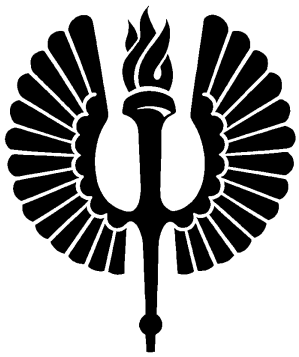

MACHINE LEARNING IN PERSONALIZED MEDICINE: MODELING MULTIPLE MYELOMA TREATMENT RESPONSES

Leo Uramoto

Master's Thesis
August 2020

DEPARTMENT OF MATHEMATICS AND STATISTICS
UNIVERSITY OF TURKU

UNIVERSITY OF TURKU
Department of Mathematics and Statistics

Leo Uramoto: Machine learning in personalized medicine: modeling multiple myeloma treatment responses
Master's thesis , 64 pages, including 3 pages of attachments
Applied Mathematics
August 2020

This thesis explores various machine learning models and attempts to use them to gain insight on individual differences in treatment responses of patients diagnosed with multiple myeloma, a hematopic malignancy representing approximately 1% of all cancers. While the 5-year survival rate exceeds 50%, treatments are not personalized and treatment results are heterogeneous.

Machine learning systems processing large amounts of patient data could help tailor the treatments to individual patients. Two research goals are set; understanding the medical trajectory of a patient by predicting the best treatment response they will reach in the future and predicting how long a given patient can remain on a given drug.

The work is split into two main parts. In the first part the mathematical concepts of machine learning and several commonly used machine learning models are introduced. In the second part the theoretical portions are applied by training the introduced models on a data set from the CoMMpass study by the Multiple Myeloma Research Foundation to answer the research questions.

The results from the medical trajectory prediction are somewhat promising and with additional research could become accurate enough for real-world use. The drug duration prediction task turns out too complicated for the limited methodology of this thesis. A discussion of the results and possible improvements on the methods are provided.

Keywords: Applied mathematics, machine learning, personalized medicine, multiple myeloma

Contents

1 Introduction

1.1 Description of the disease

Multiple myeloma (MM) is a hematologic malignancy arising from an asymptomatic condition known as Monoclonal Gammopathy of Undetermined Significance (MGUS). MGUS, one of the most common premalignant disorders in Western countries, can be found in approximately 3% in the Caucasian population above the age of 50[1]. MGUS develops into MM at a rate of approximately 1% per year.

MM accounts for approximately 1% of all cancers and 10% of hematologic malignancies [2] in the United States. MM affects mostly the elderly which is also visible in the data set in Figure 1.3. In a 2003 review of patients diagnosed with MM [3], the median age was 66 with only 2% of those diagnosed under the age of 40. Both ethnic background and sex are source of disparities in MM rates.

MM affects a type of white blood cell known as the B-cell. It causes malignant plasma cells to expand and accumulate in the bone marrow [4]. This leads to reduction in blood cells, breaking down of bone tissue and in most cases production of monoclonal protein, characteristic of plasma cell tumors. Most common symptoms of MM are pain and fatigue [5].

Diagnostic testing for MM includes blood and urine tests and bone marrow samples. A typical finding is the presence of the monoclonal protein, a high presence of bone marrow plasma cells or a detection of plasmacytoma in a biopsy [6]. Full radiographic skeletal surveys are done to detect bone lesions, osteopenia or pathological fractures.

MM with no end-organ damage is called smouldering multiple myeloma (SMM). Early treatment of SMM has shown no benefit [6]. Patients with SMM require no treatment but should be monitored for progression. End-organ damage is detected by elevated calcium levels caused by breaking down of bone tissue, renal failure, anemia and bone lesions.

Achieving a full cure or enduring remission of MM is rare [6]. Thus the main goal of treatment is managing the disease. With modern treatments the median survival of MM patients is above 5 years and the 3-year survival rates have exceeded 75% [7]. Autologous stem-cell transplantation helps prolong the survival, but has upper age limits that vary by country. Surgery and radiotherapy can be used for bone-related complications [6].

1.2 Motivation, research goals and outline of the thesis

While the expected survival duration of MM has increased in the last years, the therapies are still in general not targeted but rather used in all suitable cases [4]. A current major challenge is separating patients based on risk level and treatment tolerance [6].

For both the patient and the doctor having a realistic idea of what the future holds for the patient is of interest. Life expectancy and 5-year survival rate are common metrics when talking about the future of cancer patients. This thesis attempts to instead predict the medical trajectory of the patients. The data set contains an evaluation of the patient's response to the treatment. The encoded responses and their frequency in the data set are shown in Figure 1.1.

Research question 1.2.1. *Given full medical data of a patient* [1] *is it possible to predict what is the best treatment response the patient can reach in the future?*

When choosing a treatment option for a patient several different factors such as the expected efficacy and side effects are taken into account. For cancer treatments one factor of interest is how long a patient can remain on a treatment plan until either the cancer becomes immune to it or the side effects become unbearable. The distribution of drug durations is shown in Figure 1.2.

Research question 1.2.2. *Given a drug and full medical data of a patient is it possible to predict how long the patient can remain on the drug?*

In Chapter 2 I will present a mathematical formulation for machine learning problems and related basic concepts. In Chapter 3 I will present the models that I use in Chapter 4 in answering the research questions 1.2.1 and 1.2.2. Discussion about machine learning in personalized medicine, the results of the analysis and potential further work are discussed in Chapter 5.

[1] Full medical data refers to the collection of all measurements performed on the patients in the CoMMpass study.

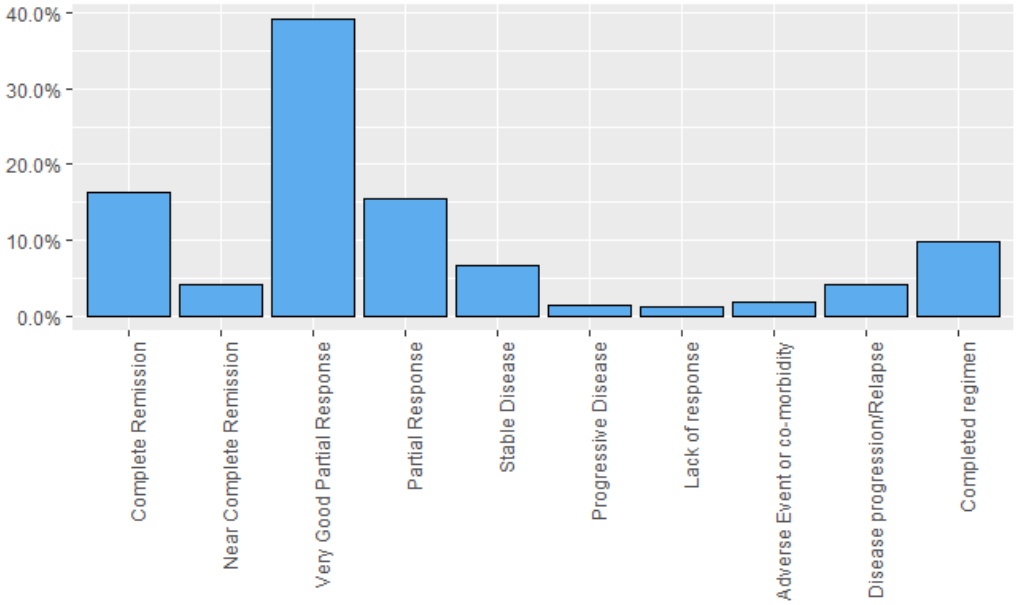

Figure 1.1: Distribution of the encoded responses

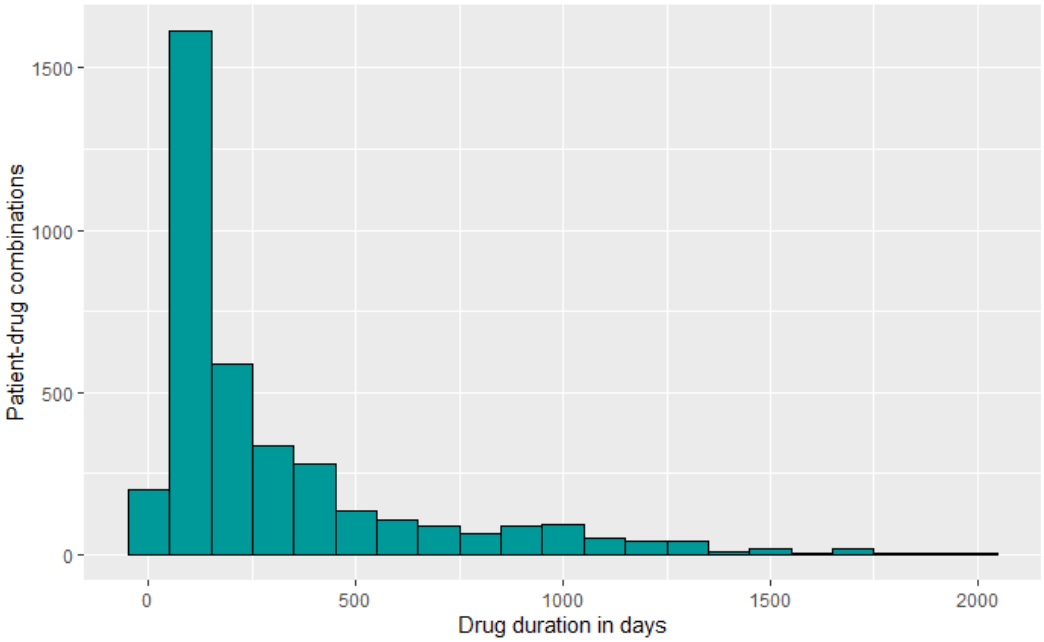

Figure 1.2: Distribution of the drug duration

1.3 Data set

The data set used in this thesis is from the CoMMpass (Relating Clinical Outcomes in MM to Personal Assessment of Genetic Profile) study by the Multiple Myeloma Research Foundation (MMRF) [2]. MMRF researchers tracked over 1000 newly diagnosed MM patients from their initial diagnosis through their treatment over a minimum of 5 years. Sequential tissue collection was done to identify the effects of patients' molecular profile on progression and treatment of the disease.

The data set covers the medical history of the patients along with the history of cancer in their family and their ethnic background. The data gives a longitudinal description of the progression of the disease for each patient with each treatment plan, hospital visit, clinical test, medical decision and outcome documented.

The data set is diverse in terms of ethnic background, age and sex. Figure 1.3 shows an approximation of the age and ethnicity distributions and 1.4 the absolute numbers for ethnic groups and sexes.

The patients in the data set were given 41 different drugs, identified only by their index $1, \ldots 41$. Many drugs were given to only a handful of patients. Figure 1.5 shows the number of patients that received a given drug at least once during their treatment. Only drugs with 10 or more patients are displayed.

Most of the initial data cleaning was done in a separate project. The original data set included numerical data, categorical data and abstract data in the form of doctor's notes. The data was spread over several files that in the initial cleaning phase were combined into one file to be used in machine learning projects. Some patients were missing a significant amount of data and were left out. Equally features with significant amounts of data missing were left out. Manual feature selection was performed to leave out irrelevant data.

The data set given to me was built to track drugs given to patients and a single visit to the facilities resulted in one row of measurements per drug received by the patient. The contained 13512 rows covering information of 1012 unique patients and 236 features.

Further processing of the data set is described in Chapter 4.

[2] `https://themmrf.org/`

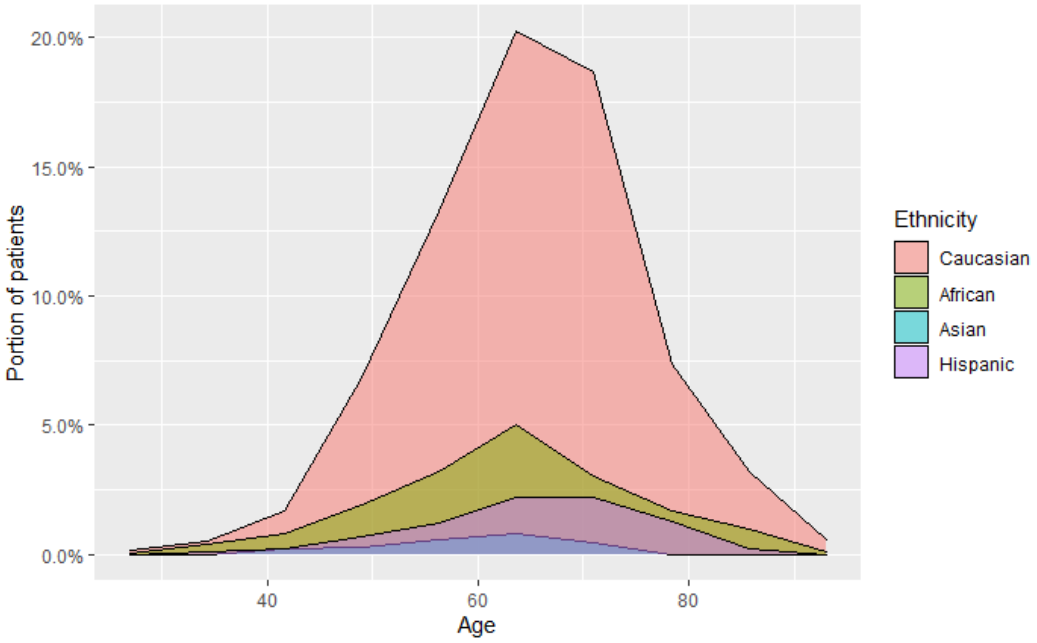

Figure 1.3: The age and ethnicity densities

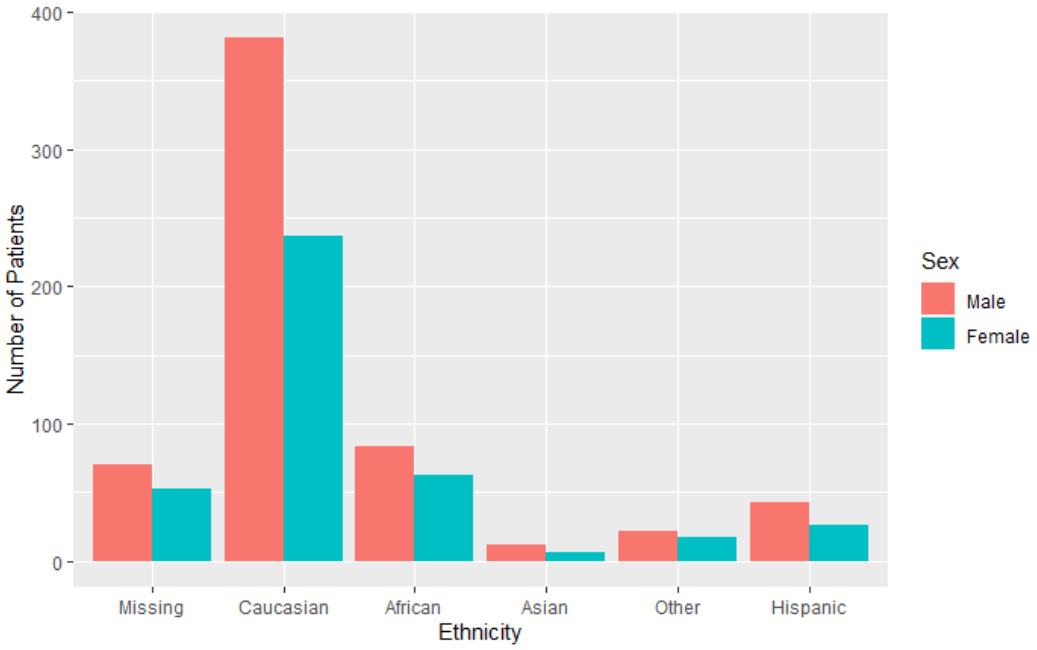

Figure 1.4: The total number of patients per ethnic background and sex

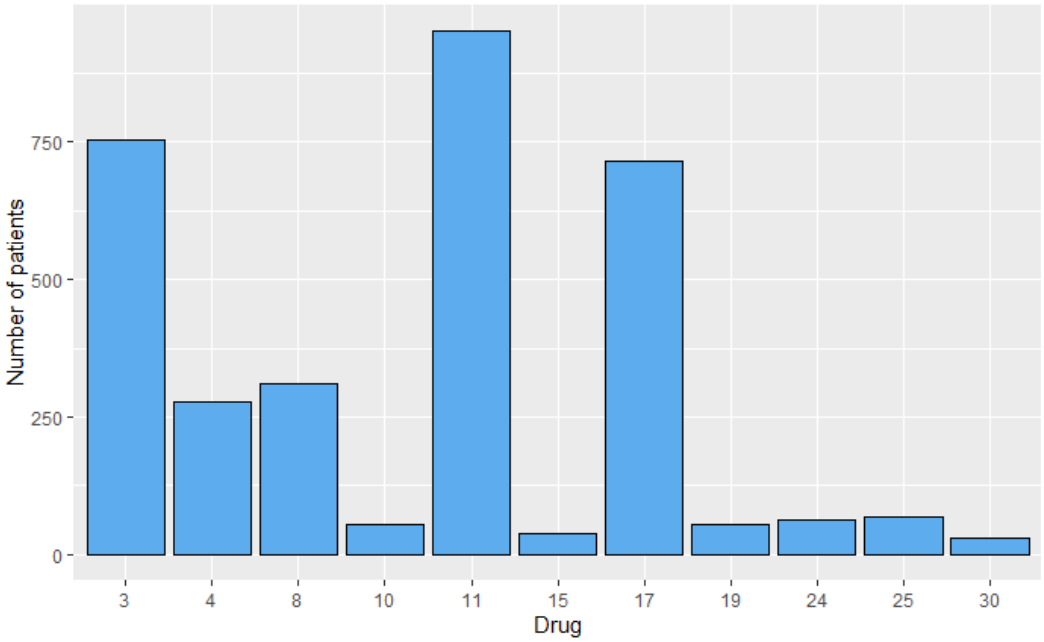

Figure 1.5: The total number of patients that received a drug

2 Machine learning basics

2.1 Representing data

Machine learning algorithms process massive amounts of data. This data can come in various forms such as images, categorical data or real-valued measurements. For a mathematical algorithm to be able to process data, it must be in a form it can understand. Images are converted to real vectors, depending on the algorithm categorical data can be processed as-is or encoded into numeric form, equally numeric data can be processed as is or converted into categorical form.

A simple example of an encoding scheme for a categoric-to-numeric mapping is to give each of the categories k_i an index i. This approach can be problematic as many algorithms would interpret the magnitude of the numbers to have some kind of inherent meaning. With some categories and indices, such as as severity of metastasis, this could make sense. However with examples such as gender and ethnicity this loses all meaning and can cause certain models to perform poorly.

A common way of avoiding this issue is the so called one-hot encoding. The idea of one-hot encoding is to add dummy variables for the categories and use an indicator function to set the dummy variables to either 0 or 1 based on the categoric value. A binary "sex" variable can be replaced with a new "is male" variable, with possible values 0 and 1. If the variable shows the person not to be male, they are known to be female. This can be extended to $c_1, \ldots, c_n$ categories being represented as $v_1, \ldots, v_{n-1}$ numeric variables. If all of the $n-1$ variables are $= 0$ then the categorical variable must be c_n.

A real-valued variable $x \in A \subset \mathbb{R}$ can be encoded into categorical data of n categories by partitioning the variable space A into n sets $A_i, \ldots, A_n$ such that $A_i \cap A_j = \emptyset$ when $i \neq j$ and $\cup_{i=1}^{n} A_i = A$. An indicator-like indexing function $ind : A \to 1, \ldots, n$ that gives the index i of the set A_i that x belongs to maps a continuous variable to a discrete space. This value is then encoded as an observation of the i:th category.

2.2 Mathematical formulation of machine learning problems

Machine learning problems can be classified [8] into three main types: supervised, unsupervised and reinforcement learning problems. Other types such as semi-supervised learning can be seen as combinations or variations of the former three. In supervised learning the input data corresponds to a human labeled output and the goal is to find a function that maps a given input to the correct output. Unsupervised learning lacks the human labeled output and instead attempts to draw conclusions about the data by e.x. finding anomalies in the data or organizing the data into clusters. Reinforcement learning deals with learning optimal behavior policies based on observations of the environment and by the feedback provided by it as it is interacted with.

This thesis is restricted to supervised learning problems and the definitions used are not general enough to necessarily cover other problem types. Supervised learning problems can be further divided into regression and classification problems based on the type of the output. In a classification problem the output is a categoric variable with m categories one-hot encoded into a vector in $\mathbb{R}^m$ with each element i corresponding to a probability of being classified as the ith category while in a regression problem the observation is a scalar value.

Definition 2.2.1. *(Data set) A data set D is a set of pairs $(\mathbf{x}_i, \mathbf{y}_i)$ where $\mathbf{x}_i \in \mathbb{R}^n$ is a collections of measurements called instances and $\mathbf{y}_i \in \mathbb{R}^m$ their corresponding observation.*

The elements $x_{i1}, \dots x_{in}$ of the instances $\mathbf{x}_i$ are typically called the independent variables as they vary independently while the observation $\mathbf{y}_i$ is called the dependent variable as it depends on $\mathbf{x}_i$.

With m being appropriate for the task at hand, the assumption is that there exists a function $g : \mathbb{R}^n \to \mathbb{R}^m$ that correctly maps the independent variables to the dependent variable. Let $\boldsymbol{\epsilon}$ be an m dimensional random vector representing the noise in the system, with each component of $\boldsymbol{\epsilon}$ a random variable of an unknown distribution with finite variance and mean 0. The relationship between the dependent and independent variables can be written as

$$\mathbf{y} = g(\mathbf{x}) + \boldsymbol{\epsilon}. \tag{2.2.1}$$

The goal is to find an approximation to g based on a data set D. The function g can be modeled by either parametric or non-parametric models. A

parametric model approximates g by a function $f(w_1, \ldots, w_k, \mathbf{x})$ with model parameters $w_1, \ldots, w_k$. A non-parametric model on the other hand builds an approximating function $f(x)$ from the data set D where the exact form of f is not known in advance.

Definition 2.2.2. *(Parametric Model, Parameter space) A parametric model $\mathscr{F}$ for vector variable $\mathbf{x} \in \mathbb{R}^n$ with parameters $w_1, \ldots, w_k \in \mathbb{R}$ is a family of functions $f_w(\mathbf{x}) = f(w_1, \ldots, w_k, \mathbf{x})$ indexed by a parameter space $W \subseteq \mathbb{R}^k$ with $\mathbf{w} = (w_1, \ldots, w_k) \in W$.*

A non-parametric model can be seen as a special case of Definition 2.2.2 with the parameter space $W = \{0\}$ consisting of a single trivial element and the function $f(\mathbf{x})$ built by the model can be interpreted to be the function $f_0(\mathbf{x})$ corresponding to the trivial element in D.

The final piece needed to properly define a machine learning problem is a way to evaluate how well the function f_w is approximating the underlying function g. A loss function $\hat{L}(f, D) \to \mathbb{R}_+$ measures how close to $\mathbf{y}$ the function f maps the inputs $\mathbf{x}$. Loss functions are discussed in more detail in Chapter 2.4. The loss functions are assumed to be expressible as sums over D, i.e.

$$\hat{L}(f, D) = \sum_{(\mathbf{x},\mathbf{y}) \in D} L(f_w(\mathbf{x}), \mathbf{y}). \tag{2.2.2}$$

It is now possible to give a general definition for a machine learning problem. Let D be a (training) data set, L a loss function, W the index set of a model and f_w the function corresponding to a given $\mathbf{w} \in W$.

$$\begin{aligned} \arg\min_{\mathbf{w}} \quad & \sum_{(\mathbf{x},\mathbf{y}) \in D} L(f_w(\mathbf{x}), \mathbf{y}) \\ \text{s.t.} \quad & \mathbf{w} \in W \end{aligned} \tag{2.2.3}$$

The function solving optimization problem in Equation 2.2.3 is denoted by $\hat{f}$. Interpreting a non-parametric model as a special case of a parametric model is clearly compatible with the formulation of the problem in Equation 2.2.3 and its solution is the function $f_0(\mathbf{x})$. During the minimization process the model is *trained* on the training set D, leading to a function $\hat{f}$ that can be said to have *learned* to approximate the underlying function g on D. The function $\hat{f}$ corresponds to an instance of the model $\mathscr{F}$ and is called a trained model.

According to the principle of empirical risk minimization [9], a function $\hat{f}(\mathbf{x})$ minimizing the loss for a training set D should also minimize the loss

for yet unseen data. If the data set was split into a training and test set as discussed in Chapter 2.3, the quality of the solution $\hat{f}$ can be tested on the test data to see how well it performs with unseen data. Good performance on the test set suggests it will perform well on data collected in the future, assuming the data set D was large enough and offered a good representation of the future data as well.

2.3 Training, validation and test sets and sampling

A data set is typically partitioned into at least two separate sets, a training set and a test set. As the names suggest the model is trained on the training set and once the training phase is complete the trained model is tested on the test set. Sometimes evaluating the performance during the training phase is useful. A part of the training set can be set aside for that purpose. This set is called a validation set.

It is important to separate the test set completely from the training data and not allow it to have any impact on the training phase. As an example many algorithms require the continuous variables to be standardized. Standardizing the training data and test sets together is a form of data leakage as the parameters used in standardizing the training data are affected by the test data. The proper way to standardize a data set is to standardize the training data separately, then standardizing the test data using the means and the standard deviations of the training data.

The partitioning is typically done by random sampling. In some cases it is useful to guarantee that the training and test sets are distributed similarly. Consider a data set on cancer screening. If the number of positives is very low and not offset by a large enough sample size, it is possible for the rate of positives in training and test sets to be differ significantly. In such a situation it is useful to divide the data set into groups or stratum that are then sampled independently to get the desired distributions in both training and test sets.

Stratified sampling is used in Chapter 4 in the classification task.

2.4 Measuring success

To be able to compare models it is necessary to have a metric for how well the underlying function g is being approximated. Naturally these metrics will be different for regression and classification tasks and are therefore discussed separately.

2.4.1 Regression

Let $\mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}$ for $i = 0, \dots k$ and f be a function $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(\mathbf{x}_i) \approx y_i$. Several metrics for the measuring how well the function f matches with the observations y_i exist. In general these metrics start by calculating the residual errors

$$\epsilon_i = y_i - f(\mathbf{x}_i). \tag{2.4.1}$$

The errors ϵ_i can be used in different ways to create a metrics with desired properties.

Mean absolute error (mae),

$$mae = \sum_{i=0}^{k} \frac{|\epsilon_i|}{k}, \tag{2.4.2}$$

calculates the average absolute deviation of the prediction from the true value. The *mae* metric has the same unit as y making it easy to interpret.

While *mae* is easy to interpret, the penalty for deviation is linear. In real world applications the cost of errors is often non-linear and large deviations between the real and the predicted value can have devastating consequences and thus oftentimes penalizing large deviations more is preferred. Mean square error (mse),

$$mse = \sum_{i=0}^{k} \frac{\epsilon_i^2}{k}, \tag{2.4.3}$$

penalizes outliers with the square term. However *mse* is not in the same unit as the target variable. Additionally in problems where y has large values the *mse* value can be very large leading to added difficulties in interpreting the value. Root-mean-square error (rmse),

$$rmse = \sqrt{mse} = \sqrt{\frac{\sum_{i=0}^{k} \epsilon_i^2}{k}}, \tag{2.4.4}$$

takes the square root of *mse*, mapping it to the same unit as y and to a more easily understandable magnitude, leading to again easier interpretation.

Minimizing *mse* is computationally simpler than *rmse* but leads to equivalent results. Thus all models were built minimizing *mse*. The *rmse* value is reported with *mae* shown alongside it.

The R^2 metric offers a different approach by measuring how much of the variance of the dependent variable can be explained by the model. With $SSE = \sum_{i=0}^{k}(\epsilon_i)^2$ being sum of squared estimation errors and $TSS = \sum_{i=0}^{k}(y_i - \bar{y})^2$ the total sum of squares

$$R^2 = 1 - \frac{SSE}{TSS}. \tag{2.4.5}$$

It can be seen that if $SSE = 0$, or, if $f(\mathbf{x}_i)$ matches perfectly with y_i, $R^2 = 1$. If the function f always returns the mean of y_i regardless of the input, $f(\mathbf{x}_i) = \bar{y}$ for all i then $SSE = TSS$ and $R^2 = 0$. There is no lower limit for R^2 as the SSE can be arbitrarily large.

The downside of R^2 is that it does not measure goodness of fit or correctness of the model. High variance leads to low R^2 values, even if the model itself is correct. In Figure 2.1 two data sets were generated from $y = x + N(0, \sigma^2)$ with two different values for σ^2 and the R^2 values calculated for the model $y = x$. While the model is correct in both cases, it can not be seen from the R^2 values.

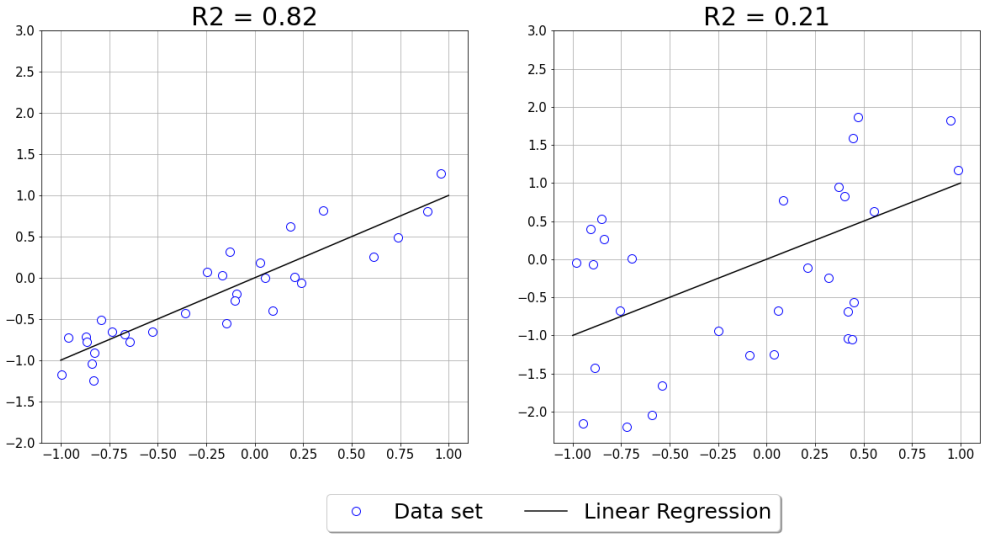

Figure 2.1: A high variance leads to a low R^2 value, even when the model is correct

As R^2 can still offer a way of comparing models built and tested on the same data the value is displayed alongside *rmse* and *mae* metrics.

2.4.2 Classification

The simplest way to measure success for classification tasks is accuracy. However as the only metric it can be very misleading. Consider a binary classification task in early cancer screening. If 98% of the patients are classified as cancer negative and 2% as positive, a trivial classifier classifying each patient as negative would be 98% accurate.

One way to understand classification accuracy better is the confusion matrix. Each row of a confusion matrix corresponds to the true class A_j and each column to predicted class A_j. It is possible to add a row and a column for totals to make it easier to compare the distributions of the actual and the predicted values and to notice degenerate behavior like always predicting the most common class. As an example a confusion matrix for a hypothetical 3-class classification task could look like this

$n = 150$	$A_{1\text{ pred}}$	$A_{2\text{ pred}}$	$A_{3\text{ pred}}$	Total
A_1	80	15	5	100
A_2	3	15	2	20
A_3	20	5	5	30
Total	103	35	12	150

It can be seen that the hypothetical model misclassified the class A_3 much more often than classes A_1 and A_2 and that the distribution of predicted values and actual values do not look the same. The true number of instances of A_3 was higher than A_2 but in the predictions A_2 was predicted almost 3 times as often as A_3.

A confusion matrix still does not offer a single simple value to compare models by. One possible metric for comparing classification models with one simple number is cross-entropy. Let q_i be an indicator of a correct classification and p_i the probability of the true class being class i. Cross entropy is defined as

$$CrossEntropy = -\sum_{i=0}^{n} q_i log(p_i). \tag{2.4.6}$$

In this thesis the classification models are built minimizing cross entropy and it is the main metric used for comparing models. However due to how difficult it is to interpret the accuracy of the models is also provided.

2.5 Optimization algorithms

Since (parametric) machine learning problems are formulated as optimization problems it is natural that optimization algorithms play a key role in machine learning. Various algorithms, each with their own up- and downsides have been proposed, commonly based on a simple algorithm called gradient descent.

Since the negative gradient points towards the direction of the deepest descent it is possible to iteratively take small steps in this direction, recalculating the gradient after each step. Let f be a differentiable function. The algorithm is defined by the sequence

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_i) \tag{2.5.1}$$

where $\eta > 0$ is a hyperparameter called the learning rate. The learning rate defines the size of the step taken in the direction of the negative gradient. Choosing this hyperparameter carefully is important. Too large learning rates can lead to oscillating around the optimal value or possibly even worsening the error while too small values can lead to slow learning or being stuck in a poor local minimum.

Consider a machine learning problem with a data set D and a loss function expressible as a sum over D. Calculating the gradient of the loss function over the whole D is computationally expensive when D is large. Stochastic gradient descent is a modification of the formerly presented regular gradient descent algorithm where the gradient of the loss function is approximated by using a single sample drawn from D instead of the whole data set [10].

The stochastic approximations to the gradient can be noisy, leading to quick changes in the direction of descent. The algorithm might also oscillate or stop in a local minimum. The concept of momentum mitigates this by tracking Δx, directions that have persistently lead to reducing the target function. Let $\mu \in [0, 1]$ be a decay rate for the momentum and $\mathbf{g}_i$ an approximation of the gradient at iteration i. The Equation 2.5.1 can be modified to [11]

$$\begin{aligned} \Delta \mathbf{x}_{i+1} &= \mu \Delta \mathbf{x}_i - \eta \mathbf{g}_i \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \Delta \mathbf{x}_{i+1} \end{aligned} \tag{2.5.2}$$

There exist several more modern versions of the gradient descent algorithm. The algorithm used with neural networks in this thesis was ADADELTA [12]. ADADELTA does not require learning or decay rates but adapts them on its own by considering the root mean square of past gradients and steps

with

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \frac{rms[\Delta \mathbf{x}]_{t_1}}{rms[\mathbf{g}]_t}\mathbf{g}_i, \tag{2.5.3}$$

where rms is the root-mean-square over last w samples.

2.6 Variance-bias trade-off and overfitting

When selecting a parametric model to solve the machine learning problem formulated in Equation 2.2.3 some kind of assumption about the complexity and the structure the underlying function g is made. A model that with all possible free parameters $\mathbf{w}$ consistently differs from g is said to be a biased as an estimator [13]. Linear regression is a biased model since it assumes there to be a linear relationship between the measurement and the observation.

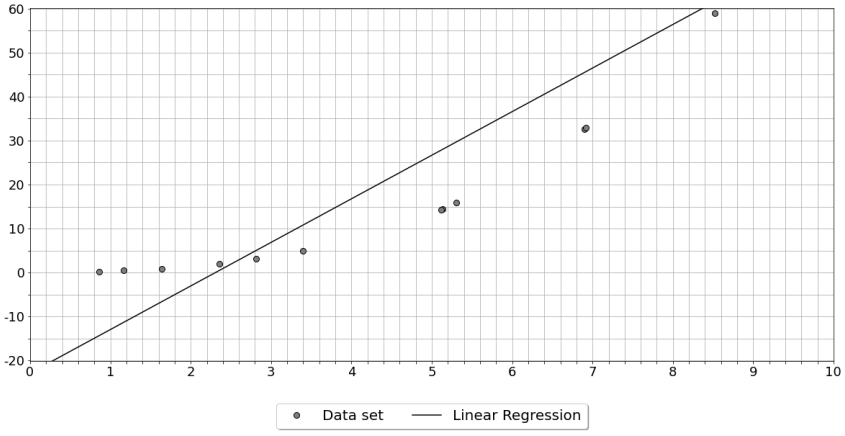

Figure 2.2: A linear model is not complex enough to create an accurate model

At the other end of the spectrum a high variance model is very sensitive to the training data. A function f_W could be an excellent approximation for g over the training set, but produce very poor results with previously unseen data. A function like this would not generalize well.

For the mse loss it is possible to decompose the the error of the model into a bias term and a variance term[13]. To emphasize dependence of the function f on the training set D it is written as $f(\mathbf{x}; D)$. With mse as the

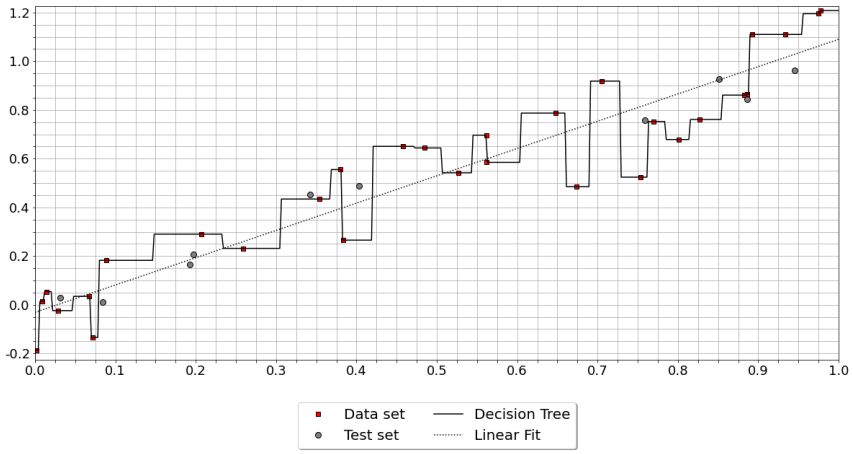

Figure 2.3: A decision tree regressor overfits to the data set. A linear model would have been a better choice.

loss function

$$
\begin{aligned}
&E_D[L(f(\mathbf{x};D),y)] \\
&= E_D[(f(\mathbf{x};D)-y)^2] = E_D[(f(\mathbf{x};D)-y+E[f(\mathbf{x};D)]-E_D[f(\mathbf{x};D)])^2] \\
&= E_D[f(\mathbf{x};D)-E_D[f(\mathbf{x};D)]+(E_D[f(\mathbf{x};D)]-y)^2] \\
&\quad + 2E_D[(f(\mathbf{x};D)-E_D[f(\mathbf{x};D)])(E_D[f(\mathbf{x};D)]-y)] \\
&= E_D[(f(\mathbf{x};D)-E_D[f(\mathbf{x};D))])^2+(E_D[f(\mathbf{x};D)-y^2 \\
&\quad + 2E_D[f(\mathbf{x};D)-E_D[f(\mathbf{x};D)]])(E_D[f(\mathbf{x};D)-y]) \\
&= E_D[f(\mathbf{x};D)-y]^2+E_D[(f(\mathbf{x};D)-E[f(\mathbf{x};D)))^2]] \\
&= E_D[f(\mathbf{x};D)-y_i]^2+Var[f(\mathbf{x};D)].
\end{aligned}
\tag{2.6.1}
$$

Interpreting the term $E[f(\mathbf{x})-y]^2$ as the bias of the model, it can be seen that the error of the model is a sum of the bias and the variance of the model. Finding the correct balance between the bias and the variance of the model is a key to building models that both learn efficiently and generalize well to unseen data.

2.7 Regularization

As mentioned finding the correct balance in model complexity is challenging. Regularization tones down the variance of a parametric model by making the solver take into account the complexity of the model.

Consider a function $R : W \to \mathbb{R}_+$ that evaluates the complexity of the function f_w for $\mathbf{w} \in W$. Taking the output of such a function into account when evaluating the loss in Equation 2.2.3 would force the optimization algorithm to take the complexity into account. Let L be a loss function, $\lambda > 0$ a regularization constant and $R(f)$ a regularizer function. A regularized loss function L_R is of the form

$$L_R(f_w(\mathbf{x}_i), y_i) = L(f_w(\mathbf{x}_i), y_i) + \lambda R(f_w). \tag{2.7.1}$$

Consider a linear model $f_w(\mathbf{x}) = w_1x_1 + \ldots w_kx_k$. Large values of w_i make the model more sensitive to small changes in x_i. A model less sensitive to changes in x_i will have lower variance, thus preferring lower values of w_i can be justified.

As the impact the parameter $\mathbf{w}$ has on a given model $\mathscr{F}$ varies by model all models have their own regularization systems. The Chapter 3.2.2 looks at ridge regression with $R(f_w) = \lambda \mathbf{w}^T\mathbf{w}$.

Early stopping [14] is a regularization method associated with neural networks, but it can be applied to any model where finding the best fitting function $\hat{f}$ is done iteratively. A small validation set is used for tracking the performance of the model periodically. If the performance of the training set begins to differ greatly from the validation set, the model is beginning to overfit the training set and instead of learning the underlying structures of the data it is fitting specifically to the training set. Allowing the algorithm to continue overfitting would often result not only in general performance not improving but sometimes it even declining. Once a certain threshold with no improvement in performance on the validation set is reached the algorithm is terminated and learning ends.

A simple neural network with 8 neurons on each of 3 layers was built to demonstrate potential issues. The model was trained on the Boston housing data set [15] to predict the median value of homes in areas of Boston from socioeconomic data. Figure 2.4 shows the *mae* values of the test and validation sets when training over 500 epochs or iterations on full data. The training and validation set *mae* values began separating quickly. Looking only at the training set makes it seem like there was an improvement in performance, but in reality the model began performing worse on unseen data.

While other regularization approaches exist for neural networks they will be omitted from this work.

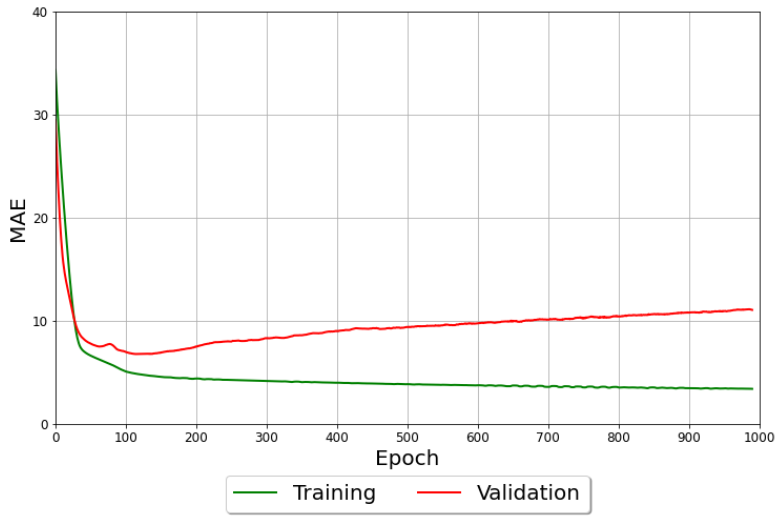

Figure 2.4: Early stopping would have prevented this model from overfitting.

2.8 Hyperparameters and cross-validation

Models often require certain decision that affect the learning process to be made before the learning begins. The choice of the optimization algorithm and constants such as the learning rate are some examples of such decisions. These are called hyperparameters of the model.

The choice of the hyperparameters can impact the performance and potentially the structure of the model built during the learning phase. To find the best performing model it is necessary to test several different sets of hyperparameters and compare the performance of the resulting trained models. Of course the actual test set can not be used for this and the training data is further split into a new, smaller training set and a test set used to evaluate the performance of the trained model for each hyperparameter set.

If several different hyperparameter sets are compared using the same test-train split it is possible for the test-train split to affect the hyperparameter selection process. The final model might perform well only on the exact test set used during hyperparameter selection but not generalize as well as expected. Having access to several training and test sets to confirm that the hyperparameters do indeed indeed build the best model in general would be preferable. However typically only a finite amount of data exists and using it efficiently is important.

k-folds cross-validation offers a way to validate the performance of a model on more than 1 train-test set. Instead of splitting the data set into one training set and one test set, the data is split into k sets called folds. The learning process is performed k times with each set being the test set once and the rest of the $k-1$ folds forming the training set. The average error of the k iterations is used to evaluate the performance of the model.

Figure 2.5 shows large performance differences between the folds observed during the analysis portion of this thesis. The figure shows a 3rd degree polynomial used in the regression task. The x-axis shows the regularization term λ. The k-folds average shows that the model performed in general poorly but on fold number 5 it did seem like a reasonable candidate. The loss curve of fold 5 is reasonably stable, small changes on λ result in only small changes in the *rmse*. However the k-folds average is highly unstable in several regions hinting at problems with the model. In this plot the neighborhood of $\lambda = 7$ corresponds to a local stable minimum and could be chosen as the optimal value.

Algorithm 2.8.1. *(k-fold cross validation)*
Let $k > 0$. Initialize the total error $E_{tot} = 0$.
(1) Split the data set into k folds $D_1, \ldots, D_k$.
(2) For $n = 1, \ldots, k$:

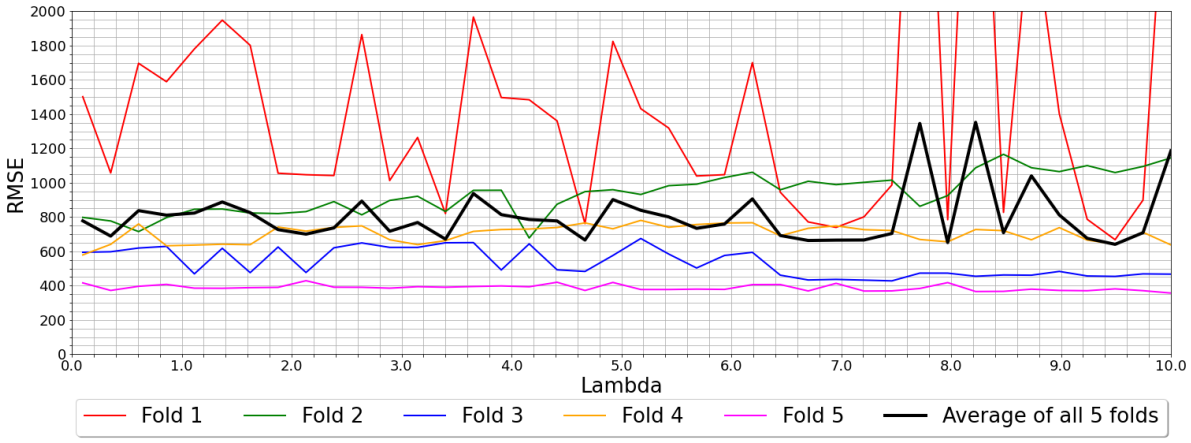

Figure 2.5: An example of significant performance differences between the folds and the average of all 5 folds

(2.1) Train the model on train data $T = D \setminus D_n$.
(2.2) Calculate the test error E_n *using* D_n *as the test set.*
(2.3) Add E_n *to* E_{tot}.
(3) Calculate the average error E_{avg}.

This leads to each element having been once in the test set and $k - 1$ times in the training set as shown in Figure 2.6.

Several improved variations of the base algorithm of Algorithm 2.8.1 exist [16]. These are typically computationally expensive and require computing capabilities deemed infeasible for the scope of this thesis.

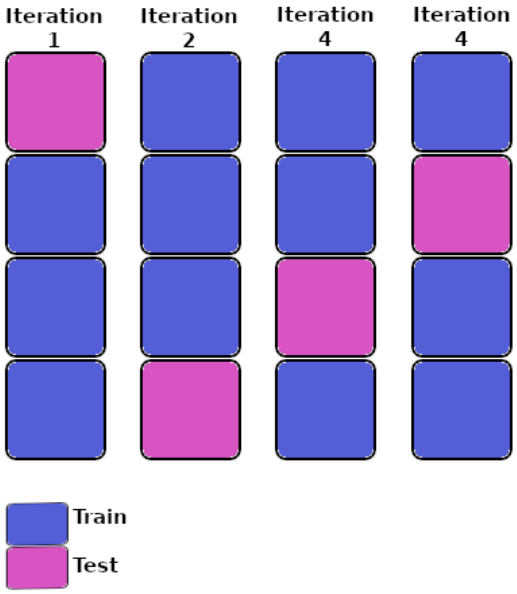

Figure 2.6: A visualization of the test-train split on a 4-fold cross validation

3 Machine learning algorithms

3.1 Decision trees and random forests

The decision tree algorithm and other algorithms derived from it are non-parametric models resembling flowcharts. They can be used for both regression and classification tasks. I will present decision trees and random forests in the context of classification problems due to the simplicity of their presentation. However both classification and regression trees are similar enough that there is no need to explore both.

3.1.1 Decision trees

A decision tree can be seen as a flowchart leading a sample to its classification. A decision tree consists of internal nodes that partition the data into subsets and terminal nodes or leaves that assign a class value to the sample and terminate the algorithm.

Decision tree is an umbrella term covering various related algorithms differing mainly on splitting and stopping criteria [17]. While multivariate splits are possible splitting is usually done univariately, based on only one feature. Let $\mathbf{x} \in X$ be an instance. A split on feature i on a value $t \in \mathbb{R}$ results in the data being split based on the condition $x_i > t$.

Choosing the feature i and the value t is done by comparing how the split reduces impurity of the post-split data. Measuring impurity of the data can be done as an example by the concept of entropy from information theory or by the Gini impurity that calculates the probability of mislabeling a random sample with a random category in the remaining instance space.

Tree building is done by recursively adding new nodes to partition the instance space. Once the remaining space can not be partitioned further based on a given stopping criteria, e.x. impurity reduction threshold value or a maximum depth is reached, a leaf node with the most common value in the remaining samples is created. The following algorithm is based on the C4.5 algorithm [18], but does not include pruning which is discussed later.

Algorithm 3.1.1. *(Decision tree)*
Let there be features $x_1, \ldots, x_n \in X$, $m : P(X) \to \mathbb{R}_+$, an impurity measure m, a stopping criterion λ for the impurity and a maximum depth d.
(1) For each feature $x_1, \ldots, x_n$ find the optimal splitting value t_i that splits the remaining data set D into $D_{i-} = \{D | x_i < t_i\}$ and $D_{i+} = \{D | x_i \geq t_i\}$ and calculate the corresponding impurities $Impurity_i = m(D_{i-}) + m(D_{i+})$ and the information gains $Gain_i = m(D) - Impurity_i$
(2) Add a split to the tree splitting the data set with index j maximizing the

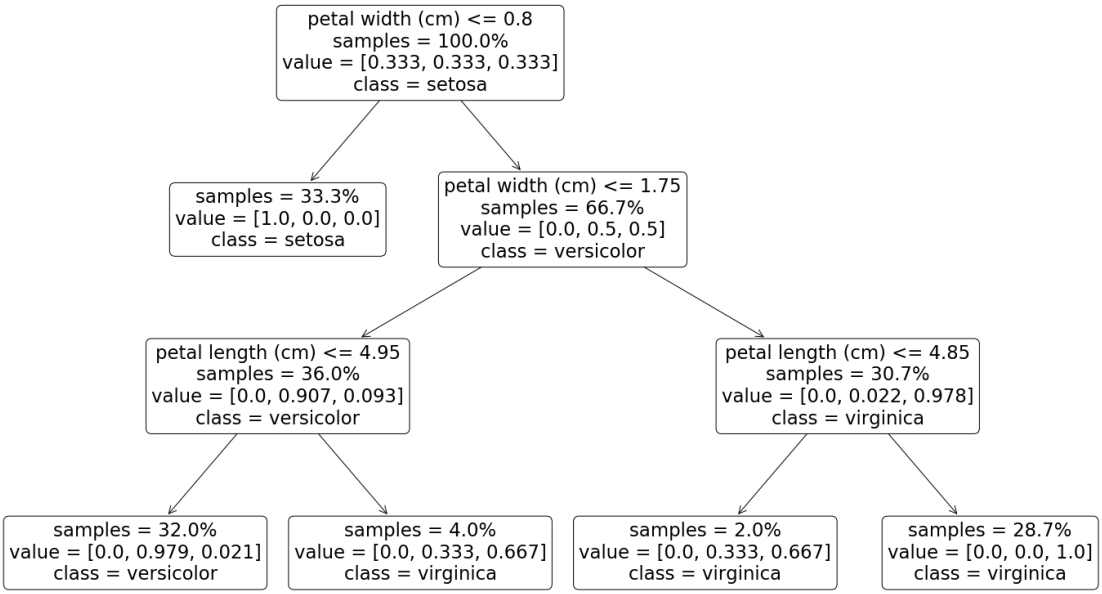

Figure 3.1: A decision tree classifier with depth 2 on the Iris data set

information gain into D_{j-} and D_{j-}.
(3) For both D_{j-} and D_{j+}: Denote the current set with T and test if the stopping criterion $m(T) < \lambda$ for impurity is fulfilled or if the maximum depth is reached. If either of the stopping criteria is met, create a leaf with value A_i where A_i is the category with the largest frequency in T. Otherwise set $D = T$ and go to (1).

The limitation of Algorithm 3.1.1 is that it can only process numeric data. It is possible to extend the algorithm to also process categorical data and to split on a single category in a node. Some implementations in the R [19] package do implement this behavior. However this thesis used the implementations of the sklearn [20] library which can only process numeric data.

One big difference between tree-based algorithms and other algorithms discussed in this thesis is that they can process categoric variables with a numeric encoding. Since several splits can occur on the same feature in one single tree, it is possible for a tree to separate classes $A_1, \ldots, A_k$ of a categoric variable encoded as $1, \ldots, k$ by splitting on the feature several times. This could have an impact with tree variants like random forests that select the features considered for a split randomly. Consider a feature set of n continuous variables and 1 categoric variable with the number of categories

being $> n$. If the categoric variable is one-hot encoded, the feature pool where the features considered for a split are drawn from will be heavily biased towards the single categoric variable. On the other hand if the depth of the tree is limited it might not be able to both perform the required number of splits on the numerically encoded variable to separate instances of a single category and to split on other features as well. It is also noteworthy that the data need not be standardized. If t splits a continuous feature z optimally and z is standardized by $z' = f(z, \sigma, \mu) = \frac{z-\mu}{\sigma}$, then $t' = f(t, \sigma, \mu)$ splits z' optimally.

Figure 3.1 shows a decision tree built with depth 3 using the Fisher's Iris data set[21]. The data set consists of length and width measurements of the sepals and petals and the species (Iris setosa, Iris virginica, Iris versicolor) of the measured flower.

Tight stopping criteria leads to small trees with low accuracy while loose criteria leads to overfitted trees that generalize poorly. Overly complex decision trees are also difficult to interpret[22]. These issues can be solved by pruning the trees by removing branches that do not significantly contribute to the accuracy of the tree. For a comparison of pruning algorithms see [22].

Regression trees differ from classification trees on two points: the impurity measurement has to be appropriate for regression data and if there are more than 1 instance in a leaf node their average is returned.

3.1.2 Random forests

Even pruned trees suffer from training bias and generalize poorly. This bias can be combated by generating multiple smaller trees and having them work together as an ensemble in making the prediction.

Bagging trees [23] algorithm builds an ensemble of independent trees, each constructed from a bootstrap sample with a given number of samples drawn with replacement from the data set D. Random forest algorithm improves the bagging trees algorithm by considering only a random subset of features at each split [24].

Algorithm 3.1.2. *(Random forest)*
Let n be the number of trees to be generated, m the number of random features used at each split and k the number of samples to draw from the data set for each tree.
(1) Draw n bootstrap samples from the data set, each with k samples from the data set.
(2) For each bootstrap sample, grow an unpruned decision tree, except at each node instead of using all features, sample m features at random without

replacement and choose the best one among those for the partitioning.
(3) In predictions use majority votes among all trees for classification and average of values for regression.

The idea behind the algorithm is that features that are good predictors of the dependent variable will be selected more often than bad predictors. Since the algorithm uses independent small random subsets of the data for each tree it should not overfit to the training set as strongly as regular decision trees. If the number of sampled features m is equal to the number of features in the data set, i.e. all features are considered, the algorithm produces a bagging tree ensemble.

Out-of-bag samples (OOB samples) are samples not in the bootstrap used in building a tree. The out-of-bag error (OOB error) of a tree is the aggregate error of the OOB samples when passed through the tree. The OOB error of a random forest is the average of the OOB errors of its trees. The OOB error of a random forest is a good estimate for the error of the forest on unseen data.

A further modification on the algorithm called extremely randomized trees or ExtraTrees [25] does not optimize the split location t but chooses it randomly.

3.1.3 Feature selection with random forests

Random forest based feature selection has been found efficient for large medical data sets [26]. Since random forests can be used for both regression and classification, random forest feature selection can similarly be used for both types of problems.

A random forest allows for ranking of variable importance [27]. Each tree t has its OOB sample OOB_t. Denote the error of the OOB sample by $errOOB_t$. The importance of a feature X^j can be calculated by randomly permuting the values of X^j in OOB_t. Denote the perturbed sample by $\widetilde{OOB_t}$ and the error of the perturbed sample $err\widetilde{OOB_t}$. The importance of a feature X^j in a random forest with n trees is

$$\text{Importance of } X^j = \frac{1}{n}\sum_t (err\widetilde{OOB_t} - errOOB_t). \tag{3.1.1}$$

The following algorithm selects k best features in predicting a dependent variable Y:

Algorithm 3.1.3. *(Random forest feature selection)*
Let D be a data set, n be the number of trees in the random forest, the

hyperparameters for the random forest be chosen and k the number of features to be selected.
(1) Build a random forest of n trees predicting the dependent variable Y.
(2) For each feature X^j calculate the importance value.
(3) Rank the features by their importances and choose the k features with the highest importance values.

3.2 Linear models

3.2.1 Linear regression

Linear regression is a simple model for regression. The model, while simple, is powerful enough as long as its assumptions are met. The model requires the following conditions to hold [28]:

(1) Vectors x_i and values y_i must have minimal measurement error.
(2) The relationship between the independent variables x_i and Y is linear, or expressible as

$$\mathbf{y} = a + \mathbf{X}\boldsymbol{\beta} + \mu, \tag{3.2.1}$$

where $\mathbf{X}$ is a matrix whose row vectors are samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and μ is a random variable with mean 0.
(3) The random variable μ is homoscedastic, or has constant variance.
(4) No autocorrelation is present, or the residuals are independently distributed.
(5) The independent variables are linearly independent of one another.

Equation 3.2.1 can be written in a simpler form by using padded vectors. Let $\mathbf{x}_i = [x_i^1, \ldots, x_i^n]$ be a sample with n features. The vector can be padded with a constant and the padded vector written as $\mathbf{x}_i = [1, x_i^1, \ldots, x_i^n]$. The vector $\boldsymbol{\beta}$ is padded in a similar way. Now Equation 3.2.1 can be expressed as a matrix multiplication and a random variable:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mu. \tag{3.2.2}$$

The loss is typically measured with *mse* which leads to the so called ordinary least squares fit formula:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{3.2.3}$$

Computing $\hat{\boldsymbol{\beta}}$ of Equation 3.2.3 with simple matrix algebra is straightforward when the number of features is low. However as the size of the matrix grows the calculation becomes prohibitively expensive. Large linear regression problems can be solved with algorithms stochastic gradient descent [29].

3.2.2 Ridge regression and kernels

Ridge regression is a regularized extension of linear regression. The higher the values β_i are, the more sensitive the model is to small changes in inputs. To create a more stable model and less sensitive model, smaller values of β are preferred.

Ridge regression modifies the loss function to the form

$$L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\beta}) = \frac{1}{2}(\mathbf{y}_i - \mathbf{x}_i^T\boldsymbol{\beta})^2 + \frac{1}{2}\lambda\boldsymbol{\beta}^T\boldsymbol{\beta}, \tag{3.2.4}$$

where λ is the regularization strength of the model. In matrix form this becomes

$$L(\mathbf{X}, \mathbf{y}, \boldsymbol{\beta}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \frac{1}{2}\lambda\boldsymbol{\beta}^T\boldsymbol{\beta}. \tag{3.2.5}$$

Taking the derivative of L with respect to $\boldsymbol{\beta}$

$$\frac{\partial L(\mathbf{X}, \mathbf{y}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} - \mathbf{X}^T\mathbf{y} + \lambda\boldsymbol{\beta} \tag{3.2.6}$$

and setting Equation 3.2.6 $= 0$ leads to

$$\begin{aligned}
&(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} - \mathbf{X}^T\mathbf{y} + \lambda\boldsymbol{\beta} = 0 \\
&(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y} \\
&\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}.
\end{aligned}$$

Thus best fit is reached with

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}. \tag{3.2.7}$$

Setting $\lambda = 0$ in Equation 3.2.7 proves the formula for ordinary least squares fit in Equation 3.2.3

Ridge regression can take advantage of the so-called kernel trick, allowing a linear model to learn non-linear behaviors without explicitly generating the non-linear model or the non-linear features. Formally this means choosing a more complex input space V over the initial input space $\boldsymbol{\chi}$ with $\phi : X \to V$ and instead of solving the ridge regression problem with input $\mathbf{x} \in \chi$, solving it with input $\phi(\mathbf{x}) \in V$. Denote new measurements by $\phi_i = \phi(\mathbf{x}_i)$ for each i and the matrix they form by $\boldsymbol{\Phi}$. If $\mathbf{X}$ was a $m \times n$ matrix of m samples and n features, $\boldsymbol{\Phi}$ is a $m \times p$ matrix with $p >> n$.

Substituting $\boldsymbol{\Phi}$ for X in Equation 3.2.7 and applying the push-through identity

$$(\mathbf{I} + \mathbf{UV})^{-1}\mathbf{U} = \mathbf{U}(\mathbf{I} + \mathbf{VU})^{-1} \tag{3.2.8}$$

yields

$$\hat{\boldsymbol{\beta}} = \boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\mathbf{I})^{-1}\mathbf{y}. \tag{3.2.9}$$

Denote $\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \mathbf{K}$. Taking a closer look at $\mathbf{K}$

$$\mathbf{K} = \begin{pmatrix} \boldsymbol{\phi}_1^T\boldsymbol{\phi}_1 & \dots & \boldsymbol{\phi}_1^T\boldsymbol{\phi}_m \\ \vdots & \ddots & \vdots \\ \boldsymbol{\phi}_m^T\boldsymbol{\phi}_1 & \dots & \boldsymbol{\phi}_m^T\boldsymbol{\phi}_m \end{pmatrix} \tag{3.2.10}$$

It can be seen that the elements of $\mathbf{K}$ are formed of all dot product combinations of $\phi(\mathbf{x}_i)$. What this means is that generating all of the features and performing the algorithm in V is not necessary, calculating the dot products of the elements in X in V is enough. In more general terms, given a function $k : X \times X \to \mathbb{R}$ and $\mathbf{x}_1, \ldots, \mathbf{x}_n \in X$, an $n \times n$ matrix with entries

$$\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_i) \tag{3.2.11}$$

is called a Gram matrix of k with respect to $\mathbf{x}_1, \ldots, \mathbf{x}_n$. In machine learning literature a Gram matrix is commonly called a kernel.

For any positive semidefinite kernel the function k can be represented as the inner product of some Hillbert space V [30]. The space V is called the Reproducing Kernel Hilbert Space induced by the kernel.

Definition 3.2.1. *Any algorithm formulated in terms of a positive definite kernel* $\mathbf{K}$ *is called kernelizable.*

The so-called "kernel trick" means creating a new algorithm by replacing the original kernel $\mathbf{K}$ of a kernelizable algorithm by a new positive definite kernel $\tilde{\mathbf{K}}$ that is the inner product of a more suitable feature space V. The inner product $\langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$ can be seen as a similarity measure of $\mathbf{x}_1$ and $\mathbf{x}_2$ under ϕ.

With n features, generating all polynomial combinations up to k degrees leads to generating $\binom{n+p}{p}$ features. This can easily lead to the number of features overtaking the number of samples a data set contains. Using the kernel trick makes the calculation simpler.

Table A.1 shows some kernels and the dimensions of V when the feature space has n dimensions.

Defining $\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}$ and substituting to Equation 3.2.9 gives

$$\hat{\boldsymbol{\beta}} = \boldsymbol{\Phi}^T \boldsymbol{\alpha}. \tag{3.2.12}$$

Predicting a new y using $\hat{\boldsymbol{\beta}}$ and the new features $\phi(\mathbf{x})$ of x is done with

$$y = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) = (\boldsymbol{\Phi}^T \boldsymbol{\alpha})^T \phi(\mathbf{x}) = \boldsymbol{\alpha}^T \boldsymbol{\Phi} \phi(\mathbf{x}). \tag{3.2.13}$$

The goal of the training phase is to find $\boldsymbol{\alpha}$. With even a moderate number of features this can not be done in a closed form and the training phase finds an approximation to $\boldsymbol{\alpha}$.

3.2.3 Logistic regression

Logistic regression despite its name is a classification algorithm. It is a linear model for the log-odds of a binary random variable, but can be extended to general finite discrete cases. Let $Y \sim Ber(p)$ be a Bernoulli distributed binary random variable, the log-odds of Y_1 are

$$l = log_b(\frac{p}{1-p}). \tag{3.2.14}$$

The choice of basis b is arbitrary for any $b > 1$, here the basis used will be e and the the natural logarithm denoted by log.

The logistic regression model assumes that the log-odds are linearly dependent on the predictor variables $x_1, \dots x_n$. As before the predictor variables are written as a padded $n+1$ dimensional column vector $\mathbf{x}$ with $x_0 = 1$ to simplify the linear model. The dependence of the resulting probability distribution is emphasized with the notation $P_\beta(Y = y)$. The final model can be written as:

$$f_\beta(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} = \beta_0 + x_1\beta_1 \dots \beta_n x_n = log(\frac{P(Y = Y_1)}{1 - P(Y = Y_1)}). \tag{3.2.15}$$

The model does not strictly classify a sample into a category, but the probability of it being in a given category can be retrieved from Equation 3.2.15. A sample can then be classified into the class with the highest probability.

Each weight vector $\boldsymbol{\beta}$ corresponds to a distribution of Y. The likelihood of observing a given pattern can be measured with the likelihood function

$$\mathcal{L}(\boldsymbol{\beta}|y_1, \dots, y_n) = P(Y = y_1 \text{ and } \dots Y = y_n|\boldsymbol{\beta}). \tag{3.2.16}$$

Since the observations are independent and identically distributed the likelihood function can be written as a product

$$\mathcal{L}(\boldsymbol{\beta}|y_1, \dots, y_n) = P(Y = y_1|\boldsymbol{\beta}) \cdot \dots P(Y = y_n|\boldsymbol{\beta}). \tag{3.2.17}$$

Maximum likelihood estimation (MLE)[31] finds the optimal value of $\boldsymbol{\beta}$ by maximizing the likelihood of observing a distribution matching the data set. A common way of finding the maximum is to apply gradient descent to the function $-\mathcal{L}$ thus maximizing $\mathcal{L}$.

Figure 3.2 shows the logistic regression model fitted to an imaginary data set shown in Table A.2 of household income and stock ownership.

For a general model with outcomes $Y_1, \dots, Y_n$, a pivot index is chosen. Because the indexing is arbitrary, choosing the last index n as the pivot leads

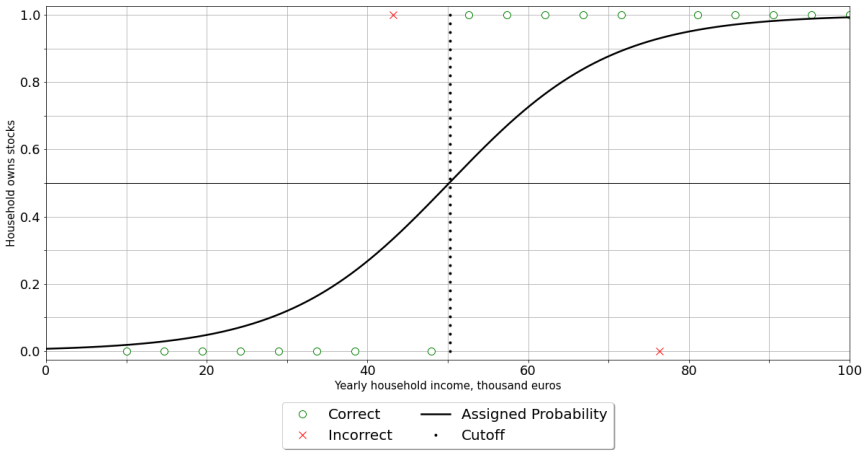

Figure 3.2: Binary classification with logicistic regression

to no loss of generality. For each index $i = 1, \dots n-1$ a model l_i is created, each with their own weights $\boldsymbol{\beta^i}$. Instead of the log-odds of $\frac{P(Y_i)}{1-P(Y_i)}$ like in the binary case, $\frac{P(Y_i)}{P(Y_n)}$ is used and assumed to be linearly dependent of the independent variables:

$$log(\frac{P(Y_i)}{P(Y_n)}) = \mathbf{x}^T \boldsymbol{\beta}^i. \tag{3.2.18}$$

Solving Equation 3.2.18 for $P(Y_i)$ yields

$$P(Y_i) = P(Y_n) e^{\mathbf{x}^T \boldsymbol{\beta}^i}. \tag{3.2.19}$$

Since the sum of probabilities $\sum_{k=1}^{n} P(Y_j) = 1$ and each $P(Y_i), i \neq n$ can be written in terms of $P(Y_n)$ and $\boldsymbol{\beta}^i$, the term $P(Y_n)$ can be written as

$$P(Y_n) = \frac{1}{1 + \sum_{k=1}^{n-1} e^{\mathbf{x}^T \boldsymbol{b}^k}}, \tag{3.2.20}$$

and for $i < n$

$$P(Y_i) = P(Y_n) e^{\mathbf{x}^T \boldsymbol{\beta}^i} = \frac{e^{\mathbf{x}^T \boldsymbol{\beta}^i}}{1 + \sum_{k=1}^{n-1} e^{\mathbf{x}^T \boldsymbol{b}^k}}. \tag{3.2.21}$$

The vector $\hat{\boldsymbol{\beta}}$ can again be found with MLE [32].

3.3 Deep neural networks

An artificial neural network (ANN) is a machine learning system based loosely on biological brains. The structure imitates neurons and the synapses connecting them [33]. Much like in biological brains, neurons can become activated causing synapses to transmit information about the activation to other neurons.

ANNs have a long history with the earliest related ideas being traceable to the 1940s and 1950s [34]. Two much later breakthroughs were necessary for their wide adoption; the backpropagation algorithm for efficient gradient calculation in 1970s and the introduction of the graphics processing unit (GPU) that, while not initially intended for machine learning, ended up providing hardware specializing in fast, parallel matrix multiplication allowing fast training of ANNs.

While ANNs are well known for their performance in image and pattern recognition they are extremely flexible having wide applications ranging from natural language processing to self-driving cars. ANNs have biomedical applications in diagnostic medicine, especially in the field of medical image recognition.

The downside of ANNs is that they are in practice black boxes. The massive amount of connections in subsequent layers scramble the data in a way that makes reasoning about their internal logic extremely difficult. In contrast models such as decision trees are easy to track and their logic is entirely transparent.

In mathematical terms an ANN is a computational graph. The base level building block of an ANN is the artificial neuron, a node in the graph, modeled by an activation function determining the neuron's output. These artificial neurons are interlinked by synapse-like connections. Three types of neurons exist; input neurons reacting to input fed from the environment, inner neurons reacting to the activation of other neurons and output neurons outputting the result of the computation. The input of a non-input neuron is the weighted sum of the activation of the neurons feeding into it and the neuron's bias term. Figure 3.3 shows how the three types of neurons link together forming a neural network.

In general neurons can be connected in any way imaginable. A special type of ANN called feed-forward network has its neurons in subsequent layers $1, \ldots, n$. The first layer, the input layer, receives the independent variables as input while the last layer, the output layer, outputs a prediction for the dependent variable. The layers between the input and the output layers are called hidden layers. An ANN with 2 or more hidden layers is called a deep neural network (DNN).

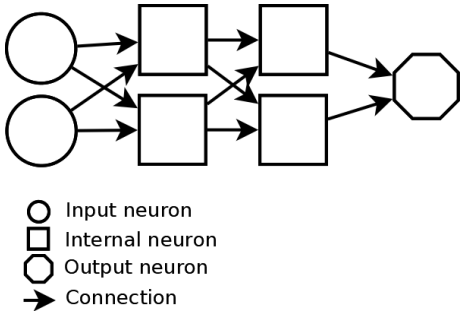

Figure 3.3: Three types of neurons are connected to form a computational graph

This thesis focuses only on feed-forward networks and the terms ANN and DNN are interpreted to refer exclusively feed-forward networks. DNNs are often massive, consisting of hundreds or even thousands input neurons and several layers. Figure 3.4 shows the 32-32-32 architecture for the classification task used in Chapter 4.

Let there be an ANN consisting of layers indexed $k \in \{1, \ldots, m\}$, each with n_k neurons. Let neuron η_j^i be the jth neuron of layer i and w_{jh}^i the weight of the connection from neuron η_j^i to neuron η_k^{i+1}. The weights of neurons in layer $i-1$ connecting to η_j^i can be written in vector form as $\mathbf{w}_j^i \in \mathbb{R}^{n_{i-1}+1}$, where the first element is the bias term.

Denote the output of neuron η_j^i by z_j^i. Again using a padded vector to allow for simple handling of the bias term the outputs of the neurons in layer k can be written as a vector $\mathbf{z}^k \in \mathbb{R}^{n_k+1}$ with $\mathbf{z}^k = (1, z_1^k, \ldots, z_{n_k+1}^k)$.

The input of neuron η_j^i denoted by a_j^i is now given by $a_j^i = (\mathbf{w}_j^{i-1})^T \mathbf{z}^{i-1}$ and the output $z_j^i = \phi_j^i((\mathbf{w}_j^i)^T \mathbf{z}^{i-1})$. Furthermore, the n_k weight vectors of layer k form a weight matrix $\mathbf{W}_k \in \mathbb{R}^{n_k \times n_{k+1}+1}$ and the input a_j^i is the jth component of the vector $\mathbf{W}_{i-1}^T \mathbf{z}^{i-1}$.

All of the neurons on a layer of an ANN typically share the same activation function. Given a shared activation function $\phi : \mathbb{R} \to \mathbb{R}$ shared by all neurons in layer k, denote $\phi_k : \mathbb{R}^n \to \mathbb{R}^n$ as the activation function applied to all of the n_k neurons of layer k.

Activation functions can take almost any form imaginable, but certain properties are desirable. At least some non-linearity is required to allow the network to approximate non-linear functions. Differentiability enables the use of gradient-based optimization algorithms for learning and monotonicity helps the algorithm converge faster.

A notable exception to the global differentiability requirement is the recti-

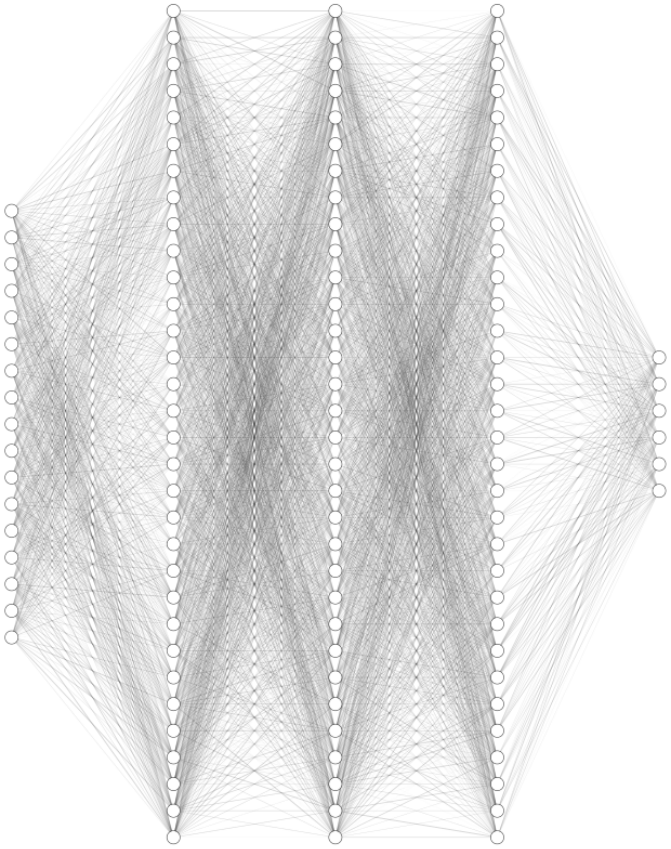

Figure 3.4: One of the neural network architectures considered in the classification task. The alpha values of the lines showing the connections were randomly sampled to prevent too many black lines from making the image impossible to interpret.

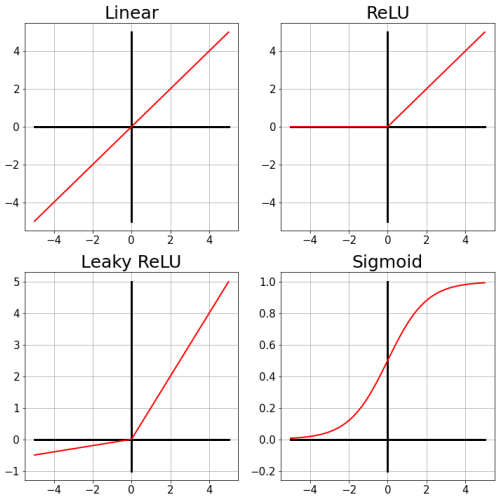

Figure 3.5: Examples of activation functions

fied linear unit (ReLU) function which is not differentiable at $x = 0$. However ANNs using ReLU activation still converge under stochastic gradient descent as long as the network is wide enough [35]. Historically the sigmoid function was the most common activaction function but ReLU has largely taken its place. Figure 3.5 shows a visual comparison of different activation functions.

For a classification task the final layer of the network should be the softmax function

$$\phi(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}} \text{ for each } i = 1, \ldots, n. \tag{3.3.1}$$

transforming the output to an n dimensional vector with components denoting the probability of the input being classified as the nth class. For a regression task the final activation function is typically linear $\phi(x) = x$ with range $(-\infty, \infty)$, however any function with the appropriate range for the task would be acceptable.

Definition 3.3.1. *(ANN) An ANN is a model $\mathscr{F}$ where the functions $f_w \in \mathscr{F}$ are of the form*

$$\begin{aligned} f_w(x) &= f(\mathbf{x}, w_1^1 \ldots w_{n_1}^1, w_1^2 \ldots, w_{n_k}^k) \\ &= \phi_k((\mathbf{W}^k)^T \phi_{k-1}((\mathbf{W}^{k-1}) \ldots \phi_1((\mathbf{W}^1)^T \mathbf{x}))), \end{aligned} \tag{3.3.2}$$

where ϕ_k is the activation function of layer k, $w_1^1, \ldots, w_{n_k}^k$ the weight vectors, $\mathbf{W}_1, \ldots, \mathbf{W}_k$ the weight matrices and $\mathbf{x}$ is the input of the network.

The layer structure with their neuron counts and activation functions defines a neural network model . The structure is often called an architecture. There are obviously an infinite number of ANN models one could choose for any problem.

Training an ANN is the task of minimizing the error produced by it over the training set. Given a loss function L, the total loss over a data set D can be written as

$$L(D) = \sum_{\{\mathbf{x}_i, \mathbf{y}_i\} \in D} L(f_w(\mathbf{x}_i), \mathbf{y}_i). \tag{3.3.3}$$

Finding the gradient of the loss function ∇L in the parameter space allows for gradient based optimization algorithms to do the training. The backpropagation algorithm [36], an efficient algorithm for calculating the gradient, arises from the chain rule and from realizing that all of the factors of the chain rule product are needed at different parts of the gradient. The backpropagation algorithm for feed-forward networks was initially described in [37] for the sigmoid activation function

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{3.3.4}$$

and the mean squared error loss function, but generalizing to any differentiable activation function and loss function expressable as a sum $L(D) = \sum_{(\mathbf{x},\mathbf{y}) \in D} L(\mathbf{x}, \mathbf{y})$ is easy.

Let there be an ANN with k layers. Denote the total error of the ANN for a sample $(\mathbf{x}, \mathbf{y}) \in D$ with E. With the chain rule the partial derivative of E with respect to a weight w_{ij}^l in layer l is

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_{ij}^l}. \tag{3.3.5}$$

The first term is called the error and denoted by δ_j^l,

$$\delta_j^l = \frac{\partial E}{\partial a_j^l}. \tag{3.3.6}$$

The second term of Equation 3.3.5 is simply

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial \mathbf{w}^k \mathbf{z}^{k-1}}{\partial w_{ij}^k} = z_i^{k-1}. \tag{3.3.7}$$

Combining equations 3.3.5, 3.3.6 and 3.3.7 yields the formula

$$\frac{\partial E}{\partial w_{ij}^l} = \delta_j^l z_i^{k-1}. \tag{3.3.8}$$

Let l be a hidden layer. Applying the chain rule to δ^l it can be written as

$$\delta_j^l = \sum_{p=1}^{n_{l+1}} \frac{\partial E}{\partial a_p^{l+1}} \frac{\partial a_p^{l+1}}{\partial a_j^l} = \sum_{p=1}^{n_{l+1}} \delta_p^{l+1} \frac{\partial a_p^{l+1}}{\partial a_j^k}. \tag{3.3.9}$$

The input in terms of the activation function is $a_p^{l+1} = w_p^{l+1} z^l = w_p^{l+1} \phi^l(a^l)$ and its partial derivative

$$\frac{\partial a_p^{l+1}}{\partial a_j^l} = w_{jp}^{l+1} \phi_l'(a_j^l). \tag{3.3.10}$$

Combining equations 3.3.9 and 3.3.10 yields

$$\delta_j^l = \sum_{p=1}^{n_{l+1}} \delta_p^{l+1} w_{jp}^{l+1} \phi_l'(a_j^l). \tag{3.3.11}$$

Thus knowing the error δ^{l+1} allows one to calculate the partial derivatives of E with respect to the weights in layer l very easily. All that is left is dealing with the output layer k. The error of neuron i of layer k is

$$\delta_i^k = \frac{\partial L(z_i^k, y)}{\partial a_{ij}^k}, \tag{3.3.12}$$

which needs to be calculated for the loss function L.

Because the requirement for the loss function L was that the total loss for a data set can be expressed as a sum of single sample losses, the differentials of the total error for a data set can be calculated as the sum of single samples. Having an algorithm for computing the gradient it is now possible to use the gradient-based optimization algorithms presented in Chapter 2.5 to minimize the error.

4 Analysis

In this chapter I present my attempts at answering the research questions posed in Chapter 1.2. The chapter is split into 3 parts; feature selection which was similar for both the regression and the classification tasks, classification models for the best response and regression models for the drug duration. A summary of the results is provided in Chapter 5.

4.1 Preprocessing the data

The data set I received had no missing values. However some continuous features such as weight and height had value 0 signifying either that the value had been missing but automatically filled or had not been copied correctly. Features where a value of 0 signified an error were identified and handled by replacing the value with the mean of the feature. No categorical features with missing values were present but had there been any the missing value would have been replaced by the mode.

From Figure 1.1 the encoded response variable was judged to have too many classes. The number of classes was dropped from 10 to 5 using encoding shown in Table A.3. The resulting distribution of encoded responses is shown in Figure 4.1.

Simple feature engineering, constructing new features from existing data, was performed. A feature *best future response* was created as the target variable for the classification task. For each patient and each treatment line the best response found on a future date was set as the value. A new category was added for cases where there were no more future visits. The encoding can be found in Table A.4. A similar feature was created for the best result the patient had reached in the past as this could help the models see if the patient's medical trajectory is declining. The *No information* class corresponds to either in the case of future responses the last visit or in the case of past responses the first visit.

A feature for *drug duration* had to be created. A treatment line consisted of periods during which a patient was given a combination of drugs. Each of these periods were documented with start and end dates and drugs given to the patient. The treatment lines changed over time with some drugs being dropped or replaced with other drugs. The starting day of a drug was defined as the start day of the first period in a treatment line where the drug was given to the patient. Similarly the last day of a drug was defined as the ending day of the last period of a treatment line where the drug was given to the patient. The drug duration was then defined as the number of days between the start and the end dates of a drug.

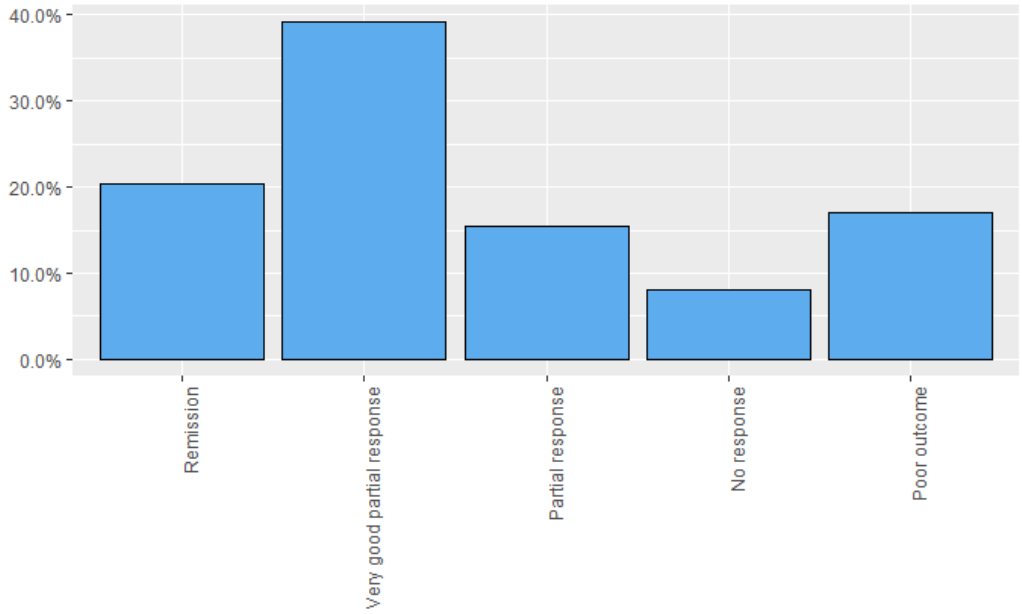

Figure 4.1: Distribution of the new encoded responses

The given definition for drug duration is not perfect. Sometimes the periods were not consecutive but had breaks between them, meaning that the drug was not given to the patient continuously during the whole duration of the drug. Other potential problems can also be imagined. Consider a drug that causes very serious side effects but is also very effective. Such drug might be dropped from a treatment line at an early stage due to unbearable side effects but later used again if the patient does not respond to other drugs. In such a case the drug duration would seem much longer than it was in reality. Other definitions were also considered, such as summing the number of days a drug was given to a patient. The other potential definitions had also their own downsides. Summing the number of days the patient was on the drug would have broken the connection between the day the observations were recorded and the drug duration. Even with its imperfections the chosen definition was judged to be the most suitable for this task.

The total number of different drugs that patients could receive was 41. As mentioned in Chapter 1.3 and shown in Figure 1.5 many drugs were given to only a small number of patients and modeling their duration would have been difficult. Drugs with less than 100 patients were left out of the study.

The two questions this study attempts to answer are very different and the data the models should and should not have access to differs on both

tasks. Hence both research questions used their own copies of the data set.

For both data sets features containing data that the practitioner could not have been able to know at the time of the observations were left out. To prevent any data leaks and to protect the integrity of the study feature selection on features lacking proper documentation erred on the side of caution.

For both data sets 20% of the patients were set aside for a test set. For the classification task stratified sampling was used to build a test set with the distribution of the dependent variable matching that of the full data set. Patients were divided into classes based on the best response they reached during any time of their treatment and from each class 20% were randomly sampled into the test set.

The test set was kept completely separate from the training process. When features were standardized the mean and the standard deviation of the training set were used for standardizing the test set.

4.2 Feature selection

The data set consisted of 236 features, many of them either correlated with other features or simply irrelevant. The workflow described in this chapter was applied independently to both the regression task and the classification task, resulting in 4 data sets for each of them. Table A.5 shows which data set was used with which algorithm.

As the goal of the regression task was to predict the time a patient can remain on a given drug, the drug the patient was receiving had to be in the feature set. To achieve this when performing feature selection in the regression task the drug feature was left out of the feature selection procedure and added later to the set of features.

Neural networks, bagging trees and random forests are able to handle large amounts of features and thus data sets with no feature selection were prepared. Neural networks require categorical features to be one-hot encoded while tree-based models could theoretically handle either numerical or one-hot encoding. As mentioned in Chapter 3.1.1 one-hot encoding could in some cases be counterproductive with tree-based models and thus a data set with numeric encoding was also prepared.

Similarly two data sets with features selected using Algorithm 3.1.3 were prepared; one with numeric encoding for categorical variables to be used only with tree-based models and one with one-hot encoding that could be used with any model. The algorithm was run with $n = 2500$ trees and the number of features considered at each split was the square root of the total number of features. The features were ranked by their importance and a number of the highest ranked features were selected. For the numerically encoded data set

the number of features selected at this stage was 30, for the one-hot encoded version it was lowered to 25 as the drug feature was one-hot encoded and added several new features. This approach lead to both data sets containing approximately the same number of features.

The remaining features were still potentially correlated. To get a set of independent features the following algorithm was applied:

Algorithm 4.2.1. *(Independent feature filtering) Let* $X^1, \dots X^k$ *be features ranked in order of importance,* $t > 0$ *a threshold and* K *a set of independent features.*
(1) *Initialize* $K = \emptyset$.
(2) *In the order of importance from the highest to the lowest, for each feature* X^j*:*
(3.1) *Calculate the correlations* $c_1, \dots, c_n$ *between* X^j *and each feature* $X^{K_0}, \dots, X^{K_n} \in K$.
(3.2) *If* $max(|c_1|, \dots, |c_n|) < t$*) add* X^j *to* K*, otherwise reject it.*

The threshold t was set to $t = 0.7$ which is commonly interpreted as the lower bound for strong correlation.

4.3 Models and hyperparameter optimization workflow

For the sake of simplicity models for both tasks were quite similar as were the hyperparameter sets considered.

For tree ensembles the number of trees was tested for the range $[5, 250]$. The maximum depth of the trees was tested with values $3, 6, 9$ and with unlimited depth. Random forests were applied to the data set with no feature selection while bagging trees were tested with both the feature selected and the raw data set. The number of features random forests sampled at each split was the square root of the total features in the data set. The number of samples in the bootstraps of the tree ensembles was the number of samples in the training data set.

A manual search for the best neural network architecture was performed. The architectures included 3 different neuron configurations for the hidden layers; 512-512, 32-32-32 and 256-256-64. All neuron configurations were tested with both ReLU and sigmoid activation functions. For the regression task the final layer had one neuron with a linear activation function while in classification the number of neurons corresponded to the classes of the dependent variable and the activation function was softmax. The fold designated as the test set was set as the validation set so that the performance of the network could be easily tracked over the learning process.

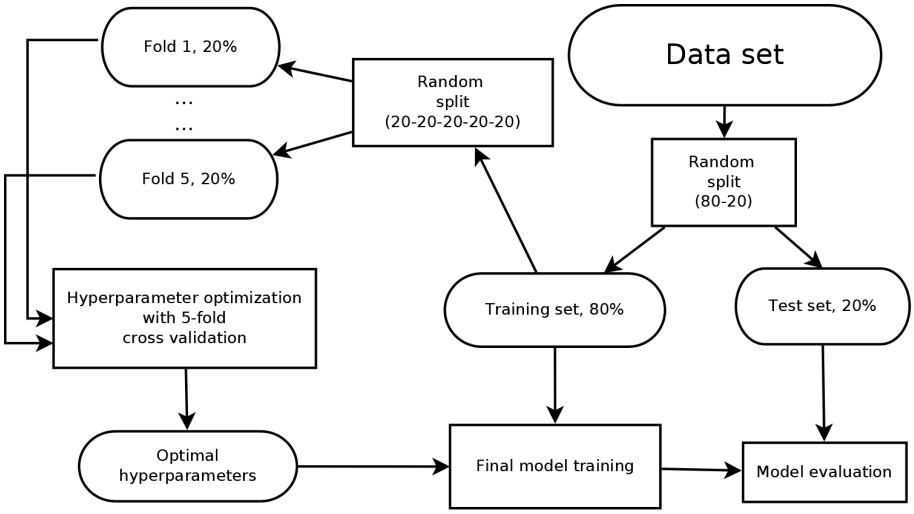

Figure 4.2: Model selection workflow

For the classification task logistic regression was tested with both l_1 and l_2 regularization with the value of λ being optimized. For the regression task kernel ridge regression was tested with 2 kernels with 4 kernel-related values each; polynomial with degrees $1, \ldots, 4$ and RBF with 4 different values from the range $(0, 1]$ for σ. For each configuration the value of λ was again varied.

The optimization workflow was straightforward and simple. The training sets were separated randomly into 5 subsets by randomly sampling patients into a subset. For the classification task stratified sampling was used to keep the distribution of the dependent variable the same over each subset. For each set of hyperparameters the performance of the resulting model was validated with a 5-fold cross validation described in Algorithm 2.8.1 using previously described subsets as the 5 folds. The best performing hyperparameters were selected and used to build the final models. The workflow is described in Figure 4.2.

The hyperparameter optimization could have been more efficient and covered a wider area had random search been used instead of the manual and grid searches performed[38]. This was however deemed computationally too expensive to fit within the scope of this work.

4.4 Predicting the best response

Unlike in the regression task including the drug the patient was receiving was not a necessity. This meant that if feature selection did not choose the drug the patient was on to be a good predictor at least the feature selected

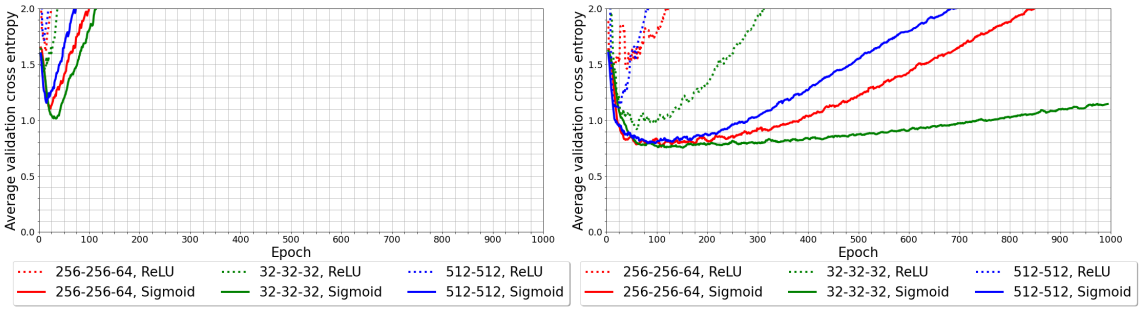

Figure 4.3: Optimizing the neural network architecture for the classification task. Networks on the left used all features while networks on the right used the feature selected data set.

data set would contain duplicate entries as the differing column, the drug, would be removed. Three options were considered; removing the drug from the features and then removing duplicates, removing duplicates if the drug the patient was on was not selected as a good predictor or simply ignoring the issue and allowing the duplicates to remain.

Each option had its own problems. Removing the treatment the patient was judged to lead to potentially losing useful information and was rejected. Removing duplicates from the feature selected set would have lead to an imbalance between the feature selected set and the raw set. Since the plan was to use both of the data sets in this task this was judged to lead to potential problems and left out. The simplest option of allowing the duplicates to remain was chosen.

The best possible option would have been to one-hot encode the drug feature and to then combine each visit to one entry such that features corresponding to drugs the patient was getting would be given a value of 1 and ones corresponding to treatments not received by the patient a value of 0. Sadly time restrictions did not permit this option.

Logistic regression was solved using a solver based on stochastic gradient descent. The default maximum iteration count was 100 but it was increased to 2500 in an attempt to ensure convergence. The stopping criteria tolerance was the default value of 10^{-4}. The model failed to converge with all values of λ with both l_1 and l_2 regularization. All values of λ lead to very similar, poor performance with both l_1 and l_2 regularization. This suggests that the relationship between the log-odds of the classes and the independent variables was not linear. The default model was chosen with l_2 regularization and $\lambda = 1$.

Neural networks shown in Figure 4.3 showed significant difference in their

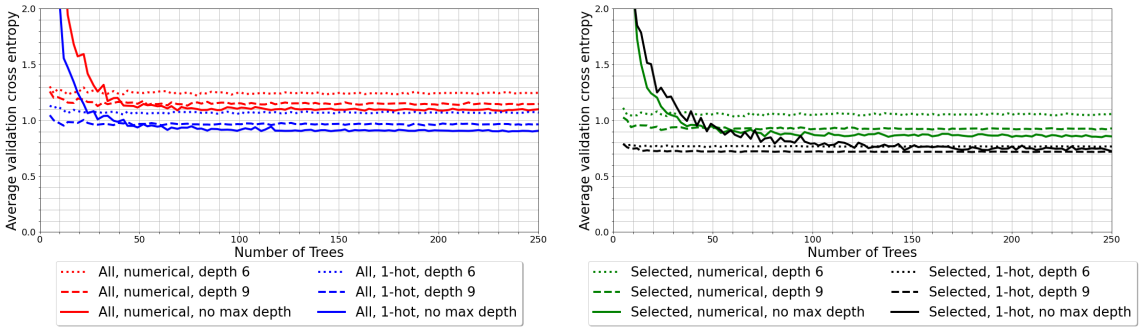

Figure 4.4: Optimizing the hyperparameters of the bagging tree model for the classification task. The feature sets, categorical variable encoding, maximum depth and number of trees were varied.

performance. The networks using all features began overfitting to the training set almost immediately. This could have been caused by the full feature set containing genomic data and the network having learned very quickly to identify individuals in the training set instead of patterns applicable to all patients. With the feature selected versions the sigmoid networks performed better than ReLU ones. All of the sigmoid networks reached similar levels in performance but the 32-32-32 network was the slowest to overfit. Overfitting slowly is a significant upside as the early stopping criterion would otherwise have to be very strict and risk random noise stopping the learning process with potential gains remaining or with loose criterion risk fast overfitting and a loss in performance.

Bagging trees shown in Figure 4.4 and random forests in Figure 4.5 performed better with one-hot encoded data. Both bagging trees and random forests performed poorly with maximum depth 3 and those models were left out from the plots. Most of the models reached stable performance when the number of trees was close to 100 with one-hot encoded bagging trees using selected features and no maximum depth as the exception requiring approximately 150 trees.

Bagging trees outperformed random forests and the best performance was reached with feature selected data set using one-hot encoded features and either a maximum depth of 9 or with unbound depth. A model using a maximum depth of 9 is significantly simpler than one with unbound depth and as such was chosen as the better model.

The final accuracy of the models was analyzed on 6 measurements. The cross entropy and the accuracy values for the full data set are provided. However further analysis can be done by considering subsets of the test set.

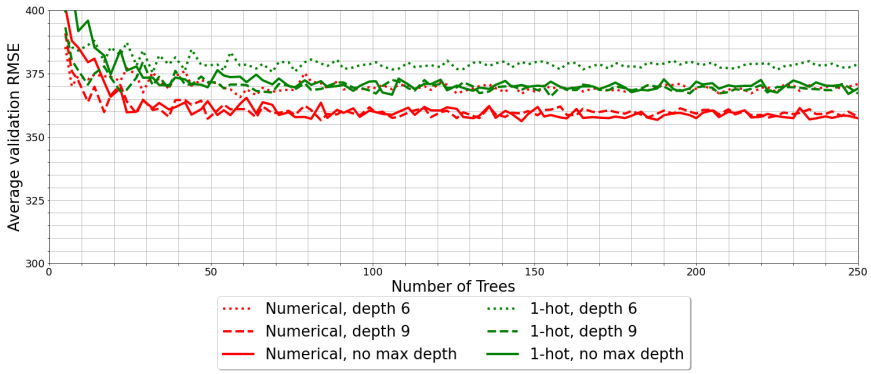

Figure 4.5: Optimizing the hyperparameters of the random forest model for the classification task. The categorical variable encoding, maximum depth and number of trees were varied.

As cross entropy is not easily interpretable only accuracy value is displayed for the subsets.

Cases where there were no future visits had the future best response *no information*. These cases are not interesting as there could be too many reasons for the visits to stop varying from the death of the patient to the visit day having been close to the end of the study. A test set of *feasible* cases was created by removing cases where true value of the best future response was *no information*.

Cases that were *feasible* and where there was a change in the response, i.e. the current encoded response did not match the best future response, were considered *unstable*. Cases that were *feasible* but not *unstable* were *stable*.

Finally seeing the prediction accuracy from the first visit was of interest. For each patient found in the *feasible* set data from their first visit was searched. These cases formed the *first visit* set.

Table A.6 shows the number of rows in each of the 5 test sets. Table 4.1 shows the performance of the models on each of the 6 metrics. The logistic regression performed the worst in all tests. On the full test set cross entropy the neural network outperformed the bagging trees. However in every other test the bagging trees performed better and were considered in general the superior model for this task.

From the performance table it can be seen that the bagging tree classifier identifies stable patients who will see no change in the response to the treatment with high accuracy but performs poorly with unstable patients. Even though the model does not classify patients with unstable trajectories correctly, it is possible to attempt to predict a simplified trajectory. New groups,

Metric/Alg	DNN	Bagging trees	Logistic regression
Cross entropy (full)	**0.79**	0.75	1.10
Accuracy (full)	0.68	**0.69**	0.65
Feasible	0.83	**0.87**	0.73
Unstable	0.22	**0.29**	0.19
Stable	0.94	**0.98**	0.82
First	0.80	**0.83**	0.72

Table 4.1: Classification model performance. Best result is bolded.

	$\text{Improving}_{\text{pred}}$	$\text{Stable}_{\text{pred}}$	$\text{Worsening}_{\text{pred}}$	Total
Improving	166	28	23	217
Worsening	0	83	10	93
Total	166	111	33	310

Table 4.2: Direction of the predicted change vs real direction on the *unstable* set

improving, *stable* and *worsening*, are defined by comparing the best future response to the current response. Looking closer at the *unstable* group Table 4.2 shows that while only 29% were initially labeled correctly, in 62% of the cases the model did predict an upcoming instability in the drug response, i.e. the predicted future best response was not the same as the current response. Of the cases where a better response was reached 76% were identified while of the cases where the patient took a turn for the worse only 5% were identified correctly with most of the patients on a downward trajectory being predicted to remain stable. The total accuracy with simplified labels on the *unstable* set was 0.57.

A confusion matrix with simplified classes of the full feasible set is provided in Table A.7. The accuracy with simplified labels for the full *feasible* set is 0.92.

While the initial model was not very accurate simplifying the model to predict only stability or direction of change improved the accuracy significantly.

4.5 Predicting the drug duration

Out of the ridge regression models the linear polynomial performed the best with *rmse* value in the $[340, 350]$ range. No significant performance difference was observed with different values of λ. Of the non-linear versions degree 2 managed to get to *rmse* 400 with high values of λ, higher degrees had much

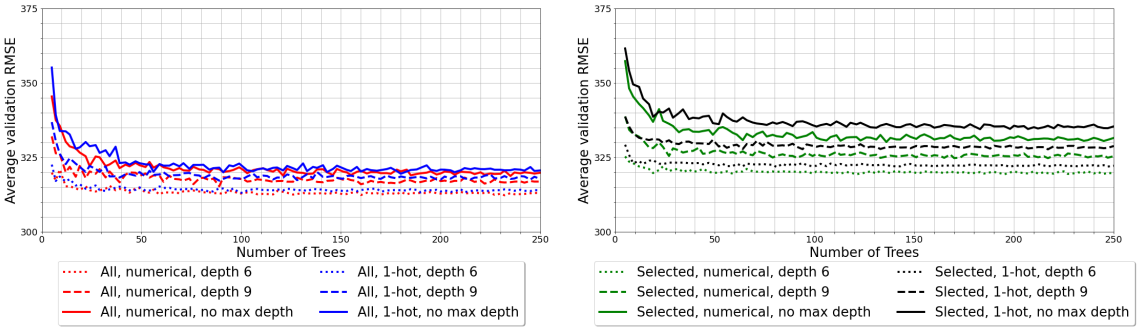

Figure 4.6: Optimizing the hyperparameters of the bagging tree model for the regression task. The feature sets, categorical variable encoding, maximum depth and number of trees varied

worse performance with *rmse* values well above 800.

Bagging trees performance is shown in Figure 4.6. The model performed the best when allowed to use all features in the data set. Both random forests and bagging trees using maximum depth 3 performed poorly and were left out of the plots. With bagging trees maximum depth of 6 lead to best performance with higher depths generalizing poorly. Random forests in Figure 4.7 on the other hand performed best with either maximum depth of 9 or with no maximum.

For both bagging trees and random forests one-hot encoded versions performed in general worse than their numerically encoded counterparts. The tree count required to stabilize the performance varied but in general no large deviations were seen with values above 100. Out of all of the tree-based models the bagging trees using numerical encoding for categorical variables, all features and a maximum depth of 6 performed the best.

Similar to Chapter 4.4 neural networks using all features in the data set began overfitting to the data set almost immediately as can be seen from Figure 4.8. Feature selection lead to better performance and the $32-32-32$ networks seem to have learned the actual underlying patterns and not just overfit into the training set as the *rmse* can be seen decreasing for some time.

The ReLU network begins overfitting earlier but performs significantly better than the sigmoid network. The $32-32-32$ architecture with ReLU activation and selected features is chosen as the best performing model, but early stopping using a validation set is clearly necessary.

The best performing models were trained on the full data. 20% of the patients in the training set were sampled as the validation set for the neural network and the tolerance for iterations with no improvement set to 25.

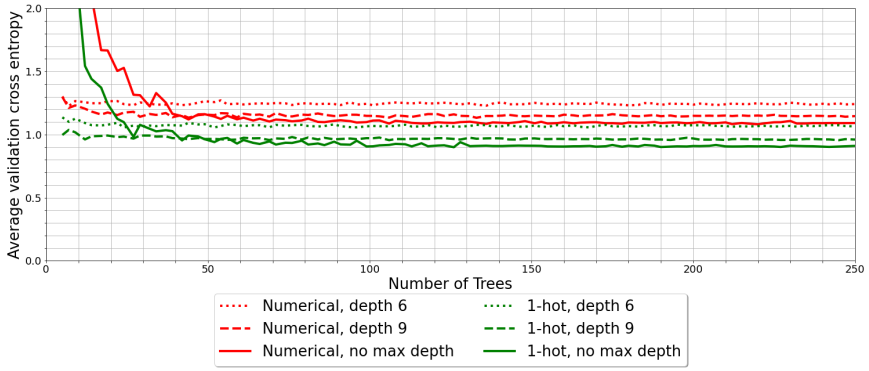

Figure 4.7: Optimizing the hyperparameters of the random forest model for the regression task. The categorical variable encoding, maximum depth and number of trees were varied.

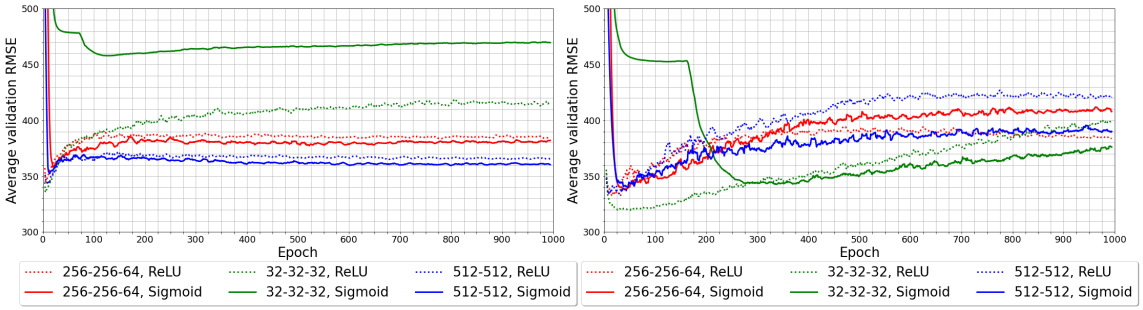

Figure 4.8: Optimizing the neural network architecture for the regression task. Networks on the left used all features while networks on the right used the feature selected data set.

Algorithm	mae	$rmse$	R^2
DNN	**234.52**	313.54	0.50
Bagging trees	236.76	**310.89**	**0.51**
Linear Ridge	249.30	330.31	0.45
Poly Ridge	249.30	330.31	0.45
RBF Ridge	339.52	452.05	-0.03

Table 4.3: Final regression models with the best value bolded

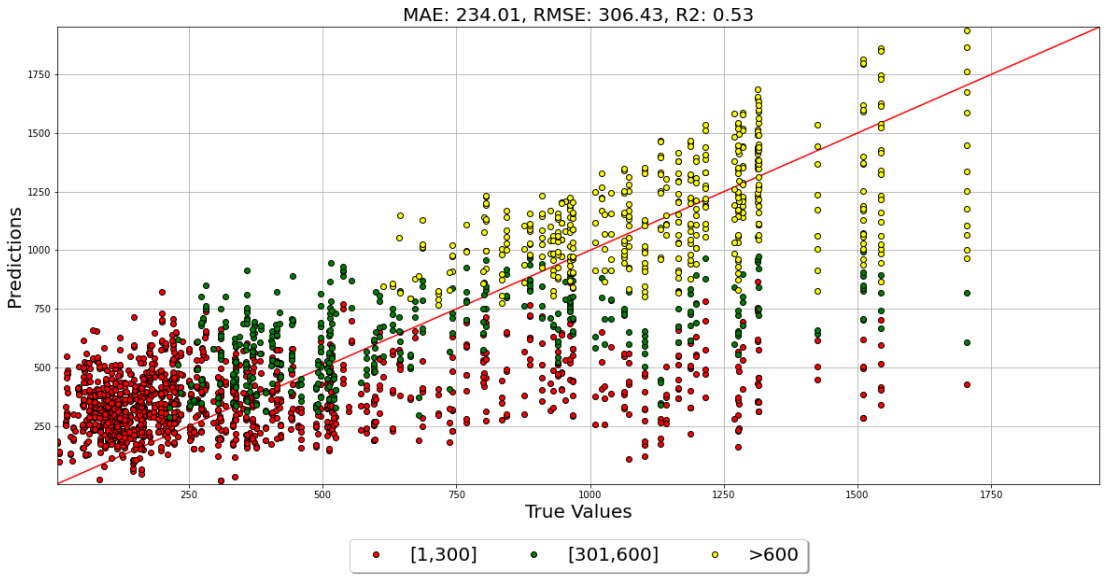

Figure 4.9: DNN predicting drug duration on the test set

The performance comparison can be seen in Table 4.3. Bagging trees and neural networks performed the best, both reaching $R^2 \geq 0.5$. While the results are not as good as one would have initially hoped, they can be seen as somewhat promising and deserve a closer look.

Figures 4.9 and 4.10 show the predicted drug duration with the neural network and bagging trees. To gain a better understanding of the performance points were color coded to show the day the patient visited the facilities, counting from the first day the patient received the drug. It seems like in both cases the 3 colored groups form bands. The red markers corresponding to visits dates ≤ 300 have their predicted drug durations largerly ≤ 500.

While the initial R^2 values seemed promising it seems like the models do perform poorly. The model can be changed to approximate the number of days remaining on the current treatment by subtracting the visit day from the predicted drug duration. This is not perfectly accurate due to the problems in defining drug duration discussed earlier. It is possible to see that in Figure 4.9 there are some green markers with true values < 300 which would mean a negative duration. In reality this means that the drug was started later in the treatment line. This same issue exists when subtracting the visit day from the true drug duration. Thus the change to approximated remaining time on a drug results in a plot that, while not a perfect representation of time remaining on the drug, is internally consistent.

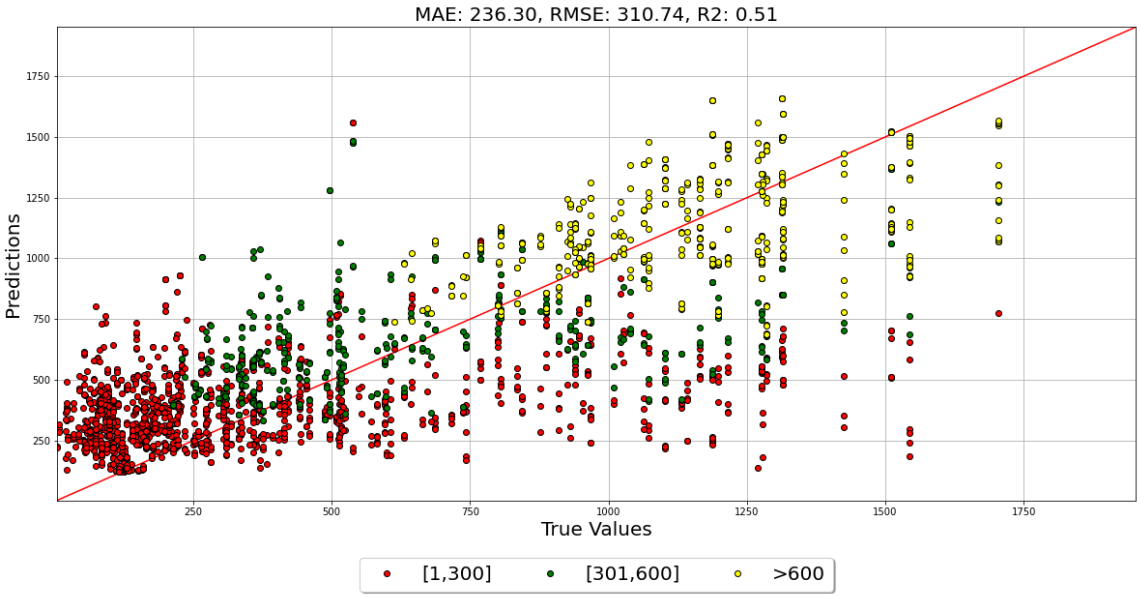

Figure 4.10: Bagging trees predicting drug duration on the test set

The performed on the predicted drug duration by bagging trees drug yields Figure 4.11. As both DNN and bagging tree models showed the same behavior when predicting performing the same analysis on DNN is left out.

The plot shows that the predictions are indeed inaccurate and the model predicts time remaining on the drug to be mostly ≤ 500 regardless of the visit day or the true approximated remaining time. The R^2 value is barely above 0 and the model works poorly. In conclusion even the best model built for Question 1.2.2 has poor accuracy and negative results are reported.

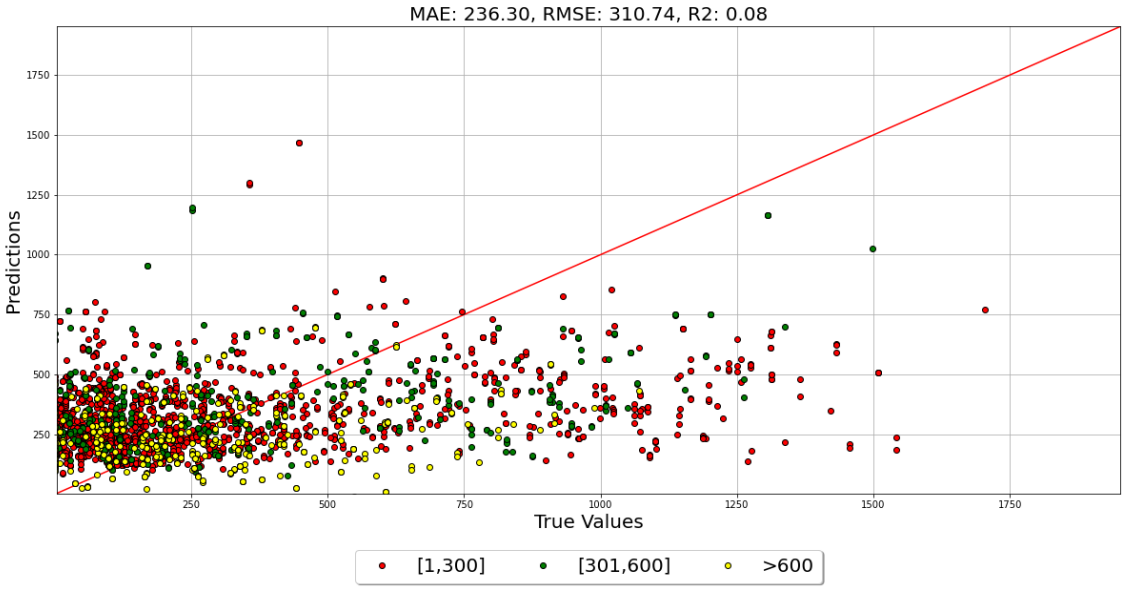

Figure 4.11: Bagging tree predictions approximating time left on a drug

5 Discussion

5.1 Machine learning in personalized medicine

Machine learning has found several applications in the medical field where it can compete human practitioners in several areas. Models have been reported to outperform human experts on tasks such as cancer screening, cardiac disease diagnosis and early diagnosis of Alzheimer's disease [39]. However areas like diagnosing neurodevelopmental disorders have seen less progress [40] and offer an interesting area for research.

Machine learning models can support the human practitioners in tasks that are difficult for humans. In areas like the aforementioned medical image recognition machine learning models can learn to detect patterns from details too small or too complex for a human to detect. The ability to process data on scales infeasible for humans allows models to find new patterns that help diagnose and treat various diseases.

Machine learning in personalized medicine can serve many goals. Repurposing old drugs can save resources in drug development, allowing for the resources to be directed at other yet unsolved problems. Many cancer drugs are currently used in all applicable cases. Being able to predict the response of a patient to a given drug benefits both the patient and the society. Cancer often requires quick responses and not wasting any time on drugs that either will not work or that will cause unbearable side effects helps the survival of the patient. Cancer drugs are often extremely expensive and saving money by not wasting drugs on patients where there will be no positive response is a massive benefit for the society. Predicting both the adverse effects and the efficacy of drugs could help tailor treatment plans matching patients' wishes on balancing expected survival and quality of life. Machine learning models analyzing genetic data gives rise to potentially tailoring treatments for individual patients based on their genotype. Accurately predicting risks of having various diseases from a genotype can personalize screening programs. This would help focus the limited resources more efficiently by not screening low-risk patients too often and simultaneously creating more efficient programs for high-risk patients leading to early intervention and hopefully better outcomes.

One of the bottlenecks for progress in the field is the limited amount of data. Many data sets are private and not freely available to researchers. Many non-profit organizations like the Multiple Myeloma Research Foundation are vital in the data gathering process.

A challenge with diseases with a slow onset and progression is that the data collection portion can last prohibitively long. The CoMMpass study has

been tracking patients for over 8 years and is still going on. This challenge is obvious with models attempting to diagnose diseases early and to provide the patient with an early prognosis as they require data sets that are extremely difficult to gather. This can in part explain why most of the progress is found in fields like medical image detection, the necessary data sets are much easier to gather.

Choosing whether to participate in research is not an easy choice for the patient either. Most likely there is no benefit for the patient as the improved treatments, drugs and techniques will become available to future patients. Participating in data gathering like this is an act of kindness and charity for future generations. While one might say that there is no downside in participating in studies and every patient should make their data available, everyone has a right to privacy. Potential downsides also exist. While researchers handle the data with care and remove identifying information, as more and more data of a patient is gathered identifying them from the data becomes a theoretical possibility. This is especially true with genetic information. Additional medical tests that have no impact on the treatment might also be performed on the patient, causing discomfort and taking time away from the patients potentially limited remaining lifetime.

5.2 Summary of the results

This thesis seeked to answer two research questions posed in Chapter 1.2. Question 1.2.1 asked if it was possible to build a model predicting the best response a patient would reach during their treatment to understand the medical trajectory of the patient. Question 1.2.2 asked if it was possible to predict how long a patient can stay on a given drug to potentially help practitioners choose personalized treatment options for patients.

Answering Question 1.2.1 is complicated. While a model predicting the best response a patient will reach in the future was built, the accuracy on patients with change from their current response was quite low. As the latter part of Chapter 4.4 shows when the categories were simplified into 3 possible outcomes, *improving*, *stable* and *worsening* when comparing to the current response, the accuracy improved to a level that can be seen as somewhat promising.

No good model for Question 1.2.2 was found. The accuracy of even the best model was poor. Chapter 5.3 discusses possible improvements to the workflow to potentially build a better model.

5.3 Further work

This work was limited in scope but it could be expanded. While the results of the classification task were promising space for improvement remains. No good model was found for the regression task, but improvements on the methodology could lead to finding a model giving accurate predictions.

Only one feature set per task was tested but varying the number of features selected could have lead to better performing models. Only one feature selection method was tested but other methods could have selected better features. Other related approaches like principal component analysis could have helped squeeze the information contained in the data set into a lower-dimensional space, leading to more information contained in each selected feature. The hyperparameter optimization scheme was limited and with more computing power could be improved.

Only a limited number of models were tested. With neural networks only a handful of architectures from the infinite number of possibilities were considered. Other activation functions could have been tested. The only regularization applied to neural networks was early stopping but other regularization approaches could have helped the network learn for a longer time without overfitting hopefully leading to improved performance. Other models such as gradient boosting and support vector machines could have been tested. Gaining a better understanding of the data set and the disease could help in feature engineering new features from the already existing data, leading to possible improvements in performance.

In the classification task the classes contained an unbalanced number of patients. Better balanced data set could have lead to improved performance. A better balance could have been achieved by undersampling the largest classes or by using algorithms like SMOTE [41] to create synthetic samples for the under-represented classes. The duplicate entry issue mentioned in 4.4 could be solved by combining each visit to one row leading to more accurately interpretable results.

The simplified classification model showed potential and redesigning the experiment with more modest goals with insight gained here could lead to a model accurate enough for practical use. Uncertainty with time spans could be an issue. A patient knowing that their medical trajectory will change in the future would benefit from knowing more accurately when this will happen. Thus a possible modification would be to limit the time span and to build separate models predicting changes in trajectory at as an example 1 and 3 years into the future.

Some research based on the CoMMpass data set has already been published. Some of the research has been focusing on the genomic data on

questions like choosing the best drug based on genetic expressions [42] and studying the link between chromosomal translocations, MM diagnosis and prognosis[43]. The CoMMpass study is still continuing with the data set growing every year. Hopefully in the future the data set can help answer several open questions related to MM. Open questions already mentioned in this thesis include estimating the treatment tolerance of patients and understanding the causes of the heterogeneity in treatment results. As MM currently has no permanent treatment and will in the end most likely lead to the death of the patient research focusing on improving the quality of the remaining lifetime could be of interest.

6 Bibliography

References

[1] R. A. Kyle, T. M. Therneau, S. V. Rajkumar, J. R. Offord, D. R. Larson, M. F. Plevak, and L. J. Melton, "A long-term study of prognosis in monoclonal gammopathy of undetermined significance," *New England Journal of Medicine*, vol. 346, no. 8, pp. 564–569, 2002, pMID: 11856795. [Online]. Available: https://doi.org/10.1056/NEJMoa01133202

[2] S. V. Rajkumar, "Multiple myeloma: 2018 update on diagnosis, risk-stratification, and management," *American Journal of Hematology*, vol. 93, no. 8, pp. 1091–1110, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ajh.25117

[3] R. A. Kyle, M. A. Gertz, T. E. Witzig, J. A. Lust, M. Q. Lacy, A. Dispenzieri, R. Fonseca, S. V. Rajkumar, J. R. Offord, D. R. Larson, M. E. Plevak, T. M. Therneau, and P. R. Greipp, "Review of 1027 patients with newly diagnosed multiple myeloma," *Mayo Clinic Proceedings*, vol. 78, no. 1, pp. 21 – 33, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0025619611618272

[4] R. Fonseca, P. L. Bergsagel, J. Drach, J. Shaughnessy, N. Gutierrez, A. K. Stewart, G. Morgan, B. Van Ness, M. Chesi, S. Minvielle *et al.*, "International myeloma working group molecular classification of multiple myeloma: spotlight review," *Leukemia*, vol. 23, no. 12, pp. 2210–2221, 2009.

[5] C. Ramsenthaler, P. Kane, W. Gao, R. J. Siegert, P. M. Edmonds, S. A. Schey, and I. J. Higginson, "Prevalence of symptoms in patients with multiple myeloma: a systematic review and meta-analysis," *European journal of haematology*, vol. 97, no. 5, pp. 416–429, 2016.

[6] C. Röllig, S. Knop, and M. Bornhäuser, "Multiple myeloma," *The Lancet*, vol. 385, no. 9983, pp. 2197 – 2208, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140673614604931

[7] S. V. Rajkumar, "Treatment of multiple myeloma," *Nature reviews Clinical oncology*, vol. 8, no. 8, p. 479, 2011.

[8] T. O. Ayodele, "Types of machine learning algorithms," *New advances in machine learning*, vol. 3, pp. 19–48, 2010.

[9] V. Vapnik, "Principles of risk minimization for learning theory," in *Advances in neural information processing systems*, 1992, pp. 831–838.

[10] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[11] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[12] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[13] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[14] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998.

[15] D. Harrison Jr and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," 1978.

[16] D. Krstajic, L. J. Buturovic, D. E. Leahy, and S. Thomas, "Cross-validation pitfalls when selecting and assessing regression and classification models," *Journal of cheminformatics*, vol. 6, no. 1, pp. 1–15, 2014.

[17] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers-a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005.

[18] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[19] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. [Online]. Available: https://www.R-project.org

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[22] J. R. Quinlan, "Simplifying decision trees," *International journal of man-machine studies*, vol. 27, no. 3, pp. 221–234, 1987.

[23] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[24] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[25] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[26] L. N. Sanchez-Pinto, L. R. Venable, J. Fahrenbach, and M. M. Churpek, "Comparison of variable selection methods for clinical predictive modeling," *International journal of medical informatics*, vol. 116, pp. 10–17, 2018.

[27] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable selection using random forests," *Pattern recognition letters*, vol. 31, no. 14, pp. 2225–2236, 2010.

[28] M. A. Poole and P. N. O'Farrell, "The assumptions of the linear regression model," *Transactions of the Institute of British Geographers*, pp. 145–158, 1971.

[29] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 116.

[30] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[31] I. J. Myung, "Tutorial on maximum likelihood estimation," *Journal of mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003.

[32] F. C. Pampel, *Logistic regression: A primer*. Sage, 2000.

[33] C. Mishra and D. Gupta, "Deep machine learning and neural networks: An overview," *IAES International Journal of Artificial Intelligence*, vol. 6, no. 2, p. 66, 2017.

[34] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[35] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning*, 2019, pp. 242–252.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[38] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[39] G. Z. Papadakis, A. H. Karantanas, M. Tsiknakis, A. Tsatsakis, D. A. Spandidos, and K. Marias, "Deep learning opens new horizons in personalized medicine," *Biomedical reports*, vol. 10, no. 4, pp. 215–217, 2019.

[40] M. Uddin, Y. Wang, and M. Woodbury-Smith, "Artificial intelligence for precision medicine in neurodevelopmental disorders," *npj Digital Medicine*, vol. 2, no. 1, pp. 1–10, 2019.

[41] J. Wang, M. Xu, H. Wang, and J. Zhang, "Classification of imbalanced data by using the smote algorithm and locally linear embedding," in *2006 8th international Conference on Signal Processing*, vol. 3. IEEE, 2006.

[42] J. Ubels, P. Sonneveld, E. H. van Beers, A. Broijl, M. H. van Vliet, and J. de Ridder, "Predicting treatment benefit in multiple myeloma through simulation of alternative treatment effects," *Nature communications*, vol. 9, no. 1, pp. 1–10, 2018.

[43] B. G. Barwick, P. Neri, N. J. Bahlis, A. K. Nooka, M. V. Dhodapkar, D. L. Jaye, C. C. Hofmeister, J. L. Kaufman, V. A. Gupta, D. Auclair *et al.*, "Multiple myeloma immunoglobulin lambda translocations portend poor prognosis," *Nature communications*, vol. 10, no. 1, pp. 1–13, 2019.

Appendix

A Tables

Kernel name	$k(\mathbf{x}, \mathbf{y})$	Dimensions
Linear	$\mathbf{x}^T\mathbf{y}$	n
Polynomial	$(\mathbf{x}^T\mathbf{y} + c)^d$	$\binom{n+d}{d}$
Gaussian	$exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$	∞
Laplacian	$exp(-\frac{\|\mathbf{x}-\mathbf{y}\|}{\sigma})$	∞

Table A.1: Kernels

Household income (10k EUR)	Owns Stocks
10	False
14	False
19	False
24	False
28	False
33	False
38	False
43	True
47	False
52	True
57	True
62	True
66	True
71	True
76	False
81	True
85	True
90	True
95	True
100	True

Table A.2: An imaginary data set for household income and stock ownership

Encoding	Original explanation	New encoding	New explanation
Y_0	Complete remission	Y_0	Remission
Y_1	Near complete remission	Y_0	Remission
Y_2	Very good partial response	Y_1	Very good partial response
Y_3	Partial response	Y_2	Partial response
Y_4	Stable disease	Y_3	No response
Y_5	Progressive disease	Y_3	No response
Y_6	Lack of response	Y_4	Poor outcome
Y_7	Adverse event or co-morbidity	Y_4	Poor outcome
Y_8	Disease progression or relapse	Y_4	Poor outcome
Y_9	Completed regimen	Y_4	Poor outcome

Table A.3: Old and new encodings of the treatment response

Encoded value	Explanation
Y_0	Remission
Y_1	Very good partial response
Y_2	Partial response
Y_3	No response
Y_4	Poor outcome
Y_5	No information

Table A.4: Encoding of the best future/past response

Algorithm	All, numeric	All, one-hot	Selected, numeric	Selected, one-hot
Neural networks		✓		✓
Random forest	✓	✓		
Bagging trees	✓	✓	✓	✓
Logistic regression				✓
Kernel ridge				✓

Table A.5: Data sets, indicated by feature sets and encoding of categorical variables, and algorithms applied to them

Full	Feasible	Unstable	Stable	First visit
2726	2044	310	1734	428

Table A.6: Number of cases in each test set

	$\text{Improving}_{\text{pred}}$	$\text{Stable}_{\text{pred}}$	$\text{Worsening}_{\text{pred}}$	Total
Improving	166	28	23	217
Stable	15	1697	22	1734
Worsening	0	83	10	93
Total	181	1808	55	2044

Table A.7: Direction of the predicted change vs real direction on the *feasible* set