm models & cached data
   ├─ orthoplanner.py
   ├─ dataparser.py
   ├─ LearningModule.py
   └─ databasehandler.py
```

**Figure 3.1:** Directory tree of the Python project

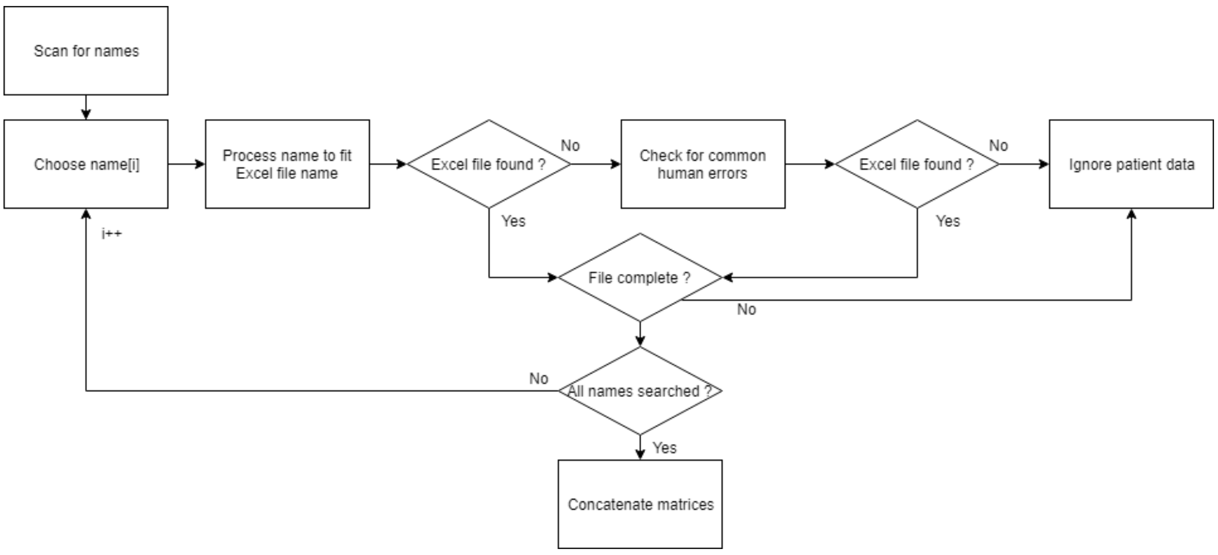

**Figure 3.2:** Flowchart for importing patient data from Excel files

Figure 3.2 shows a flowchart representing the function responsible for importing data. First, the data from `clinical_measurements.xlsx` is stored into a matrix and is scanned for patient names. The patient names are stored in a vector through which an iteration process starts. Secondly, every name in the vector is transformed to fit the relative directory of the Excel file containing cephalometrical data. If the Excel file is not found, the same name is checked for common human entry errors such as leading or trailing spaces, swapping first and last name etc. If the Excel file is found, its contents are scanned to make sure every feature is present. If the Excel file is not present or lacks important information, the patient's data is discarded. Finally, the data from `clinical_measurements.xlsx` and the data from the cephalometrical tracing files are concatenated to construct one input feature matrix. The output decisions are stored in a separate matrix.

The feature matrix is constructed by transforming the Excel file data into usable, numerical data. This is done by checking certain keywords in each cell and by transforming the data accordingly. For example, if one of the Excel file cells contains the value `"intrusion 2mm"`, it is transformed to the numerical value `-2`. Discrete values were given a value of either `0` or `1` for binary values, `'-1, 0, 1'` for `'setback, nothing, advancement'`, `'0, 1, 2'` for `'no, one piece, multiple pieces'` and custom chosen classes for surgery decisions.

## 3.2 Algorithm Types and Experimentation

### 3.2.1 Experimentation in MATLAB

In the second stage of the research, several algorithms have been experimented with in MATLAB. MATLAB is chosen because it is also the language used in the machine learning course and due to its fast prototyping abilities [2]. Different approaches have been taken to find the most optimal algorithm and learning structure. For the first three algorithms, the best model is chosen based primarily on F1-score, and secondarily based on RCCE.

### 3.2.2 Types of tested algorithms

#### Classification (surgery type)

The first algorithm prototype classifies patients per surgery type. An SVM with both linear and RBF kernel are used and optimized using grid search.

#### Classification (surgery type) based on gender

The second algorithm tries to find a relation by dividing the original dataset based on gender. Again, an SVM with both linear and RBF kernel are used.

#### Classification (divided surgery types)

The third algorithm groups and divides the output data. Operations including and excluding the chin are grouped, resulting in only three output classes instead of six. For each of these classes, an SVM with RBF kernel is trained and optimized.

The following structures have been experimented with when it was clear that the previous algorithms did not accurately reflect the surgeon's decision-making pattern. The surgery decision is now based on the algorithm's output for maxilla, mandible and chin adjustments.

#### Regression (Maxilla and Chin combined)

The first regression algorithm is based on Support Vector Regression (SVR) with RBF kernel. The chosen features were determined in consultation with the external supervisor. The output layer consists of six nodes, one for every necessary output value.

#### Regression (Maxilla and Chin combined) based on malocclusion category

The second regression algorithm first divides the data set into class 2 or 3 malocclusion patients. Class 2 patients are defined as having a positive overjet and overbite, whereas class 3 patients have a negative overjet and overbite. The regression is performed using a neural network with four hidden layers, including 120 nodes per layer. The output layer consists of six nodes, one for every necessary output value.

#### Regression (Maxilla and Chin combined) based on 20 best features

The third regression algorithm uses all the patients and choses the 20 best features based on the correlation between input and output values [12]. The used algorithm is also a neural network with four hidden layers, including 20 nodes per layer. The output layer consists of six nodes, one for every necessary output value.

#### Regression (One algorithm per output feature) based on 20 best features

The final regression algorithms uses all the patients and choses the 20 best features per output type based on the correlation between input and output values [12]. The regressors for chin predictions also take the predicted values of the maxilla into account. This results in six neural networks with each four hidden layers, including 20 nodes per layer. The output layer consists of one node.

#### Regression (Maxilla Advancement) based on 20 best features and allowed error margin

Due to the importance of the maxilla advancement, a different approach has been taken to improve the performance. Instead of using the MSE as cost function, a self-implemented cost function has been developed to allow for some deviation. The pseudo-code for this cost function is given in listing 3.1.

**Listing 3.1:** Pseudo code for MSE with error margin

```
if y_pred >= y_true - 1 and y_pred <= y_true + 1:
  return 0
else:
  return MSE(y_true, y_pred)
```

#### Classification (discrete maxilla & mandible values) based on 20 best features

The final classification algorithms for the two discrete output values use an SVM with linear kernel. The input features consist of the 20 best features per output based on the correlation between input and output values [12]. In addition to all patient features, the mandible classification algorithm also adds the output values from the maxilla classifiers and regressors to its input. Each discrete output value has one responsible SVM.

### 3.2.3 Optimal model selection

For the final algorithms, the most optimal kernel and hyperparameters are chosen automatically by performing a grid search based on the relevant performance metrics for that type of algorithm. For classification based algorithms, grid search finds the best combination of kernel and hyperparameters primarily based on the F1-score and secondarily on the highest RCCE. For regression based algorithms, the hyperparameters are primarily based on the coefficient of determination and secondarily on the lowest MSE.

## 3.3 Python Implementation

The final stage consists of implementing the algorithms discussed above into a user-friendly script. Python is chosen as programming language due to its versatility and easy implementation server-side. Due to a Bachelor's Thesis developing a front-end interface, an implementation without a Graphical User Interface (GUI) is chosen. The command-line script is called by the web interface and the output is stored in a database (see section 3.3.2).

### 3.3.1 Usage

Listing 3.2 shows the script's output when supplying `-h` or `--help` as arguments. The default and required argument for the script is `-e EAD` which evaluates a patient using the already trained algorithms. The parameter `EAD` is a patient-specific identification string. To prevent compatibility or library issues, the script should be run using the provided virtual environment. This is easily done by calling `orthoplanner.py` using the `python` script within `/venv/Scripts/`.

The algorithm can be retrained by calling the script with argument `-r`. This will gather data from the database and import them into usable arrays. It is possible to also use the local data in the `/data` directory by supplying the extra argument `--USE_LOCAL_DATA`. Whenever any of these functions are called, the script automatically saves these arrays into a file with an easily readable data format (`.npy`). When rerun using extra argument `--USE_CACHED_DATA`, the script will automatically load these saved files instead. This allows the user to bypass the Excel file or database importing stage and save time when adjusting the algorithm's parameters.

### 3.3.2 Project structure

The project's directory structure is shown in figure 3.1. This section will briefly explain the task of every script file.

#### orthoplanner.py

This is the file that should be called from the virtual environment. This script parses the provided arguments and acts accordingly.

#### dataparser.py

Dataparser either searches for Excel files, or queries the database to import the data into usable arrays. When reading a string value, the data is transformed into numerical data (eg. `"intrusion 2mm" -> -2`). Data is sanitized by removing invalid patient examples and is ultimately divided into a feature matrix and decision matrix.

#### LearningModule.py

LearningModule implements all the final versions of the algorithms. It makes use of the 'sklearn' and 'keras' libraries to implement the SVM and deep neural networks respectively. When retrained, all algorithm models are saved in `/bin/script_data` using `.h5` for sklearn models or `.pkl` for keras models as data format. This allows the user to evaluate the patient relatively quickly.

#### databasehandler.py

Databasehandler secures a connection between the application and a MySQL database containing patient data and algorithm predictions. For evaluation, several `SELECT` statements are issued to retrieve both clinical measurements and cephalometrical data. When inserting predictions into the database, a check is first performed to see if the patient already has a prediction. In the current implementation, any previous predictions are overwritten. It is also possible to gather all patient information for retraining the algorithms.

**Listing 3.2:** Script output when calling with arguments -h or –help

```
C:\Users\Danilo\Desktop\MASTERPROEF\CONF\venv64\Scripts\python.exe
C:/Users/Danilo/Desktop/MASTERPROEF/CONF/Script-Python/bin/orthoplanner.py
--help

Using TensorFlow backend.
usage: orthoplanner.py -e EAD [-v]

Using SVM and Deep Learning to facilitate orthognathic surgery planning - A
Master's Thesis project by Danilo Peeters. In cooperation with CoSenS and
University Hospitals Leuven.

optional arguments:
  -h, --help          show this help message and exit
  -e PATIENT_EAD      evaluate using the current algorithm's parameters.
                      Required argument: Patient EAD string (eg. 123456V789)
  -r                  retrain the algorithm using data from the database
  -v, --verbose       increase output verbosity
  --USE_CACHED_DATA   reuse cached data when retraining the algorithm
  --USE_LOCAL_DATA    use the data found in ./data to retrain the algorithms
                      (NOT RECOMMENDED)
```

Chapter 4

# Results

This chapter will discuss the performance of the previously discussed algorithms. Due to the wide range of the algorithms used, only results from the latest iterations are given and discussed. These algorithms proved to be an improvement on their previous iterations which was either due to better performance measures or due to changing requirements.

## 4.1 Regression based algorithms

For every analogue output value a neural network has been implemented using the 20 best input features. For each network, the amount of layers and neurons are computationally defined by training and evaluating each network based on the coefficient of determination. Table 4.1 shows the performance metrics for each neural network by varying the amount of neurons between 1 and 25. The output for one such test is shown in listing 4.1. Each neural network has a total of 6 layers (N=4) and uses a linear rectifier (Rectified Linear Unit (ReLU)) as activation function for the hidden nodes. Results may vary when redoing these tests due to the randomization of the initial value of $\Theta$ for neural networks.

**Table 4.1:** Performance metrics for constant amount of layers (N=4)

| | **Maxilla Advancement** | **Maxilla Advancement (margin)** | **Maxilla Anterior** | **Maxilla Posterior** | **Maxilla Midline Rotation** | **Chin Advancement** | **Chin Intrusion/Extrusion** |
|---|---|---|---|---|---|---|---|
| # neurons | 6 | 6 | 14 | 4 | 1 | 9 | 1 |
| $R^2_{train}$ | 0.606 | 0.580 | 0.745 | 0.168 | -0.006 | 0.524 | -0.050 |
| $R^2_{test}$ | 0.425 | 0.438 | 0.499 | 0.100 | -0.066 | 0.045 | -0.050 |
| MSE | 5.055 | 4.009 | 4.381 | 2.897 | 0.658 | 4.987 | 0.065 |
| MAE | 1.635 | 1.615 | 1.501 | 1.197 | 0.241 | 1.408 | 0.068 |

For maxilla advancement, figure 4.1 shows the amount of testing examples within a certain range of deviation. Table 4.2 shows these amounts for deviation ranges within 1 and 2 millimeters. These values are averaged over five trained models and have been gathered from a testing set consisting of 61 patients.

**Table 4.2:** Mean values for percentage of testing examples with deviation ranges

| **Deviation** | **Percentage of testing examples** |
|---|---|
| ±1 mm | 37.37% |
| ±2 mm | 65.57% |

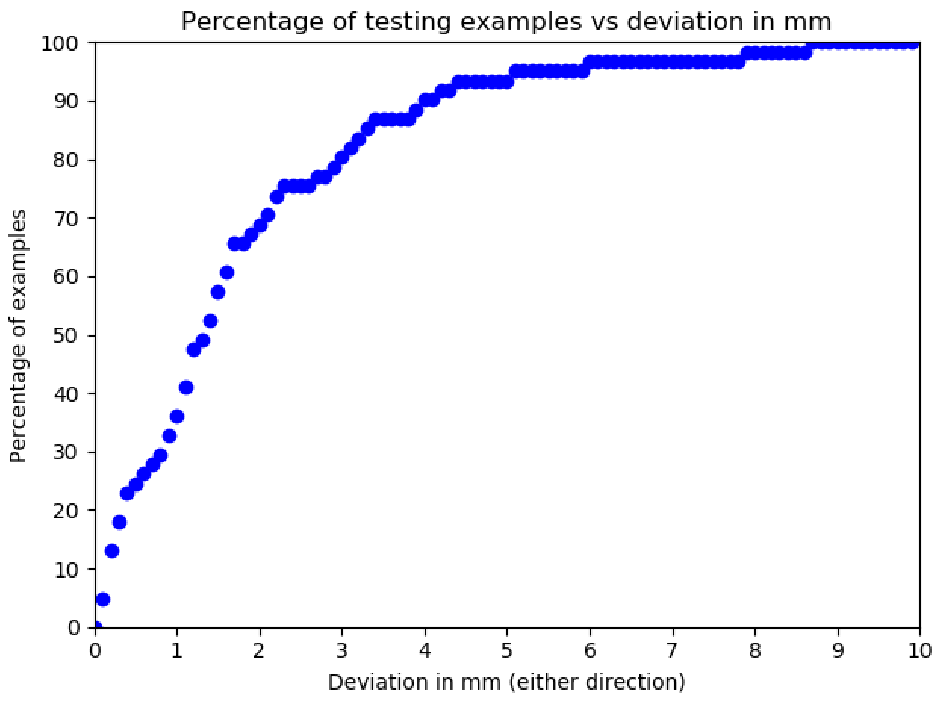

**Figure 4.1:** Percentage of testing examples within a certain range of deviation

**Listing 4.1:** Test output for determining amount of nodes for maxilla advancement regression

```
Best: 0.425258 using {'neurons': 6}
Train: −0.435154 (0.033008) // Test : −0.451080 (0.141512) with: {'neurons': 1}
Train: −0.236605 (0.377447) // Test : −0.307745 (0.382349) with: {'neurons': 2}
Train: 0.481554 (0.090270) // Test : 0.344272 (0.135833) with: {'neurons': 3}
Train: 0.147526 (0.474924) // Test : −0.024030 (0.407038) with: {'neurons': 4}
Train: 0.521925 (0.082398) // Test : 0.346980 (0.137241) with: {'neurons': 5}
Train: 0.606069 (0.043646) // Test : 0.425258 (0.107502) with: {'neurons': 6}
...
Train: 0.875921 (0.014413) // Test : 0.289328 (0.141174) with: {'neurons': 19}
Train: 0.852235 (0.055685) // Test : 0.251638 (0.246964) with: {'neurons': 20}
Train: 0.876141 (0.028955) // Test : 0.331270 (0.167496) with: {'neurons': 21}
Train: 0.857931 (0.052627) // Test : 0.338754 (0.063567) with: {'neurons': 22}
Train: 0.874847 (0.023300) // Test : 0.376806 (0.141168) with: {'neurons': 23}
Train: 0.880406 (0.031549) // Test : 0.338291 (0.085950) with: {'neurons': 24}
Train: 0.901908 (0.029407) // Test : 0.322130 (0.266266) with: {'neurons': 25}
```

## 4.2 Classification based algorithms

The classification based algorithms are constructed for parameters `Maxilla one/more pieces` and `Mandible Advancement/Setback`. Table 4.3 shows the kernel and hyperparameters which have been found by grid search (example output, see listing 4.2). These results are based on a cross-validation set with an amount of 20% of the training set examples. Given that the training contains 80% of all data samples, the cross-validation set globally consists of 16% of all samples. Table 4.4 and table 4.5 show the performance measures for the `Maxilla one/more pieces` and `Mandible advancement/setback` SVM respectively.

**Table 4.3:** Hyperparameters for both SVMs

| | **Maxilla one/more pieces** | **Mandible advancement/setback** |
|---|---|---|
| Kernel | Linear | RBF |
| C | 1.000 | 2.184 |
| $\gamma$ | - | $1.207\times10^{-3}$ |

**Table 4.4:** Performance measures for one/more maxilla pieces SVM

| **Class** | **Precision** | **Recall** | **F1-score** | **RCCE** | **# Testing examples** |
|---|---|---|---|---|---|
| Nothing | 0.78 | 1.00 | 0.88 | | 28 |
| One piece | 0.96 | 0.80 | 0.87 | | 30 |
| More pieces | 1.00 | 0.50 | 0.67 | | 6 |
| average / total | 0.88 | 0.86 | 0.85 | 0.86 | 64 |

**Table 4.5:** Performance measures for mandible advancement / setback SVM

| **Class** | **Precision** | **Recall** | **F1-score** | **RCCE** | **# Testing examples** |
|---|---|---|---|---|---|
| Nothing | - | - | - | | 6 |
| Setback | 0.68 | 0.93 | 0.79 | | 14 |
| Advancement | 0.89 | 0.91 | 0.90 | | 44 |
| average / total | 0.76 | 0.83 | 0.79 | 0.83 | 64 |

**Listing 4.2:** Grid search for determining best hyperparameters and kernel based on F1-score

```
0.774 (+/-0.056) for {'kernel': 'linear', 'C': 1.0}
0.778 (+/-0.055) for {'kernel': 'linear', 'C': 1.5918367346938775}
0.778 (+/-0.055) for {'kernel': 'linear', 'C': 2.183673469387755}
0.778 (+/-0.055) for {'kernel': 'linear', 'C': 2.7755102040816326}
0.774 (+/-0.066) for {'kernel': 'linear', 'C': 3.36734693877551}
...
0.762 (+/-0.076) for {'kernel': 'linear', 'C': 28.816326530612244}
0.762 (+/-0.076) for {'kernel': 'linear', 'C': 29.408163265306122}
0.762 (+/-0.076) for {'kernel': 'linear', 'C': 30.0}
0.786 (+/-0.070) for {'kernel': 'rbf', 'gamma': 0.001, 'C': 1.0}
0.786 (+/-0.070) for {'kernel': 'rbf', 'gamma': 0.0010689655172413793, 'C': 1.0}
0.790 (+/-0.061) for {'kernel': 'rbf', 'gamma': 0.0011379310344827587, 'C': 1.0}
0.795 (+/-0.061) for {'kernel': 'rbf', 'gamma': 0.001206896551724138, 'C': 1.0}
...
0.778 (+/-0.069) for {'kernel': 'rbf', 'gamma': 0.002862068965517241, 'C': 1.0}
0.778 (+/-0.069) for {'kernel': 'rbf', 'gamma': 0.0029310344827586207, 'C': 1.0}
0.778 (+/-0.069) for {'kernel': 'rbf', 'gamma': 0.003, 'C': 1.0}
0.794 (+/-0.057) for {'kernel': 'rbf', 'gamma': 0.001, 'C': 1.5918367346938775}
0.794 (+/-0.057) for {'kernel': 'rbf', 'gamma': 0.0010689655172413793,
'C': 1.5918367346938775}
...
```

## 4.3 Script output

### 4.3.1 Evaluating a patient

Patient evaluation is performed by calling the script with argument `-e EAD`. Listing 4.3 shows the script's output when evaluating a patient. The data is exported to the database in numeric values.

**Listing 4.3:** Evaluation output for a patient

```
C:\Users\Danilo\Desktop\MASTERPROEF\CONF\venv64\Scripts\python.exe
C:/Users/Danilo/Desktop/MASTERPROEF/CONF/Script-Python/bin/orthoplanner.py -e 60360414
Using TensorFlow backend.
========================STEP 1: IMPORTING DATA========================
Querying database...
...done !
======================STEP 2: EVALUATING PATIENT======================
Evaluating patient...
...done !
========================STEP 3: EXPORTING DATA========================
Maxilla Advancement: 1.8724944591522217mm
Maxilla Pieces: More pieces
Maxilla Anterior: Extrusion 2.328683376312256mm
Maxilla Posterior: Extrusion 0.9665279984474182mm
Maxilla Midline Rotation: Rotate left 0.04467182233929634mm
Mandible Advancement/Setback: Advancement
Chin Advancement: Advancement 0.4156898558139801mm
Chin Intrusion/Extrusion: Intrusion 0.03682597726583481mm

Exporting data to database...
We already predicted some data for patient 60360414. Overwriting previous result...
Data exported successfully
Goodbye
```

### 4.3.2 Retraining the algorithms

Due to the length of the script's output for retraining, this section has been moved to appendix A. This non-verbose output shows a summary of the results for each algorithm including the performed gridsearches.

Chapter 5

# Discussion

This chapter will discuss the results from chapter 4 for both types of algorithms. It will primarily focus on the regression based algorithms as they are the most difficult to predict. Secondly, it will briefly discuss the results of both SVM implementations for the discrete value predictions.

## 5.1 Regression based algorithms

### 5.1.1 Poorly performing algorithms

Table 4.1 generally shows poor results for every regression algorithm in terms of $R^2$-values. More specifically, `Maxilla Posterior`, `Maxilla Midline Rotation`, `Chin Advancement` and `Chin Intrusion/Extrusion` seem to be predicting random values. After multiple tests this is believed to be caused by the reasons below.

#### Relative low amount of data

A total of 320 valid samples are used as dataset, of which 80% are used to train both classification and regression algorithms (including cross-validation set). From these 320 patients, only a fraction undergo certain surgeries and provide relevant data. For example, only 51 patients (of all patients, including invalid examples) underwent chin surgery where their chin received an advancement. The algorithm for predicting the value for chin intrusion / extrusion only consists of six patients who underwent this type of surgery.

#### Statistical inconsistencies and non-exact data

Due to the nature of surgery, one patient might receive a different surgery based on several factors. These factors include surgeon experience, patient wishes and geographical location. This results in statistical inconsistencies which has diminishing results on the algorithm's performance.

#### Human error

The clinical measurements are based on surgeon's hand-made measurements and were afterwards recollected into one Excel file. This process may produce typographical errors.

### 5.1.2 Moderately performing algorithms

The results for `Maxilla Advancement with(out) margin` and `Maxilla Anterior` seem to have a higher correlation with the training and test data than the examples discussed above. These values, however, are still relatively low. This is most likely also caused by the issues discussed above. As said before, the model's correlation factor is based on how well the model represents the training & test sets. Given that multiple surgeries are possible per patient, it might have been more interesting to use a different performance metric as the coefficient of determination does not provide enough information.

The model has acceptable scores concerning the deviation in millimeters. Due to surgical limitations it has been decided that a deviation of 1mm in either direction was allowed. Currently, table 4.2 suggests that 37% of all patients fall within this category. Figure 4.1 also shows a rapidly increasing ratio of training examples with an increasing deviation.

## 5.2 Classification based algorithms

As shown in tables 4.4 and 4.5, both SVM implementations show promising results regarding RCCE and F1-score. It is to be expected that these models perform better than the regression models as classifying examples allows greater error ranges (eg. both predicted values 0.6 and 0.8 are classified as the same class for $k = 0.5$). The one/more pieces SVM reports a RCCE value of 0.86 with average F1-score of 0.85. These values are acceptable, but could be improved upon. Most patients only need their maxilla to be cut into one piece instead of more pieces.

The mandible advancement classifier reports no scores for `nothing` as there were too few training examples to produce a meaningful model. It generally provides a good prediction for the advancement or setback of the mandible. This is most likely due to its location-based dependence on the maxilla.

Chapter 6

# Conclusion & Future research

## 6.1 Conclusion

This thesis proposes two promising types of machine learning algorithms to facilitate orthognathic surgery planning. Using clinical measurements and cephalometrical data of patients, multiple supervised learning algorithms have been constructed to predict surgery decisions.

For a total of seven analog output parameters, several neural networks with 4 hidden layers have been constructed for each of the output parameter separately. The 20 inputs of these neural networks consist of the best correlated input features including previously predicted values. Although some models fail to find a good correlation, others show promising results with regard to the allowed deviation. It would be beneficial for these models to gather additional data to accurately reflect the complex decision-making.

Two discrete output parameters have been predicted using two separate SVM models. The inputs for these models also consist of the 20 best correlated input features, whereas the kernel type and hyperparameters are determined computationally by performing grid search. Both models have a correct classification rate of over 83%.

## 6.2 Future research

Even though the classification based algorithms shows promising results, it should be noted that the regression based models still require work in order to more accurately reflect the surgeon's decision. On top of solving the issues mentioned in 5.1.1, a different method is proposed that may improve the accuracy of these models.

#### Classifying regression parameters

Due to the relative high amount of allowed deviation and low value for the regression parameters, it could be beneficial to create custom classes. The model instead predicts which class the patient belongs to instead of calculating an analogue value. Table 6.1 shows an example which could be used to classify maxilla advancement parameters.

**Table 6.1:** Classifying values for maxilla advancement

| **Class range** | **Class number** |
|---|---|
| no advancement | 0 |
| $[0, 2[$ | 1 |
| $[2, 4[$ | 2 |
| $[4, 6[$ | 3 |
| $[6, +\infty[$ | 4 |

# Appendix A

# Script output for retraining

**Listing A.1:** Retraining output

```
C:\Users\Danilo\Desktop\MASTERPROEF\CONF\venv64\Scripts\python.exe
C:/Users/Danilo/Desktop/MASTERPROEF/CONF/Script-Python/bin/orthoplanner.py -r
Using TensorFlow backend.
==================STEP 1: IMPORTING DATA==================
Okay! We are reimporting all the data...
Querying database...
...done!
=================STEP 2: TRAINING ALGORITHMS=================
-------------------Training Maxilla Regression (1/4)-------------------

61/61 [==============================] - 0s 0us/step

61/61 [==============================] - 0s 0us/step

61/61 [==============================] - 0s 0us/step

61/61 [==============================] - 0s 16us/step
-----------------Training Maxilla Classification (2/4)-----------------
# Tuning hyper-parameters for f1

Best parameters set found on development set:

{'kernel': 'linear', 'C': 1.0}

-----------------Training Mandible Classification (3/4)-----------------
# Tuning hyper-parameters for f1

C:\Users\Danilo\Desktop\MASTERPROEF\CONF\venv64\lib\site-packages\sklearn\model_selection\
_split.py:605: Warning: The least populated class in y has only 2 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=3.
\% (min_groups, self.n_splits)), Warning)
Best parameters set found on development set:

{'kernel': 'rbf', 'C': 2.183673469387755, 'gamma': 0.001206896551724138}

--------------------Training Chin Regression (4/4)--------------------

61/61 [==============================] - 0s 801us/step

61/61 [==============================] - 0s 965us/step
==================STEP 3: REPORTING STAGE==================
------------------------Maxilla Classification------------------------
Model accuracy: 85.9375%
--------------------------Maxilla Regression--------------------------
Maxilla Advancement
Model parameters: loss: 3.925313949584961. mean_absolute_error: 1.6489484310150146.
mean_squared_error: 4.328970432281494.
Maxilla Anterior
Model parameters: loss: 5.531023979187012. mean_absolute_error: 1.7098286151885986.
mean_squared_error: 5.531023979187012.
Maxilla Posterior
Model parameters: loss: 2.532546261591051e-07. mean_absolute_error: 0.0005029244930483401.
mean_squared_error: 2.532546261591051e-07.
Maxilla Midline Rotation
```

```
Model parameters: loss: 0.25399526953697205. mean_absolute_error: 0.14487725496292114.
mean_squared_error: 0.25399526953697205.
-----------------------------Mandible Classification------------------------------
Model accuracy: 82.8125%
----------------------------------Chin Regression-----------------------------------
Chin Advancement
Model parameters: loss: 3.7077255249023438. mean_absolute_error: 1.0736597776412964.
mean_squared_error: 3.7077255249023438.
Chin Intrusion/Extrusion
Model parameters: loss: 0.2004764974117279. mean_absolute_error: 0.0962613970041275.
mean_squared_error: 0.2004764974117279.
Goodbye
```

# Bibliography

[1] G. R. Swennen, *3D Virtual Treatment Planning of Orthognathic Surgery.* Springer Berlin, 2016.

[2] A. Ng, "Machine learning," https://www.coursera.org/learn/machine-learning/home/welcome, [Online; accessed September 2017].

[3] A. Bhande, "What is underfitting and overfitting in machine learning and how to deal with it." https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76, March, [Online, accessed 11 May 2018].

[4] K. Takada, "Artificial intelligence expert systems with neural network machine learning may assist decision-making for extractions in orthodontic treatment planning," *Journal of Evidence Based Dental Practice*, vol. 16, no. 3, pp. 190 – 192, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532338216300951

[5] T. M. Mitchell, *Machine Learning.* WCB McGraw-Hill, 1997.

[6] K. Phil, *MATLAB Deep Learning.* Berkeley, California: Apress Media LLC, 2017.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[8] E. B. Britannica Academic, "Coefficient of determination," academic-eb-com.kuleuven.ezproxy.kuleuven.be/levels/collegiate/article/coefficient-of-determination/605386, December 2013, [Online; accessed 10 May 2018].

[9] sci-kit learn developers, "sklearn.metrics - file:regression.py," https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/metrics/regression.py#L444, [Online, accessed 11 May 2018].

[10] ——, "sklearn.metrics.r2_score," http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html, [Online, accessed 11 May 2018].

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," 2014.

[12] sci-kit learn developers, "sklearn.feature_selection.f_regression," http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html, [Online, accessed 16 May 2018].

# Auteursrechtelijke overeenkomst