Pavel Viitasalo

Applying Machine Learning to Predict Mortality of Heart Failure

A Design Science Case Study

School of Technology and
Innovation
Master of Science in Economics
and Business Administration
Information Systems

Vaasa 2025

UNIVERSITY OF VAASA
School of Technology and Innovation
Author: Pavel Viitasalo
Title of the thesis: Applying Machine Learning to Predict Mortality of Heart Failure: A Design Science Case Study
Degree: Master of Science in Economics and Business Administration
Discipline: Information Systems
Supervisor: Timo Mantere
Year: 2025 **Pages:** 77

ABSTRACT:

Tässä tapaustutkimuksessa sovelletaan koneoppimista kliiniseen tietoaineistoon ennustamaan sydämen vajaatoiminnasta johtuvaa kuolemaa. Työn rajaus on soveltaa muuttujien karsimista ja hyperparametrien optimointia kaikille koneoppimismalleille. Tavoitteena on rakentaa toimiva luokitteleva koneoppimismalli ja tutkia, parantaako muuttujien karsiminen ja mallien hyperparametrien optimoiminen ennustustarkkuutta verrattuna optimoimattomiin ja kaikille muuttujille ennustaviin malleihin. On myös oleellista tutkia, voiko koneoppimista soveltaa kliiniseen tietoaineistoon onnistuneesti.

Työ toteutetaan suunnittelututkimuksen (design science) ja tapaustutkimuksen hybridinä. Ensin määritellään tutkimuksen kannalta oleelliset käsitteet ja tuodaan esille aiempi aiheeseen liittyvä tutkimus. Sen jälkeen suoritetaan datan keruu, prosessointi, mallien rakennus, muuttujien karsiminen, hyperparametrien optimointi, ja lopuksi esitetään tulokset vertailemalla optimoimattomien mallien sekä optimoitujen mallien ennustustarkkuutta. Tutkimuksessa otetaan kantaa myös aiemman tutkimuksen mallien suoriutumiseen ja vertaillaan niiden sekä tämän tutkimuksen tuloksia. Työ päätetään johtopäätöksiin ja keskusteluun. Koneoppimismallit rakennetaan Python-ohjelmointikielen koneoppimiskirjastoilla. Muuttujien valinnassa käytetään korrelaatiotutkimusta ja Random Forest -koneoppimismallin mean decrease in impurity ja permutaatiomerkityksiä. Muuttujat valitaan lopuksi näiden menetelmien valitsemasta kokonaisuudesta. Edeltäviä tutkimuksia käytetään myös hyödyksi erottamaan tärkeimmät muuttujat. Hyperparametrien optimoinnissa käytetään sen sijaan RandomizedSearchCV-algoritmia.

Muuttujien karsimisen ja hyperparametrien optimoinnin jälkeen parhaan tarkkuuden saavuttanut koneoppimismalli on K-Nearest Neighbors. Malli saavuttaa ennustustarkkuuden 81,67 %. Valitut muuttujat ovat ejektiofraktio, kreatiniinin määrä veressä ja ikä. Muuttujien määrä saadaan pienennettyä yhdestätoista kolmeen. Muuttuja seuranta-aika jätetään pois jo aiemmassa vaiheessa, sillä selviytyminen on riippuvainen tämän muuttujan korkeasta arvosta. Lisäksi lähes kaikkien koneoppimismallien ennustustarkkuus paranee muuttujien karsimisen ja hyperparametrien optimoinnin jälkeen; yhdenkään mallin tarkkuus ei heikenny. K-Nearest Neighbors on myös yksi heikoimman tarkkuuden omaavista koneoppimismalleista ennustettaessa kaikkien muuttujien avulla. Täten muuttujien karsiminen ja hyperparametrien optimointi todetaan toimiviksi menetelmiksi tähän aineistoon sovellettaessa. Koneoppimista voidaan myös soveltaa tähän kliiniseen tietoaineistoon onnistuneesti. Jotta lopullista koneoppimismallia voitaisiin käyttää oikeassa tilanteessa, kuten sairaalassa, tulee mallia kuitenkin testata tässä tutkimuksessa käytetyn aineiston ulkopuolisella datalla. Lisäksi aineistossa on vain 299 tietopistettä, joka voi rajoittaa mallin kykyä ennustaa suuremmassa mittakaavassa.

KEYWORDS: machine learning, classification, mortality, prediction, data

Contents

Figures

Tables

Algorithms

Abbreviations

ANN	Artificial Neural Network
ETC	ExtraTreesClassifier
FN	False negative
FP	False positive
GBC	GradientBoostingClassifier
GSCV	GridSearchCV
KNN	K-Nearest Neighbors
ML	Machine Learning
RSCV	RandomizedSearchCV
SVM	Support Vector Machine
TN	True negative
TP	True positive
XGBoost	Extreme Gradient Boosting

1 Introduction

According to OECD and European Commission (2024), the European health personnel face a severe shortage. *Twenty EU countries reported a shortage of doctors in 2022 and 2023, while 15 countries reported a shortage of nurses. Based on minimum staffing thresholds for universal health coverage (UHC), EU countries had an estimated shortage of approximately 1.2 million doctors, nurses and midwives in 2022* (OECD/European Commission, 2024). Furthermore, the United States has spent an average of 2.7 trillion USD annually on chronic disease treatment, which comprises 18% of the annual GDP of the US (Chen et al., 2017). According to them (Chen et al., 2017), the problem is also prevalent in China, where 86.6 % of deaths are caused by chronic diseases.

The situation is especially severe in Eastern Finland, where the emergency room shutdowns of Varkaus and Iisalmi at the start of year 2026 are expected to lengthen the already long queues further (Nyyssönen, 2025). The health personnel of Kuopio University hospital are exhausted, and even the regional administration senior medical officer is worried about the situation (Nyyssönen, 2025). Fortunately, artificial intelligence trials are being tested in some regions in, e.g., automation of assessment of need for medical services and patient information systems, as well as multilingual customer service (Roiha, 2025).

Ahmad et al. (2017a) describe heart failure as a state in which the heart wall's muscles fade and enlarge, thus limiting blood pumping. As the ventricles of heart get inflexible and don't fill properly between beats, the heart doesn't support the blood demand in the body, and consequently patient feels difficult to breath (Ahmad et al., 2017a). Breathing difficulties can arise while resting when the disease progresses (Lommi et al., 2023). Furthermore, according to Lommi et al. (2023), swelling can appear on ankles shins, and body weight can increase by a couple of kilograms in a short period due to liquid gathering in the body as the heart's pumping power.

According to McDonagh et al. (2024), the prevalence of heart failure is between 1 and 2% of the whole population. The figure is probably larger however, since it is based on diagnoses which are verified (Lommi et al., 2023). Heart failure is described as a disease of the elderly, since approximately 1% of population under 55-year-olds and approximately 10% of population of 70-year-olds suffer from it (Lommi et al., 2023). According to Lommi et al. (2023), heart failure is separated into diastolic and systolic, where the former is more common and less serious alternative. In diastolic heart failure, the left ventricular ejection fraction is normal or mildly decreased ($\geq 50\%$ or $41-49\%$) whereas in systolic heart failure it's decreased ($\leq 40\%$) (Lommi et al., 2023). Ejection fraction means the percentage of blood leaving the heart at each contraction (Chicco & Jurman, 2020). It is estimated that as the prevalence of heart failure increases in the next decades (as much as 50%), the demand for hospital care will increase rapidly (Lommi et al., 2023). This is one of the main motivations for the thesis; to make the work of healthcare professionals easier as the need for hospital visits, and consequently labor increases.

Today, machine learning (ML) is used extensively in medicine applications to diagnose and identify disease, discover and manufacture drugs, image medical data, and predict diseases (Rahmani et al., 2021, p. 3). As an example, Al Younis et al. (2024) have developed a Decision Tree model to heart failure patients with distinct left ventricular ejection fraction levels using circadian ECG features and ML. The Decision Tree reaches an accuracy of 91.2 %. More examples, especially related to heart failure and heart failure mortality, are introduced in chapter 2.4, where prior research on applied ML on heart failure will be done.

The goal of this thesis is to apply ML to predict heart failure mortality using the Heart Failure Clinical Records data set (2020) from UCI Machine Learning Repository. Well-known ML algorithms from earlier research will be used, and later feature selection and hyperparameter tuning will be applied on all the models. As a personal motivation, this thesis also aims to ease the burden of healthcare personnel and professionals by possibly

identifying the risk of heart failure death before a patient's situation will become worse. The target audience is therefore healthcare personnel and professionals, data scientists in healthcare, as well as researchers who are developing ML models for heart failure mortality prediction further.

The research gap is applying RandomizedSearchCV (RSCV) (Bergstra & Bengio, 2012) on the ML models implemented in the data set Heart Failure Clinical Records (2020). Since Chicco and Jurman (2020) used GridSearchCV (GSCV) during hyperparameter tuning, this thesis uses a different ML approach process. They also applied GSCV only in the initial stage, where all the features were present. Tuning hyperparameters with selected features could uncover additional prediction power, which will be done in this thesis. Furthermore, there has not been much published research on the data set. The data set is also a more challenging one than many in the Prior Research chapter (2.4), since the number of data points (patients) is very low in comparison. The number of positive target variable instances (deceased during the follow-up period in this case) is 96, while 203 survived, so there should be enough data for effective training of the ML models. This will be further elaborated in chapter 4, where the data will be preprocessed and analyzed.

The research questions to be answered at the end of the thesis are:

1. Can ML predict heart failure patient mortality from a clinical data set?
2. Can feature selection and hyperparameter tuning improve prediction accuracy?
3. Which ML models yield the best prediction accuracy?

Lastly, the goal is to create an ML model that will predict heart failure mortality from a clinical data set. The main inspiration and comparison will be Chicco and Jurman's (2020) study on the same data set, where they discovered that only two features with a stratified logistic regression algorithm can classify heart failure with 83.8% accuracy. However, other prior research will be considered during comparison as well. Additional

sources for already existing case studies on similar data sets include but aren't limited to studies of Adler et al. (2020), Sarijaloo et al. (2021) and Escolar et al. (2022).

The thesis will have the following structure:

- Research literature will be reviewed to define the essential topics in ML, the utilized ML algorithms and validation techniques, and bring up prior research on heart failure and ML
- The research methodology will be defined
- The data will be cleaned and preprocessed
- The ML models will be created, improved and evaluated
- The results will be presented, and the research questions will be answered
- Lastly, discussion and conclusions will be brought up.

2 Literature review

This section discusses and explains the most important literature topics regarding the thesis. These include ML, different types of learning, ML models used in the thesis, and the validation methods used after training the models. Lastly, prior research, where ML is applied on similar data, will be discussed.

2.1 Machine learning

According to Alpaydin (2020, p. 21), ML has been created for a problem that does not have a distinct algorithm to solve it. For example, we can play chess and hold conversations in foreign languages without a struggle, but a machine can't. Therefore, a very general model with many parameters is first introduced to the data, and after training it with the data in question, the tweaked model will become the algorithm for the problem (Alpaydin, 2020, p. 21).

A more concrete example could be the following:

> [--] you might write a program to perform some accounting tasks for your business. In this case, the data collected would include your sales records, your inventory lists, and so on. [--] Using the same accounting example, with the machine learning paradigm, you would take the detailed sales records (which are collectively both the data and output) and use them to derive a set of rules to make predictions. You may use this model to predict the most popular items that will sell next year, or which items will be less popular going forward (Lee, 2019, p. 15).

A more detailed explanation of ML would include that every practical ML method uses a particular hypothesis space out of which the hypothesis h is chosen. This hypothesis space of candidates for the hypothesis map is an important design choice and might be based on domain knowledge (Jung, 2022, p. 14). Therefore, as Jung (2022) reasons, an example of predicting daytime temperature based on the previous day's minimum and

maximum temperatures, a higher minimum daytime temperature yields a higher maximum daytime temperature. Furthermore, a linear model can be assigned to this problem, since an increase in days increases the temperature relatively. According to Jung (2022, p. 14) the linear model could look like this:

$$h(x) \coloneqq w_1 x + w_0;\ w_1 \in R_+, w_0 \in R, \quad (1)$$

where w_1 is the relative increase in the maximum daytime temperature for an increased minimum daytime temperature, *x* is the minimum temperature for a given day, and w_0 is the maximum daytime temperature that we anticipate for a day with minimum daytime temperature equal to 0, i.e. the intercept.

These are only a handful of examples of ML implementation, and even very deep neural networks belong under this umbrella term. Generative Pre-trained Transformer 3 (GPT-3) consists of approximately 175 billion parameters (Välisuo, 2023a) which, e.g., Large Language Models such as Chat-GPT is based on and belongs under Artificial Neural Networks (ANNs) category. They are too, an ML algorithm, though much more complex than a linear model. More ML methods and algorithms will be discussed in the following chapters.

2.2 Supervised learning

According to Chopra and Khurana (2023, p. 74), Supervised Learning is an ML technique that involves mapping of input data with the corresponding output. Supervised learning is furthermore performed using an algorithm that provides a mapping function which can input with the output (Chopra & Khurana, 2023, p.74). Supervised learning is done on labeled data points, whose values are known, and are often labeled by human experts (Jung, 2022, p. 27).

The previous linear daytime algorithm predictor (Jung 2022, p. 14) belongs in this ML methodology. Since the model is trying to predict the maximum temperature for a specific day with the function:

$$y \approx w_1 x + w_0; w_1 \in R_+, w_0 \in R, \quad (2)$$

The model is in fact trying to predict a distinct, continuous value. Furthermore, since there exists a real, or correct, value for each prediction, the previous linear model belongs under supervised learning. Linear regression is also one of the simplest algorithms that you can apply to a data set to model the relationships between features and labels (Lee, 2019, p. 75).

Alpaydin (2020, p. 40) introduces a classification example, where a group of people has labeled cars that they believe are family cars as positive, and the other cars as negative examples. To have a simple example model, only car price and engine power are used as independent variables, and the other variables are deemed irrelevant. They (Alpaydin, 2020, p. 40) denote two numerical values to represent each car, and a label to denote the type:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad r = \begin{cases} 1 \; if \; \boldsymbol{x} \; is \; deemed \; a \; family \; car \\ 0 \; if \; \boldsymbol{x} \; is \; deemed \; a \; non-family \; car \end{cases}, \quad (3)$$

where x_1 is the price (e.g., in US dollars), x_2 is the engine power and r is the class that a group of people think that a given car belongs to; 1 is a family car and 0 is not. Each car has these attributes as an ordered pair (Alpaydin, 2020, p. 40). This way, the cars can be plotted in two dimensions and classified according to their classes:

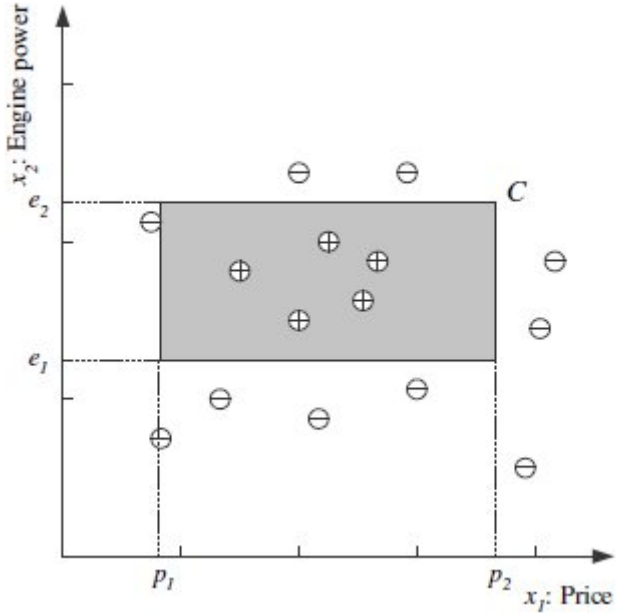

Figure 1. A two-dimensional representation of cars; price on the x-axis, engine power on the y-axis and the class marked with a + (family car) or a – (non-family car) (Alpaydin, 2020, p. 40-42).

The gray rectangle in the middle of the figure represents a hypothesis range, in which the family cars should be in upon further investigation (Figure 1). The learning algorithm then finds the hypothesis, $h \in \mathcal{H}$ where the latter is the hypothesis class that is specified by the rectangle, that approximates the real class as closely as possible (Alpaydin, 2020, p. 41).

Since in real life it is not possible to use the population of cars, it is necessary to use only a subset of them, as in the example above. Once the hypothesis class is restricted to a particular range, however, the problem becomes easier. Only the restricted area needs to be considered in this example, and the most specific hypothesis can be derived (Alpaydin, 2020, p. 41). An error hypothesis given the training set X could look like this:

$$E(h|X) = \sum_{t-1}^{N} 1(h(x^t) \neq r^t, \qquad (4)$$

where h is the hypothesis, x^t is an example car, r^t is the real type of the car, and t denotes the car's index. If the hypothesis classifies the car incorrectly, in this instance the function yields a 1, and if it classifies the car correctly, the function yields a 0 (Alpaydin, 2020, p. 41).

This method can then be specified and tried with different variations, e.g. a sphere instead of a rectangle, to make the hypothesis more accurate. (Alpaydin, 2020, p. 42). According to Jung (2022, p. 28), supervised learning tries to learn a hypothesis with minimum discrepancy between predictions and the true labels of the training set. To implement a curve for labeled data points in a training set, a loss function is needed to quantify the fitting error. Supervised ML methods differ in the loss function but follow the same basic idea to measure discrepancy between the prediction and the truth. (Jung 2022, p. 28). In short, the model learns by minimizing the error function, i.e. the average loss of each prediction and the true label (Jung, 2022, p. 15).

2.2.1 Linear Regression

Linear Regression is one of the simplest regression algorithms and it depicts the relationship between an independent variable *x* and a dependent variable *y* (Chopra & Khurana, 2023, p. 35). According to Chopra and Khurana (2023, p. 35), linear regression states that the relationship that exists between one or more input features and the relative output or target vector is approximately linear in nature. As a simple linear regression with one variable was already presented, the cost function associated with linear regression is very often average squared error loss, or mean squared error:

$$\arg\min \frac{1}{m} \sum_{i=1}^{m} \left(y^i - h(x^i) \right)^2, \qquad (5)$$

where m is the total number of inputs, y^i is the target variable and $h(x^i)$ is the output of the linear regressor (Jung, 2022, p. 98). This is what a simple linear regression $f(x) = x + 1$ looks like (see Figure 2):

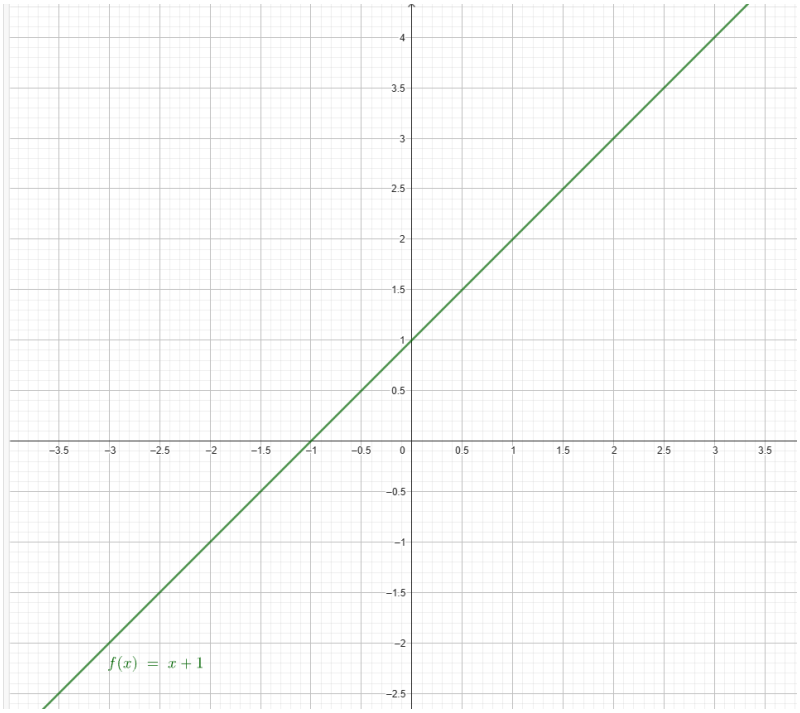

Figure 2. A figure of a simple linear regression, $\boldsymbol{f(x) = x + 1}$.

2.2.2 Logistic Regression

Logistic Regression is a statistical model that uses the logistic function to model a binary dependent variable. Its output is discrete as opposed to linear regression, where it's continuous (Chopra & Khurana, 2023, p. 38). Moreover, according to Lee (2019, p. 92), logistic regression yields a probability that a given input point belongs to a specific class. Logistic regression uses the sigmoid function, which is an inverse of the logit function, to distribute the odds of the classification that looks like this:

$$P = \frac{1}{(1 + e^{-(\beta_0 + x\beta)})}, \qquad (6)$$

where β_0 is the intercept and $x\beta$ is the coefficient. They are unknown until the training process and must be estimated with a technique known as Maximum Likelihood Estimation (Lee, 2019, p. 93). When plotted, the sigmoid function transforms the values

on $(-\infty, \infty)$ into numbers on $(0,1)$ and forms an S-like shape when plotted on the x and y-axis:

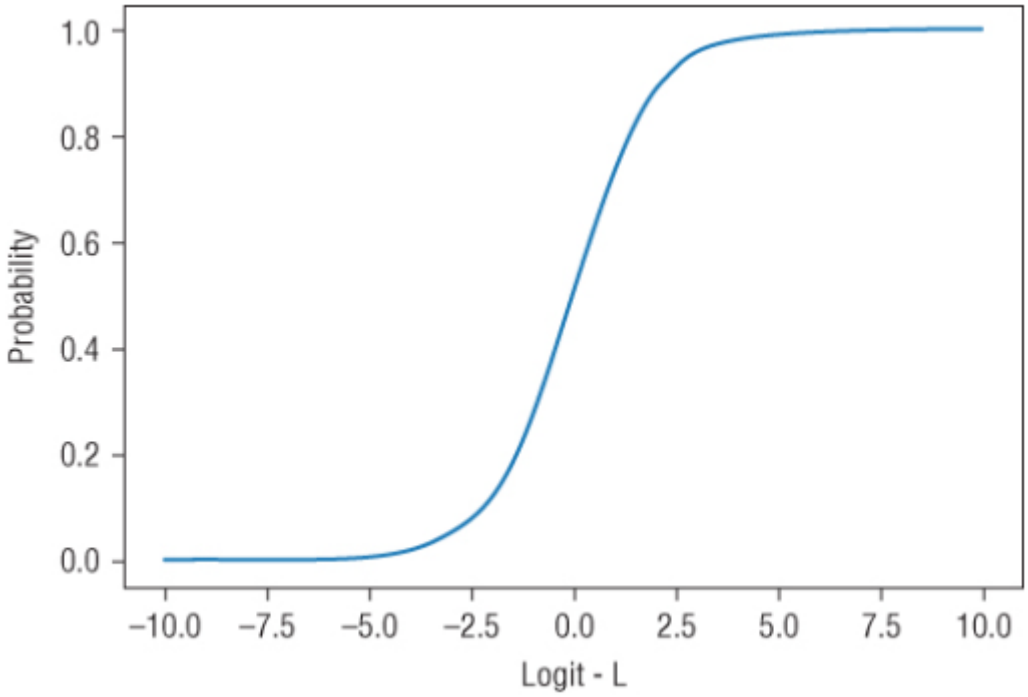

Figure 3. A plotted sigmoid function curve (Lee, 2019, p. 93-94).

where the curve fits through all the data points given during the training process (Figure 3; Lee, 2019, p. 93).

2.2.3 Gradient descent

To understand the training process and the error function, gradient descent is presented before more complicated ML algorithms. Gradient descent is a necessary method when minimizing the value of the cost function (Chopra & Khurana, 2023, p. 37). According to Alpaydin (2020, p. 187), when w denotes the set of parameters and $E(w|X)$ is the error with parameters w on the given training set X, we are trying to find

$$w^* = \arg\min_w E(w|X), \quad (7)$$

where, as earlier stated, $E(w|X)$ is the error with parameters on the given training set. There is no analytical solution, and it needs to be iteratively optimized, following that gradient descent must be used (Alpaydin, 2020, p. 187). When $\mathrm{E}(\mathrm{w})$ is a differentiable function of a vector of variables, the gradient vector looks like

$$\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_d}\right]^T, \quad (8)$$

where the inside of the brackets is composed of all the partial derivatives (Alpaydin, 2020, p. 187). Gradient descent starts minimizing the error E from a random set of parameters w and updates the latter at each step in the opposite direction of the gradient:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i; w_i = w_i + \Delta w_i, \qquad (9)$$

where η is the learning factor that determines how much to move in that direction, and where w_i and Δw_i refer to the updated and the previous gradient descent (Alpaydin, 2020, p. 187; Chopra & Khurana, 2023, p. 37). The basic idea of ML methods is to learn a hypothesis whose predictions yield minimum loss (Jung, 2022, p. 166), and gradient descent does just that.

2.2.4 K-Nearest Neighbors

According to Lee (2019, p. 118), K-Nearest Neighbors (KNN) is an algorithm that compares the query instance's distance to the other training samples and selects the K-nearest neighbors. He adds that it then takes the majority of these K-neighbor classes to be the prediction of the query instance. An example with KNN classifier with $k = 3$ and $k = 5$ is shown below:

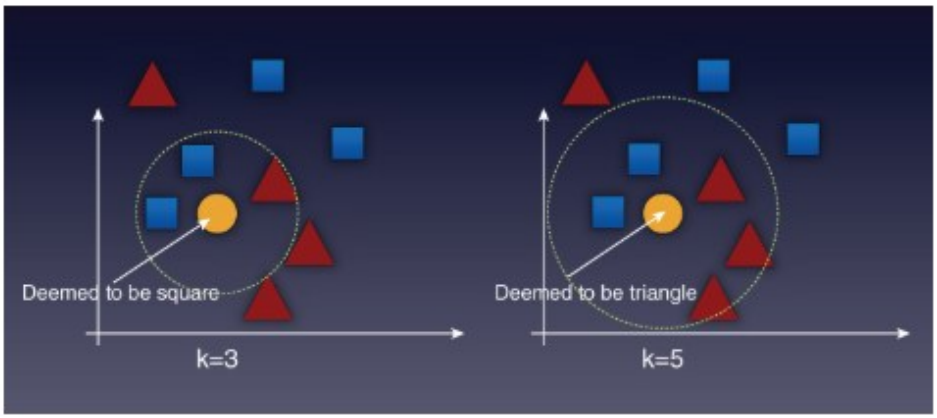

Figure 4. KNN classifier with squares, triangles and a circle, and k=3, k=5 Neighbor values (Lee, 2019, p. 118).

As Chopra and Khurana (2023, p. 77) state, the object (the circle in this instance) is classified according to the majority voting of k number of Neighbors. In the first case, the circle would be classified as a square and in the second case, it would be classified as a triangle (see Figure 4). If $k = 1$, the input would be classified according to the closest neighbor with no further consideration. In regression using KNN moreover, the output is found by taking an average of K-Nearest Neighbors values (Chopra & Khurana, 2023, p. 77). Chopra and Khurana (2023, p. 177) add that large value of *k* helps in the reduction of noise during the classification process but may also lead to more boundaries between the classes which are less different. Furthermore, KNN is said to be a non-parametric algorithm, since it does not make any pre-assumptions on the underlying data set (Chopra & Khurana, 2023, p. 178).

According to Alpaydin (2020, p. 152), the kernel estimator for KNN class-conditional density is

$$p(x|C_i) = \frac{k_i}{N_i V^k(x)}, \qquad (10)$$

where k_i is the number of neighbors out of the k nearest that belong to the class C_i, N_i is the number of labeled instances belonging to C_i and $V^k(x)$ is the volume of the d-

dimensional hypersphere centered at *x*, with radius $r = \left||x - x(k)|\right|$ where $x(k)$ is the *k*-th nearest observation to *x*. The dimension changes the volume of the unit sphere of $V^k(x)$ accordingly (Alpaydin, 2020, p. 152), but this doesn't need to be explored in more detail to get the basic idea.

2.2.5 Support Vector Machine

According to Jung (2022, p. 108), Support Vector Machines (SVMs) are an ML method for learning a hypothesis to predict a binary label y of data point based on its features x. The binary labels take values in the label space $y = \{-1,1\}$, without loss of generality (Jung, 2022, p. 108). The principal idea is to draw a line between two or more classes in the best possible way and predict future data with it. This technique can be utilized in, e.g., a cat and dog classifier, which makes a prediction based on snout length and ear geometry (Lee, 2019, p.105).

SVMs try to find maximum separability, i.e. separate data point into two (or more) distinct groups with the widest possible margins (Lee, 2019, p. 105). According to Lee (2019, p. 105), *each of the two margins touches the closest point(s) to each group of points and the center of the two margins is known as the hyperplane*:

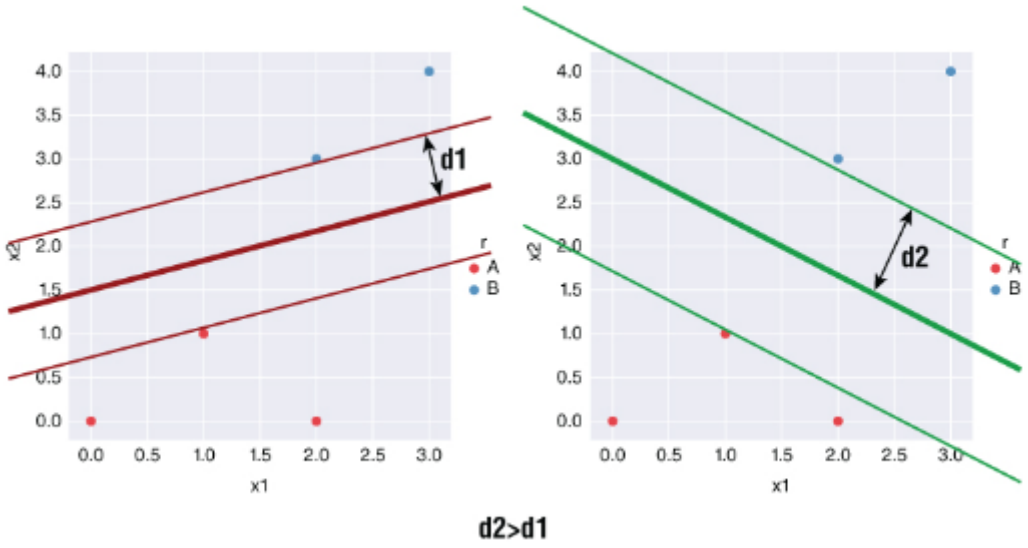

Figure 5. Two possible linear support vector machine margins and hyperplanes, d1 and d2 (Lee, 2019, p. 105-106).

Here, the d2 option is selected as the support vector line, since it clearly splits the groups with a wider margin than the d1 line (Figure 5; Lee, 2019, p. 105). According to Lee (2019, p. 106), the formula for the hyperplane is given as:

$$g(x) = \overrightarrow{W_0}x_1 + \overrightarrow{W_1}x_2 + b, \qquad (11)$$

where x_1 and x_2 are the inputs, $\overrightarrow{W_0}$ and $\overrightarrow{W_1}$ are the weight vectors, and b is the bias term. In Figure 3's instance, the upper line represents the margin for the first class, and the below line represents the margin for the second class. Furthermore, if value of $g \geq 1$, the point belongs to the first, and if $g \leq -1$, it belongs to the second class (Lee, 2019, p. 105). Finally, calculating the optimization problem requires the Langrage Multipliers technique (Lee, 2019, p. 106) and can be formulated as:

$$\min \frac{1}{2}\left|\left|w\right|\right|^2 \; subject\ to\ r^t(w^T x^t + w_0) \geq +1, \forall t, \qquad (12)$$

where w is the normalized weight vectors, r^t is the class (-1 or 1), w^T is the weight vector and x^t is the input (Alpaydin, 2020, p. 280).

The above examples present a linear kernel for the SVM, but there are also other alternatives: such as Radial Basis Function (RBF, or Gaussian) kernel, polynomial kernel and Sigmoid kernel (Lee, 2019, p. 114). This is especially useful, when the underlying data is not simply separable by a linear hyperplane, as in the below figure:

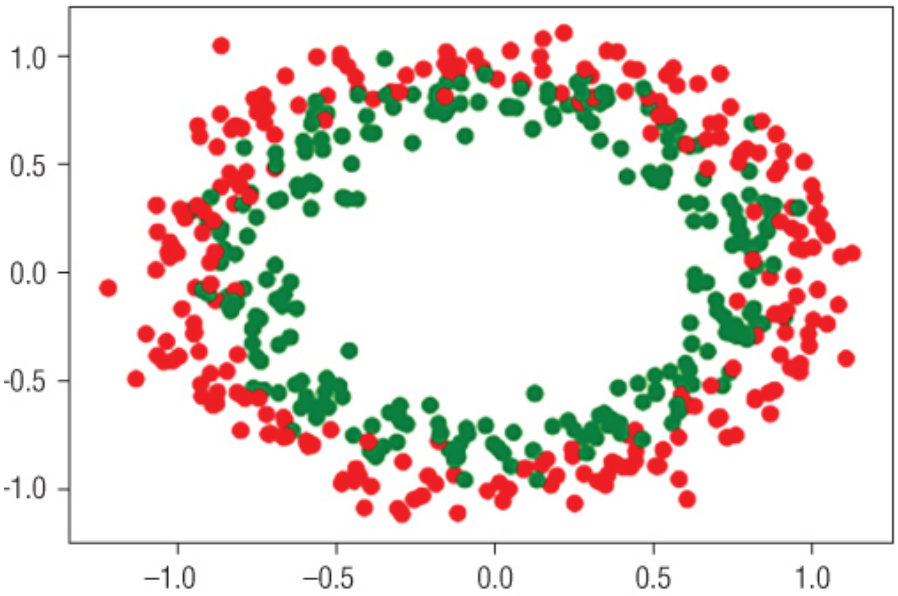

Figure 6. A scatter plot of two groups of points distributed in a circle, separated by the color (Lee, 2019, p. 109).

Figure 6 presents a scatter plot, where the data is clearly not separable linearly. Thus, a technique called the kernel trick is used to transform the data into a higher dimension so that it has a clear dividing margin between classes (Lee, 2019, p. 109). The higher, or third, dimension can in this instance be calculated with the formula below:

$$z = x^2 + y^2, \qquad (13)$$

where x and y are the values of the input and target variable (Chopra & Khurana, 2023, p. 52). Now, with the third variable, a 3D-figure can be plotted of the circular data points:

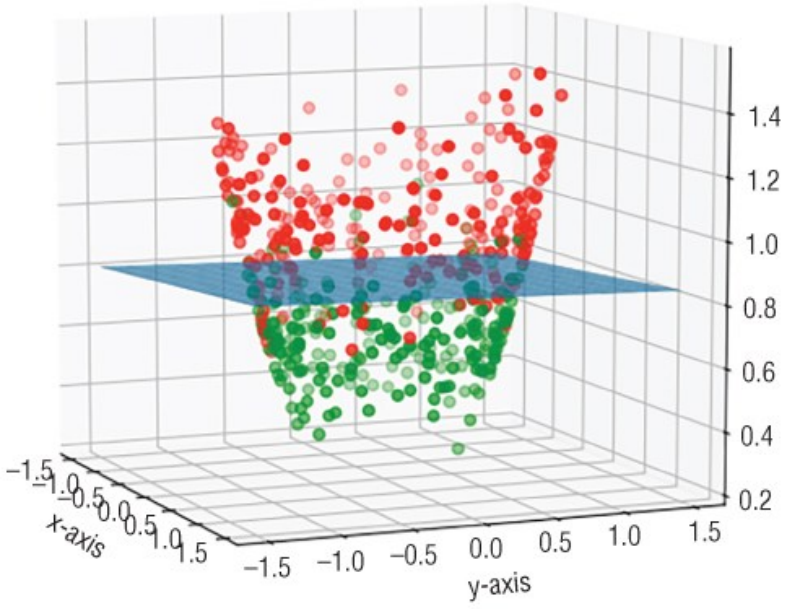

Figure 7. A 3D plot of the circular data transformed into a third dimension, with the linear hyperplane cutting through the two sets of points (Lee, 2019, p. 112).

Here (Figure 7), the earlier data can be seen in a 3D-environment and separated by a linear hyperplane (Lee, 2019, p. 112). The hyperplane can be found with the formula below:

$$\begin{aligned} &\overrightarrow{w_0}x_1 + \overrightarrow{w_1}x_2 + \overrightarrow{w_2}x_3 + b = 0 \\ &\overrightarrow{w_2}x_3 = -\overrightarrow{w_0}x_1 - \overrightarrow{w_1}x_2 - b \\ &x_3 = \frac{-\overrightarrow{w_0}x_1 - \overrightarrow{w_1}x_2 - b}{\overrightarrow{w_2}}, \end{aligned} \qquad (14)$$

where $\vec{w}_n$ are the vectors, x_n are the variables including x, y and z, and b is the intercept (Lee, 2019, p. 111). Since there exist kernel function alternatives, such as RBF, polynomial and Sigmoid, it is not necessary to map them in the new higher dimensional space at all since the algorithms have been *kernelized* (Alpaydin, 2020, p. 286). It is important to note that different kernel algorithms and multi-class problems can make the SVM separate the classes in a seemly complicated way:

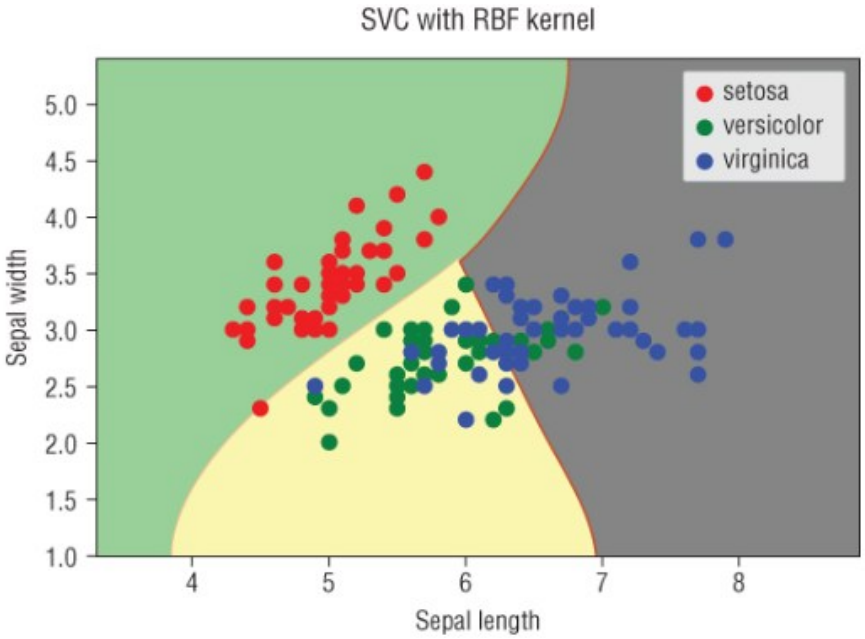

Figure 8. Support Vector Classifier with an RBF kernel and three different classes (Lee, 2019, p. 114).

Here (Figure 8), the RBF kernel function gives a value to each point according to the distance from the origin or a fixed center, commonly on Euclidean space (Lee, 2019, p. 114). Other parameters of SVM include C, which controls the tradeoff between a smooth decision boundary and classifying the training points correctly, and Gamma, which defines how far the influence of a single training example reaches (Lee, 2019, p. 113-115). These parameters, as well as the kernel choice, will be the ones that will be optimized when applying hyperparameter tuning on SVM in chapter 4.4.

2.2.6 Decision Tree

Decision Tree refers to the non-parametric supervised learning method, where the process is begun from the root node and followed to the branch according to the condition or the property satisfied by the corresponding data set (Chopra & Khurana 2023, p. 57). According to Jung (2022, p. 116), a Decision Tree consists of nodes which are connected by directed edges; they can be thought of as an instruction for how to compute the function value $h(x)$ given the features $x \in X$ of a datapoint. A simple Decision Tree is shown below:

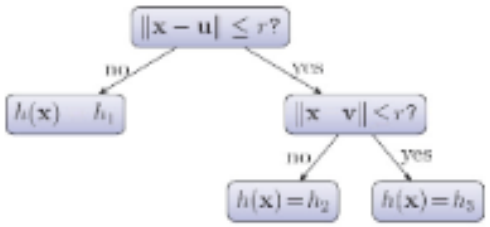

Figure 9. A simple decision tree which regresses or classifies according to the node instructions (Jung, 2022, p. 118-120).

where x is the input, u and r are some arbitrary values, and h_n are an arbitrary value or a class (Figure 9). According to Alpaydin (2020, p. 164), a Decision Tree does not assume any parametric form for the class densities and the tree structure is not fixed a priori, but the complexity of the data affects the tree's growth by adding new branches and leaves while learning. The test functions, $f_m(x)$, divide and further subdivide the data set into a series of simple decisions, which finally reach an output label that is a numeric value in regression and a class code in classification (Alpaydin, 2020, p.164). Since each leaf node of a Decision Tree corresponds to one region (Jung, 2022, p. 119), each node then defines a localized region in the input space where instances have the same labels or very similar numeric outputs (Alpaydin, 2020, p. 164).

Decision Trees are simple and easy to understand (Chopra & Khurana 2023, p. 57), since they can be converted to a set of IF-THEN rules (Alpaydin, 2020, p. 166). Furthermore, in the case of limited computational resources, they can be shallow with the cost of less accuracy (Jung, 2022, p. 116); but even making them deeper and more complex for a more accurate prediction (Jung, 2022, p. 116) won't eat up too much computational resources since the trees employ greedy search and divide and conquer approach to solve the problem (Chopra & Khurana 2023, p. 57).

Loss functions used to measure the quality of a Decision Tree include, e.g., squared error loss for numeric labels and the impurity of individual decision regressions for discrete labels (Jung, 2022, p. 116). The mean square error looks like:

$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(x^t), \qquad (15)$$

where m is the node, N_m is the number of training instances reaching node m, r^t is the correct value, g_m is the estimated value, b_m is 1 if x reaches node m and 0 otherwise, and x^t is the input (Alpaydin, 2020, p. 168). To measure impurity, one can use entropy:

$$\mathrm{I}_m = -\sum_{i-1}^{K} p_m^i \log_2 p_m^i, \qquad (16)$$

where p_m^i is the probability of a given class when node reaches m, K is the number of classes and i denotes the number of nodes under node m (Alpaydin, 2020, p. 166). Lastly, pruning in a Decision Tree involves the elimination of unwanted branches of a tree which will never contribute to the resultant path, and it is a common method (Chopra & Khurana, 2023, p. 57).

2.2.7 Random Forest

According to Alpaydin (2020, p. 178), Random Forest (or a decision forest) is an ensemble of Decision Trees, which is one of the most popular models to combine a single ML algorithm. Random Forests use bagging to learn an ensemble of decision trees, after which individual predictions obtained from the trees in a Random Forest are combined, using an average in regression or a majority vote in binary classification, to obtain a final prediction (Jung, 2022, p. 233). According to Breiman (2001), in bagging:

1. A random partition of data is drawn from the training data to bootstrap the tree structure

2. The tree may use all features or only a random subset of available features
3. The output is again aggregated from all predictors
4. The two sources of randomness stabilize the tree structure and reduce overfitting.

Voting corresponds to taking a linear combination of the learners and a sum of this looks like:

$$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}\,, \qquad (17)$$

where L is the total number of voters and d_{ji} is the vote of learner for class C_i (Alpaydin, 2020, p. 367). Other classifier combination rules include weighted sum, median, minimum, maximum or product, but sum is the most intuitive and the most widely used in practice (Alpaydin, 2020, p. 367). Bagging is useful, since it reduces the variance in the predictor generation process by introducing some randomness (Breiman, 2001).

2.2.8 Gradient boosting

In boosting, complementary base-learners are generated by training the next learner on the mistakes of the previous learners (Alpaydin, 2020, p. 372). This idea is presented by Freund and Schapire (1999) in the form of a boosting algorithm, which combines learners whose error rate is a bit less than 50 %, i. e. a weak learner, to generate a strong learner. Gradient boosting builds an additive approximation of $F^*(x)$, which maps instances of x to output values of y, as a weighted sum of functions

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x), \qquad (18)$$

where ρ_m is the weight of the m^{th} function, $h_m(x)$ (Bentéjac et al., 2021). According to Bentéjac at al. (2021), the first approximate model and subsequent ones are expected to minimize a training loss, but instead of solving the problem directly, each h_m is in this

instance a greedy step in gradient descent optimization for F^*. As each model is trained on a new data set $D = \{x_i, r_{mi}\}_{i=1}^{N}$, the pseudo-residuals are calculated by

$$r_{mi} = \left[\frac{\partial L\big(y_i, F(x)\big)}{\partial F(x)}\right]_{F(x)=F_{m-1}(x)}, \qquad (19)$$

where L is the loss function (Bentéjac et al., 2021). The original boosting method's weakness is that it requires a very large training sample, and the subsequent classifiers are only trained on a subset on which the previous ones err (Alpaydin, 2020, p. 372).

2.2.9 Artificial Neural Network

A simple example of an ANN, which takes its inspiration from the human brain, is a perceptron (Alpaydin, 2020, p. 198). A perceptron is based on a neuron, which is the basic processing unit of the brain (Chopra & Khurana, 2023, p. 65). There are approximately 10^4 of them in a human brain, which operate in parallel, i.e. simultaneously (Alpaydin, 2020, p. 198). According to Chopra and Khurana (2023, p. 65) an action signal is fired by a neuron when a particular threshold is met by a cell. In a perceptron's case, it accepts the input and the weight, performs the weighted sum of inputs, and applies an activation function over it. (Chopra & Khurana, 2023, p. 66). An example figure of a perceptron includes the input units, labelled x_0 (which is the bias unit and is always 1) and all the way to x_d. Each input unit is associated with its own weight unit of the directed connection, w_0 to w_d, from x to the output y (see Figure 10; Alpaydin, 2020, p. 202):

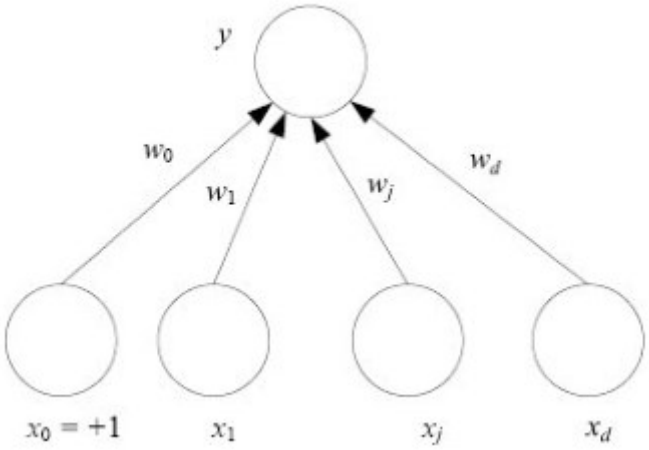

Figure 10. A simple perceptron representation with the input units x, weights w and the output unit y (Alpaydin, 2020, p. 201?).

The perceptron's output formula is formed as follows:

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \,, \qquad (20)$$

where d is the index of the input, w_j is the connection weight of an input, x_j is the input and w_0 is the bias unit (Alpaydin, 2020, p. 201). According to Alpaydin (2020, p. 202), in case of one input a linear fit can be implemented; in case of more than one input, a multivariate linear fit can be implemented, in which the line becomes a hyperplane.

However, ANNs are formed deeper, i. e. having 10 or more hidden layers, and with more neurons to solve more complex problems (Jung, 2022, p. 120). This is called deep learning in ANN terminology. According to Jung (2022, p. 120), it can be shown that an ANN with only one single, but arbitrarily large hidden layer can approximate any given hypothesis map $h: X \to Y = R$ to any given accuracy. Furthermore, success of ANNs with many hidden layers might be attributed to the fact that the network representation of hypothesis maps is beneficial for the computational implementation of ML methods (Jung, 2022, p. 120).

Backpropagation, or gradient computation, is used to train the perceptron (Alpaydin, 2020, p. 198) and it includes two steps: gradient computation phase and weight updating phase (Chopra & Khurana, 2023, p. 66). According to Chopra and Khurana (2023, p. 66), in the first step of backward propagation the input is the propagation's output activation that is sent to the ANN, and it generates the deltas of all the output and hidden neurons. Subsequently,

1. The gradient of the weight is calculated by multiplying the output delta and the input activation.
2. A ratio or percentage of the gradient is subtracted from the weight. This ratio or percentage affects the quality of learning and speed. It is referred to as the learning rate. A neuron is able to train faster if the learning rate is higher. If the learning rate is lower, then the training is considered accurate (Chopra & Khurana, 2023, p. 65-67).

According to Alpaydin (2020, p. 208), the first-layer weights' gradient is calculated by the chain rule:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}}, \qquad (21)$$

where E is the error, w_{hj} are the weights, y_i is the output and z_h are the hidden units or inputs. Non-linear regression with a single output is calculated by

$$y^t = \sum_{h=1}^{H} v_h z_h^t + v_0\,, \qquad (22)$$

where v_h is the weight for the hidden unit and z_h^t being the hidden unit or input (Alpaydin, 2020, p. 208). The error function over the whole sample in regression is

$$E(W, \boldsymbol{v}|X) = \frac{1}{2}\sum_t (r^t - y^t)^2, \qquad (23)$$

and the second-layer weights are updated by the least-squares rule with

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t, \qquad (24)$$

where η is the learning factor, where the sum calculates the error for the hidden unit h and other variables are explained prior to this (Alpaydin, 2020, p. 208). Finally, backpropagation is used to propagate from the output y back to the inputs which cover the input layer and the hidden units. To calculate the direction and magnitude in which each parameter needs to be changed, batch learning, online learning or minibatch learning is used:

- In batch learning by accumulating these changes over all patterns and making the change once after a complete pass over the whole training set is made.
- In online learning, updating the weights after each pattern, thereby implementing stochastic gradient descent. A complete pass over all the patterns in a training set is called an epoch, and in this case the learning factor, η, should be chosen smaller in this case and patterns should be scanned in a random order.
- Minibatch learning lies in middle of these, where we take small batches of randomly picked training instances, calculate the gradients on those, and do a single update with their average. With this approach, learning is less affected by outliers since averaging is used (Alpaydin, 2020, p. 208).

2.3 Unsupervised learning

In this chapter, unsupervised learning will be introduced shortly, since it won't be used in the thesis.

According to Lee (2019, p. 127), in supervised learning labeled data is considered and estimated, whereas in unsupervised learning unlabeled data is used. Unsupervised methods must rely solely on the intrinsic structure of data points to learn a good hypothesis and thus, unsupervised methods do not need a teacher or domain expert who provides labels for data points (Jung, 2022, p. 28).

Consequently, a metric that measures similarity between objects needs to be defined (Chopra & Khurana, 2023, p. 81). Välisuo (2023b) demonstrates this with an example, where *to implement a machine learning algorithm performing the same task, one could measure the hue of the color (H), and the parameter describing surface roughness (S) and then plot each apple and orange in the orthogonal 2D H-S coordinates. The similarity between objects can be for example the Euclidean distance between objects in this 2D space.* The figure below displays oranges and two kinds of apples in a 2D space, where similar fruit are grouped or clustered (see Figure 11):

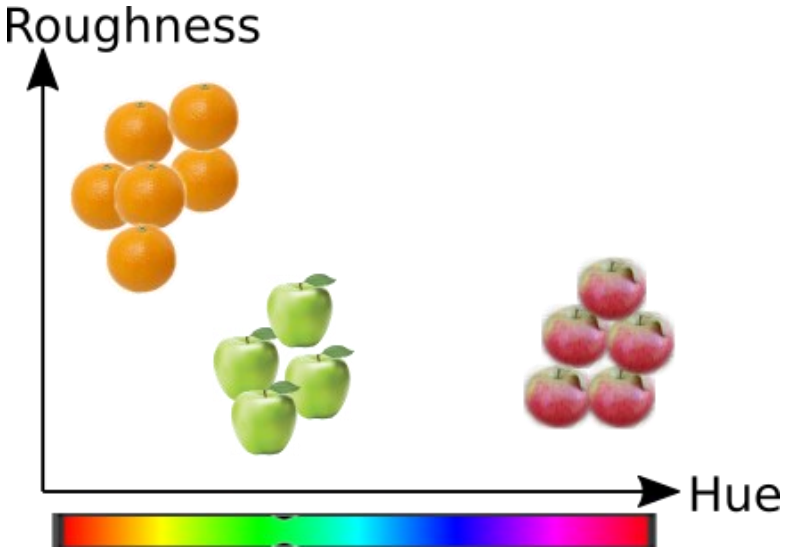

Figure 11. Fruit in a 2D space, Roughness on the y-axis and Hue on the x-axis (Välisuo, 2023b).

According to Välisuo (2023b), clustering recognizes a dense group of points surrounded by more sparsely populated areas. The clustering algorithm clusters the data in the design matrix $X = [H^T, S^T]$ into cluster memberships, c_i:

$$\begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = f\left(\begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}\right), \qquad (25)$$

where the function is given the datapoints $x_{11} \dots x_{np}$ as inputs to then be clustered in the memberships $c_1 \dots c_n$ (Välisuo, 2023b).

Unsupervised learning isn't limited to clustering, but includes feature learning methods as well (Jung, 2022, p. 27). According to Jung (2022, p. 28), *feature learning methods determine numeric features such that data points can be processed efficiently using these features. Two important applications of feature learning are dimensionality reduction and data visualization.* According to Alpaydin (2020, p. 137), clustering and dimensionality reduction methods can first be used for data exploration and to understand the structure of the data. Secondly, it can be used to map data to a new

space where supervised learning is easier (Alpaydin, 2020, p. 137), i.e. reducing dimensions.

Clustering algorithms include k-Means Clustering, Hierarchical Clustering (Chopra & Khurana, 2023, p. 81) and Gaussian mixture model (GMM), whereas dimensionality reduction algorithms include Principal Component Analysis (PCA), Independent Component Analysis (ICA), t-Distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation (UMAP) (VanderPlas, 2016).

2.4 Validation techniques

An ML model's generalization ability of, i.e. the quality of its inductive bias, can be measured by validating the trained model with data outside the training set (Alpaydin, 2020, p. 51). According to Alpaydin (2020, p. 51), this can be simulated by dividing the data set into two parts: the training set and the testing or validation set. According to Välisuo (2023c), a usual ML data splitting process for training models looks simply like the following:

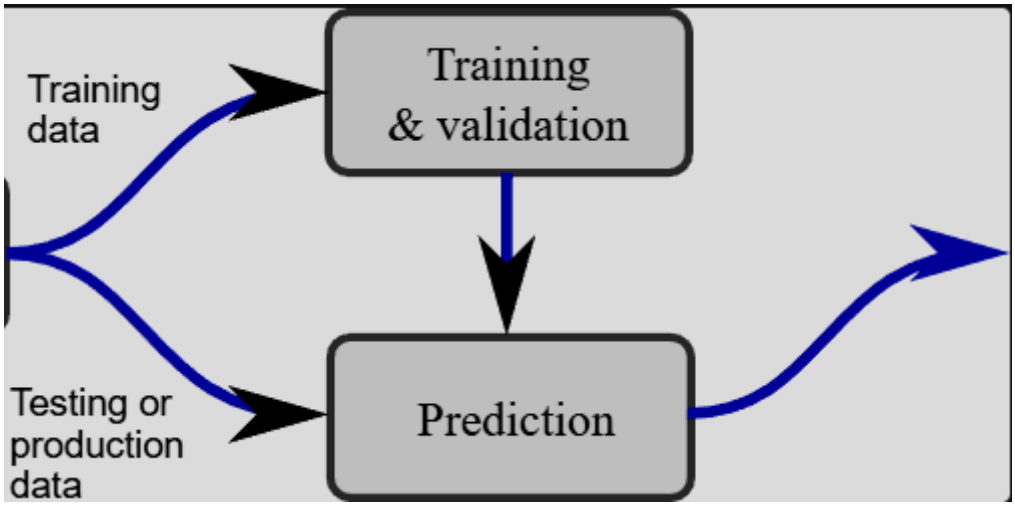

Figure 12. An ML process graph to show the division of data into a training and testing data set (Välisuo, 2023c).

In the above Figure (Figure 12), the original data is divided into a training set and a testing set, which are used to train the ML model and to test the prediction accuracy of the model, respectively (Välisuo, 2023c). The model cannot be tested with the data that was used to train it, since that would lead to overfitting and a too optimistic prediction accuracy of the model (Alpaydin, 2020, p. 51). Lee (2019, p. 123) adds that an overfitted model will not work with new, unseen data, since the model captures all the noises and fluctuations in the training data. Underfitting, on the other hand, happens when the model cannot capture the underlying trend of the data accurately (Lee, 2019, p. 123). Chopra and Khurana (2023, p. 24) emphasize the importance of the quality and quantity of the training set, since its accuracy and success are entirely dependent on it.

A good addition to only using a test set for the accuracy of the model is to use a confusion matrix which gives one a clearer picture of the prediction power (Lee, 2019, p. 99). It shows the number of actual and predicted labels, and how many of them are classified correctly (Lee, 2019, p. 99). Here is an example of one:

```
[[187  16]
[ 28  68]]
The accuracy is 0.85
```

The number 187 comprises of the True Negatives (TN), 16 of the False Negatives (FN), 28 of the False Positives (FP) and 68 of the True Positives (TP), that the ML model predicted (Lee, 2019, p. 100). In this instance, positive means dead and negative survived patients. The correctly predicted labels can thus be seen diagonally starting from the upper left corner and ending in the lower right corner of the matrix. Hence, the accuracy can be calculated by the sum of all correct predictions divided by the total number of predictions (Lee, 2019, p. 101), which yields approximately 85% in this instance. Here is also the formula for calculating accuracy:

$$Accuracy = \frac{TN + TP}{TN + FN + FP + TP}, \qquad (26)$$

This is, furthermore, the main metric when model evaluation is done in chapter 4.6. Another type of validation technique is cross-validation. It is often used when the underlying data is not distributed evenly, making the test data too hard or too simple to predict (Lee, 2019, p. 123). Jung (2022, p. 198) adds that one might be very unlucky and divide data points for the validation, which are outliers and don't represent the overall distribution of the data. As opposed to dividing the data into a training and a test set, the data set can be evenly distributed into *k* subsets which are referred to as folds (Jung, 2022, p. 198). According to Jung (2022, p. 198), the training and validation is repeated *k* times, where one fold is used as a test (validation) set and the other *k – 1* folds are used as the training set. Once this is done *k* times and for each possible fold, the training and validation error is averaged for all repetitions (Jung, 2022, p. 198; Lee, 2019, p. 123). Cross-validation is a more robust technique to estimate the accuracy or the expected loss of the model, than a single split validation (Jung, 2022, p. 198). A visual representation of a cross-validation looks like the following:

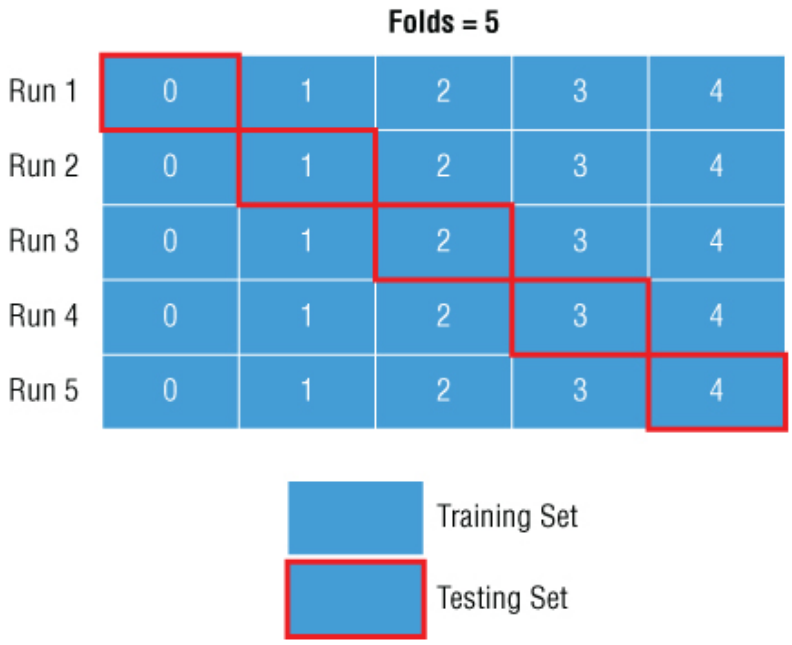

Figure 13. A visual representation of a 5-fold cross-validation (Lee, 2019, p. 123).

In Figure 13, the data is evenly split into five parts. In each training and testing sequence, the red border part of the data is used as the testing set. Before the next sequence, or row, the model is validated. Lastly, each sequence's test score is averaged into one cross-validation score (Lee, 2019, p. 123). In this thesis, the data will be split into a training and test set, where the models never see the testing data; they only make predictions on it. More on this in chapter 4.3. Cross-validation will only be used during the use of RSCV, since it is difficult to split the data many times so that the model has not seen it at all.

2.5 Prior research

To have the most recent and relevant research as the background, only literature that is published on, and after the year 2020, is reviewed in this chapter.

This case study's main inspiration and benchmark is Chicco and Jurman's (2020) case study, which discovered that the data also used in this study can efficiently predict heart failure using stratified logistic regression and with only two of the data set's variables: serum creatinine and ejection fraction. The ML models that were used during the study were Random Forest, Decision Tree, Gradient Boosting, Linear and Logistic regression, One rule, Artificial Neural Network, Naïve Bayes, Support Vector Machine with a Linear and Radial kernel, and K-Nearest Neighbors (Chicco & Jurman, 2020, p. 7). For feature selection they first used a traditional univariate biostatistics analysis including Mann-Whitney U-test, Pearson correlation coefficients and Shapiro–Wilk tests, Chi squared test, and Random Forests feature selection aggregate ranking. Even survival analysis was performed with the follow-up period variable using a stratified logistic regression algorithm, since the fifth month reported fewer surviving patients than the fourth month (Chicco & Jurman, 2020, p. 10). Here, the two selected variables and the applied ML algorithms are of importance in this thesis, since the data set is the same.

Adler et al. (2020) have done a case study using ML to predict heart failure, but the data set, and the cohorts they used were different from the one used in this thesis. The number of patients was 5822 and variables were different as well. The variables included

diastolic blood pressure, creatinine, blood urea nitrogen, hemoglobin, white blood cell count, platelets, albumin and red blood cell distribution width (Adler et al., 2020, p. 140). Different variables from the data set used in this study were, e.g., diastolic blood pressure mmHg, hemoglobin g/dL, white blood cell count × $10^3/L$ and blood urea nitrogen mg/dL (Adler et al., 2020, p. 141). The ML model used in their study was a boosted Decision Tree to relate a subset of the patient data to low and high-risk groups. The data was collected between 2006 and 2017 from three distinct patient populations: from retrospective analysis of a cohort of patients followed at the University of California, San Diego; the University of California, San Francisco Medical Center and from the European A systems Biology Study to Tailored Treatment in Chronic Heart Failure (BIOSTAT-CHF) project, which enrolled from 69 centers in 11 European countries (Adler et al. 2020, p. 140). They add that using data from different cohorts was possible due to similar underlying electronic medical record systems. Although the data set differs from the one used in this study, the results and ML models used can be utilized in this thesis as well.

Escolar et al. (2022) have done a study to predict heart failure decompensations using ML techniques. The ML algorithms used were Naïve Bayes, Decision Tree, Random Forest, Support Vector Machines, ANNs and Class Balancing. As with the previously introduced study, the data set is different from the data set used in this thesis. The baseline characteristics of each patient were much more detailed than in both previously mentioned studies, which included among others age, sex, height, weight, cardiopathy values, rhythm, devices and different blood analysis values. A supporting health questionnaire was included, remote patient telemonitoring data was used, and patients' heart failure decompensations (i.e., emergency admissions, hospital admissions, and home care interventions) were used to support the ML methods (Escolar et al., 2022, p. 432). The telemonitoring data included spontaneous bacterial peritonitis, diastolic blood pressure, heart rate and oxygen saturation. The best performing models are Random Forest and Naïve Bayes with Bernoulli method according to Escolar et al. (2022, p. 435). Even though Random Forest gets the best scores, they select Naïve Bayes as the

classification algorithm, since Random Forest has a high standard deviation and thus Naïve Bayes is more stable (Escolar et al., 2022, p. 435). The data set is very different from the one used in this thesis, but the notion of high standard deviation in Random Forest is important to note.

Sarijaloo et al. (2021) have done a study on predicting 90-day acute heart failure readmission and death using ML-supported decision analysis. The final utilized ML algorithm was a Logistic Regression with Stepwise Selection and variable selection was done by the least absolute shrinkage and selection operator (LASSO) algorithm (Sarijaloo et al., 2021, p. 232-233). Other variable selection ML algorithms were Support Vector Machine, Random Forest and Gradient Boosting Machine, but LASSO was deemed best by a 4-Fold Cross Validation technique according to Sarijaloo et al. (2021, p. 233). The data set was quite large; it included 4368 consecutive patients from the University of Florida Health Shands Hospital. The variables were similar to Escolar et al.'s (2021) data set's variables. Identified risk factors included reduced left ventricular ejection fraction, elevated blood pressure, elevated blood urea nitrogen, reduced albumin, abnormal sodium or bicarbonate, and lung disease (Sarijaloo et al., 2021, p. 230). According to them, the final model yielded a 71% accuracy. The first two are also present in the data set used in this thesis, and thus, they can be of importance.

Xu et al. (2025) have researched ML-based risk factor analysis and prediction model construction for mortality in chronic heart failure during a 10-year follow-up. They utilized seven ML models: Decision tree, Random Fores, XGBoost, Adaptive Boosting (AdaBoost), Support Vector Machine, Naïve Bayes, and multilayer perceptron, i.e. ANN (Xu et al., 2025). According to them, the data was retrieved from the Jackson Heart Study database hosted by the National Heart, Lung, and Blood Institute, which comprises data on 3883 African American adults aged 35–84 years, from which 489 patients had chronic heart failure. The data contained standardized medical assessments, advanced imaging techniques, behavioral factors, and environmental variables totaling to 58; thus, the data set was multidimensional and multimodal (Xu et al., 2025). Variables included, among

others, age, body mass index, use of antihypertensive, hypoglycemic and antiarrhythmic medications, systolic blood pressure, left ventricular ejection fraction, favorite food stores within three kilometers, and proportion of the population living in poverty in the area (Xu et al., 2025). There are some similar variables as this thesis' data set, including age, diabetes and ejection fraction. Since there were so many variables in the data set, Xu et al. (2025) generated a feature heat map and sequentially eliminated features with high collinearity to mitigate the risk of overfitting. Furthermore, they used the Standard Scaler technique to standardize the diverse data from the cohort. These should prove useful in this thesis as well, however there are far less variables so the former, i.e. eliminating features with high collinearity, might not be necessary. Xu et al. (2025) discovered that the best performing ML algorithm for classifying chronic heart failure was XGBoost, yielding an accuracy of 81.58%. To further boost classification accuracy, they tuned the hyperparameters by GSCV and 10-fold cross-validation, which improved the model's accuracy to 88.64%.

Al Younis et al. (2024) have done a case study of predicting heart failure patients with distinct left ventricular ejection fraction levels using circadian electrocardiogram (ECG) features and ML. They utilized K-Nearest Neighbors, ANN, SVM and Decision Tree algorithms to classify ejection fraction. The goal was to classify heart failure patients into three subgroups based on their ejection fraction: reduced, mid-range and preserved, to succeed in diagnosis, risk assessment, treatment choice, and the ongoing monitoring of heart failure (Al Younis et al., 2024). According to them, the data was a combination of American patients from the University of Rochester Medical Centre's Telemetric and Holter ECG Warehouse, and Greek patients sourced from seven cardiology departments as part of the PRESERVE EF trial in Greece. The data set encompassed individuals aged 33 to 88 years and some with coronary artery disease (Al Younis et al., 2024). Since the training data was partly clinical, e.g. age, gender, body mass index, smoking, diabetes, etc., and partly ECG features from hourly intervals using 24-hour ECG recordings, e.g. TP which denotes the period when the ventricles are repolarized and at rest (Al Younis et al., 2024), the models are not completely comparable to the one created in this thesis.

However, utilized algorithms and cross-validation techniques are of interest; the best classification algorithms for the problem were Decision tree and K-Nearest Neighbors, which yielded a 91.2% and 90.9% accuracy, respectively. The utilized cross-validation was a nested cross-validation framework (Al Younis et al., 2024). According to them,

> *[--] the outer loop follows the "Leave-One-Out Cross-Validation" (LOOCV) approach, where in each iteration, one patient is withheld for testing, while the remaining (n-1) patients are used for training. This process is then repeated for all n patients in the dataset. [--] Within the outer loop, there is an inner loop that utilizes a 5-fold cross-validation strategy for hyperparameter tuning. In this inner loop, the (n-1) patients from the training set are further divided into five subsets, and the model is trained and validated across these subsets iteratively. [--] Finally, the average of these metrics is calculated to provide a comprehensive and representative assessment of the model's performance across all patients in the dataset.*

The LOOCV approach is useful especially when data set utilization needs to be maximized due to it being small, as is the case in this thesis, provides an unbiased evaluation, offers high variability insights, tests robustness, ensures consistent performance estimation, is applicable to imbalanced data sets, and maintains simplicity in its implementation (Al Younis et al., 2024).

To summarize the chapter, only one of the articles utilized the same data set as this thesis (Chicco & Jurman, 2020). Therefore, other articles cannot be used as a benchmark as much as the aforementioned. However, many data sets that the articles were using had some similar variables. The most successful algorithms were often also tree-based (Decision Tree, Random Forest and XGBoost), which is good to note before the ML algorithms are deployed. Lastly, many good methodologies were deployed, such as the utilization of cross-validation, dropping correlated variables and scaling the data before training the models (Al Younis et al., 2024; Xu et al., 2025).

3 Methodology

The thesis will be performed using a design science research-based case study methodology hybrid, where ML classifier models will be created and deployed to predict heart failure mortality from the Heart Failure Clinical Records data set (2020). Justification for the hybrid methodology is that known ML algorithms, which can be thought of as basic research, are applied to a specific context and a data set (Järvinen, 1999, p. 103). Furthermore, according to Järvinen (1999, p. 78), since a population, a ready data set with selected patients in this instance, is selected to replicate previous cases or extend emergent theory, a case study methodology could be useful. Since the ML model creation will follow the instructions and examples of Chopra and Khurana (2023), Lee (2019), and Välisuo (2023), an appropriate guiding theory for helping in construction of an emergent model or theory is selected, and thus, a case study design methodology is used (Järvinen, 1999, p. 77). The ML model creation will also draw inspiration from prior research done on heart failure (see chapter 2.4). The data set has not been researched afterwards much; thus, the use of case study methodology is justified. Dresch et al. (2015, p. 1122) also state, that a small sample size is usually associated with a case study, which further justifies the use of the methodology.

Design science, as the other side of the information system research cycle, creates and evaluates IT artifacts intended to solve identified organizational problems. Such artifacts are represented in a structured form that may vary from software, formal logic, and rigorous mathematics to informal natural language descriptions (Hevner et al., 2004, p. 77). According to Hevner et al. (2004, p. 82), design science is a problem-solving process. The fundamental principle of design-science research is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact (Hevner et al., 2004, p. 82). In the context of the thesis, the ML models will be the artifacts to be built and applied. The problem that the ML models are meant to solve is found in the research questions; mainly to predict heart failure mortality from the Heart Failure Clinical Records (2020).

Hevner et al. (2004) present several Guidelines to Design-Science Research. These describe how one, among other things, effectively describes and addresses the problem when implementing the artifact, demonstrates the utility of a design artifact via well-executed evaluation methods, and provides clear contributions (Hevner et al., 2004). The Guidelines will be the framework that guides this thesis in creating the artifact, i.e. ML models. The ML applications will be done with the thesis' research questions in mind and each step in programming process will be documented thoroughly. The classifying algorithms will include those defined under chapter 2.2.

In the next chapter, the heart failure data set will be loaded, preprocessed, and the ML algorithms defined under chapter 2.2 will go through a learning process on the training data set. Applied ML part will follow the instructions and examples of Chopra and Khurana (2023), Lee (2019), and Välisuo (2023). After evaluating the initial models, a subset of features will be selected by examining the correlations between features and Random Forest Gini and Permutation feature importance. After feature selection, models' hyperparameters will be tuned by RandomizedSearchCV (RSCV) algorithm to possibly further increase the prediction accuracy (Bergstra & Bengio, 2012). In contrast to GSCV, which was also considered for hyperparameter tuning, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled randomly from the specified distributions. The number of parameter settings that are tried is given by *n_iter* (Scikit-Learn, n.d. a). This forms the final ML models, which will then be evaluated with the same data as initial models, only with selected features. In Chapter 4.6 the accuracy of non-optimized models will be compared to the accuracy of optimized models, and in Chapter 6 the accuracy of optimized models will be compared against the accuracy of prior literature's models. Below is a figure (Figure 14) of the ML process workflow, where each step is described:

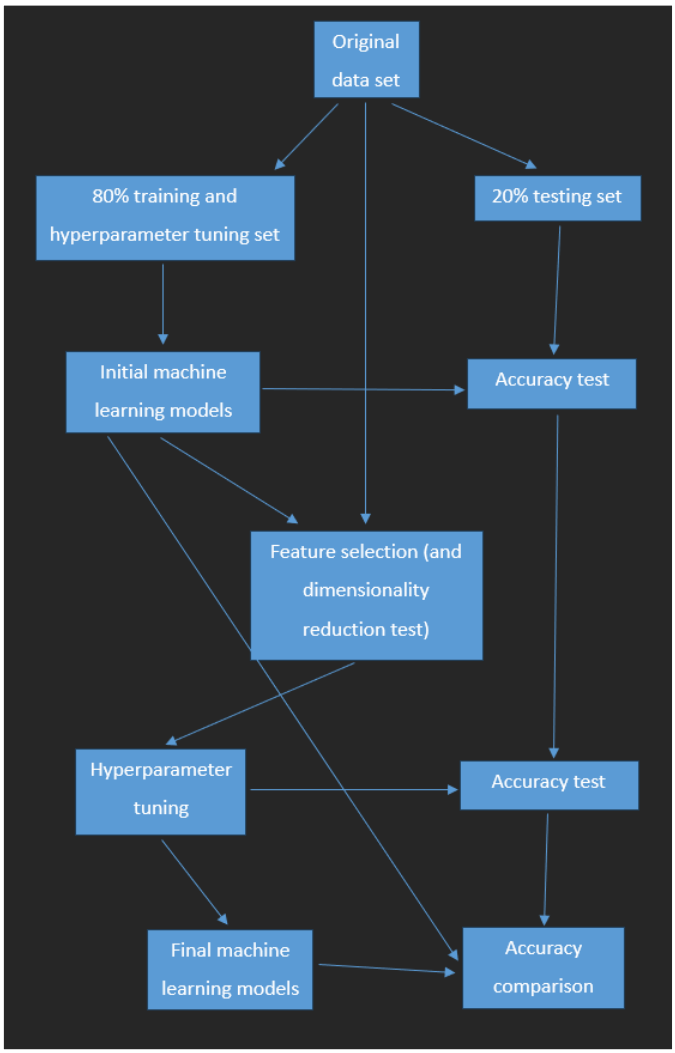

Figure 14. ML process workflow.

4 Model creation, improvement and evaluation

In this chapter, Python Jupyter Notebook environment will be used to load and preprocess the data, as well as build, tune and evaluate the ML models. The process will follow a design science research approach, where the models will be created in phases, according to the chapters they are in. Lee (2019, p. 148-154) has dedicated a whole chapter to creating ML models with code examples in Python, and thus it will be used as an example when creating models in this chapter. Chopra and Khurana (2023), and Välisuo (2023) also present examples of ML code in Python, which will be utilized.

Chopra and Khurana (2023, p. 25) present guidelines to make an ML system effective. According to them, one must develop a data pipeline as early as possible, determine high variance or high bias and take steps accordingly, and analyze missed records or patterns. According to Chopra and Khurana (2023, p. 25), in developing a pipeline one must

- Eliminate outliers or noisy data
- Deal with null values by removing or replacing null values by mean or default values
- Normalize the numerical values
- Encode the categorical attributes
- Split the data into 3 parts: 70 % training set, 15 % testing set and 15% cross validation set
- Fit and perform predictions using a suitable ML model.

Since many parts are similar to what was deemed good in the Prior research chapter (2.5), the above steps will be implemented during the ML model building process. However, some steps will be revised, such as the splitting of the data. The training and testing set will be split into 80% and 20% of the data, respectively. The training set will be used in hyperparameter tuning as well. The idea is to keep the training data separate from the testing data to test the model with unseen or novel data (Jung, 2022, p. 195).

During the splitting the data will be stratified, i.e. the class imbalance is kept similar in both sets (Lee, 2019, p. 97). The testing set will be identical before and after feature selection and hyperparameter tuning, to be able to compare the models before and after. Chicco and Jurman (2020, p. 6) use a similar division of the data, without stratification, however.

4.1 Data collection and introduction

The data set (Heart Failure Clinical Records, 2020) was originally published in Public Library of Science supplement to Ahmad et al.'s (2017a) case study and was later donated to UC Irvine Machine Learning Repository in 2020. The data was downloaded from the latter (Heart Failure Clinical Records, 2020) but is the exact same one as the former (Ahmad et al., 2017b). The data set contains the medical records of 299 heart failure patients situated in Faisalabad Institute of Cardiology, and Allied Hospital in Faisablad, Punjab, Pakistan (Ahmad et al., 2017a). The number of positive target variable instances (deceased during the follow-up period in this case) is 96, while 203 survived (Ahmad et al., 2017a). This translates to 32.11% deceased and 67.89% survived patient rates. According to Ahmad et al. (2017a), 194 of the patients are males and 105 females. All of them are over 40 years old and suffer from left ventricular systolic dysfunction (LVSD). The disease, LVSD, was diagnosed by cardiac echo report or from notes written by a physician.

Other variables, other than the event of death and age, include serum sodium, serum creatinine, gender, smoking, Blood Pressure (BP), Ejection Fraction (EF), anemia, platelets, Creatinine Phosphokinase (CPK) and diabetes (Ahmad et al., 2017a). According to them, the features were considered potential variables explaining mortality caused by chronic heart disease (CHD). *The information related to risk factors were taken from blood reports while smoking status and blood pressure were taken from physician's notes* (Ahmad et al., 2017a). Below are all the variables listed and detailed in a table (Table 1):

Table 1. Heart Failure Clinical Records (2020) features table.

Feature	**Details**	**Data type**	**Min/Max**
Age	Patient's age	Integer (years)	40/95
Anemia	Decrease of red blood cells or hemoglobin (1 if hematocrit < 36)	Boolean	0 (no)/1 (yes)
(High) BP	If the patient has hypertension	Boolean	0 (no)/1 (yes)
CPK	Level of the CPK enzyme in the blood	Integer (mcg/L)	23/7 861
Diabetes	If the patient has diabetes	Boolean	0 (no)/1 (yes)
EF	Percentage of blood leaving the heart at each contraction	Integer (percentage)	14/80
Platelets	Platelets in the blood	Floating point (kiloplatelets/mL)	25 100/850 000
Serum creatinine	Level of creatinine in the blood	Floating point (mg/dL)	0.5/9.4
Serum sodium	Level of sodium in the blood	Integer (mg/dL)	113/148
Sex	Male or female	Boolean	0 (female)/1 (male)
Smoking	If the patient smokes	Boolean	0 (no)/1 (yes)
Time	Follow up time	Integer (days)	4/285
(Target) death event	If the patient died during the follow-up period	Boolean	0 (survived)/1 (died)

Since surviving (death event = 0) is dependent on a long follow-up time, the *Time* feature will be dropped from the data set. It may be present in some future figures in chapter

4.4., but it will not be used in the training process of the models. Some of the features are significantly bigger than others in scale, e.g. platelets and serum creatinine. Therefore, features will be scaled before applying them to the ML algorithms (Xu et al., 2025).

4.2 Data cleaning and preprocessing

The ML model building process starts by downloading the data locally and importing the data to Python with the Pandas library's read_csv() function (Lee, 2019, p. 148). In addition, Numpy and Matplotlib.Pyplot are loaded for data manipulation and visualization. The file is named *data* for convenience:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data=
pd.read_csv('heart_failure_clinical_records_dataset.csv')
```

The resulting Pandas DataFrame (*data*) has 299 rows and 13 columns, of which *DEATH_EVENT* is the target variable of diseased heart failure patients, i.e. the one that will be the subject of prediction later. Next, the Pandas info() function will be used on the data, presenting the column names, non-null counts and datatypes. There are no null values in the data set, as is stated in the data set description (Heart Failure Clinical Records, 2020). Therefore, no data imputation with mean values is needed, like in the example (Lee, 2019, p. 149). Furthermore, we check that there are no values captured in the columns where data type is some other than Boolean with the eq(0) function (Lee, 2019, p. 149). Since the result shows that only Boolean data type columns have null values, the data cleaning is completed (Lee, 2019, p. 149).

Next, the data is split to the target variable array *y* and the feature array *x*. The target data must be in a separate array so that the ML algorithms are not fitted with the whole

data set, so that the algorithms have the target to predict on. This is done with the following code:

```
y = data['DEATH_EVENT']
x = data.iloc[:,:11]
```

The code also drops the follow-up time feature. The code creates an array *y* which has only the target variable's Boolean values and an array *x* which excludes the target variable and has the remaining columns of the data set. The scaling of numerical variables will lastly be done with StandardScaler algorithm (Scikit-Learn, n.d. b) through the Scikit-Learn Pipeline, where the scaling of features and classification can be combined into one line of code (Scikit-Learn, n.d. c). This is demonstrated in the following chapter.

4.3 Model training

The data will be split into a training and testing set, which will be divided into 80% and 20% of the data, respectively. The split will use a random state, so that the splitting of the data can be repeated identically afterwards. This is achieved with the following code:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=.8, random_state=0, stratify=y)
```

Where the *random_state* input can be any integer. The *stratify* parameter keeps the class imbalance similar to the original data set (Lee, 2019, p. 97), therefore both the training and testing set will have approximately the same ratio of survived and diseased patients. Here is a Pipeline, where the scaling of features and a Logistic Regression model are combined into one line of code (Scikit-Learn, n.d. c):

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
lr_pipeline = Pipeline([('Scaling', StandardScaler()),
('Logistic Regression',
LogisticRegression())]).fit(x_train,y_train)
```

The fit() function trains the Pipeline's Logistic Regression model and scales the features in *x_train* array. The Pipeline can now be used for prediction. Therefore, a confusion matrix is printed with each associated accuracy score to see how the models classify the testing data. This is demonstrated in the below code for Logistic Regression for example (Algorithm 1):

```
Code:
print(metrics.confusion_matrix(y_true=y_test,
y_pred=lr_pipeline.predict(x_test)))
print("The accuracy of Logistic Regression is %4.2f" %
metrics.accuracy_score(y_true=y_test,
y_pred=lr_pipeline.predict(x_test)))
lr_score = metrics.accuracy_score(y_true=y_test,
y_pred=lr_pipeline.predict(x_test))
results = []
results.append(lr_score)

Output:
[[36  1]
 [12 11]]
The accuracy of Logistic Regression is 0.78
```

Algorithm 1. Logistic Regression confusion matrix code and output.

The accuracy is stored in a variable called *lr_score*, which is then stored in an array of all the results, called *results*. The confusion matrix output denotes that 36 patients were classified correctly as not diseased, and 1 diseased patient was incorrectly classified as survived. On the contrary, 11 diseased patients were correctly classified as diseased, while 12 diseased patients were incorrectly classified as survived. The above process is repeated for each model.

Following the example of Lee (2019, p. 152), the first ML models to be trained are Logistic Regression, KNN and SVM with a radial basis function (RBF) kernel as default kernel. The models will be trained with the training data and without modifying the parameters, such as in the above example of a Logistic Regression Pipeline. After them, Tree-based

and ANN ML models will be trained on the data. The Tree-based models include a Decision Tree, Random Forest, ExtraTreesClassifier (ETC), and Gradient Boosted Classifier (GBC). In GBC, the predictor is an aggregation of many weak individual predictors, often small decision trees (Friedman, 2001). ETC was trained on the data to see if randomly selected threshold rules for randomly selected features (Geurts et al., 2006) can be a good predictor.

After training all the models, they are evaluated with the testing set. Since the hyperparameters will also be optimized with the training set, the models will never see the testing data outside of prediction, which makes the prediction fair for each model and doesn't make the accuracy biased (Välisuo, 2023d).

4.4 Feature selection

To select the most relevant features for heart failure mortality prediction, pairwise collinearity, Random Forest Gini impurity and Random Forest Permutation feature importance are utilized. The most relevant features will then be selected in aggregate according to these techniques, which Lee (2019, p. 150), and Chicco and Jurman (2020, p. 5) also utilize.

First, pairwise collinearity is examined between the independent variables and the target variable (Lee, 2019, p. 150), i.e. whether the heart failure patient died during the follow-up time. This can be done with the corr() function of the Pandas DataFrame, which calculates the pairwise correlations for each column. Instead of simply showing the correlation matrix, the correlations will be plotted in a heatmap, where strong correlations (to both directions) will have the most color and weaker correlations will have less color (Lee, 2019, p. 151).

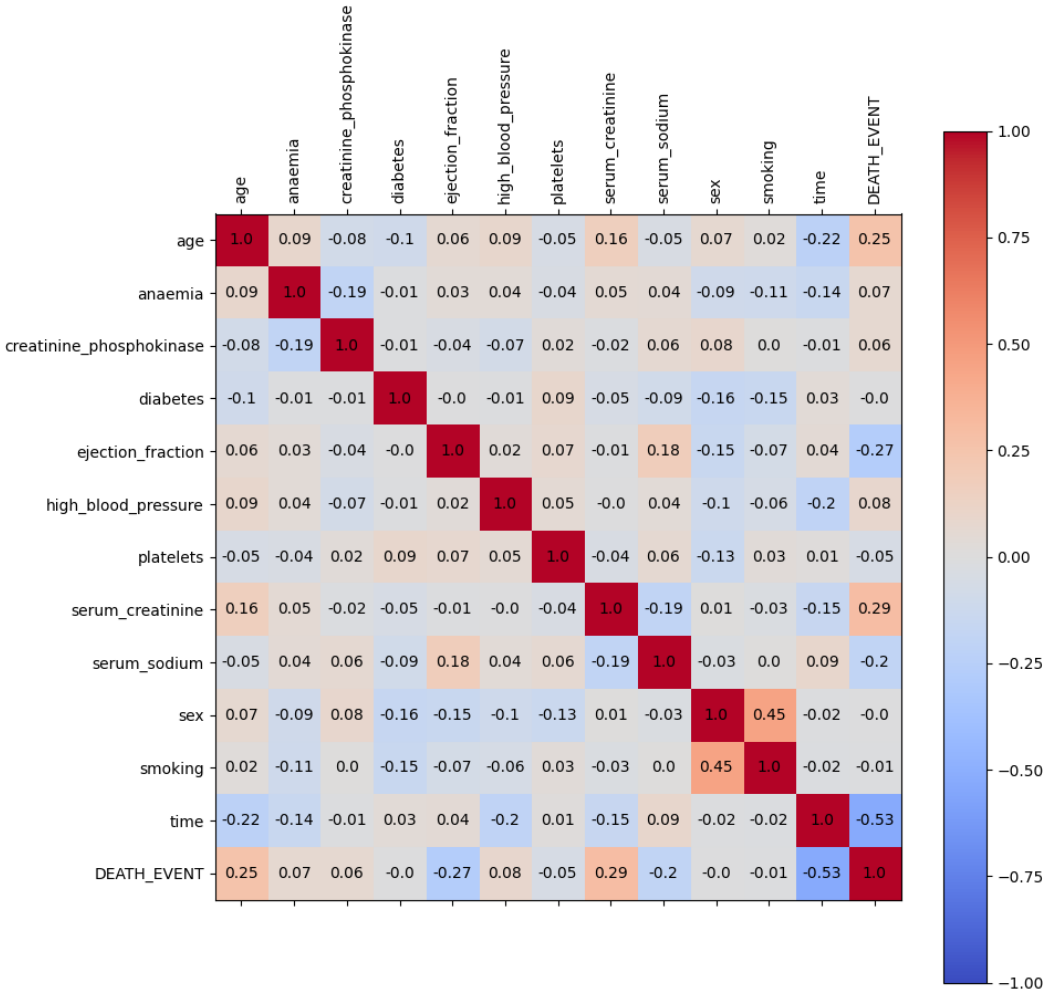

Figure 15. Pairwise collinearity heatmap between the data set's variables; the stronger the color, the stronger the correlation.

As evident from Figure 15, only the follow-up time has an absolute correlation stronger than 0.5 in relation to the target variable. This can be explained by most survived patients in the data set (203 out of 299) having a long follow-up time, therefore there exists a negative correlation. However, it won't be considered as a predictor, since surviving is dependent on a long follow-up time. Furthermore, only serum creatinine, ejection fraction and age have an absolute correlation equal to or stronger than 0.25. The following code and output show the most correlated variables and the values associated with them:

```
Code:
abs_corr = abs(data.corr())
print(abs_corr.nlargest(5, 'DEATH_EVENT').index)
print(abs_corr.nlargest(5,  'DEATH_EVENT').values[:,12])
Output:
Index(['DEATH_EVENT',    'time',    'serum_creatinine',
'ejection_fraction', 'age'], dtype='object')
[1.         0.52696378 0.29427756 0.26860331 0.25372854]
```

The code follows the example of Lee (2019, p. 152) but considers that there are also negatively correlated variables. As stated above, the three strongest correlations to the target variable, excluding itself and follow-up time, are serum creatinine, ejection fraction and age. Since ejection fraction has a negative correlation, it means that as it increases, the target variable becomes less likely. Thus, as the percentage of blood leaving at each contraction or time increases, the likelihood of dying decreases, and vice versa. On the contrary, since serum creatinine and age had a positive correlation, the likelihood of dying increases as these variables increase, and vice versa. Serum creatinine means the level of creatinine in the blood (Ahmad et al., 2017a).

Random Forest feature importance provides multiple ranking techniques, but two are utilized here: the built-in mean decrease in impurity (MDI) for forest of trees and permutation importance (Breiman, 2001). According to Breiman (2001), feature importance based on MDI is computed as the mean and standard deviation of accumulation of the decrease in impurity within each tree (Breiman, 2001). Permutation feature importance, on the other hand, measures the contribution of each feature to a fitted model's statistical performance on a given dataset (Breiman, 2001). It randomly shuffles the values of a single feature and observes the resulting degradation of the model's score; this way, it determines how much a model relies on a particular feature (Breiman, 2001). Both MDI and permutation importances will be shown in figures beneath:

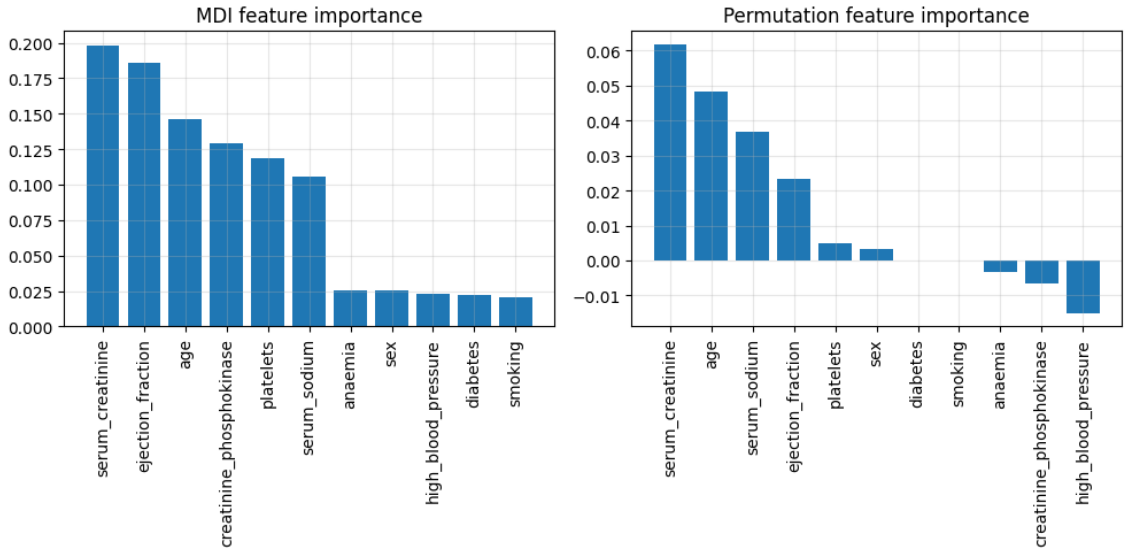

Figure 16. Random Forest feature importance by MDI and permutation in decreasing order, higher value is more important for both. Follow-up time is excluded.

The most important features are serum creatinine, ejection fraction and age in both collinearity matrix and Random Forest feature importance (Figure 15 and 16). Thus, these three features, excluding follow-up time, will be the ones that will be selected, i.e. others will be dropped. Dropping the least important features and storing the most important ones in an array is done with the following code:

```
smaller_x = x.iloc[:,[0,4,7]],
```

where the original feature array *x* is stored in a smaller array *smaller_x* with the selected features. Lee (2019, p. 152) also uses the most influential variables to train the models in order to reduce the amount of noise in the data. This also abides by Chopra and Khurana's (2023, p. 25) guidelines in developing an ML pipeline.

4.5 Hyperparameter tuning

In this chapter, each earlier ML model's hyperparameters will be optimized by the Scikit-Learn (n.d. a) RSCV algorithm (Bergstra & Bengio, 2012). Each model is optimized to have the best possible prediction accuracy with the underlying data. The parameters can be

found on each ML algorithm's website, e.g. DecisionTreeClassifier's parameters on Scikit-Learn's website (Scikit-Learn, n.d. d). Before applying RSCV to the models, the training data must be scaled since a Pipeline with StandardScaler cannot be searched during RSCV. Therefore, this code is run before anything else:

```
scaled_smaller_x = StandardScaler().fit_transform(smaller_x)
scaled_x_train,       scaled_x_test,       scaled_y_train,
scaled_y_test=train_test_split(scaled_smaller_x,          y,
train_size=.8, random_state=0, stratify=y)
```

The above code scales the selected features before dividing them into a training and a testing set so that RSCV can capture the optimal parameters correctly. After finding the suitable parameters from DecisionTreeClassifier's guide (Scikit-Learn, n.d. d), RSCV is run on Decision Tree with the following code (Algorithm 2):

```
from scipy.stats import randint
dt_grid = [{'max_depth': randint(1,10),
            'min_samples_split': randint(1,10),
            'criterion': ['gini', 'entropy', 'log_loss'],
            'splitter': ['best', 'random'],
            'min_samples_split': randint(1,10),
            'min_samples_leaf': randint(1,10),
            'max_features': randint(1,12)}]
dt_search   =   RandomizedSearchCV(DecisionTreeClassifier(),
dt_grid,  cv=10,  scoring='accuracy',  verbose=1,  n_jobs=-1,
n_iter=100, random_state=0)
dt_search.fit(scaled_x_train,scaled_y_train)
optimal_dt_param = dt_search.best_estimator_
optimal_dt = Pipeline([('Scaling', StandardScaler()), ('DT',
optimal_dt_param)]).fit(x_train, y_train)
```

Algorithm 2. Decision Tree RSCV hyperparameter optimization implementation.

The optimizable hyperparameters are stored in the *dt_grid* array and RSCV makes a randomized search through them to find the best predictor in 100 iterations (*n_iter=100*), paired with a 10-fold cross-validation (*cv=10*). The optimal Decision Tree parameters that RSCV finds, which yield the best prediction accuracy with 10-fold cross-validation with the training set, are stored in *optimal_dt_param* to then use for prediction when implementing the Pipeline. Optimal parameters are searched and stored for each model

identically, only searched parameters are different. Below are all the optimal parameters (Table 2):

Table 2. Models' optimal hyperparameters by RSCV and with 3 most significant features, follow-up time excluded.

Model	**Optimal hyperparameters by RSCV**
Logistic Regression	C=np.float64(11.288378916846883), max_iter=2500
SVM	C=np.float64(0.00026366508987303583), max_iter=1000, solver='liblinear'
KNN	metric='manhattan', n_neighbors=19, weights='distance'
Decision Tree	criterion='entropy', max_depth=24, max_features=2, min_samples_leaf=3, min_samples_split=3, splitter='random'
Random Forest	criterion='entropy', max_depth=45, n_estimators=25
ETC	criterion='entropy', max_depth=12, min_samples_split=4, n_estimators=29
GBC	criterion='squared_error', max_depth=33, min_samples_leaf=4, min_samples_split=3, n_estimators=34
ANN	activation='logistic', hidden_layer_sizes=(3,), learning_rate_init=1, solver='sgd'

RSVC found many parameters which are not set by default, e.g. both Logistic Regression and SVC have 1 as default value of *C* (Table 2). Furthermore, KNN, tree-based models and ANN had many parameters which were different in the initial models by default, such as *n_neighbors* (5 by default), *criterion* (*gini* by default), and *learning_rate* (*constant*

by default). In the next chapter, results will be presented before and after feature selection and hyperparameter tuning.

5 Results

The initial unoptimized models' accuracy scores are compared in the following table:

Table 3. Unoptimized all features models' accuracy in decreasing order.

Rank	Model	Accuracy
1.	Logistic Regression	78.33%
2.	SVM (RBF kernel)	76.67%
3.	GBC	76.67%
4.	Random Forest	76.67%
5.	ANN	76.67%
6.	ETC	71.67%
7.	KNN	68.33%
8.	Decision Tree	65%

With all the features and without optimization, Logistic Regression achieves the best accuracy (Table 3). SVM, GBC, Random Forest and ANN are not far behind (<5%), however. Logistic Regression is the best predictive model in this instance, since it is moreover the simplest and requires the least computational resources (Alpaydin, 2020, p. 36; Välisuo, 2023e). The worst performing models are Decision Tree and KNN at this stage, but each model will be considered after feature selection and hyperparameter tuning. This is also crucial to answer the third research question.

During feature selection, it was discovered that the most significant features were serum creatinine, ejection fraction and age. Pairwise collinearity, Random Forest MDI and permutation feature importance were used to detect the most significant features in the data set. Follow-up time was dropped already before the model training stage (chapter 4.3), since surviving is dependent on a long follow-up time. After feature selection all, except the most significant features were dropped (8 out of 11 features). Therefore, only three features were used in RSCV and the final optimized models. The following table

presents the accuracy scores of each optimized model, evaluated with the testing set which the models have never seen during the training and optimization process:

Table 4. Optimized models' accuracy and comparisons to unoptimized models (Table 3).

Rank	**Rank change**	**Model**	**Accuracy**	**Accuracy change**
1.	**+6**	**Optimized KNN**	**81.67%**	**+13.33%**
2.	-1	Optimized Logistic Regression	80%	+1.67%
3.	-1	Optimized SVM (linear kernel)	80%	+3.33%
4.	+4	Optimized Decision Tree	78.33%	**+13.33%**
5.	-1	Optimized Random Forest	78.33%	+1.66%
6.	-3	Optimized GBC	76.67%	-
7.	-2	Optimized ANN	76.67%	-
8.	-2	Optimized ETC	75%	+3.33%

Rank and accuracy changes from chapter 4.3 are included in the above table (Table 4) to demonstrate the shifts in prediction accuracy score in Table 3. KNN has achieved the best prediction accuracy of 81.67%, while Logistic Regression, SVM, Decision Tree, and Random Forest are almost as accurate. Logistic Regression and SVM are the second-best models according to accuracy (Table 4), and thus they have been generally good predictive models with this data set.

Furthermore, the scores improved most on the models which had the weakest prediction accuracy in the initial stage, mainly KNN and Decision Tree. Their accuracy improved by 13.33% (Table 4). This is impressive, since both models had also the highest

jump in ranking, moreover KNN having the best accuracy of all models. SVM and ETC also improved their accuracy. Every model except ANN and GBC achieved an improved prediction accuracy compared to prior results (Table 3).

The optimized results' confidence intervals will be presented to give an error margin for the predictions (Chopra & Khurana, 2023, p. 92). If the distribution for the predictions is estimated to be normal (Alpaydin, 2020, p. 423), a 95 percent confidence level *z-value* and the 95 percent confidence interval for Logistic Regression can be calculated with the below code:

```
import scipy.stats
confidence = 0.95
z_value = scipy.stats.norm.ppf((1 + confidence) / 2.0)
logreg_ci_length = z_value * np.sqrt((optimized_logreg * (1
- optimized_logreg)) / y_test.shape[0])
logreg_ci_lower = optimized_logreg - logreg_ci_length
logrer_ci_upper = optimized_logreg + logreg_ci_length
```

The interval is then appended to a confidence interval list to present each of them at once. Since the length of the testing set is only 60, the confidence bounds will have some distance between them. The table below shows these intervals for each model (Table 5):

Table 5. Models' accuracy confidence interval with a normal distribution assumption.

Model	**Lower confidence bound**	**Upper confidence bound**	**Accuracy**
KNN	71.55%	91.79%	81.67%
Logistic Regression	69.88%	90.12%	80%
SVM	69.88%	90.12%	80%
Decision Tree	68.21%	88.45%	78.33%
Random Forest	67.91%	88.76%	78.33%
GBC	65.96%	87.37%	76.67%
ANN	65.96%	87.37%	76.67%
ETC	64.04%	85.96%	75%

Though accuracy was the main metric that was considered at the end, it is important to see how the ML models classified positives and negatives through the confusion matrices (Lee, 2019, p. 100). Each model was associated with a confusion matrix in the initial stage (chapter 4.3), and after feature selection and hyperparameter tuning. Since almost every model improved its accuracy, consequently the number of correctly classified instances in confusion matrices were also improved. Below is a table where each optimized model's TP, TN, FP, FN figures can be seen in Table 4's order:

Table 6. Optimized models' confusion matrix figures in decreasing accuracy order.

Model	**TP**	**TN**	**FP**	**FN**
KNN	**63.2% (12)**	90.2% (37)	**31.6% (7)**	9.8% (4)
Logistic Regression	47.4% (9)	**95.1% (39)**	52.6% (10)	**4.9% (2)**
SVM	52.6% (10)	92.7% (38)	47.4% (9)	7.3% (3)
Decision Tree	52.6% (10)	82.9% (37)	47.4% (9)	17.1% (4)
Random Forest	**63.2% (12)**	85.4% (35)	**31.6% (7)**	14.6% (6)
GBC	57.9% (11)	85.4% (35)	42.1% (8)	14.6% (6)
ANN	**63.2% (12)**	80.5% (34)	**31.6% (7)**	19.5% (7)
ETC	52.6% (10)	82.9% (35)	47.4% (9)	17.1% (6)

In the above table, the correctly and incorrectly classified cases are in parenthesis next to the percentage figures. Interestingly, the models that classified diseased patients best were not the best performing models (Table 5). KNN, Random Forest and ANN had a TP rate of 63.2%. Therefore, they were the most successful models in capturing the relevance of positive instances from the data. Furthermore, Logistic Regression classified survived patients the best, at 95.1% accuracy. This is almost 5% better than KNN's TN rate as well. However, at the end KNN was the most accurate model in total.

The answers to the research questions are

1. ML can predict heart failure mortality from a clinical data set with 75% and higher accuracy, no matter what the model is in this thesis.
2. Feature selection and hyperparameter tuning were successful techniques in improving prediction accuracy.
3. The best accuracy was yielded by KNN: 81.67%.

Furthermore, the goal of creating an ML model that can predict heart failure mortality from a clinical data set was successful. Thus, it can be concluded that with the chosen

ML algorithms, both feature selection and hyperparameter tuning have mostly succeeded in improving prediction accuracy with the data set, as well as in removing less significant features; the features used in making the prediction have been reduced to 3 features or approximately 1/4 of the original data set, which generalizes the models significantly and makes them more accessible. Moreover, the reduced number of features has made the prediction more accurate (Table 4). Since the research questions could be answered and the thesis' goal was accomplished, the thesis can be deemed successful too.

6 Discussion & conclusions

Lastly in this chapter, the results of the thesis will be compared to prior research on the data set and on similar ones. By using Python's ML libraries, feature selection and hyperparameter tuning by RSCV, it was discovered that the best performing optimized ML model predicting on three features in this thesis was KNN with an accuracy of 81.67%.

While the accuracy score achieved by the most accurate model in this thesis (KNN) is lower than Chicco and Jurman (2020) achieved with a stratified Logistic Regression (83.8%), it is much higher than what they achieved with other ML models prior. For example, KNN, with optimized parameters by GSCV, was their worst performing model (62.4% accuracy) while predicting on all the features. This is due to most features being non-significant for the prediction of mortality, as discovered during feature selection in chapter 4.4. Furthermore, only Random Forest, Gradient Boosting and SVM with an RBF kernel were considered while they predicted on ejection fraction and serum creatinine alone, in addition to stratified Logistic Regression (Chicco & Jurman, 2020, p. 7-10). The former models trained on two features yielded 58.5%, 58.5% and 54.3% accuracy (Chicco & Jurman, 2020, p. 10), which are significantly weaker than what the same ML models achieved during this thesis (Table 4). It is important to note, however, that Chicco and Jurman (2020) used a mean of 100 executions validation technique, and thus the results are not directly comparable.

Furthermore, in this thesis it was proved that with a three-feature combination, i. e. serum creatinine, ejection fraction and age, and by optimizing hyperparameters with RSCV, many initially worse performing models could achieve significant improvements (Table 4). Chicco and Jurman (2020, p. 8-9) identified ejection fraction and serum creatinine as the most significant features, while Ahmad et al. (2017a) identified age, serum creatinine, high blood pressure, ejection fraction and anemia as top features. Therefore, the results in feature selection differ slightly from prior research but are at least in line with Chicco and Jurman (2020).

Prior research, which uses accuracy as a metric in the results of the final models, include Sarijaloo et al. (2021), Al Younis et al. (2024) and Xu et al. (2025). Their final models were Lasso Logistic Regression, Decision Tree and XGBoost, with accuracy of 71%, 91.2% and 81.58%, respectively. Therefore, the final KNN model in this thesis surpasses the model of Sarijaloo et al. (2021) and Xu et al. (2025) in accuracy. However, Sarijaloo et al. (2021) and Xu et al. (2025) had both data sets with more than 3500 data points, while Al Younis et al. (2024) had a similar number of data points (303) but much more detailed features in their data set, which can be extremely significant for the prediction of heart failure.

As Chicco and Jurman (2020, p. 13) state, there are limitations to the data set and ML models applied on it. The number of studies on the data set (Heart Failure Clinical Records, 2020) is still very low and further confirmatory studies are required to take any of the ML models into clinical practice. However, at least ejection fraction and serum creatinine have been identified as significant features in this thesis, and in studies of Chicco and Jurman (2020) and Ahmad et al. (2017). ML has also been applied successfully in both this thesis and in Chicco and Jurman's (2020) work. Furthermore, as this thesis proves, initially weak predictive models should not be discarded before feature selection and hyperparameter tuning, as the best model in this thesis was the least accurate initially. KNN might have more use cases in more detailed data sets, moreover.

As it is impossible to test the final KNN model in another cohort, e.g. a data set with same features but from a different region (Chicco & Jurman, 2020, p. 13), it is not possible to validate it with out-of-sample data, and thus its robustness and generalization cannot be perfectly tested. Furthermore, as Chicco and Jurman (2020, p. 13) argue, more features of patients' physical features, such as body mass index, would have been useful to further improve the accuracy of the models. Lastly, as stated in earlier parts of the thesis, the data set is very small (Chicco & Jurman, 2020, p. 13), having only 299 data points in total. With more data points and a more balanced data set the ML models could have had more accurate predictions. Thus, the ML model is not

ready yet for clinical implementation, without further testing. However, with as few as three features, it would be easy to test with out-of-sample data.

Since RSCV was applied exhaustively on selected features and on all models, further research on the data set and similar ones should consider and develop hybrid ML models, such as Xu et al. (2025) have done in their research. Their model (Xu et al., 2025) was constructed using synthetic minority over-sampling technique combined with edited nearest neighbors (SMOTE-ENN) processing and applying XGBoost model. However, more research on hybrid models is necessary to make heart failure mortality prediction more accurate and robust. Otherwise, more high-quality data is necessary for non-hybrid models to achieve better accuracy in estimating heart failure mortality.

References

Adler, E.D., Voors, A.A., Klein, L., Macheret, F., Braun, O.O., Urey, M.A., Zhu, W., Sama, I., Tadel, M., Campagnari, C., Greenberg, B. and Yagil, A. (2020). Improving risk prediction in heart failure using machine learning. Eur J Heart Fail, 22: 139-147. https://doi.org/10.1002/ejhf.1628

Ahmad T., Munir A., Bhatti S. H., Aftab M. & Raza M. A. (2017a). Survival analysis of heart failure patients: A case study. PLoS ONE 12(7): e0181001. https://doi.org/10.1371/journal.pone.0181001

Ahmad T., Munir A., Bhatti S. H., Aftab M. & Raza M. A. (2017b). DATA_MINIMAL.. PLOS ONE. Dataset. https://doi.org/10.1371/journal.pone.0181001.s001

Alpaydin, E. (2020). *Introduction to machine learning, fourth edition*. MIT Press.

Al Younis, S. M., Hadjileontiadis, L. J., Khandoker, A. H., Stefanini, C., Soulaidopoulos, S., Arsenos, P., . . . Tsioufis, K. (2024, May 15). Prediction of heart failure patients with distinct left ventricular ejection fraction levels using circadian ECG features and machine learning. *PloS one, 19*(5), e0302639. https://doi.org/10.1371/journal.pone.0302639

Bentéjac, C., Csörgő, A., & Martínez-Muñoz, G. (2021, March 1). A comparative analysis of gradient boosting algorithms. *The Artificial intelligence review, 54*(3), 1937-1967. https://doi.org/10.1007/s10462-020-09896-5

Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13, null (3/1/2012), 281–305. Retrieved 2025-11-05 from https://dl.acm.org/doi/10.5555/2188385.2188395

Breiman, L. (2001). Random Forests. *Machine Learning* **45**, 5–32. https://doi.org/10.1023/A:1010933404324

Chen, M., Hao, Y., Hwang, K., Wang, L., & Wang, L. (2017). Disease Prediction by Machine Learning Over Big Data From Healthcare Communities. *IEEE access, 5*, 8869-8879. https://doi.org/10.1109/ACCESS.2017.2694446

Chicco, D. & Jurman, G. (2020). Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. *BMC Med Inform Decis Mak* 20, 16. https://doi.org/10.1186/s12911-020-1023-5

Chopra, D., & Khurana, R. (2023). *Introduction to machine learning with python*. Bentham Science Publishers.

Dresch, A., Pacheco Lacerda, D., & Cauchick Miguel, P. A. (2015). A Distinctive Analysis of Case Study, Action Research and Design Science Research. *Revista brasileira de gestão de negócios, 17*(56), 1116-1133. https://doi.org/10.7819/rbgn.v17i56.2069

Escolar, V., Lozano, A., Larburu, N., Kerexeta, J., Álvarez, R., Echebarria, A. & Azcona, A. (2022). Prediction of heart failure decompensations using artificial intelligence and machine learning techniques. Cardiology Department, School of Medicine, Basurto University Hospital, Bilbao. Rev Colomb Cardiol. 2022;29(4):431-440. https://doi.org/10.24875/RCCAR.21000023

Freund, Y. & Schapire, R. E. (1999) A short introduction to boosting. J Jpn Soc Artif Intell 14(5):771–780. Retrieved 2025-09-17 from https://docslib.org/doc/11394707/a-short-introduction-to-boosting

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Statist*. 29 (5) 1189 - 1232. https://doi.org/10.1214/aos/1013203451

Geurts, P., Ernst., D. & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1), 3-42, 2006. https://doi.org/10.1007/s10994-006-6226-1

Heart Failure Clinical Records [Dataset]. (2020). UCI Machine Learning Repository. https://doi.org/10.24432/C5Z89R.

Hevner, A., Alan R., March, Salvatore, T., Park, J. & Sudha, R. (2004). Design Science in Information Systems Research. Management Information Systems Quarterly. 28. 75-. https://doi.org/10.2307/25148625

Jung, A. (2022). *Machine learning : The basics*. Springer.

Järvinen, P. (1999). *On Research Methods.* University of Tampere.

Lee, W. (2019). *Python machine learning*. John Wiley & Sons, Incorporated.

Lommi, J., Lassus, J., Brinkmann, S., Junttila, J., Kokkonen, J., Koljonen, E., Mervaala, E., Mustonen, P., Pentikäinen, M., Posio, P. & Ukkonen, H. (2023, December 19). Heart Failure (Sydämen vajaatoiminta). Current Care Guidelines. Retrieved 2025-10-19 from https://www.kaypahoito.fi/hoi50113?tab=suositus

McDonagh, T. A., Adamo, M., Butler, J., Chioncel, O., Crespo-Leiro, M. G., Farmakis, D., . . . Witkowski, A. (2024). 2023 Focused Update of the 2021 ESC Guidelines for the diagnosis and treatment of acute and chronic heart failure. *European journal of heart failure, 26*(1), 5-17. https://doi.org/10.1002/ejhf.3024

Nyyssönen, T. (2025, September 5). Ennemmin tai myöhemmin tapahtuu katastrofi, sanoo hoitaja – päivystykset karmeassa tilanteessa Pohjois-Savossa. Yle. Retrieved 2025-09-07 from https://yle.fi/a/74-20180699

OECD/European Commission. (2024). *Health at a Glance: Europe 2024: State of Health in the EU Cycle*. OECD Publishing. Paris. https://doi.org/10.1787/b3704e14-en

Rahmani, A. M., Yousefpoor, E., Yousefpoor, M. S., Mehmood, Z., Haider, A., Hosseinzadeh, M., & Ali Naqvi, R. (2021, November 1). Machine Learning (ML) in Medicine: Review, Applications, and Challenges. *Mathematics (Basel), 9*(22), 2970. https://doi.org/10.3390/math9222970

Roiha, M. (2025, January 17). Tekoäly testiin hyvinvointialueilla, tarkoitus on etsiä helpotusta ammattilaisten työhön. Yle. Retrieved 2025-09-07 from https://yle.fi/a/74-20137674

Sarijaloo, F., Park, J., Zhong, X. & Wokhlu, A. (2021). Predicting 90 day acute heart failure readmission and death using machine learning-supported decision analysis. *Clinical cardiology (Mahwah, N.J.), 44*(2), 230-237. https://doi.org/10.1002/clc.23532

Scikit-Learn. (n.d. a). RandomizedSearchCV. Retrieved 2025-11-05 from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

Scikit-Learn. (n.d. b). StandardScaler. Retrieved 2025-11-21 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler

Scikit-Learn. (n.d. c). Pipeline. Retrieved 2025-11-21 https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Scikit-Learn. (n.d. d). DecisionTreeClassifier. Retrieved 2025-10-30 from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

VanderPlas, J. (2016). Python Data Science Handbook. Retrieved 2025-09-24 from https://jakevdp.github.io/PythonDataScienceHandbook/index.html

Välisuo, P. (2023a). Applied Machine Learning. Retrieved 2025-08-31 from https://pevalisuo.github.io/AML/NeuralNetworks.html

Välisuo, P. (2023b). Applied Machine Learning. Retrieved 2025-09-23 from https://pevalisuo.github.io/AML/Clustering.html

Välisuo, P. (2023c). Applied Machine Learning. Retrieved 2025-10-14 from https://pevalisuo.github.io/AML/Preprocessing_and_feature_extraction.html

Välisuo, P. (2023d). Applied Machine Learning. Retrieved 2025-10-14 from https://pevalisuo.github.io/AML/SupervisedMachineLearningTerminology.html

Välisuo, P. (2023e). Applied Machine Learning. Retrieved 2025-10-19 from https://pevalisuo.github.io/AML/Regression.html

Xu, Q., Yu, R., Cai, X., Chen, G., Zheng, Y., Xu, C., & Sun, J. (2025). Machine learning-based risk factor analysis and prediction model construction for mortality in chronic heart failure. *Journal of global health, 15*, 04242. https://doi.org/10.7189/jogh.15.04242

Appendices

Appendix 1. Python Jupyter Notebook initial models code

```
# Import the libraries used for data pre-processing and
extraction.
import pandas as pd

# Read the data and view it.
data                                                    =
pd.read_csv('heart_failure_clinical_records_dataset.csv')
data

# Next, the data types. .info() also checks the number of
null values, which should be zero according to the repository.
data.info()

# Let's also check that there are no values captured in the
columns where the data type is some other than boolean. Here
one can also see the number of survived patients.
data.eq(0).sum()

# Lastly, let's see some descriptive statistics of the data
set.
data.describe()

# Let's separate features and the target (DEATH_EVENT/if the
patient died during the follow-up period; 0 if no, 1 if yes)
to own arrays, the target array y:
y = data['DEATH_EVENT']
y.describe()
y

# And the feature array x, excluding follow-up time:
x = data.iloc[:,:11]

# Next, before building the machine learning models,
numerical variables will be scaled in the Pipelines so that
none of them has too large of an impact
# on the models. The required libraries are imported for
this.
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics

# The data is split into a training set (80% of the data)
and a testing set (20% of the data).
x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=.8, random_state=0, stratify=y)
```

```
# We'll create a Pipeline for Logistic Regression with a
StandardScaler.
lr_pipeline = Pipeline([('Scaling', StandardScaler()),
('Logistic Regression',
LogisticRegression())]).fit(x_train,y_train)

# We'll print the confusion matrix and the accuracy
associated to the matrix.
print(metrics.confusion_matrix(y_true=y_test,
y_pred=lr_pipeline.predict(x_test), labels=None,
sample_weight=None))
print("The accuracy of Logistic Regression is %4.2f" %
metrics.accuracy_score(y_true=y_test,
y_pred=lr_pipeline.predict(x_test)))
lr_score = metrics.accuracy_score(y_true=y_test,
y_pred=lr_pipeline.predict(x_test))

# Create an array to store the accuracy scores for later and
append Logistic Regression score.
results = []
results.append(lr_score)

## The above is repeated for each model.

# Let's print the results of each model's accuracy thus far.
algorithms = ["Logistic Regression", "K-Nearest Neighbors",
"SVM (RBF)", "Decision Tree", "Random Forest",
"ExtraTreesClassifier", "Gradient Boosting Classifier",
"ANN"]
cv_mean = pd.DataFrame(results,index=algorithms)
cv_mean.columns=["Accuracy"]
cv_mean.sort_values(by='Accuracy',ascending=False)
```

Appendix 2. Python Jupyter Notebook feature selection code

```
# Import the libraries used for data pre-processing,
extraction, and visualization.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read the data and view it.
data =
pd.read_csv('heart_failure_clinical_records_dataset.csv')
# Let's separate features and the target (DEATH_EVENT/if the
patient died during the follow-up period; 0 if no, 1 if yes)
to own arrays, the target array y:
y = data['DEATH_EVENT']
# And the feature array x:
x = data.iloc[:,:11]
x
```

```
# Let's next examine the pairwise correlations between
independent variables and store them in an array.
corr = data.corr()


# A collinearity figure is printed of the data.
fig, ax =plt.subplots(figsize=(10,10))
cax = ax.matshow(corr,cmap = 'coolwarm',vmin = -1,vmax = 1)
fig.colorbar(cax)
ticks = np.arange(0,len(data.columns),1)

ax.set_xticks(ticks)
ax.set_xticklabels(data.columns)
plt.xticks(rotation = 90)

ax.set_yticklabels(data.columns)
ax.set_yticks(ticks)

for i in range(data.shape[1]):
    for j in range(13):
        text  =  ax.text(j,  i,  round(corr.iloc[i][j],2),
ha='center', va='center')


# With the following code, one can print the five most
correlated variables and their values.
abs_corr = abs(data.corr())
print(abs_corr.nlargest(5, 'DEATH_EVENT').index)
print(abs_corr.nlargest(5, 'DEATH_EVENT').values[:,12])

# Next, let's use Random Forest-based feature importance to
see the most important variables.
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
# The data is split into a training set (80% of the data)
and a testing set (20% of the data).
x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=.8, random_state=0, stratify=y)

# Random Forest is trained with the training data.
rf_features                                                =
RandomForestClassifier(random_state=0).fit(x_train,y_train)
# MDI feature names and importances will be appended and
sorted in rf_importances list.
rf_importances_list                                        =
list(zip(rf_features.feature_names_in_,
rf_features.feature_importances_))
rf_importances  =  sorted(rf_importances_list,  key=lambda
x:x[1], reverse=True)
print(rf_importances)
```

```
# To later plot the names on the x and importances on the y-axis.
x_val = [x[0] for x in rf_importances]
y_val = [x[1] for x in rf_importances]
print(x_val, y_val)

# Permutation feature importance
from sklearn.inspection import permutation_importance
result = permutation_importance(rf_features, x_test, y_test, n_repeats=10, random_state=0, n_jobs=-1)
perm_imp_df = pd.DataFrame({'Feature': rf_features.feature_names_in_, 'Permutation Importance': result.importances_mean}).sort_values('Permutation Importance', ascending=False)
print(perm_imp_df)


# As discovered in the correlation matrix, the most important features are almost the same. However, creatinine phosphokinase and platelets are surprisingly higher here.
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)  # 1 row, 2 columns, plot 1
plt.bar(x_val,y_val)
plt.title('Gini impurity feature importance')
plt.grid(visible=True,alpha=.3)
plt.xlabel=data.columns
plt.ylabel=('Gini importance')
plt.xticks(rotation=90)
plt.tight_layout()

plt.subplot(1, 2, 2)  # 1 row, 2 columns, plot 2
plt.bar(perm_imp_df['Feature'], perm_imp_df['Permutation Importance'])
plt.title('Permutation feature Importance')
plt.xticks(rotation=90)
plt.grid(visible=True,alpha=.3)
plt.ylabel=('Permutation importance')
plt.tight_layout()

plt.show()


# The most significant features will be selected, i. e. age, ejection fraction and serum creatinine.
smaller_x = x.iloc[:,[0,4,7]]
smaller_x
```

Appendix 3. Python Jupyter Notebook hyperparameter tuning and final models code

```
# Import the libraries used for data pre-processing, extraction, and visualization.
import pandas as pd
import numpy as np
from scipy.stats import randint

# Read the data and view it.
data = pd.read_csv('heart_failure_clinical_records_dataset.csv')
# Let's separate features and the target (DEATH_EVENT/if the patient died during the follow-up period; 0 if no, 1 if yes) to own arrays, the target array y:
y = data['DEATH_EVENT']
# And the feature array x:
x = data.iloc[:,:11]
# All but the 4 most relevant features will be dropped:
smaller_x = x.iloc[:,[0,4,7]]
# For RandomizedSearchCV fitting, the feature array will be scaled.
from sklearn.preprocessing import StandardScaler
scaled_smaller_x = StandardScaler().fit_transform(smaller_x)
# For optimized results
optimized_results = []

# Next, before building the machine learning models, numerical variables will be scaled in the Pipelines so that none of them has too large of an impact
# on the models. The required libraries are imported for this.
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics

# Splitting the trainind and testing data, as well as scaled training and testing data.
x_train, x_test, y_train, y_test = train_test_split(smaller_x, y, train_size=.8, random_state=0, stratify=y)
scaled_x_train, scaled_x_test, scaled_y_train, scaled_y_test=train_test_split(scaled_smaller_x, y, train_size=.8, random_state=0, stratify=y)

from sklearn.linear_model import LogisticRegression

# The optimizable parameters for Logistic Regression RandomizedSearchCV.
"""lr_grid = [
```

```
    {'penalty':['l1','l2','elasticnet','none'],
    'C': np.logspace(-4,4,20),
    'solver':                          ['lbfgs','newton-
cg','liblinear','sag','saga'],
    'max_iter': [100,1000,2500,5000]}]"""

# The RandomizedSearchCV will be run and optimal parameters
will be printed thereafter. Cross-validation will be done in
10 folds.
"""lr_search   =   RandomizedSearchCV(LogisticRegression(),
lr_grid,     verbose     =     1,     cv=10,     n_jobs=-
1,random_state=0,n_iter=100)
lr_search.fit(scaled_x_train,scaled_y_train)
optimal_lr_param = lr_search.best_estimator_
print(optimal_lr_param)"""

# The originally found optimal parameters for Logistic
Regression.
optimal_lr_param                                          =
LogisticRegression(C=np.float64(0.00026366508987303583),
max_iter=1000, solver='liblinear', random_state=0)
lr_pipeline   =   Pipeline([('Scaling',   StandardScaler()),
('Logistic                              Regression',
optimal_lr_param)]).fit(x_train,y_train)

# We'll print the confusion matrix and the accuracy
associated to the matrix.
print(metrics.confusion_matrix(y_true=y_test,
y_pred=lr_pipeline.predict(x_test),            labels=None,
sample_weight=None))
print("The accuracy of Logistic Regression is %4.2f" %
metrics.accuracy_score(y_true=y_test,
y_pred=lr_pipeline.predict(x_test)))
optimized_logreg   =   metrics.accuracy_score(y_true=y_test,
y_pred=lr_pipeline.predict(x_test))
optimized_results.append(optimized_logreg)

## The above is once again repeated for each model.

# The final table to present the accuracy of optimized models.
algorithms = ["Optimized Logistic Regression", "Optimized
KNN", "Optimized Decision Tree", "Optimized Random Forest",
"Optimized ETC", "Optimized GBC", "Optimized ANN", "Optimized
SVM"]
cv_mean = pd.DataFrame(optimized_results, index=algorithms)
cv_mean.columns=["Accuracy"]
cv_mean.sort_values(by='Accuracy', ascending=False)

# To estimate confidence intervals for the predictions.
import scipy.stats
confidence = 0.95
z_value = scipy.stats.norm.ppf((1 + confidence) / 2.0)
print(z_value)
```

```
# For confidence intervals.
confidence_intervals = []
# Let's calculate the confidence interval for Logistic
Regression.
logreg_ci_length = z_value * np.sqrt((optimized_logreg * (1
- optimized_logreg)) / y_test.shape[0])
logreg_ci_lower = optimized_logreg - logreg_ci_length
logrer_ci_upper = optimized_logreg + logreg_ci_length
confidence_intervals.append(['Logistic
Regression',logreg_ci_lower,logrer_ci_upper])

## The above is repeated for each model.

for i in range(len(confidence_intervals)):
    print(confidence_intervals[i][0])
    print("Lower                                    bound:",
float(confidence_intervals[i][1]*100),    "Upper    bound:",
float(confidence_intervals[i][2]*100))
```