Acute Myeloid Leukemia Risk Group Prediction from Gene Expression Data with Feed-Forward Neural Networks

Master's Thesis in Data Science
Heli Leskelä
Faculty of Science
University of Oulu
Summer 2022

Tiivistelmä

Geeniekpressiodatalle on tyypillistä, että muuttujia on kerätty kymmeniä tuhansia, kun taas havaintoja on vain muutama sata. Tämän takia luokkien ennustaminen geeniekpressioista on monimutkainen tehtävä, jota vaikeuttaa epätasapaino enemmistö- ja vähemmistöluokkien välillä. Epätasapaino vaikeuttaa geenien välisten yhteyksien oppimista, ja kun luokkia on useampi, yleisesti käytetyt arviointimenetelmät piilottavat huonon luokittelukyvyn vähemmistöluokille. Näiden ongelmien lisäksi akuutti myelooinen leukemia (AML) tuo omat haasteensa potilaiden välillä olevien molekyylisten tekijöiden heterogeenisyyden vuoksi. Tämän seurauksena ennusteiden tekeminen ja hoitokeinojen suunnittelu geenien pohjalta on haastavaa.

Menetelmien valitseminen edellä mainittujen ongelmien ratkaisemiseksi riippuu suoraan käytettävissä olevista resursseista. Tämän työn tavoite on löytää kustannustehokkaat menetelmät datan epätasapainon korjaamiseen ja ylimääräisten muuttujien poistamiseen, sekä luoda useamman luokan luokittelija uudelle AML riskiryhmälle. Uusi riskiryhmä luodaan kahdesta muusta muuttujasta selviytymisaikojen perusteella. Yhteensä kuutta eri tilannetta tarkastellaan eteenpäinsyöttävillä neuroverkoilla. Ensin alkuperäistä AML geenieksprессiodataa käytetään ennustamaan riskiryhmä ilman aineiston esikäsittelyä. Tämän jälkeen aineiston epätasapaino korjataan simuloimalla vähemmistöluokalle uusia havaintoja käyttäen SMOTE- ja ADASYN-algoritmeja. Viimeiset kolme aineistoa saadaan pudottamalla muuttujia edellisistä aineistoista RFE-algoritmia hyödyntäen.

Eteenpäinsyöttävien neuroverkkojen optimaaliset hyperparametrien arvot haetaan 100:sta parmetrikombinaatiosta, jotka on valittu satunnaisesti noin 3 000:n kombinaation ryhmästä. Valittujen neuroverkkojen tuloksia verrataan kokonaistarkkuuden, sekä jokaisesta ryhmästä erikseen saatavan F_1-suureen perusteella. Parhaimpien mallien joukosta löytyi esikäsiteltyjen aineistojen lisäksi prosessoimattomia aineistoja, mikä viittaa siihen, että neuroverkkojen oikean rakenteen valitseminen on yhtä tärkeää kuin datan esikäsittely. Uuden riskiryhmän luokittelu antoi lupaavia tuloksia, joten ennustaminen pelkästään geenien pohjalta näyttäisi olevan mahdollista myös vähäisillä resursseilla.

Abstract

Predicting from gene expression data remains a complex task due to it characteristically having large dimensionality and small sample sizes. Creating classifiers in these settings is a non-trivial task, which is complicated by the presence of multi-class imbalance. The imbalance hinders the feed-forward neural network's ability to learn patterns from the data, and the multi-class structure makes common evaluation metrics hide the network's poor performance in the minority classes. For Acute Myeloid Leukemia (AML) these issues are magnified by the fact that the underlying molecular factors are heterogeneous from patient to patient, which makes treatment and prognosis difficult.

Having limited resources has a direct impact on which methods can be used to tackle these problems. In this thesis, the goal is to find cost-effective methods to balance the data, remove unnecessary features and to create a multi-class classifier for AML risk group. The risk group is created using two variables based on survival times. In total six scenarios are compared for creating the optimal feed-forward neural network. First, the original gene expressions are used as the predictors without any pre-processing. The following two scenarios fix the class imbalance using SMOTE and ADASYN. Finally, RFE is used to reduce dimensions in all previous scenarios to get the last three data sets.

The feed-forward neural network is tuned separately for each scenario. In total 100 parameter combinations are chosen randomly from around 3 000 possible model configurations, and the resulting models are evaluated based on overall accuracy and F_1-score for each class. The results show that while ADASYN, SMOTE, and RFE help the networks yield better results, having the right network structure is just as important. This is demonstrated by the fact that some models using the unprocessed data set were found among the best-performing models. Furthermore, based on high accuracy in classification, predicting the new AML risk category based only on genes seems possible even with limited resources.

Supervisors
Mikko Sillanpää
Valerio Izzi
Juho Kontio

Acknowledgements

I would like to thank my supervisors Mikko Sillanpää, Valeario Izzi and Juho Kontio for giving me such an interesting topic to work on. Special thanks to Juho Kontio, who gave me insightful feedback in the beginning of the writing process, and to Mikko Sillanpää whose advice helped me finalize the thesis.

Secondly, I want to give big thanks to Nicholas Larsen for proofreading the full thesis, and I also thank Jenni Harju, Sara Harris, Kerry Bernard and Konstantin Podleski for giving feedback on parts of the thesis during the writing process. Another big thanks to my parents and little sister for supporting me throughout my studies, and to my friends for all the fun activities outside of school. Lastly, huge thanks to AbdulRahman Alaedi, who has supported me every step of the way.

August 7, 2022
Heli Leskelä

Contents

1 Introduction

In order to develop reliable treatments for different cancers, the research needs to be done on the genetic level. One method is to analyse gene-expression data, which often has a large number of features compared to the observations. While simpler models fail to learn the complex patterns in the gene expressions, deep learning methods have the potential to uncover the underlying connections between the genes responsible for cancer. While this sounds promising, learning from imbalanced data with only few observations is a major issue, especially when the feature space is large. This thesis aims to address these problems by implementing sampling and feature selection methods for a multi-class classifier.

1.1 Motivation

Neural networks are a part of machine learning algorithms, which are used to make predictions using low-level information and signals (Theodoridis, 2020). Most machine learning algorithms are *shallow models*, meaning that they use only a couple of layers to find the optimal output (Bomheke and Greenwell, 2020). These kind of models function with a small number of features, but when the feature space becomes large, the optimal output might become impossible to find (Bomheke and Greenwell, 2020), and in these situations *deep learning* methods can be utilized.

As characterized by Bishop (2006), deep neural networks are a series of functional transformations, that utilize multiple layers for learning. They are able to give complex representations for high-dimensional data, as well as model non-linear relationships between variables (Bomheke and Greenwell, 2020). Artificial neural networks are able to approximate any well-behaving system to any level of precision, as long as the number of hidden layers and nodes can be increased (Baldi and Hatfield, 2002). According to the same authors, this has been proven for feed-forward neural networks.

Based on the characteristics of neural networks, they show clear promise for dealing with gene-expression data, where the relationships between different genes can be extremely complex. In case of AML the molecular features are extremely heterogeneous between patients, and therefore the prognosis and treatment varies from patient to patient (Noren et al., 2016). Moreover, the imbalances in the data sets make classification to risk groups difficult, which is why balancing techniques are important.

Though neural networks automatically filter out less-important features from the data (Bomheke and Greenwell, 2020), large feature spaces require a lot of computing power to yield accurate results, which is not always available.

Therefore, it becomes necessary to exclude some of the dimensions. This is not only because the needed computing power might surpass the limits of available resources (Bolón-Canedo et al., 2015), but also because the curse of dimensionality causes the error rate of the used algorithm to increase, and the model to overfit (Venkatesh and Anuradha, 2019). As time progresses the data sets will continue to have larger and larger dimensions due to the advancements in technology (Luengo et al., 2020), which in turn will intensify the importance of feature selection. This holds true for the biological research as well, as new inventions introduce new complexities to the existing problems (Baldi and Hatfield, 2002).

1.2 Goals and Outline

The goal of this thesis is to create a multi-class classifier that is able to predict the risk group of AML patients using only gene expression data. This is done by comparing the outcomes of six data sets. First, the data is divided into training and test sets using stratified sampling, and the feed-forward neural network classifier is tuned with original gene expression data. Next the imbalance in the training set is fixed with SMOTE and ADASYN, and the network is tuned separately for both data sets using the same set of hyperparameter configurations as before. Next, the dimensionality of the original data set is reduced with RFE. After this, SMOTE and ADASYN are used to balance the data. The reason why the dimensionality is reduced before balancing the data sets is because both SMOTE and ADASYN use the Euclidian distance of all features to create new minority samples. Finally, after all data sets have been used to train the classifier, the best network configuration for each scenario is selected and compared.

The structure of the thesis is as follows. In Chapter 2, the theoretical background on gene expression data, feed-forward neural networks, imbalanced classification and feature selection are discussed. Chapter 3 introduces the data set, along with initial data exploration and new variable creation. Chapter 4 gives details on the implementation, going through the technical information, feature selection procedure, data sets, and finally network architectures. In Chapter 5 the results of implementation are presented, and Chapter 6 concludes the thesis with final thoughts and discussion.

2 Theoretical Background

This chapter goes through the theoretical background of the thesis. The basic concepts of gene expression data and acute myeloid leukemia are introduced first. Then, the feed-forward neural networks are discussed in detail, going through the architecture, training, and tuning processes. After this the sampling methods for imbalanced data are examined, including Synthetic Minority Oversampling TEchnique (SMOTE), Adaptive Synthetic Sampling Approach (ADASYN) and evaluation metrics. Finally, the feature selection is discussed, and Recursive Feature Elimination (RFE) method is presented.

2.1 Gene Expression Data

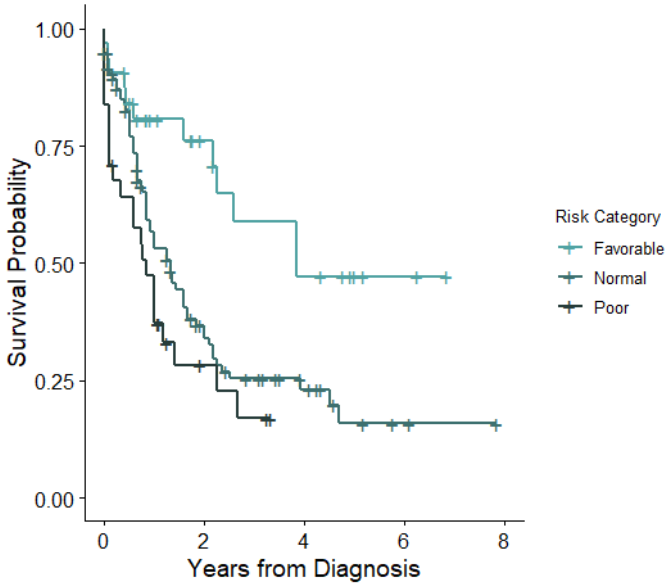
Analysing vast amounts of raw data produced by *DNA arrays* is a major challenge in bio-informatics (Baldi and Hatfield, 2002). In the past, DNA was studied with a microscope, and the produced image could be interpreted by a human. Nowadays the interpretation cannot be done like before, but different technologies are used to produce images, which are stored in large files. Analysing these files is a multidisciplinary problem, which requires knowledge from both biological sciences and statistics (Baldi and Hatfield, 2002).

2.1.1 Basic Concepts

Deoxyribonucleic Acid, or DNA for short, has the potential to reveal the underlying biological processes even in the most complex diseases (Baldi and Hatfield, 2002). In short, DNA is a double-helix shaped molecule that carries information for encoding biological information (Bates, nd). To extract the information, *DNA microarrays* can be used. In summary, they are unique fragments of DNA in a known sequence, which can include anywhere between 10 000 to 100 000 molecule combinations (Baldi and Hatfield, 2002). As described by Smith (nd), the approach for using microarrays starts by attaching them on to a solid surface, which is then bathed with RNA or DNA isolated from a sample. After this, the microarrays pair with the sample RNA or DNA, and emit light through fluorescence (Smith, nd). This light can be detected using specialized machines, and the resulting images are saved as numbers in large files (Baldi and Hatfield, 2002).

The method above can be used to measure *gene expressions* for any given gene (Smith, nd). This is called *gene expression profiling*, which means determining the *messanger RNA* levels in the living cells (Baldi and Hatfield, 2002). As described by Sen (nd), the messenger RNA, or mRNA for short,

Figure 1: Example of a Kaplan–Meier curve for a three-class variable.



carries information out from the cell's nucleus in order to create proteins. The gene expressions are processes that control how much RNA and proteins are created (National Human Genome Research Institute, ndb), and therefore the profiling can be done by measuring the mRNA expression levels in a living cell (Baldi and Hatfield, 2002).

The resulting data from mRNA expression levels can be used in diagnosis of many types of diseases, for example in classifying tumor types (Bolón-Canedo et al., 2015). As mentioned before, one characteristic of gene expression data is large dimensionality p and small sample size n. The reason for this, is that while the gene expression values can be collected simultaneously for a large number of different genes, usually the number of patients is very small in comparison (Bolón-Canedo et al., 2015). The same authors continue, that class-imbalance is another typical problem with gene expression data. This problem occurs in classification tasks, where the less-interesting majority class dominates the data. Bolón-Canedo et al. (2015) explain, that this can cause a bias towards the majority class, so that the minority class samples are miss-classified more often.

Lastly, when recording survival times for patients, often some of them are *censored* (Clark et al., 2003). This means, that the study was interrupted before the event of interest occurred (Jager et al., 2008). This can happen for example when the patient leaves the study before the observation period ends,

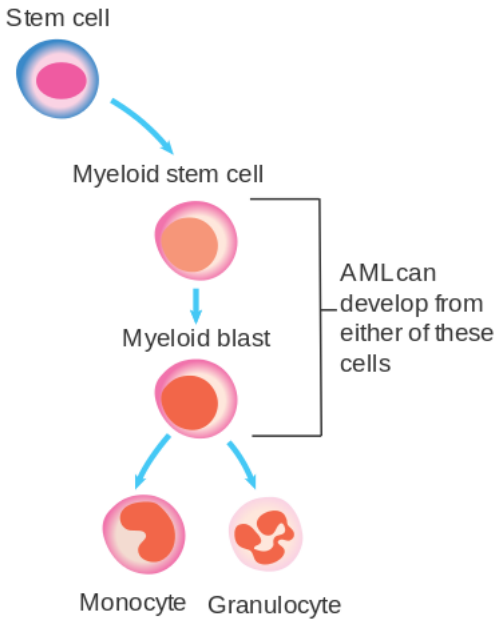


Figure 2: Development of AML.

and the time of death cannot be recorded. Handling the censored values needs to be done with special methods as described by Clark et al. (2003). One of these methods is the *Kaplan-Mayer estimator*, which approximates the survival probability for both censored and uncensored data. The estimated probabilities can be plotted against time to get *Kaplan–Meier curves* which can be used to compare survival times between groups. Figure 1 shows an example of a Kaplan-Mayer curve. More information about the Kaplan-Mayer estimators can be found in (Clark et al., 2003; Jager et al., 2008).

2.1.2 Acute Myeloid Leukemia

Noren et al. (2016) characterize AML as a fatal cancer, with a survival rate of around 5 years. It effects the production of myeloid cells in the bone marrow, but the underlying molecular factors vary significantly from patient to patient. For this reason, the prognosis is very difficult to establish (Noren et al., 2016). Figure 2 from Cancer Research UK / Wikimedia Commons shows the development of AML from the stem cells.

The diagnosis of AML has shifted to using genetic anomalies based on the recommendations of World Health Organization (WHO) (Noren et al., 2016).

While this shift has revealed mutations that are commonly found in AML, the prognosis and classification remain difficult due to the heterogeneity between patients. As discussed by Noren et al. (2016), especially prediction for patients in normal *cytogenetic risk group* is a major challenge. Without going into much detail, cytogenetics is the study of chromosones (National Human Genome Research Institute, nda), and hence the risk groups are created based on differences in them.

Kantarjian et al. (2021) introduce 13 commonly found mutations in AML patients. These are FLT3-ITD, SCT, FLT3-TKD, NPM1, CEBPA, DNMT3A, RUNX1, ASXL1, KIT5, NRAS, IDH2, IDH1, TET2, and TP53. They are a good starting point when analysing AML gene expression data, but not every patient has mutations in these genes due to the heterogeneity mentioned before. Also, other important genes for AML might exist, but they are yet to be discovered. Having said that, AML remains a complex disease which offers great challenges to scientists alike.

2.2 Feed-Forward Neural Networks

The feed-forward neural networks consist of multiple layers of logistic regression models (Bishop, 2006), which can be used for both regression and classification problems (Bomheke and Greenwell, 2020). According to Theodoridis (2020), the multi-layer structure of neural networks allow them to achieve significantly better performance than methods with a single layer. Each layer is built on top of a previous one, so adding more layers allows the model to learn more and more abstract patterns from the data (Theodoridis, 2020). The right network architecture allows the model to yield accurate results, but wrongly chosen parameters make the network unusable.

2.2.1 Network Architecture

Feed-forward neural networks consist of an *input layer*, one or more *hidden layers*, and an *output layer* as seen in Figure 3. Each layer has a set of nodes that are connected to other nodes in successive layers. In feed-forward neural networks there are no connections between nodes in the same layer.

The input layer includes the original predictor variables $(x_1, x_2, ..., x_D)$, and output layer returns the final predictions $(y_1, y_2, ..., y_K)$. Most of the learning happens in the hidden layers, where the number of nodes and layers define the model's learning capacity. The possibility to learn new features is higher when there are more nodes and layers. The number of connections between layers define the density of the network (Bomheke and Greenwell,

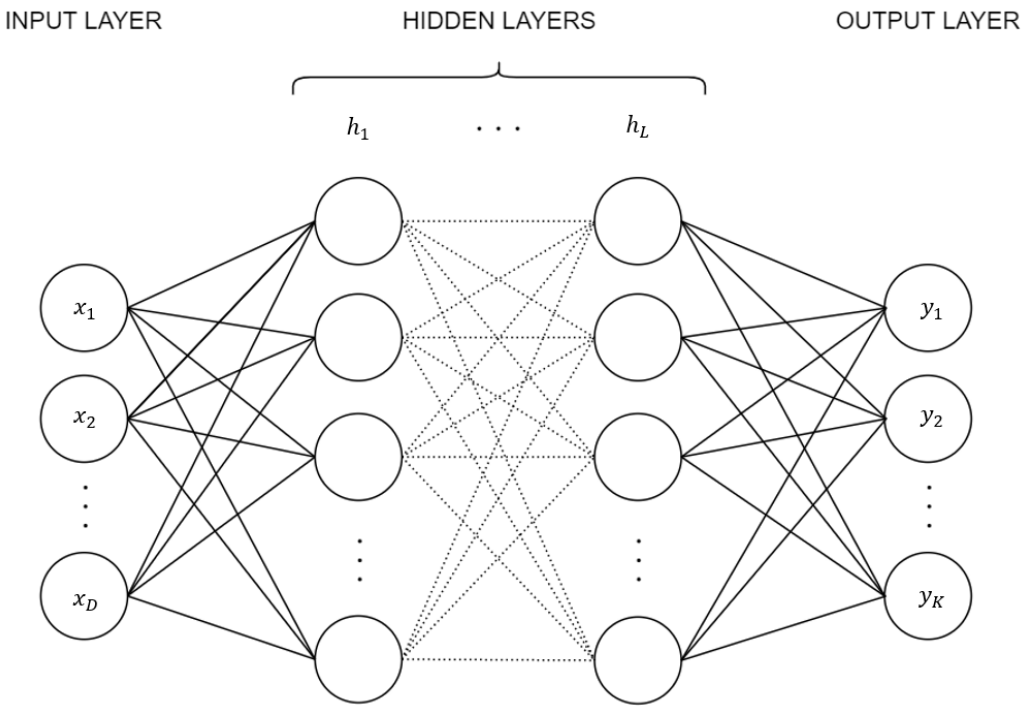


Figure 3: Feed-forward neural network structure.

2020). Figure 3 is considered dense because all nodes are connected between each successive layer.

The number of nodes in the output layer is chosen based on the learning task. When dealing with regression problems, the number of nodes in the output layer is one. This node gives the final predicted value. If the task is a multinomial classification problem, the nodes should be equal to the number of classes. This way, the probability of the observation to belong in each class is obtained. When predicting binary output, the output layer will have one node that predicts the probability for success (Bomheke and Greenwell, 2020). The number of nodes in the input layer is equal to the number of predictor variables.

2.2.2 Training Process

The neural network training process usually utilizes iterative algorithms, which adjust the weights in order to minimize the chosen *cost function* (Bishop, 2006). In short, the cost functions evaluate the performance of the network by calculating errors with a chosen metric for predicted and true values (Bomheke and Greenwell, 2020). Feed-forward neural networks use a

technique called *back propagation* for training, which can be divided into two distinct stages; cost function derivative evaluation and weight adjustments.

The First Stage of Back Propagation. As stated in Bishop (2006), the first stage evaluates the derivatives of the cost function with respect to the weights. The process starts with randomly assigning the weights to each connection of the network, after which the predictor vector $\mathbf{x}$ is given as an input. It is then *forward propagated* through the network to get an initial prediction $\mathbf{y}$.

During forward propagation the inputs are transformed into hidden units with an *activation function*. In short, the activation functions work as filters between layers by determining if a node has enough information to be passed on to the next layer (Bomheke and Greenwell, 2020). One summed input, or *activation*, is constructed from previous layer's nodes by multiplying them with the assigned weights, then summing them together. Following the terminology in Bishop (2006), the forward propagation can be explained as follows. If M_1 is the number of nodes in the first hidden layer, each activation a_j can be written as

$$a_j = \sum_{i=1}^{M_1} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^{M_1} w_{ji}^{(1)} x_i, \tag{2.1}$$

where $j = 1, .., M_1$, and M_1 is the number of input variables. Parameter $w_{ji}^{(1)}$ is weight between i^{th} input variable and j^{th} hidden node. The term $w_{j0}^{(1)}$ is the bias and the superscript $^{(1)}$ refers to the first layer of the network. On the right side of the Equation (2.1), the bias has been included in the weights by starting the sum from 0 and having $x_0 = 1$.

The activation function of the first hidden layer $h_1(\cdot)$ is used to transform the activations into hidden units z_j so that

$$z_j = h_1(a_j), \tag{2.2}$$

where $j = 1, ..., M_2$, and M_2 is the number of hidden units in layer 2. These hidden units are used as the input for the next hidden layer, and the transformations are repeated until the output layer is reached. Figure 4 shows how the inputs are turned into hidden units.

The final predictions y_k from the first iteration are

$$y_k = f\left(\sum_{i=0}^{M_L} w_{ki}^{(L+1)} z_i\right), \tag{2.3}$$

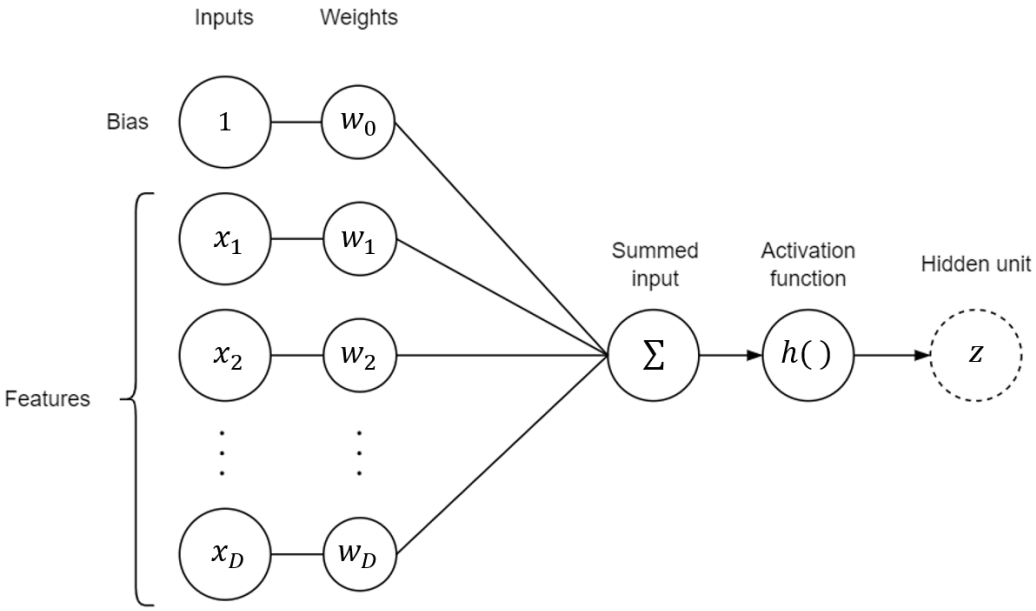


Figure 4: Turning inputs into a hidden unit.

where $k = 1, ..., K$, $f(\cdot)$ is the activation function of the last layer, M_L is the number of nodes in the last hidden layer, $w_{ki}^{(L+1)}$ is the weight between i^{th} hidden node and k^{th} output node, and z_i is the i^{th} hidden unit from the last hidden layer. When decomposing all the hidden units the final outputs y_k can be written as

$$y_k = f\left(\sum_{i_L=0}^{M_L} w_{ki_L}^{(L+1)} h_L\left(\sum_{i_{L-1}=0}^{M_{L-1}} w_{i_L i_{L-1}}^{(L)} \cdots h_1\left(\sum_{i_1=0}^{M_1} w_{i_2 i_1}^{(1)} x_i\right)\right)\right), \tag{2.4}$$

where $f(\cdot)$ is the activation function on the output layer and $h_l(\cdot)$ is the activation function on the l^{th} hidden layer where $l = 1, ..., L$. The term $w_{i_l i_{l-1}}$ is the weight of hidden units in two consecutive hidden layers and x_i is the input. The term M_l is the number of hidden nodes in a hidden layer l.

After getting the predictions y_k, the difference between y_k and the true values t_k is calculated to get the errors

$$\delta_k = y_k - t_k, \tag{2.5}$$

and the algorithm starts going backwards in the network. The error δ_j for each hidden unit is acquired using the *back propagation formula*

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k, \tag{2.6}$$

where $h'(\cdot)$ is the first derivative of the activation function, a_j is the j^{th} activation, w_{kj} is the weight between k^{th} output and j^{th} hidden unit, and δ_k is the error calculated in the previous step. Lastly in the first stage, equation

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i, \tag{2.7}$$

is used to evaluate the derivatives of the chosen cost function E_n with respect to the weights w_{ji}. More details of the first stage can be found in (Bishop, 2006).

The Second Stage of Back Propagation. The second stage of back propagation is to make adjustments to the weights using an *optimizer*. The most popular algorithm for optimizing neural networks is *gradient descent* (Ruder, 2016). In short, gradient is a differential operator that calculates partial derivatives for all variables of a function in Cartesian coordinates (Buono, 2016). The gradient operator ∇ is defined as

$$\nabla := \left(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_p}\right),$$

where $\mathbf{x} = (x_1, ..., x_p)$ are the variables. The gradient of a function $f(\mathbf{x})$ where $\mathbf{x} = (x_1, ..., x_p)$ is

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_p}\right),$$

which is a vector indicating both rate and direction of the fastest increase. This property is utilized in gradient descent algorithms to find a local minima for a cost function, by going the opposite direction of the gradient (Ruder, 2016). The simplest gradient descent algorithm

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{2.8}$$

works by calculating the gradient of a loss function $\nabla E(\mathbf{w}^{(\tau)})$ which is scaled by a predefined learning rate η, and subtracted from the weight vector $\mathbf{w}^{(\tau)}$. This yields an updated weight vector $\mathbf{w}^{(\tau+1)}$ for the next iteration, which is closer to minimizing the chosen cost function (Bishop, 2006).

According to Bishop (2006), the simple gradient descent is far from optimal, and a variety of algorithms have been developed to optimize it. Ruder (2016) gives an overview of these algorithms, from which Adaptive Moment Estimation (Adam) is chosen for closer inspection. Adam stores the averages for the past gradients $m^{(\tau)}$ and past squared gradients $v^{(\tau)}$, which are

exponentially decaying (Ruder, 2016). They are written as

$$m^{(\tau)} = \beta_1 m^{(\tau-1)} + (1-\beta_1)\nabla E(\mathbf{w}^{(\tau)}), \tag{2.9}$$

and

$$v^{(\tau)} = \beta_2 v^{(\tau-1)} + (1-\beta_2)(\nabla E(\mathbf{w}^{(\tau)}))^2, \tag{2.10}$$

where β_1 and β_2 are decay rates. The bias-corrected first and second moment estimates are

$$\hat{m}^{(\tau)} = \frac{m^{(\tau)}}{1-\beta_1^\tau} \qquad \hat{v}^{(\tau)} = \frac{v^{(\tau)}}{1-\beta_2^\tau} \tag{2.11}$$

which are implemented in the Adam learning scheme

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \frac{\eta}{\sqrt{\hat{v}^{(\tau)}}+\epsilon}\hat{m}^{(\tau)}. \tag{2.12}$$

The term ϵ is a smoothing parameter which is used to avoid division by zero. As stated by Theodoridis (2020), Adam seems to be one of the most popular optimizers for deep neural networks. To further justify choosing Adam, according to Ruder (2016), the optimization scheme works well in practice, and might be the best choice for optimization due to the bias-correction.

After using the optimizer to get new weights for each connection, the back propagation algorithm goes back to the first stage by forward propagating the predictor vector through the network Bishop (2006). The information is sent iteratively forwards and backwards through the network in these two stages, until the cost function is minimized, desired accuracy is reached, or requirements for other *stopping criteria* is met (Theodoridis, 2020).

2.2.3 Choosing Activation and Cost Functions

Before training the neural network, adequate activation and cost functions should be chosen. Each hidden and output layer of the network is assigned with an activation function, as seen in Equation (2.4). The cost function on the other hand is set for the whole network. It is possible to utilize multiple cost functions in one network (Bomheke and Greenwell, 2020), but one is sufficient for this thesis.

In the literature terms cost function, loss function and error function are used interchangeably. In Bomheke and Greenwell (2020), the cost function is called loss and objective function, Bishop (2006) is using the term error

function, and Theodoridis (2020) uses both cost and loss function. To be exact, all of these terms have sightly different definitions, but usually their usage is clear based on the context.

The choice of activation and cost functions should be done based on the assumed distribution of target variables and the nature of the data (Bishop, 2006). For this reason, the activation function for the output layer and the cost function are often chosen in pairs. For hidden layers though, rectified linear unit (ReLU) is a common choice as an activation function (Bomheke and Greenwell, 2020). It is defined as

$$h(a) = \begin{cases} 0, & \text{for } a < 0 \\ a, & \text{for } a \geq 0 \end{cases}, \tag{2.13}$$

which takes an activation a as an input and changes all the negative values to zero.

Regression. For regression problems linear identity is a common choice as the activation function for the output layer, which can be paired with mean-squared-error cost function (MSE) (Bomheke and Greenwell, 2020). They are defined as

$$f(a) = a, \tag{2.14}$$

and

$$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (y_n - t_n)^2, \tag{2.15}$$

respectively. The term a is the activation, $\mathbf{w}$ is the weight matrix of the network, N is the number of observations, y_n is the predicted output, and t_n is the true value.

Binary Classification. Binary classification problems often utilize sigmoid activation function and cross-entropy cost function (Bomheke and Greenwell, 2020). They are written as

$$f(a) = \frac{1}{1 + e^{-a}}, \tag{2.16}$$

and

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \big(t_n \ln(y_n) + (1 - t_n) \ln(1 - y_n) \big), \tag{2.17}$$

respectively. Here e is Euler's number and $\ln(\cdot)$ is the natural logarithm.

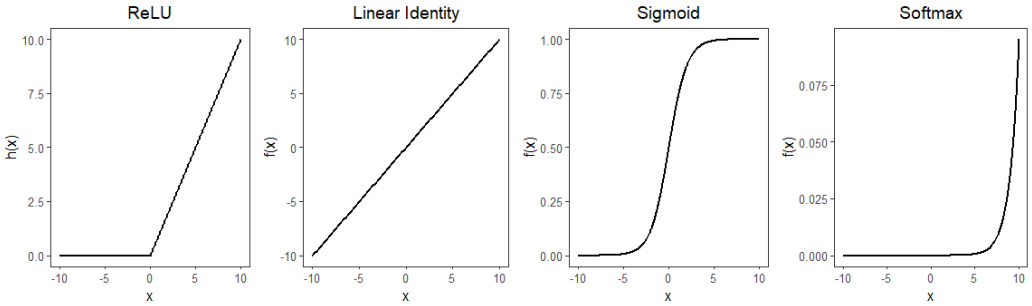


Figure 5: Most common activation functions.

Multinomial Classification. Finally, multinomial classification problems often utilize softmax function on the output layer, and multicategorical cross-entropy as the cost function (Bomheke and Greenwell, 2020). They are defined as

$$f(a) = \frac{e^{a_i}}{\sum_j e^{a_j}}, \tag{2.18}$$

and

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\big(t_{kn}\ln(y_{kn})\big). \tag{2.19}$$

respectively. Here N is the number of observations, and K is the number of classes. Figure 5 shows the most common activation functions as a function of x.

2.2.4 The Bias-Variance tradeoff

Prediction errors are composed of two main components: *bias* and *variance*. The bias tells how far off an average prediction is from the correct value, and it determines how well the model is able to depict the structure of underlying data (Bomheke and Greenwell, 2020). Variance measures how much individual data sets vary around their average, and hence it shows how sensitive the model is to the choice of the data set (Bishop, 2006). Ideally, both of these should be minimized to get small prediction error, but as discussed in Theodoridis (2020), it is not possible with a fixed amount of training data.

Decreasing the bias term without increasing the number of training data points can be done by increasing the complexity of the model. The problem with this approach is that it would result in a higher variance. As stated in Theodoridis (2020), the only way to decrease both bias and variance is

to increase both the number of training data points, and the model complexity in a controlled manner. Increasing both arbitrarily can increase the overall error of the model (Theodoridis, 2020). This balancing problem is called the *bias-variance trade-off*, and it is always present in estimation tasks (Theodoridis, 2020). Figure 6 shows how bias and variance work together in an estimation task. The inner circle represents perfect predictions, and the predictions get worse and worse towards the outer circle.

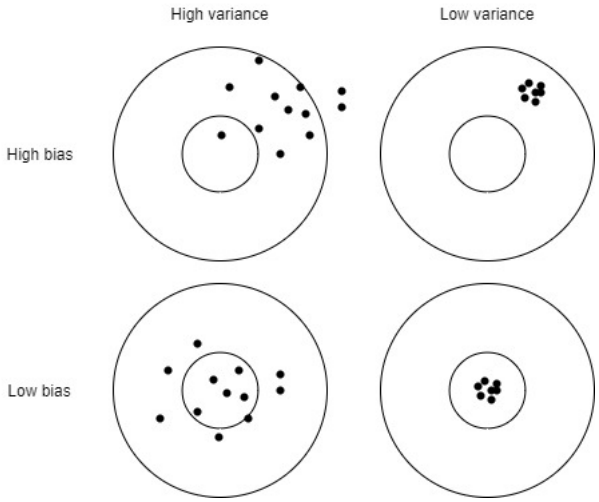


Figure 6: Illustration of bias and variance.

The expected loss can be divided into bias, variance and noise, so that

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}. \tag{2.20}$$

According to Bishop (2006), rigid models have low variance and high bias, and flexible models have the opposite. To find an appropriate balance between the bias and variance, the *Occam's razor rule* can be used. In the context of model selection, it means selecting the simplest possible model that is able to explain the data (Theodoridis, 2020). This rule is helpful during the neural network hyperparameter tuning.

2.2.5 Tuning the Hyperparameters

In order to find the optimal neural network structure, the *hyperparameters* need to be tuned. These hyperparameters define the networks optimization style, regularization methods and architecture. Especially in deep neural networks the number of possible hyperparameter combinations is huge (Feurer and Hutter, 2019). Although these parameters can be chosen manually, there

are automated ways that improve the performance of the network, promote the reproducibility of the results, and reduce the needed human effort (Feurer and Hutter, 2019).

As discussed by Bomheke and Greenwell (2020), often the tuning process starts by choosing the hyperparameters for the model capacity, which include the number of hidden layers and the number of nodes in each hidden layer. The number of layers defines the *depth* of the network, and the number of hidden nodes defines the *width* (Theodoridis, 2020). More nodes and layers increase the model's capacity, which allows the model to learn more patterns in the data (Bomheke and Greenwell, 2020). It is noted in Bomheke and Greenwell (2020) that too much learning capacity results in a model that overfits the training data, and hence the goal of tuning is to find the best performing model that is as simple as possible.

A general guideline is to set the number of hidden nodes to less than or equal to the number of features, but it can be adjusted arbitrarily (Bomheke and Greenwell, 2020). A sufficient number of layers is often two to five, but when dealing with data that has many features, it is advisable to have more layers and less nodes in order to reduce the computational complexity (Bomheke and Greenwell, 2020). This is also discussed by Theodoridis (2020), who states that for feed-forward neural networks increasing the number of layers by one is exponentially more valuable than increasing the number of nodes in the layers.

Apart from network architecture, other tunable parameters include learning rate, number of epochs and batch size. Too small a learning rate results in a slow convergence, and too high value causes oscillatory convergence or even for the algorithm to diverge (Theodoridis, 2020). This problem is solved by the Adam optimizer, as it adapts the learning rate over time, and hence extensive tuning is not needed (Ruder, 2016). According to Ruder (2016) good results can be achieved even with the algorithm's default value.

As defined by Bomheke and Greenwell (2020), the number of epochs means how many times the network uses the entire data set for training. The number should be high enough to minimize the cost function, but small enough that the model does not overfit (Theodoridis, 2020). Bomheke and Greenwell (2020) also mention that the more complex the data is, the more epochs are needed to learn the relationships and features. The optimal number might be difficult to choose, so *early stopping* can be used to reduce the number of epochs. As described by (Bomheke and Greenwell, 2020), it works by halting the training if the loss function stops improving for certain number of epochs.

The last hyperparameter to tune is the batch size. Instead of using the full training set for each epoch, the training can be done in batches. Usually

batch size is a power of two in order to take full advantage of the hardware (Bomheke and Greenwell, 2020). As described by Bomheke and Greenwell (2020), high batch sizes give less feedback, and small batch sizes are computationally more expensive.

When the neural network has multiple layers, the distribution of the input changes for each layer (Theodoridis, 2020). This can make the training process sensitive to initial values of parameters, and might require using smaller learning rates. This in turn makes the training process last longer. Theodoridis (2020) describes that *batch normalization* helps with these issues by scaling the activations in each layer. This is also discussed by Bomheke and Greenwell (2020), who state that as the number of layers increases, the more important batch normalization becomes.

To further constrain the model's complexity, a regularization can be added to the hidden layers. Bomheke and Greenwell (2020) explain that *dropout* is one of the most used regularization methods, which randomly removes nodes in hidden layers. They continue by stating, that it helps with overfitting, and avoids the network from learning random noise.

After choosing the hyperparameters for tuning, different combinations should be tested to find the best performing model. Small networks can be tested manually, but for larger models automated methods like *grid search*, *random search* or *Bayesian optimization* are preferred.

As described by Feurer and Hutter (2019), grid search and random search are model-free black-box methods, where the hyperparameters are chosen from a finite set of user-defined values. Grid search evaluates each combination of given hyperparameters (Feurer and Hutter, 2019), which is why it requires a lot of computing power when the number of parameters is large. Random search is more feasible with limited resources, since the number of randomly selected configurations can be adjusted to fit the needs (Feurer and Hutter, 2019). More information about different hyperparameter tuning methods can be found in (Feurer and Hutter, 2019).

2.3 Classification with Imbalanced Data

Before performing predictions with neural networks, the data is split into *training* and *test sets*. The training set is used to train the algorithm and tune the hyperparameters, and the test set is used to assess the performance of the model (Bomheke and Greenwell, 2020). It is also possible to use a separate *validation* set for optimizing the model, but according to Bishop (2006) it is often too wasteful since it reduces the amount of training data.

Usually the data is split so that 70% of the observations are assigned for training and 30% for testing, but it is also acceptable to have $60\% - 40\%$

or $80\% - 20\%$ split between training and test sets (Bomheke and Greenwell, 2020). As discussed by the same authors, when the number of features is larger than the number of observations the $80\% - 20\%$ split performs better. They further explain that problems arise when the test set is less than 20%, because the model might not be generalizable even if it performs well during training.

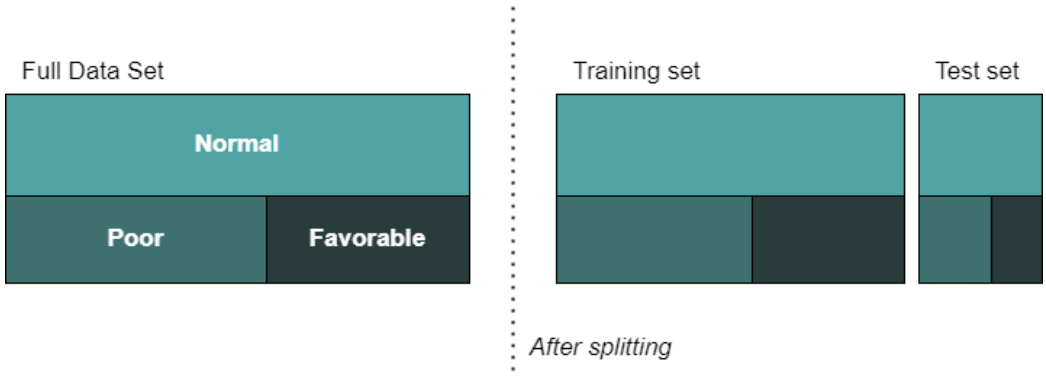
Simple random sampling splits the data completely at random, so the distribution of the response in training and test sets cannot be controlled. As discussed in Bomheke and Greenwell (2020), *stratified sampling* is an alternative splitting technique, which can be used to control the distribution of the response variable. It works by splitting the data so that the test and training sets have similar distributions for the output variable. Figure 7 shows how the split is done for a three-class response variable.

Though stratified sampling ensures that both training and test sets have similar distributions, it does not balance the data. To balance the data in terms of response variable classes, methods that remove observations from the majority class or create new observations for minority classes can be used. As described by Fernández et al. (2018), this is an important step when classifying from imbalanced data, since the classifiers tend to overlook the minority class, which hinders the performance. Many times the majority class has great accuracy, while the minority classes get accuracies close to 0% (Fernández et al., 2018).

Under-sampling is a data-level method, which randomly eliminates observations from the majority class (Kotsiantis et al., 2006). While it is able to balance the data, at the same time it discards potentially important examples from the data set, and increases the variance (Fernandez et al., 2018). Especially when the imbalance is large, removing data points might hinder the generlizability of the model (Fernandez et al., 2018). When dealing with a very limited data set, this method is not optimal. As described in Fernandez et al. (2018), the problems associated with undersampling motivated researchers to develop *oversampling methods*.

Random oversampling is a technique that randomly creates extra copies of the minority class observations (Fernández et al., 2018). While it does not discard potentially important information, it increases the likelihood of overfitting because of the duplicate observations (Fernández et al., 2018). Oversampling methods like SMOTE and ADASYN fix this problem by creating completely new data points.

Figure 7: Stratified sampling.



2.3.1 SMOTE

The *Synthetic Minority Oversampling TEchnique*, SMOTE for short, is a method published in 2002 (Chawla et al., 2002). Instead of duplicating observations, it creates synthetic instances from the minority class (Luengo et al., 2020). As described by Fernandez et al. (2018), it is considered a benchmark method for imbalanced data prepossessing.

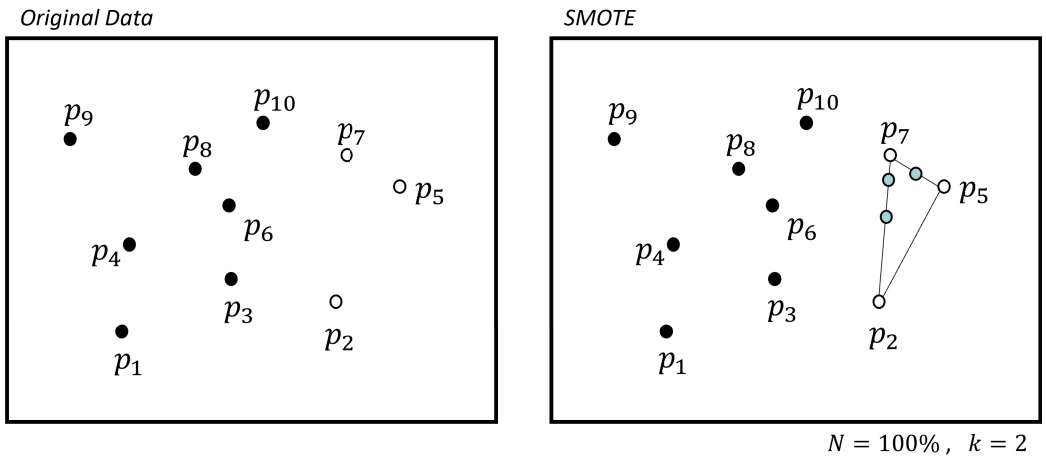
The SMOTE utilizes the *k-nearest-neighbors* (KNN) algorithm to calculate new points. In short KNN is a simple method for identifying the k closest observations to a selected data point (Bomheke and Greenwell, 2020). The distance measure for KNN in SMOTE is Euclidean distance (Chawla et al., 2002), which is defined by Bomheke and Greenwell (2020) as

$$d = \sqrt{\sum_{j=1}^{p}(x_{aj} - x_{bj})^2} \ . \tag{2.21}$$

Here p is the number of features, and x_{aj} and x_{bj} are the points for which the distance is calculated.

The following walk-through of SMOTE is based on Fernández et al. (2018). First, the oversampling percentage N is chosen. Usually it is set so that the majority and minority classes will have approximately equal number of observations after the sampling is complete. If N is less than 100%, then N of the minority class samples are randomly chosen for simulation. When N is 100% or larger, each minority observation is used to create $N/100$ new samples. In this case N is assumed to be a multiple of hundred. For example $N = 200\%$ would yield 2 new samples for each minority observation.

Figure 8: Example of balancing data with SMOTE.



After choosing the oversampling percentage, one minority class example p_i is chosen from the data, $i = 1, ..., n$, where n is the number of minority samples, and its k nearest neighbors from the minority class are obtained. Then, depending on the amount of oversampling N and the number of neighbors k, all or some of the neighbors are selected for interpolation. Note that k should be smaller than the number of minority samples in the data. The distances from point p_i to these neighbors are calculated. Then, a random number r is chosen between 0 and 1, which is multiplied by the distance and added to the original point p_i. This creates a new sample, that is in the line between p_i and its neighbor. Given that x is one of the selected neighbors, a new value p can be formally written as

$$p = p_i + r \cdot x. \tag{2.22}$$

When $N > 100\%$, this procedure is repeated for each minority sample p_i, $i = 1, ..., n$, so that in the end $n \cdot N/100$ new samples are created. When $N \leq 100\%$ the procedure is repeated for each randomly chosen minority sample, to get $N \cdot n$ new samples.

Figure 8 illustrates how the new points are chosen for a data set with 7 majority class instances and 3 minority instances. The black dots represent majority samples, white dots are minority samples, and cyan dots are new samples. Here, the sampling degree is set to 100%, meaning that each minority sample is used to create one new sample. Since the neighbors are chosen randomly, both p_2 and p_7 created a new sample between each other,

and there are no new samples between points p_2 and p_5. The number of new samples is $N \cdot n = 100\% \cdot 3 = 3$.

2.3.2 ADASYN

Adaptive Synthetic Sampling Approach, or ADASYN for short, is an oversampling method that generates minority samples based on their distribution (Fernández et al., 2018). More specifically, ADSYN creates more samples based on observations that are difficult to learn, than easily learnable observations (Fernández et al., 2018). This helps defining the decision boundary while reducing the imbalance.

The following description of ADASYN is based on Fernández et al. (2018). The first step in ADASYN is to calculate the class imbalance

$$d = m/n, \tag{2.23}$$

where m and n are the number of observations in a minority and majority classes. Additionally, $N = m + n$ is the total number of observations. Next, the number of samples needed to balance the data is calculated with

$$G = (n - m) \cdot \beta, \tag{2.24}$$

where parameter $\beta \in [0, 1]$ specifies the level of balance. If the wanted output is fully balanced data the β is set to 1. After this, the KNN is calculated using Euclidian distance for each minority example p_i, $i = 1, ..., m$. Here the neighbors are chosen from both minority and majority samples. Next, a ratio of majority class neighbors to all neighbors is calculated for each minority sample

$$r_i = \Delta_i / k, \tag{2.25}$$

where Δ_i is the number of majority samples in the KNN of p_i, $i = 1, ..., m$. After this, each ratio r_i is normalized

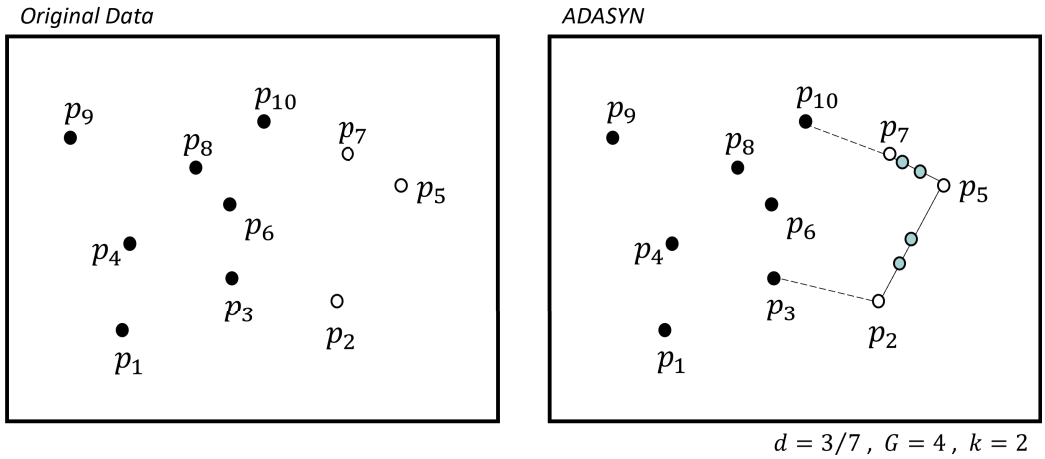
$$\hat{r}_i = \frac{r_i}{\sum_{i=1}^{m} r_i}, \tag{2.26}$$

and the number of samples to generate from each minority class sample p_i is calculated with

$$g_i = \hat{r}_i \cdot G. \tag{2.27}$$

Finally, the g_i new minority samples are generated from each minority observation p_i. This is done in a similar fashion as in SMOTE. First, a random

Figure 9: Example of balancing data with ADASYN.



minority-class neighbor p_{zi} is chosen from the KNN of p_i. Then, a new point is randomly chosen on a line between p_{zi} and p_i so that

$$s_i = p_i + (p_{zi} - p_i) \cdot \Lambda, \tag{2.28}$$

where Λ is a random value between 0 and 1. This is repeated g_i times for each p_i to get a total of G new samples.

Figure 9 shows an example of how ADASYN creates new samples for a data set where number of samples in the majority and minority classes are 7 and 3. As in SMOTE example, the black dots are majority samples, white dots are minority samples, and cyan dots are newly generated samples. If the above formulas are applied to this data, the number of new samples created from minority observations p_2, p_5 and p_7 would be $g_2 = 2$, $g_5 = 0$ and $g_7 = 2$. As seen in the figure, both p_2 and p_7 have only one minority class neighbor from the 2 nearest neighbors, so the all new points will be between these samples.

2.3.3 Evaluation Metrics for Multi-Class Imbalance

Assessing the performance of a multi-class classifier for imbalanced data can be difficult, because some common metrics might hide the poor predictive performance of the minority classes (Kotsiantis et al., 2006). Moreover, binary classification techniques cannot be directly applied to the multi-class scenario, which makes the imbalance even more difficult to deal with (Kotsiantis et al., 2006). In the worst case scenario, the classifiers cannot generate

Table 1: Confusion matrix for three classes. The focus is on class A.

	True value		
Prediction	A	B	C
A	TP	FP	FP
B	FN	TN	TN
C	FN	TN	TN

	True value		
Prediction	A	B	C
A	n_{AA}	n_{AA}	n_{CA}
B	n_{AB}	n_{BB}	n_{CB}
C	n_{AC}	n_{BC}	n_{CC}

any rules to predict the minority classes (Kotsiantis et al., 2006), and for these reasons, multiple metrics should be utilized, so that a full understanding of the classifier's performance is obtained.

Confusion Matrix. To see how the model is performing for each class, a confusion matrix can be created. It shows all relevant information about the prediction, namely the number of classified instances between predicted and true classes (Grandini et al., 2020). In a confusion matrix the diagonal axis represents the number of correct classifications, one axis represents the predicted classes, and the other axis represent the true classes (Fernández et al., 2018). Here, columns are used for true classes and rows for predicted classifications. Table 1 shows an example of a confusion matrix for a three-class variable.

For binomial classification, the confusion matrix includes false positives (FP) and false negatives (FN), which are incorrect classifications, and true positives (TP) and true negatives (TN), which are correct classifications (Fernández et al., 2018). When there are more than two classes, these values need to be defined differently by focusing on one class at a time (Grandini et al., 2020). If the focus is on a class K, the true positive can defined as correctly classified instances for the class K. The false positives are wrongly classified in class K, and false negatives are class K observations that were wrongly classified to other classes. The true negatives are all the remaining cells in the confusion matrix. As described by, Grandini et al. (2020) these values are the building blocks for many other evaluation metrics.

Accuracy. To know how big a percentage of observations are classified correctly, accuracy can be used. It is one of the most popular metrics for classification, and it gives the probability for the prediction to be correct (Grandini et al., 2020). The formula can be written as

$$\text{Accuracy} = \frac{\text{Number of Correct Classifications}}{\text{Number of All Classifications}}, \tag{2.29}$$

which is

$$\text{Accuracy} = \frac{n_{AA} + n_{BB} + n_{CC}}{n} \tag{2.30}$$

for the example in Table 1, where n is the total number of classifications.

Accuracy can be used when the distribution of classes is not important (Grandini et al., 2020). It is a good measure when only individual classifications are the focus, but it might give deceptively good results for the imbalanced response variable. As explained by (Grandini et al., 2020), this is because the biggest classes get the highest weights which hides the errors of the smallest classes. For example, if a model is able to only predict majority class, the accuracy might be 90% even if none of the minority class observations are predicted correctly. Though accuracy is an intuitive metric, it fails to capture the errors in minority classes (Grandini et al., 2020), which is why other metrics should also be considered.

Precision. One metric that takes the class imbalance into account is precision. It indicates how much the model can be trusted when predicting a specific class (Grandini et al., 2020). It is a fraction of the correct classifications in a class, divided by the number of all classifications in the same class, and is written as

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{2.31}$$

For the example confusion matrix in Table 1, the precision for class A would be

$$\text{Precision}^{(A)} = \frac{n_{AA}}{n_{AA} + n_{BA} + n_{CA}}.$$

Calculating the precision for each class separately gives an idea of how well the model classifies samples to class K, when the correct class is K (Grandini et al., 2020).

Recall Intuitively speaking, recall means how well the model is able to find samples belonging to the class of interest (Grandini et al., 2020). It is defined as the number of correct classifications in class K, divided by number of all observation in class K. Based on confusion matrix it can be written as

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{2.32}$$

which is

$$\text{Recall}^{(A)} = \frac{n_{AA}}{n_{AA} + n_{AB} + n_{AC}}.$$

for class A in Table 1. Recall and precision are used to create many other metrics, few of which will be introduced next.

Balanced Accuracy. If recall is calculated separately for all classes, and their average is taken, the resulting metric is *Balanced Accuracy*, which can be used for both binary and multi-class classification (Grandini et al., 2020). For the example matrix in Table 1, it is defined as

$$\text{Balanced Accuracy} = \frac{\text{Recall}^{(A)} + \text{Recall}^{(B)} + \text{Recall}^{(C)}}{3}. \tag{2.33}$$

If the data has very little imbalance, the balanced accuracy and accuracy will have similar values. On the other hand, balanced accuracy tends to work better than accuracy when the data is imbalanced and the correct classification of minority classes is important.

F1-Score. The F1-score is a weighted average of recall and precision, which has its worst values close to 0 and its best values near 1 (Grandini et al., 2020). For multiple classes, the F1-score needs to be calculated separately for each class. Formally, it can be defined as

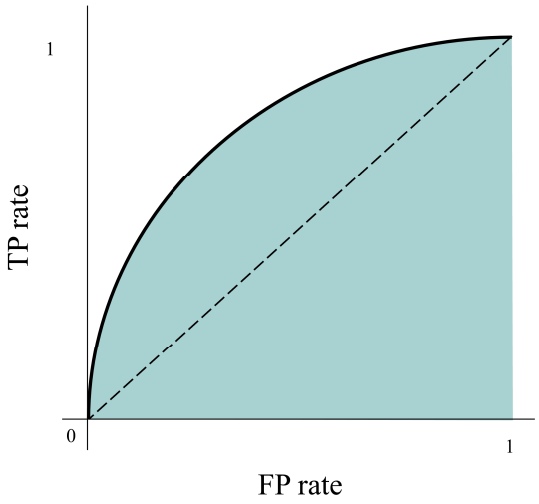
$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{2.34}$$

which is

$$F_1^{(A)} = 2 \cdot \frac{\text{Precision}^{(A)} \cdot \text{Recall}^{(A)}}{\text{Precision}^{(A)} + \text{Recall}^{(A)}}$$

for class A in example 1. It is possible to calculate *Macro* and *Micro F1-Score* for all classes simultaneously, but more information about these methods can be found in (Grandini et al., 2020).

Figure 10: ROC and AUC.



ROC curve and AUC. According to Kotsiantis et al. (2006), Receiver Operating Characteristic (ROC) and Area Under the ROC Curve (AUC) are among the most common metrics for classification. The ROC curve is plotted using TP and FP rates. They are defined as

$$\text{TP rate} = \frac{TP}{TP + FN} = \text{recall}, \tag{2.35}$$

and

$$\text{FP rate} = \frac{FP}{TN + FP}. \tag{2.36}$$

The TP rate is exactly the same as recall, and FP rate is the number of samples miss-classified as class K, divided by all classifications that do not truly belong to class K. They are used so that FP rate is on the x-axis, and TP rate is on the y-axis. The AUC is the area under this curve (Kotsiantis et al., 2006). As with the previous metrics, ROC and AUC need to be calculated individually for each class when dealing with multiple classes. Figure 10 shows an example of a ROC curve, where the cyan are is the AUC. The dashed line represents a random classification.

2.4 Feature Selection

When dealing with high dimensional data, the needed computing power might surpass the limits of available resources, and the *curse of dimensionality* might cause the the error rate to increase and the model to overfit (Venkatesh and Anuradha, 2019). For this reason, excluding some of the dimensions of high dimensional data is necessary. *Feature selection* does this by creating a data set that ideally includes only the relevant features from the original data (Bolón-Canedo et al., 2015). As discussed by the same authors, using feature selection techniques allows for the creation of simpler models, which in turn reduces the needed memory and computing power.

The feature selection approach should be chosen based on the problem setting at hand. The approaches can be divided into *filter*, *embedded* and *wrapper methods*, which all have different types of relationships to the machine learning algorithm (Bolón-Canedo et al., 2015). Filters perform feature selection based only on the training data, which means that they don't take any feedback from the predictors (Bolón-Canedo et al., 2015). Advantages of this type of feature selection includes low computational costs as well as generalizanibiliy. Wrapper methods are so called black box methods, in which the feature subset is selected based on some performance metric (Bolón-Canedo et al., 2015). Wrappers tend to yield better performance than filter methods, but the computational costs are higher. In embedded methods, the feature selection is integrated into the model building (Liu and Motoda, 2008). One major advantage of embedded methods is the ability to find dependencies with less computational costs than wrappers (Bolón-Canedo et al., 2015).

As described by Luengo et al. (2020), the property of having overwhelming amount of features is called *Big Dimensionality*. Some of the features bring only a small amount of value to the predictions, and some just add noise. These features are considered irrelevant or redundant, and feature selection aims to remove them from the data (Luengo et al., 2020).

2.4.1 Feature Relevance and Redundancy

Features can be grouped into three categories based on how relevant they are. The first group includes strongly relevant features, the second weakly relevant and the third irrelevant features (Kohavi and John, 1997).

Based on Kohavi and John (1997), a definition of strongly and weakly relevant features can be written as follows. Let $S_i = \{X_1, ..., X_{i-1}, X_{i+1}, ..., X_n\}$ be a subset of all features except X_i. Let $s_i = \{x_1, ..., x_{i-1}, x_{i+1}, ..., x_n\}$ be the assigned values and Y the response variable. Now a feature X_i is *strongly*

relevant if and only if there exists some x_i, y_i and s_i so that

$$p(Y = y|X_i = x_i, S_i = s_i) \neq p(Y = y|S_i = s_i), \tag{2.37}$$

where $p(X_i = x_i, S_i = s_i) > 0$. Now let S'_i be a subset of S_i. A feature X_i is *weakly relevant* if and only if there exists some x_i, y_i and s'_i so that

$$p(Y = y|X_i = x_i, S'_i = s'_i) \neq p(Y = y|S'_i = s'_i), \tag{2.38}$$

where $p(X_i = x_i, S'_i = s'_i) > 0$. Collectively, strongly and weakly relevant features are defined as *relevant*. If a feature is not relevant, it is defined as *irrelevant*.

According to Bolón-Canedo et al. (2015), if the values of two features are completely correlated, they are *redundant* with one another. In other words, removing one of the features will not affect the model's performance even if both of the features are relevant (Liu and Motoda, 2008). The optimal outcome of feature selection includes only strongly relevant, weakly relevant and nonredundant features (Bolón-Canedo et al., 2015). Therefore, the goal of feature selection is to remove all the irrelevant, weakly relevant and redundant features, so that the remaining subset is able to describe the original problem (Luengo et al., 2020).

2.4.2 Recursive Feature Elimination

Recursive Feature Elimination, (RFE) is a simple feature selection algorithm that creates a feature subset based on classification accuracy of learned model (Jeon and Oh, 2020). Newer approaches are based on feature importance inside the model, which use, for example, Support Vector Machine (SVM), Random Forest (RF) or Gradient Boosting Machine (GBM). These are discussed in more detail by Jeon and Oh (2020).

The basic importance based RFE algorithm works as follows. First, the classifier is trained to obtain the weights for each feature. Then, the features are ranked based on the weights, and the feature with the lowest weight is removed from the data set. The classifier is then trained again, and the feature with the lowest weight is removed. This procedure is repeated until all features have been deleted from the data, and the final ranking is given by the algorithm. As discussed by Jeon and Oh (2020), this RFE is an embedded feature selection method, that performs stronger than the wrapper and filer methods.

According to Bolón-Canedo et al. (2015) SVM-RFE is among the most commonly use feature selection methods for microarray data sets, but for

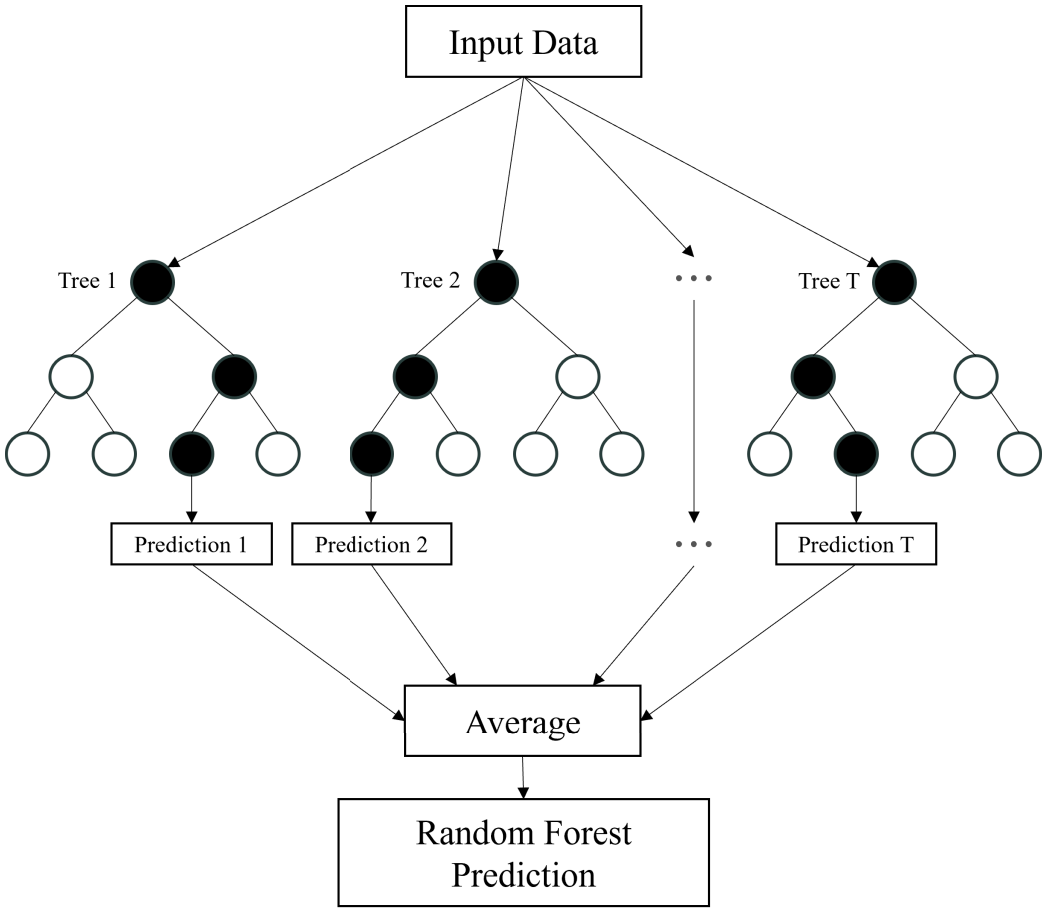
this thesis, RF based method is used. As discussed by Jeon and Oh (2020) and Theodoridis (2020), RF is a method that combines bagging and decision trees, and has demonstrated high predictive accuracies in many applications. The RF algorithm *bootstraps* copies of the training data, and builds multiple trees on these copies (Bomheke and Greenwell, 2020). In this context, bootstrapping means creating multiple uniformly sampled copies from the training set with replacement (Theodoridis, 2020). The creation of trees works by selecting a random collection of features, which is split based on a chosen splitting rule (Bomheke and Greenwell, 2020). One possible rule is mean decrease accuracy (MDA), which is used for calculating variable importances. It is defined as

$$\text{MDA} = \frac{\sum_{t \in B} VI^{(t)}(X_i)}{n}, \tag{2.39}$$

where n is number of trees, and $VI^{(t)}(X_i)$ is the variable importance for feature X_i in tree t.

After the trees have been created, the predictions are aggregated across all trees (Bomheke and Greenwell, 2020) to get the final prediction. Figure 11 shows an example of a random forest with T trees. For classification, the average is replaced with a majority vote (Theodoridis, 2020). Embedding these predictions to the RFE algorithm creates a feature selection algorithm, where a new random forest is used every time a feature is dropped.

Figure 11: Random Forest structure.



3 Data

This chapter introduces the data set used for implementation. First, the data retrieval process is explained, along with the data preparation methods. Next, a summary of the data is given, and some explanatory analysis is done. Finally, a new variable is created from two other variables, which will be used as a response variable during implementation.

3.1 Retrieval and Preparation

The data set is downloaded from *The Cancer Genome Atlas* (TCGA) with a custom function provided by Juho Kontio. The TCGA Research Network[1] includes over 2.5 petabytes of data from 33 cancer types, from which AML is chosen for the implementation.

The download function returns a data set including 172 observations and 20 636 variables for AML. The first 106 columns are clinical data, and the last 20 530 variables include gene expression values from different genes. The gene expression data have been automatically $\log_2(x + 1)$ transformed and mean-centered by the download function. The data set is made publicly available at (https://github.com/HeliJulia/AML-Risk-Group-Prediction). More information on how the mRNA expression data was gathered can be found at National Cancer Institute GDC Documentation[2].

Since the focus of feature selection will be on the genes, some initial selection is done for the clinical variables. From 106 clinical variables 6 are kept for data exploration, and their names are changed. The original and new names of these variables can be seen in Table 2.

Next, variables *risk category*, *censoring*, *gender* and *subtype* are turned

[1] https://www.cancer.gov/tcga

[2] https://docs.gdc.cancer.gov/Data/Introduction/

Original name	*New name*
OS	censoring
OS.time	survival_time
gender	gender
age_at_initial_pathologic_diagnosis	age
acute_myeloid_leukemia_calgb_cytogenetics_risk_category	risk_category
Subtype_Selected	subtype

Table 2: Changed variable names.

into factors. The labels of risk category are changed from [Favorable, Intermediate/Normal, Poor] to [Favorable, Normal, Poor]. Two new variables *age group* and *new risk category* are also created, which will be discussed in detail during the data exploration stage. The variable *new risk category* will be used as the response variable, so all missing values from it are removed. This removes one observation from the data set.

After these preparations the data set includes 171 observations and 20 538 variables. From these, 20 530 are gene expression levels and 8 are clinical variables. The eight clinical variables, their data types, and the value ranges are shown in Table 3.

3.2 Data Summary and Exploration

The variable *survival time* is the number of days from the initial diagnosis to time of death. In the cases where *censoring* is zero, the *survival time* means how many days a patient was alive from diagnosis until the experiment was interrupted. When *censoring* is one, the *survival time* is the time from diagnosis until death.

Figure 12 shows the distribution of *survival time* in years. From all the observations 34.5% are censored. The longest survival time is 7.8 years, and the shortest is 0 years. The mean of the survival times is 1.57 years and median is 1.00 years. The first quartile is 0.42 years, and the third quartile is 2.17 years. The distribution of the survival times is heavily skewed to right, with around 50% of patients dying or ending the study in under a year.

The third variable *gender* is divided into two groups, male and female. From 160 patients 45.6% are female and 54.4% are male, so the gender distribution is relatively balanced. Variable *age* tells how old the patient was at the initial diagnosis, measured in years. The age distribution is left-skewed,

variable name	*data type*	*value range*
censoring	Factor (2 levels)	0, 1
survival_time	Integer	min: 0, max: 2861
gender	Factor (2 levels)	Female, Male
age	Integer	min: 18, max: 88
age_group	Factor (3 levels)	Young, Adult, Old
risk_category	Factor (3 levels)	Favorable, Normal, Poor
subtype	Factor (7 levels)	M1, M2, M3, M4, M5, M6, M7
new_risk_category	Factor (3 levels)	Favorable, Normal, Poor

Table 3: First eight variables of cleaned AML data set.

Figure 12: Distributions of survival time, age, age group and gender.

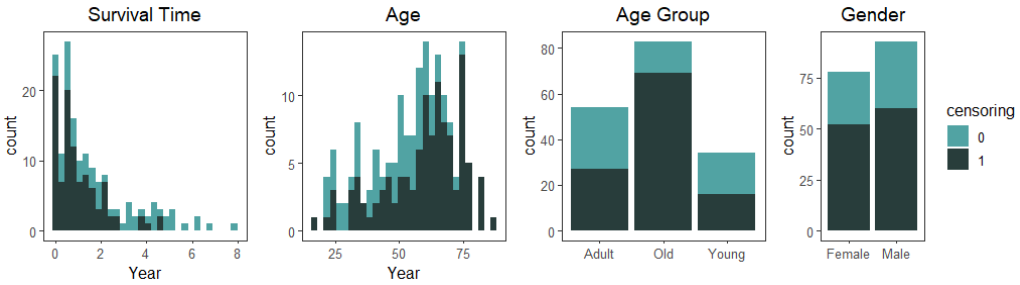
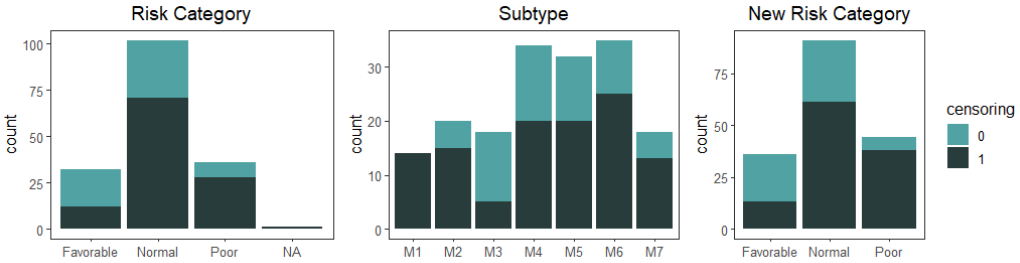


Figure 13: Distributions of risk category, subtype and new risk category.



which can be seen in Figure 12. The mean age is 55.56 years, and the median is 58 years, meaning that 50% of the patients are 58 or younger. The age of the first quartile is 45.00 years, which means that 25% of the patients were younger than 45 years at initial diagnosis. Third quartile is 67.00 years, so 75% of the patients are younger than 67 at initial diagnosis. The youngest patient was diagnosed at the age of 18 and the oldest at 88.

Newly constructed variable *age group* is divided into young, adult and old. The first group includes everyone 40 or younger, old includes everyone 60 or older, and everyone else is in adult category. The old category includes 48.5% of the observations, adult includes 31.6% and young has the least amount of observations at 19.9%. The distribution can bee seen in Figure 12.

Variable *risk category* is a three-level factor indicating favorable, normal or poor cytogenetic risk. The favorable class includes 18.7% of the observations and poor includes 21.1%, while normal has most of the observations at 59.6%. There is also one missing observation which can be seen in Figure 13. Variable *subtype* has seven levels, which indicate different AML subtypes.

The largest subtype M6 has 20.5% of the observations, and smallest group M1 has 8.2% records. These two variables will be used to create the new response variable.

3.3 Defining a New Response Variable

The response variable *new risk category* is created based on variables *risk category* and *subtype*. First, the risk category is duplicated, and all observations where subtype is M3 are changed to favorable. Then, records with subtype M1 are changed to poor in the new risk group. The reasoning behind this can be seen in the Kaplan–Meier curve 12. The subgroup M1 has the lowest survival probability out of all subgroups. On the other hand, those with subgroup M3 have the highest survival probability.

The Kaplan-Meier plots of risk category and new risk in Figure 12, show that groups poor and normal separate better in the new risk variable compared to the original risk category. This gives support for defining a new response variable. Variables survival time, and gender were also considered to be used as a response variable, but they were deemed as unusable. The survival times had a lot of censored values, which would have added extra complexity to the methods, and differences between male and female patients were not found in regards to survival probabilities on Kaplan-Meier curves.

In order to use the multiclass variable for neural networks in Keras, the *new risk category* needs to be encoded. Instead of having data with labels favorable, normal, poor, it is turned into zeros and ones. This is done by creating three columns, which represent the labels, and assigning 1 to the column for which the observation belongs. Figure 15 shows an example of how the encoding is done for a three-class variable.

Figure 14: Kaplan–Meier plots of risk category, subtype and new risk group.

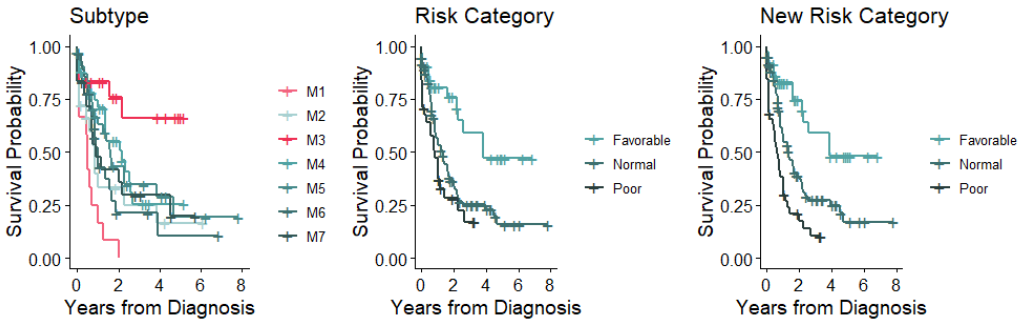
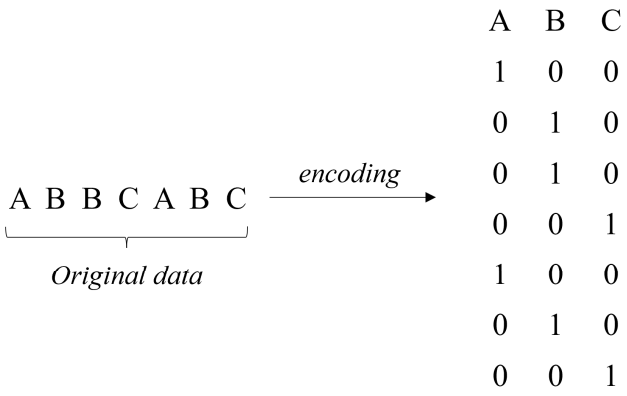


Figure 15: Example of encoding a three-class variable.



4 Methodological Contribution

In this chapter the implementation details are introduced. The coding environment is presented in the beginning, followed by the feature selection implementation. Then, the six data sets are introduced, and lastly the neural network architectures are presented.

4.1 Technical Information

The implementation is done in RStudio 2022.02.2, with version 4.0.5 of R. The use of R was decided in order to utilize a custom function developed by Juho Kontio and his colleagues. The artificial neural network is implemented with Keras version 2.6.1.9000, and version 2.7.0 of Tensorflow. Keras is a neural network API developed on top of TensorFlow using Python. Thus, using Keras requires the installation of Python for which version 3.9.7 is used. The implementation is done in a 64-bit Windows 10 laptop that has 8.00 GB of RAM, and an Intel Core i5-1035G1 Processor.

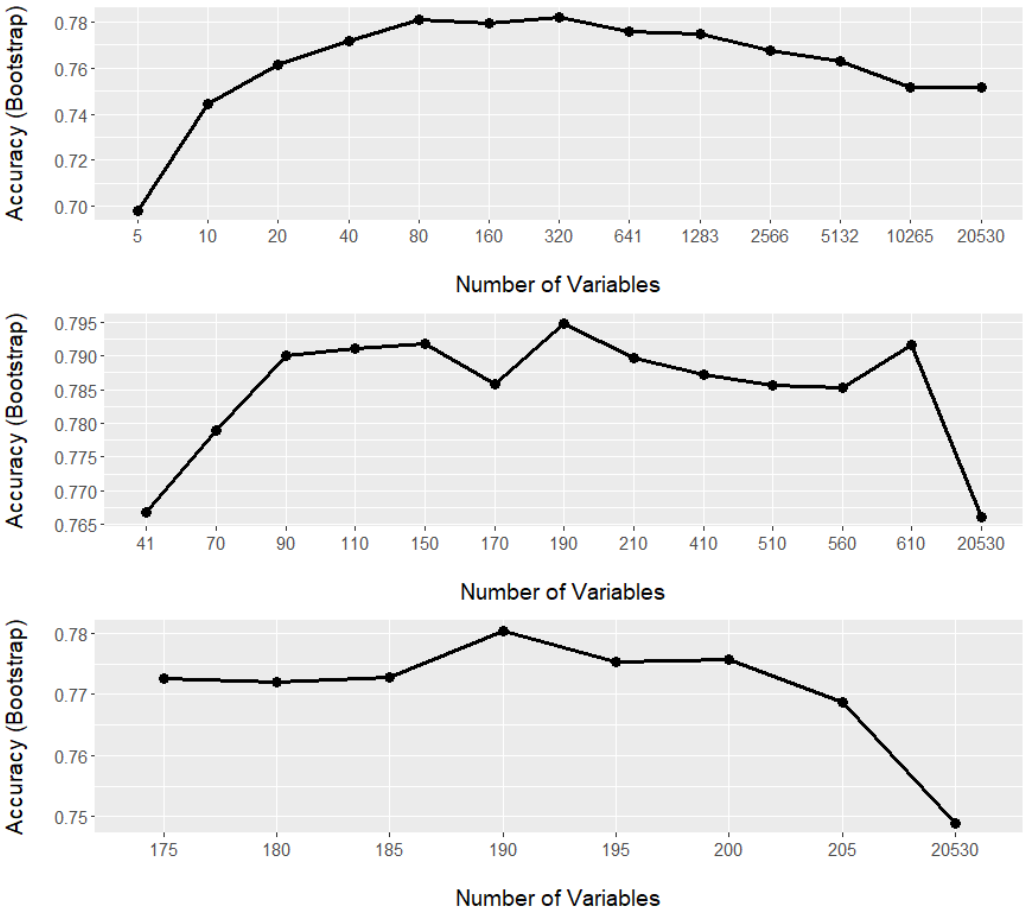
4.2 Feature Selection

The feature selection is implemented with R function **rfe** [3], by using random forest for the classification task. The function requires to set the number of features manually, which were tuned over multiple iterations. In the first iteration, the original number of features was divided by two, to get 10265, 5132, 2566, 1283, 641, 320, 160, 80, 40, 20, 10, and 5 features. The best accuracies seemed to come between 40 and 641 features.

In the next iteration, numbers 41, 70, 90, 110, 150, 170, 190, 210, 410, 510, 560, and 610 were tested. The results showed a small spike between 170 and 210, so the next iteration was done with sequence from 175 to 205 with step size of 5. As seen in Figure 17, the second plot shows a small spike also around 610. Values around this were also tested, but the accuracy did not improve. Therefore based on maximum bootstrapped accuracy 190 genes were selected. A list of the selected genes can be found in the GitHub repository[4].

[3] http://topepo.github.io/caret/recursive-feature-elimination.html
[4] https://github.com/HeliJulia/AML-Risk-Group-Prediction

Figure 16: Accuracy for different number of variables chosen with RFE.



4.3 The Data Sets

The data is split into training and test sets using stratified sampling with *new risk category* as the response variable. After the split 80% of the observations belong to the training set, and 20% to the test set. The number of observations in the sets are 135 and 36 respectively. The sets are saved into separate files to make sure that the split stays the same for each model. Table 4 shows all six training sets, along with the number of observations in poor P, normal N and favorable F classes, as well as the number of samples and variables. Note that the response variable is counted as well. The number of observation in the test set stays the same for each method, so that the poor category has 9 observation, normal category has 19, and the favorable has 8 observations.

The functions used for SMOTE and ADASYN are able to create samples only for a binary variable. Therefore, the training data needs to be split into two data sets before creating new samples. The first data set includes all observations belonging to favorable and normal classes, and the second set includes the samples from normal and poor categories. This way, the minority samples can be created separately for the poor and favorable classes.

For all the methods, the number of nearest neighbors is set to 5. Additionally, the oversampling percentage for SMOTE is set to 200%. For ADASYN, the algorithm is run twice for both the poor and favorable classes to get enough observations to balance the data. After the oversampling is complete, the new samples are selected randomly so that the training sets are balanced.

Data Set	*P*	*N*	*F*	*Samples*	*Variables*
no pre-processing	35	72	28	135	20 531
SMOTE	72	72	72	216	20 531
ADASYN	72	72	72	216	20 531
RFE	35	72	28	135	191
RFE + SMOTE	72	72	72	216	191
RFE + ADASYN	72	72	72	216	191

Table 4: Class distribution in the training sets.

4.4 Neural Network Architectures

In order to compare the performance of different sampling methods with and without feature selection, multiple neural network architectures are tested. For all of these architectures, the following characteristics stay the same. The output layer has three nodes which represent favorable, normal and poor groups in *new risk category*. Each model also uses multi-categorical cross-entropy as the cost function, and ReLu as the activation function for the hidden layers. The output layer in each network uses softmax activation function, and after every hidden layer batch normalization and dropout are used. The number of nodes in the input layer is 20 150 without feature selection and 190 with feature selection. This number represents the genes to predict from.

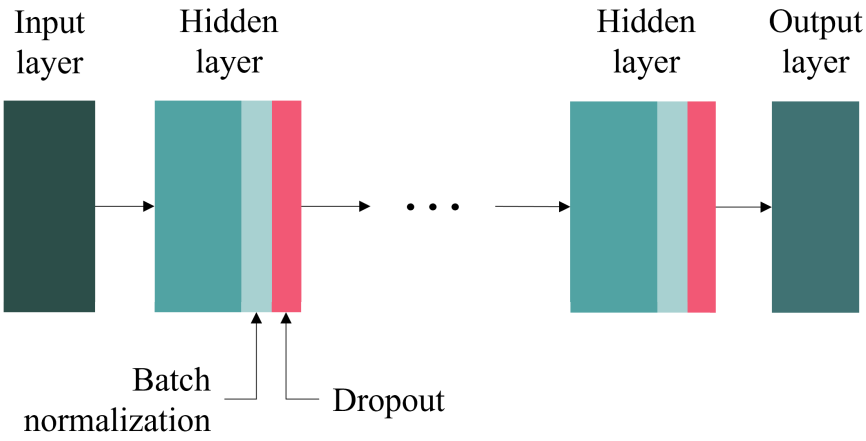
Apart from these characteristics, five hyperparameters are tuned. The parameters include number of hidden layers, number of hidden nodes in each layer, dropout rate, learning rate and batch size. These parameters created 3 168 possible model configurations, and Table 5 shows which parameter values were used to create the configurations. The number of nodes in each hidden layer stays the same for the same network. As mentioned before, the network utilizes optimizer Adam, which is used with the default values, the learning rate being an exception to this.

From the full set of configurations, a total of 100 models are randomly chosen for tuning. Each model was trained for 200 epochs with the data sets that do not use RFE. The data sets with reduced features use 190 as the number of nodes in the hidden layers. This is to reduce unnecessary complexity as per Occam's razor rule. Finally, each model uses accuracy as the stopping criteria, so that the training is halted if accuracy stops improving for 20 epochs. The set of 100 randomly chosen configurations can be found in the GitHub repository.

Table 5: Set of hypermarameters for tuning.

parameter	*values*
Number of hidden layers	3, 4, 5, ..., 12, 13
Number of hidden nodes	256, 512, 1024, 2048
Drop rate	0, 0.1, 0.2, 0.3, 0.4, 0.5
Learning Rate	0.0001, 0.001, 0.01
Batch Size	32, 64, 128, 171

Figure 17: Neural network structure.



5 Results

This chapter summarizes the findings from the implemented networks. The results are divided into models without feature selection, models with feature selection, and overall best performing models.

5.1 Models without feature selection

Based on accuracies in Table 9, the best performing model was a 7-layer neural network, with 1024 units and ADASYN sampling. The overall accuracy of this method was 0.83.

On Table 10, the best F_1-score for the favorable category was 0.93, with 4 hidden layers, 256 units and SMOTE sampling. While the F_1-score for favorable category was high, the accuracy was only 0.72, and F_1-scores for normal and poor risk groups were 0.69 and 0.64. This model is able to classify the favorable risk group quite well, but the normal and poor categories do not perform as well. For all the models in Table 10, the poor category has a low F_1 score. The accuracies of all of these models is lower than the accuracies in Table 9.

The models with the best normal F_1 scores for the normal category can be seen in Table 11. Two of the best performing models are the same as in Table 9. Finally, the Table 12 shows the highest F_1-scores for the poor category. The second best performing model is also found as the best based on overall accuracy and F_1-score for the normal risk group.

The model that maximizes the F_1-scores for all categories will be considered as the best model, because it indicates that the classification is successful for each class. Out of all the models in Tables 9, 10, 11 and 12, the model with 7 hidden layers, 1024 hidden units, drop-rate of 0.5, learning rate 0.01, batch size 32, and ADASYN as the sampling method, has the highest accuracy, and performs well for each category.

5.2 Models with feature selection

Adding feature selection to the data preprocessing changes the results. The accuracy seems to be higher in the models, and improvement in performance can be seen especially in the poor category.

The Table 13, ranks the models based on the best overall accuracy. Here, the best model has accuracy of 0.861, while without feature selection the best value is 0.833. Surprisingly, in Table 14, none of the best methods have sampling. The best F_1 score for the favorable category is a little bit smaller than without feature selection, but the overall accuracy is higher.

Looking at Table 15, the best model uses SMOTE for balancing the data, and has F_1-score of 0.89. This is a little bit higher than the F_1-score for normal category without RFE in Table 11. Finally, the poor category shows notably better results with feature selection. The highest F_1-score is 0.84, while without feature selection the best achievable value was 0.76.

The best performing model is chosen on the same basis as before. The highest accuracy was found with a model consisting of 11 hidden layers, 190 hidden units, drop rate of 0, learning rate 0.01, batch size 32, and SMOTE as the sampling method. Next, this model is compared with the best model without feature selection.

5.3 The Best Models

Tables 6 and 7 show the confusion matrices on the test set for the best models without and with feature selection. The number of predictions for the observations that are truly in the favorable class is identical. The number of correct classifications on favorable class is 7, and only one observation is miss-classified as poor with both models. The confusion matrices are also identical when it comes to the number of samples that were classified as favorable. With both models, two observations from the normal class were wrongly classified as favorable. Because these numbers are identical, the precision and recall in the favorable class are 0.778 and 0.875 for both models.

When looking at the normal class in the confusion matrices, both models miss-classify two observations as normal. The model without feature selection seems to perform a little bit better since all remaining observations are correctly classified as normal. The model with feature selection miss-classified one observation as poor. The precisions for the models were 0.84 and 0.88, the better value belonging to the model without feature selection.

On the other hand, when looking at the number of classifications for the normal class, the model with feature selection seems to perform better. While the model with feature selection wrongly classified one sample from the poor class as normal, the model without feature selection wrongly classified three. The recalls for the models were 0.94 and 0.82, and this time the model with feature selection performed better.

Lastly, the number of correct classifications in the poor class was six without and eight with feature selection. Without feature selection, one sample from the favorable class was wrongly classified as poor, and three observations in the poor class were miss-classified as normal. With the second model, there was only one miss-classified poor sample, and two wrongly classified as poor. The recall of the poor category was better for the model without feature selection, but the model with feature selection performed better in terms

of precision. The values for recall were 0.93 and 0.96 and for the precision 0.96 and 0.90.

The confusion matrices 6 and 7 are used to create the metrics found in Table 8. The values were obtained with function **confusionMatrix**[5]. From the two models, the best overall accuracy was achieved with feature selection. The 95% confidence interval (CI) for the accuracy was (0.71, 0.95) with feature selection and (0.67, 0.94) without. With feature selection, the CI is smaller. The p-values were also automatically calculated with a one-sided test for the accuracy to be better than the no-information rate. Both of the p-values are quite small, which indicates that the accuracies of the models are higher than the no-information rate. This, in turn, means that the model is predicting more than just the majority class. While the p-values should never be taken as the absolute truth (Amrhein et al., 2019), the confusion matrix shows that the models can give correct classifications also for the minority classes.

Since the p-value and confidence interval with feature selection are smaller than without, the model with feature selection is performing better. This finding is supported by the fact that the AUCs are higher in the normal and poor classes for the RFE model. In the favorable class, the AUC is the same in both models. It is important to note that the number of samples in the test set is small, so the results might be different if the number of observations was increased. Nevertheless, both of the best models can predict each class with quite a high accuracy, so the network tuning was successful.

[5]https://www.rdocumentation.org/packages/caret/versions/3.45/topics/confusionMatrix

Table 6: Confusion matrix for the best model without feature selection.

	True value		
Prediction	Favorable	Normal	Poor
Favorable	7	2	0
Normal	0	17	3
Poor	1	0	6

Table 7: Confusion matrix for the best model with feature selection.

	True value		
Prediction	Favorable	Normal	Poor
Favorable	7	2	0
Normal	0	16	1
Poor	1	1	8

Table 8: Comparison between the best models.

Metric	ADASYN	RFE+SMOTE
Overall accuracy	0.83	0.86
95% CI	(0.67, 0.94)	(0.71, 0.95)
p-value	0.00013	2.6e-05
No Information Rate	0.53	0.53
Balanced accuracy Favorable	0.90	0.90
Balanced accuracy Normal	0.86	0.89
Balanced accuracy Poor	0.81	0.90
AUC Favorable	0.90	0.90
AUC Normal	0.86	0.89
AUC Poor	0.81	0.91

Table 9: Models with the highest overall accuracy.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
7	1024	0.5	0.01	32	ADASYN	0.833	0.82	0.87	0.75
12	2048	0.1	0.01	32	SMOTE	0.806	0.82	0.85	0.67
5	256	0.4	0.001	64	SMOTE	0.806	0.82	0.83	0.74
3	512	0.5	1e-04	32	original	0.778	0.82	0.81	0.67
3	512	0.5	1e-04	32	original	0.778	0.82	0.81	0.67

Table 10: Models with the highest F_1 score for favorable category.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
4	256	0.1	0.001	32	SMOTE	0.722	0.93	0.69	0.64
8	512	0.3	0.01	32	original	0.694	0.88	0.72	0.5
12	2048	0	0.01	32	original	0.639	0.88	0.67	0.4
7	2048	0.3	1e-04	32	original	0.750	0.88	0.79	0.56
4	256	0.1	0.001	32	original	0.722	0.88	0.74	0.57

Table 11: Models with the highest F_1 score for normal category.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
7	1024	0.5	0.01	32	ADASYN	0.833	0.82	0.87	0.75
12	2048	0.1	0.01	32	SMOTE	0.806	0.82	0.85	0.67
5	512	0.4	0.01	64	original	0.750	0.55	0.83	0.67
5	512	0.4	0.01	64	original	0.750	0.55	0.83	0.67
13	2048	0	0.01	32	ADASYN	0.778	0.88	0.83	0.53

Table 12: Models with the highest F_1 score for poor category.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
8	1024	0.2	0.01	32	ADASYN	0.778	0.78	0.79	0.76
7	1024	0.5	0.01	32	ADASYN	0.833	0.82	0.87	0.75
5	256	0.4	0.001	64	SMOTE	0.806	0.82	0.83	0.74
5	256	0.1	1e-04	32	original	0.722	0.70	0.74	0.71
5	256	0.1	1e-04	32	original	0.722	0.70	0.74	0.71

Table 13: Models with the highest accuracy.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
11	190	0	0.01	32	RFE+SMOTE	0.861	0.82	0.89	0.84
4	190	0.1	0.01	128	RFE	0.833	0.78	0.86	0.82
3	190	0.5	1e-04	32	RFE	0.833	0.82	0.86	0.78
7	190	0.1	1e-04	171	RFE+ADAS	0.833	0.88	0.86	0.74
10	190	0.3	0.01	64	RFE+ADAS	0.833	0.82	0.86	0.78

Table 14: Models with the highest F_1 score for favorable category.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
6	190	0.1	1e-04	128	RFE	0.778	0.89	0.79	0.63
4	190	0.2	1e-04	128	RFE	0.750	0.88	0.78	0.60
5	190	0.2	1e-04	32	RFE	0.778	0.88	0.81	0.63
8	190	0.1	0.001	64	RFE	0.750	0.88	0.81	0.43
3	190	0.1	0.001	32	RFE	0.750	0.88	0.76	0.64

Table 15: Models with the highest F_1 score for normal category.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
11	190	0	0.01	32	RFE+SMOTE	0.861	0.82	0.89	0.84
8	190	0	0.01	32	RFE+SMOTE	0.833	0.82	0.87	0.75
4	190	0.1	0.01	128	RFE	0.833	0.78	0.86	0.82
9	190	0	0.01	64	RFE	0.806	0.74	0.86	0.78
3	190	0.5	1e-04	32	RFE	0.833	0.82	0.86	0.78

Table 16: Models with the highest F_1 score for poor category.

layers	units	drop	lr	batch	method	accuracy	f1_f	f1_n	f1_p
11	190	0	0.01	32	RFE+SMOTE	0.861	0.82	0.89	0.84
4	190	0.1	0.01	128	RFE	0.833	0.78	0.86	0.82
5	190	0.4	0.01	64	RFE	0.806	0.78	0.82	0.80
12	190	0.2	0.01	128	RFE+SMOTE	0.833	0.82	0.86	0.80
9	190	0	0.01	64	RFE	0.806	0.74	0.86	0.78

6 Discussion and Conclusions

This thesis demonstrated how RFE, SMOTE, and ADASYN can be implemented for a multi-class classification problem, where the data is imbalanced. The results showed slightly better performance when using SMOTE and RFE, compared to using only ADASYN. While each method yielded better results compared to the original data set, surprisingly, the networks were able to give high accuracies even without any preprocessing. This demonstrates why finding the proper network structure is important, especially if the data set includes complex connections.

The neural networks were able to give high accuracies for the minority classes, which indicates that predicting the newly created risk group for AML patients based only on genes is possible. Furthermore, when using optimizer, batch normalization, and regularization techniques, even with limited resources the complex structures can be modeled using the feed-forward neural networks.

Lastly, the results demonstrated why comparing multiple evaluation metrics is important. For some models, recall might be high even if the precision is low in a class, and vice versa. Looking at the confusion matrix gives a good idea of how the models are performing, and shows if some classes are systematically classified wrong. Therefore, the confusion matrix is a good starting point for assessing the classifier's performance.

A major challenge throughout this thesis was the lack of computing power, which limited the choice of usable algorithms. While testing some of the more computationally expensive feature selection methods, the capacity of the laptop exceeded in minutes. For example R-package *FSelector* had a lot of interesting algorithms to choose from, but each of the methods was unable to run with the full data set.

Another challenge was finding the right network structure. In the literature, a common guideline is to include only 2-5 layers in the network. Though I believe this guideline is valid for most scenarios to avoid overfitting, genes have such complex structures and connections between one another that a few layers is not enough. If neural networks are to be used to predict from the gene expressions, adding more layers and nodes should not be feared.

Having said that, with proper equipment and knowledge about genes, using neural networks for prognosis and finding treatments for AML seems possible based on the results. Here only gene expressions were used as predictors, but adding clinical variables to the mix might yield even better results. With the addition of clinical variables and removal of redundant features, the dimensions to sample from changes. This can effect the nearest neighbors in ADASYN and SMOTE algorithms, which in turn changes where the new

samples are created.

In addition to these points, scaling the variables before predicting is something that is missing from this thesis. Also, AML was the only cancer considered, so using these methods for different cancers could be attempted. Additionally, while neural networks are designed to learn patterns from the data, they give no information on which genes are used to form patterns. The genes do not function alone, so this an important point to consider. Finally, only one type of neural network was tested within this thesis. There are many more network architectures available, including different optimisation schemes, activation functions and hyperparameter tuning options.

To conclude, predicting from 20 000 variables using 200 observations with limited resources is a difficult, but possible task. It requires putting special attention to details and utilizing multiple methods in each step. While this thesis was done without any prior knowledge about gene expressions, the results might give new insights into cancer research, and provide ideas on how to efficiently utilize neural networks on high-dimensional data.

References

Amrhein, V., Greenland, S., and McShane, B. (2019). Retire statistical significance. *Springer Nature Limited*, 567:305–307. https://doi.org/10.1038/d41586-019-00857-9.

Baldi, P. and Hatfield, G. W. (2002). *DNA microarrays and gene expression: From experiments to data analysis and Modeling*. Cambridge University Press. https://doi.org/10.1017/CBO9780511541773.

Bates, S. A. (n.d.). *Deoxyribonucleic acid (DNA)*. National Human Genome Research Institute. Retrieved August 4, 2022, from https://www.genome.gov/genetics-glossary/Deoxyribonucleic-Acid.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer New York. https://link.springer.com/gp/book/9780387310732.

Bolón-Canedo, V., Sánchez-Maroño, N., and Alonso-Betanzos, A. (2015). *Feature selection for high-dimensional data*. Springer Cham. https://doi.org/10.1007/978-3-319-21858-8.

Bomheke, B. and Greenwell, B. (2020). *Hands-on machine learning with R*. Chapman & Hall/CRC. https://bradleyboehmke.github.io/HOML/.

Buono, P.-L. (2016). *Advanced calculus: Differential calculus and Stokes' theorem*. De Gruyter. https://doi.org/10.1515/9783110438222.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357. https://doi.org/10.1613/jair.953.

Clark, T. G., Bradburn, M. J., Love, S. B., and Altman, D. G. (2003). Survival analysis part I: Basic concepts and first analyses. *British Journal of Cancer*, 89(2):232–238. https://doi.org/10.1038/sj.bjc.6601118.

Fernandez, A., Garcia, S., Herrera, F., and Chawla, N. V. (2018). SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, 61(1):863–905. https://doi.org/10.1613/jair.1.11192.

Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., and Herrera, F. (2018). *Learning from imbalanced data sets*. Springer Cham, 1st edition. https://doi-org.pc124152.oulu.fi:9443/10.1007/978-3-319-98074-4.

Feurer, M. and Hutter, F. (2019). *Hyperparameter optimization*, pages 3–33. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-05318-5$_1$.

Grandini, M., Bagli, E., and Visani, G. (2020). Metrics for multi-class classification: An overview. *arXiv*, 05756. https://doi.org/10.48550/arxiv.2008.05756.

Jager, K. J., van Dijk, P. C., Zoccali, C., and Dekker, F. W. (2008). The analysis of survival data: The Kaplan–Meier method. *Kidney International*, 74(5):560–565. https://doi.org/10.1038/ki.2008.217.

Jeon, H. and Oh, S. (2020). Hybrid-recursive feature elimination for efficient feature selection. *Applied Sciences*, 10(9):3211. https://doi.org/10.3390/app10093211.

Kantarjian, H., Kadia, T., DiNardo, C., Daver, N., Borthakur, G., Jabbour, E., Garcia-Manero, G., Konopleva, M., and Ravandi, F. (2021). Acute myeloid leukemia: Current progress and future directions. *Blood Cancer Journal*, 11(2):41. https://doi.org/10.1038/s41408-021-00425-3.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324. https://doi.org/10.1016/S0004-3702(97)00043-X.

Kotsiantis, S. B., Kanellopoulos, D. N., and Pintelas, P. E. (2006). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.9248&rep=rep1&type=pdf.

Liu, H. and Motoda, H. (2008). *Computational Methods of Feature Selection*. Chapman & Hall/CRC.

Luengo, J., García-Gil, D., Ramírez-Gallego, S., García, S., and Herrera, F. (2020). *Big data preprocessing: Enabling smart data*. Springer Cham, 1st edition. https://doi-org.pc124152.oulu.fi:9443/10.1007/978-3-030-39105-8.

National Human Genome Research Institute (n.d.a). *Cytogenetics*. Retrieved August 8, 2022, from https://www.genome.gov/genetics-glossary/Cytogenetics.

National Human Genome Research Institute (n.d.b). *Gene Expression*. Retrieved August 4, 2022, from https://www.genome.gov/genetics-glossary/Gene-Expression.

Noren, D. P., Long, B. L., Norel, R., Rrhissorrakrai, K., Hess, K., Hu, C. W., et al. (2016). A crowdsourcing approach to developing and assessing prediction algorithms for AML prognosis. *PLoS Computational Biology*, 12(6):1004890. https://doi.org/10.1371/journal.pcbi.1004890.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv*, 1609.04747. http://arxiv.org/abs/1609.04747.

Sen, S. K. (n.d.). *Messenger RNA (mRNA).* National Human Genome Research Institute. Retrieved August 4, 2022, from https://www.genome.gov/genetics-glossary/messenger-rna.

Smith, M. (n.d.). *Microarray technology.* National Human Genome Research Institute. Retrieved August 4, 2022, from https://www.genome.gov/genetics-glossary/Microarray-Technology.

Theodoridis, S. (2020). *Machine learning: A bayesian and optimization perspective*. Academic Press, 2nd edition. https://doi.org/10.1016/C2019-0-03772-7.

Venkatesh, B. and Anuradha, J. (2019). A review of feature selection and its methods. *Cybernetics and Information Technologies*, 19(1):3–26. https://doi.org/10.2478/cait-2019-0001.