Chair of Automation

Master's Thesis

Classification of Multivariate Time Series Data using Machine Learning and System Redundancy Analysis

Elliot Lang, BSc

January 2024

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 29.01.2024

Unterschrift Verfasser/in
Elliot Lang

Acknowledgement

I would like to acknowledge and thank eSENSIAL Data Science for providing the data, which forms the basis of this work.

Kurzfassung

Diese Arbeit untersucht das Maß an Redundanz innerhalb von Chargen von Sensorsignalen. Das explizite Ziel besteht darin, die Reaktion des Systems auf die Inaktivität einer oder mehrerer Sensor-Einheiten zu bewerten. Darüber hinaus erforscht diese Arbeit die Anwendbarkeit und Machbarkeit verschiedener maschineller Lernalgorithmen zur Klassifizierung des Zustands der analysierten Maschine auf der Grundlage von Chargen von Beschleunigungsdaten. Die Redundanz in den Daten wird durch die Quantifizierung der inhärenten Dimensionsabdeckung gemessen Basierend auf dem Ergebnis kann die Dimensionalität der Daten reduziert werden. Mögliche Permutationen des Verlusts einer oder zwei der sechs verfügbaren Sensor Einheiten werden im Zusammenhang mit der verbleibenden Dimensionsabdeckung analysiert. Dies gibt einen Hinweis darauf, mit welcher Sicherheit Analysen auf Grundlage der reduzierten Daten bewertet werden können.
Darüber hinaus werden mehrere mögliche überwachte maschinelle Lernalgorithmen identifiziert, die zur Mehrklassenklassifizierung in der Lage sind, und deren Anwendbarkeit auf die gelabelten Datenchargen bewertet wird. Eine Reihe vielversprechender Klassifikationsmethoden wurden identifiziert und auf jede Datencharge angewendet. Die erfolgreiche Klassifizierung wird durch die Messung der Vorhersagegenauigkeit jeder Methode, der Anzahl der korrekt identifizierten Maschinenzustände und der Trainingszeit jeder Methode einschließlich der Optimierung ihrer Hyperparameter quantifiziert.
Die Ergebnisse beider Prozesse zeigen eine Robustheit gegenüber dem Verlust von Sensoren innerhalb des Systems, unabhängig von deren räumlicher Lage, sowie einige vielversprechende maschinelle Lernklassifikationsalgorithmen, die in der Lage sind, den Zustand der Maschine zu identifizieren.

Abstract

This thesis investigates the level of redundancy within batches of sensor data; the explicit goal of which being to evaluate the system's reaction to the inactivity of one or more sensor units. Moreover, this thesis explores the applicability and viability of different machine learning algorithms for classifying the state of the analysed machine based on batches of acceleration data. The redundancy within the data is measured by quantification of the inherent dimensional coverage; based on the result, the data dimensionality can be reduced. Possible permutations of the loss of one or two of the six available sensor units are analysed in the context of the remaining dimensional coverage. This gives an indication of the certainty with which analyses based on the reduced data can be evaluated.

Furthermore, several possible supervised machine learning algorithms, capable of multi-class classification, are identified and their applicability to the labelled data batches is assessed. An array of promising classification methods were identified and applied to each batch of data. The successful classification is quantified by measuring each method's predictive accuracy, the number of correctly identified machine states, and the training time each method requires including the optimization of its hyperparameters.

The results of both of these processes show a robustness to the loss of sensors within the system, independent of spatial location, as well as some promising machine learning classification algorithms, capable of identifying the machine's state.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

This thesis addresses two issues related to machine condition monitoring using batch data from multiple sensors; an analysis of the level of redundancy within the data and the application of machine learning algorithms in order to identify the state of the machine.

The evaluation of the redundancy within the data is achieved by applying dimensionality reduction methods to the data set in the form of principal component analysis and investigating the correlation between available data channels and informational coverage within the data set. This is done to quantify the impact of the loss of data, due to damage or loss of sensor units, on the information contained within the data set and consequently on the certainty with which the classification process can take place. A comprehensive analysis of different permutations of inactive sensors will be conducted in order to gain an insight into the spatial redundancies present in the distribution of sensors. The investigation of this property within the data is of great significance, due to industrial environments in which the sensor units are mounted, which makes the possibility of sensor failure relevant.
An aspect, on which research on redundancy within sensor system is often focused is redundancies within systems of heterogeneous sensors for anomaly detection [1]. Another feature, often focused on is sensor redundancy analysis to ensure the maintenance of long lifetimes in sensor networks [2] or to improve a system's energy efficiency [3]. All these aspects view redundancy in sensor systems as an overlap of what is measured by different sensors. In contrast, the nature of the system redundancy analysed here deals with the loss of data, which is not replaced by any other sensor within the network and thus the spatial redundancy on the machine must be investigated.

The second aspect focussed on within this thesis is the classification of the machine state through the application of machine learning algorithms. The aim here is to explore suitable and applicable machine learning algorithms, which can be trained with labelled input data and are capable of multi-class classification. This can offer the basis on which a condition monitoring system for the machine can be based. A challenging element of this is presented by the lack of any previous insights into data produced by the machine, due to the lack of any condition monitoring solutions

being present, prior to the sensor system analysed here. Consequently, there has been no previous insight into the data analysed here.
When the application of machine learning techniques for condition monitoring is discussed, it is often in the context of ”Industry 4.0” and ”Internet of Things” (IoT) applications [4, 5]. The application investigated within this thesis stands in contrast to this, focussing on the applications of sensors for condition monitoring on a machine, which was not designed with condition monitoring or predictive maintenance in mind.

1.1 Goals

The aim of the work conducted in this thesis is two-fold. The first goal within the project is to analyse the level of data redundancy present in a multi-sensor machine condition monitoring system, in order to quantify the impact a partial loss of data would have on the information, that can be extracted from the sensor data. This will be achieved by applying dimensionality reduction techniques to the data and investigating the dimensional coverage present as a function of the available data channels. The second objective contained within this project is to exploring the application of different machine learning algorithms for the classification of the machine state based on batches of sensor data. The approach employed to accomplish this will entail the partitioning the data batches into training data and testing data. Following this, various machine learning algorithms suitable for multi-class classification problems will be trained using the training data and then tested on the portion of the data reserved for testing. The results will be compared based on the accuracy of prediction and the time taken to train each algorithm.

1.2 Structure

This work is structured as follows: Part 2 will give some background information on the principles of machine learning and provide some important definitions. Part 3 will the definition and the impact of data quality and the principles of data mining. In Part 4 the data exploration phase of the project will be presented, which begins with the data from its raw form and end with the data ready to be worked with, while in Part 5 each of the applied machine learning algorithms will be described in detail. Part 6 will show and compare the application process and the results for each algorithm and Part 7 summarizes and concludes this project.

Chapter 2
Machine Learning Background

The term Machine learning (ML), describes the computational capacity of machines to gain insights from historical datasets, subsequently enabling the formulation of predictive models. This process entails the derivation of mathematical models reflective of patterns and structures inherent in the training data. Central to the establishment of a robust predictive system is the discernible influence of data volume and quality, as these factors significantly impact the system's ability to generate accurate and reliable predictions. The ever-increasing importance of machine learning is underscored by its unique capability to undertake intricately convoluted tasks that can generally not be performed by humans, thereby increasing operational efficiency [6].

2.1 Applications of Machine Learning

Nowadays, machine learning is quite broadly known and widely used. Consequently, there are many examples for the application of machine learning algorithms. Some examples of areas in which machine learning is applied are:

1. Image and Speech Recognition
   The possibility of using machine learning, in this case mainly deep learning algorithms, has been vital for speech and image recognition tasks used in virtual assistants, automated image tagging or facial recognition systems.[7]

2. Natural Language Processing (NLP)
   NLP uses machine learning to interpret and understand language and for generative language systems. This is probably one of the most widely known uses for machine learning, as it is implemented in chat bots, language translation or for extracting information from text. [8]

3. Predictive Analytics
   Machine learning can also be very useful for making predictions. This can be applied, for example, in predictive maintenance in industry for optimizing maintenance schedules and to

minimize the risk and costs of machine failure. Further usage of predictive analytics are in health care for disease prediction, diagnostic imaging and optimizing treatment plans. [9]

2.2 Types of Machine Learning Tasks

Machine learning techniques can be utilised in a wide array of different tasks, the most common of which are listed here [6, 10]:

1. Clustering
   Clustering aims to find groups of similar data points within a set based on certain characteristics without pre-defined labels. This is done in order to find previously undetected structures and features within the dataset.

2. Classification
   Classification is the task of assigning data to pre-defined categories based on certain characteristics. This process works with a labelled input data set with the goal of finding a decision boundary, which separates different classes.

3. Anomaly Detection
   Anomaly detection specifically focuses on data points, which significantly deviate from the norm, indicating unusual events or behaviour of the systems.

4. Regression
   Regression represents a task, which involves the prediction of a continuous output variable based on one or more input features. The aim is to gain insights about the relationship between input and output.

2.3 Types of Learning

Another important aspect, which needs to be considered regarding machine learning is the variety of different learning types [6, 11].

1. Supervised Learning
   In supervised learning, the machine learning model is trained on a labelled data set, where the input is paired with the corresponding output. The information learned about the relationship between input and output is then mapped onto the training data set and the deviation from the known labels gives a measure for the accuracy of this method.

2. Unsupervised Learning
   In unsupervised learning, the model is trained with unlabelled data and therefore with no information about the system output. Here, the algorithm explores the inherent structure and patterns in the data without knowing the output.

3. Reinforcement Learning
   In reinforcement learning, the system receives ”rewards” or ”penalties” based on the actions it takes. The aim is for it to learn a strategy, which allocates states to actions in order to maximize the ”reward” over time.

4. Ensemble Learning
   Ensemble learning combines multiple machine learning models in order to improve the overall performance and system robustness.

2.4 Interpretability

In machine learning, there still exists a trade-off between a model's accuracy and its interpretability. Many basic models can be understood and interpreted well, whereas more complex models lack intelligibility. However this increase in difficulty of understanding more complex models is accompanied by increasing predictive accuracy [12]. For this reason, this thesis will be focussed on an experimental approach to understanding the types of machine learning algorithms, which are applicable and useful.

2.5 Applicability

The task, which is the focus of this thesis will be analysed through the lens of a machine learning classification task, using supervised learning. This is due to the nature of the task at hand, the aim of which is to sort each instance within the data into a class, denoting the state the machine was in for that time. Furthermore, the existence of labelled data, showing the state of the machine, makes a supervised learning approach useful and efficient.

Chapter 3
Data Quality and Data Mining

Analysing sensor data has become ubiquitous in most industries in order to reduce downtime by improving maintenance strategies and increasing overall efficiency. However, this can only be effectively applied, when the data used is of sufficient quality. Problems with the quality of data can lead to wrong conclusions being drawn, even if the applied analysis is correct.
A data set can be deemed to be of sufficient quality if it meets the specifications of the task it will be used for [13]. This definition by Olson (2003), includes the intended use of the data in the assessment of data quality.

3.1 Data Quality Dimensions

When attempting to quantify the quality of a data set, there are numerous aspects, which need to be considered. Depending on the task, these may vary in importance.

3.1.1 Accuracy

Ensuring the data set is of sufficient accuracy is vital in order to prevent incorrect analyses and consequently erroneous decision-making based on an inaccurate data set. Accuracy in this context refers to the correctness of the individual data points within the set. Some influencing factors for the accuracy of a data set include the reliability of the data source itself and the data entry process, as well as the precision of measurements. [14] Generally, accuracy of a data set can be defined as the extent to which the data set is able to reflect reality. [15]

3.1.2 Completeness

Completeness with regard to data quality, refers to the extent to which all required data points are actually present within the set. Incomplete data can introduce unintended biases or incorrect conclusions. [15]

3.1.3 Time-Related Dimensions

The way in which data set changes over time is a vital aspect, which needs to be considered when judging its quality. In [15], three types of time related dimensions are considered: currency, timeliness and volatility. Currency refers to how promptly data is updated, which can be very relevant in many applications, such as status monitoring. A data set's volatility describes the rate at which a data set varies over time. Lastly, timeliness refers to how current the information within the data set is in relation to the task at hand. This means, that the data may be updated and therefore current, and yet outdated for the task at hand if the frequency of updates does not match the timeliness requirements of a task [15].

3.1.4 Consistency

Consistency, in this context refers to how well data from different sources is combined and integrated. A data set of high quality displays a consistency of units, formats and data types [13, 15].

3.2 Data Mining

Data mining is defined as the automatic extraction of new information from large data sets. Its origin can be traced to the fields of statistics, probability theory and artificial intelligence [16]. When talking about the term data mining, a distinction must be made between scientific data mining for the purpose of scientific research, which exists in contrast to market-driven data mining and is distinct from the latter due to the nature of the data sets it is applied to [16].

3.2.1 The Standard Process of Data Mining

The standard process of data mining begins with the gathering of data in order to form the initial raw data set. This is followed by data cleansing, preprocessing and transforming a sub-set of the initial data set into a flat file. Once the relevant data has been moulded into a workable and clean form, one or more methods to extract information from the data set are applied. Examples of these are predictive methods, clustering methods or visualisations of the data. The last step in the process is the interpretation of the results of the methods applied in the previous step to actually extract knowledge from the data set [16].

3.2.2 The Data Mining Wisdom Pyramid

An alternative way of representing the process of data mining is the data mining wisdom pyramid, also known as the DIKW-Pyramid, as shown in figure 3.1. f

Fig. 3.1: This figure, inspired by [17], shows the hierarchical structure inherent in a data set, which is extracted by data mining methods. A raw data set starts out at the bottom level of the pyramid. Through progressively transforming, cleansing, applying predictive methods and visualising, the pyramid culminates in wisdom being extracted from the data.

The exact meaning of each layer is disputed among sources but there is generally some common ground within various definitions. The different levels of the pyramid describe the following states of the data set:

1. Data
   A data set is considered this level of the pyramid, when it is uncategorised, unprocessed and unorganized. The raw data set itself has no value or meaning, as it is missing any context and is just the direct product of observation. [17, 18, 19]

2. Information
   This level generally refers to data, which has been structured and organized in some way. The processing step applied to create this structure within the data, gives the data set context and thereby increases its value and usability [18].

3. Knowledge
   The concept of knowledge in this context is typically defined as adding understanding, experience and skills to to learn more about the data set [18]. Knowledge can thus be described as a property of the people or systems working with the data set, rather than a property inherent to the data set itself [20].

4. Wisdom
   Wisdom in this context, requires the judgement of the person working with the data set and refers to an ability to increase effectiveness [19].

3.3 Relevance for this Thesis

When analysing and processing the data sets within the scope of this thesis, the dimensions of data quality is used to judge the data's quality and usability. Moreover the processing and extracting of information from the data is guided the concept of data mining.

Chapter 4
Data Exploration

This chapter documents the data exploration phase of this thesis, from some initial analyses up to the point at which the data is ready for the application of machine learning algorithms.

This is an essential step in the early stages of any data science project in order to understand the quality of the data and to discover potential issues or challenges, which could affect the way the data can be used and to make sure the right conclusions can be drawn upon completion of the project.

Another aspect described in this chapter is the level of redundancy within the data, analysed by evaluating the dimensional coverage. This is done to investigate the correlation between the retention of functionality and reduction of certainty by losing sensor units.

4.1 Data Structure

The data used for this work was attained from 6 sensor units, mounted symmetrically along a waste processing machine. The data was measured in 8 batches recorded between the 6th of March 2023 and the 5th of April 2023, with each batch being stored in a JSON file. The files vary in length from the smallest file containing 209 samples, covering a time period of approximately 4.5 hours up to the largest file, containing 1439 samples, spanning 24 hours with each providing a timestamp for each sample. Crucially, for each timestamp, the state of the machine has been manually labelled, meaning the data is suitable for training and testing purposes. In addition to the manually added state variable, the file contains 144 channels, 24 per sensor. However, for this work, only the three acceleration measurements per sensor were analysed, leaving 18 data channels, three acceleration channels per sensor.

After the recording of the data batches, one of the six sensor units failed and was consequently lost. After this incident, two additional batches of data were recorded analogous to the 8 original data batches but with only 15 data channels. These will be evaluated separately in the later chapters of this thesis.

4.2 Industrial Context

The industrial context from which the data sets for this thesis originate is within a metal-waste treatment facility. Specifically, the sensor units are mounted on a metallic waste separation machine. Said machine works by receiving both metal scraps and carrying fluid on a conveyor belt. The function of the fluid is to suspend the smaller parts of the metal scrap, in order to facilitate the flow of material. The conveyor belt leads to a vibrating sieve, which is meant to separate the metal scraps by size.

4.2.1 The States of the System

The labels added to the data specify the state the machine was in at a specific time stamp. There is a distinction made between four different states the machine can be in.

State Number	State Description
0	In state 0, the machine is switched off.
1	State 1 describes the machine running idly with no waste or fluid on the machine.
2	In state 2, there is only fluid but no waste running across the machine.
3	State 3 describes the machine working fully with both waste and liquid.

Table 4.1: This table displays all considered machine states, which form the basis for the machine learning classification task discussed in this work.

These four different states form the basic class structure for the classification task analysed in this thesis.

4.3 Data Ingestion

The data analysed in this work originates from six sensor units mounted on different parts of the machine. In the ingestion process, the data is mapped to a set of objects, which are collected in a timetable with attached metadata. This implicitly introduces time into the system. The sensor units are positioned symmetrically at the inlet, middle and outlet of the machine, each on the left and right side of the machine. They are abbreviated in the following way throughout this document.

Sensor Position	Abbreviation
Inlet Left	IL
Middle Left	ML
Outlet Left	OL
Inlet Right	IR
Middle Right	MR
Outlet Right	OR

Table 4.2: This table lists the names of the sensors and the corresponding abbreviations. Each of the listed sensors contains three accelerometers, one for each axis, meaning e.g. OL_x refers to the sensor on the left of the outlet and the accelerometer in direction x.

Each contains three accelerometers; these are the measurements on which this project is focused. The channels are henceforth denoted with the sensor abbreviations, as listed in 4.2 and the axis, e.g. IR_y referring to the sensor on the right side of the machine's inlet and the accelerometer measuring the y axis. The data analysed is batch data, with the addition of manually created labels, describing the state the machine was in at each time stamp.
Throughout this chapter, an exemplary data set, recorded on the 5th of April 2023, is used to visualise the progress of the data exploration and preparation phase.

4.4 Issues with the Data

4.4.1 NaNs in the Data

One issue, which requires immediate attention before proceeding is the prevalence of "Not a number" (NaN) entries in the dataset. These are especially present at the beginning and end of the data, when the machine is turned off, and sometimes such entries occur throughout the dataset. Two different approaches were selected in order to remove the NaN entries in the data set. For the beginning and end of the data set, the NaNs are replaced with zeros, due to the machine being turned off at that time. These artificially created zeros are necessary for the machine learning algorithms but are not taken into account when performing calculations, such as calculating the mean of the data, as they would cause potential problems. The NaNs contained within the measurements are linearly interpolated based on the values preceding and following the NaN value(s).

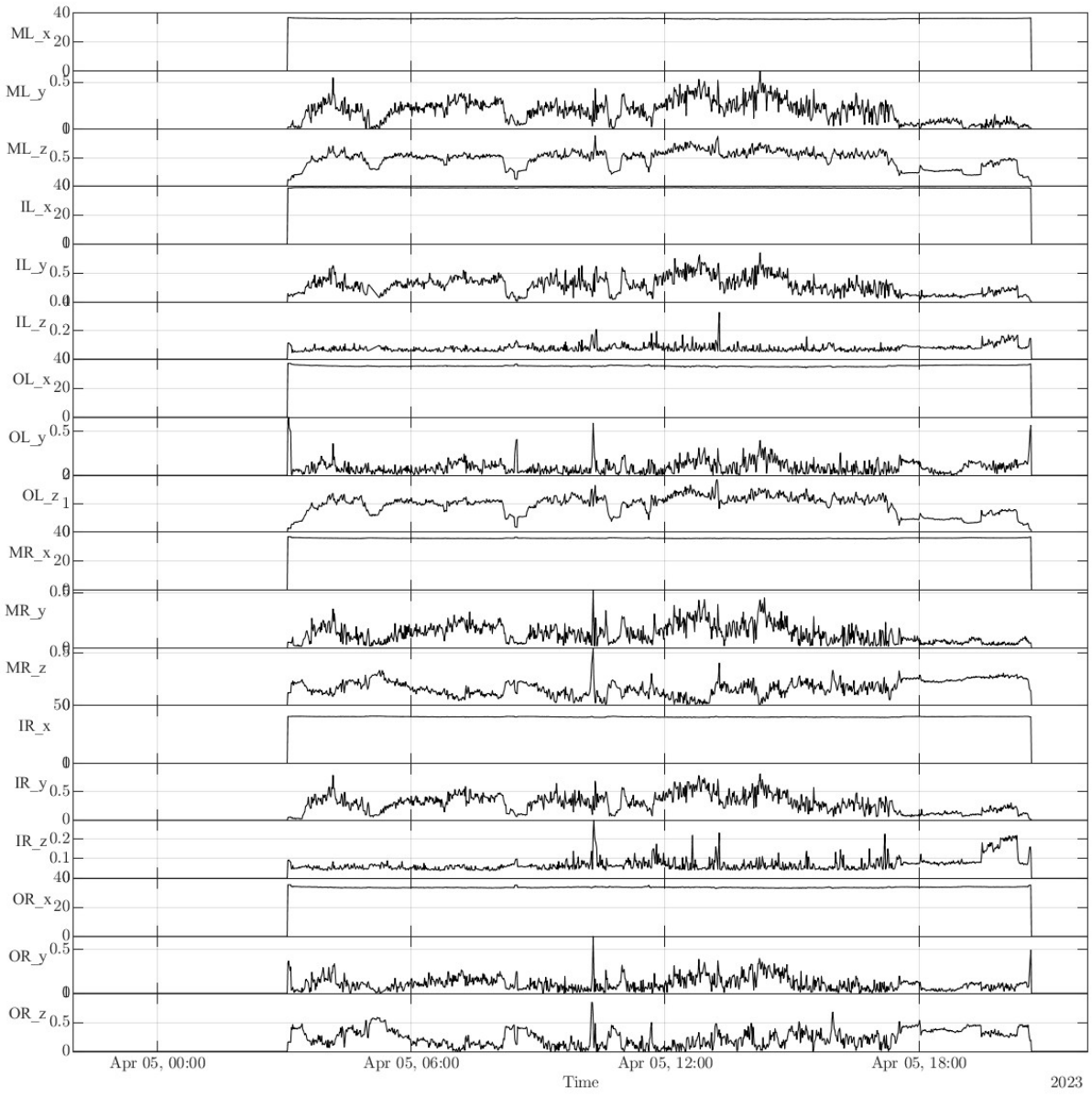


Fig. 4.1: This figure shows the raw acceleration data from each sensor unit. Three acceleration dimensions are attained from each unit, resulting in 18 channels. The data is unprocessed at this point, apart from the interpolation of all NaN values within the dataset and the replacement of NaNs at the beginning and end with zeros.

4.4.2 Time Discrepancies

Another issue, which must be kept in mind throughout this project, is the potential for small time shifts between the machine data and the manually created labels. These are primarily due to the very nature of manual labelling, which can incur latency in comparison to the machine switching states. This does not require immediate correction, however it must be kept in mind for contextu-

alising the results and can offer an explanation for minor differences in predicted data and labelled data.

4.5 Dimensionality Reduction

The next step in preparing the data for processing, is to check the dimensional coverage of the sensors. This is done, in order to discern, whether portions of the data add little to no extra information and don't need to be considered further. Reducing the volume of data required will increase the efficiency of working with the data set. If there are redundancies in the data, one can consider whether or not sensors can be removed without significantly impacting the certainty with which this data can be used to identify the state of the machine. Furthermore, this will show whether the application of dimensionality reduction measures is possible without a significant loss of information.

4.5.1 Singular Value Decomposition (SVD)

To achieve a reduction in data dimensionality, principal component analysis (PCA) is applied. This is done to identify the main direction of vibration and to adjust the frame of reference accordingly, in order to identify the main vibration axis.
In order to be able to implement PCA in a later step, as well as check the inherent dimensional coverage prior to the application of reduction measures, singular value decomposition (SVD) is performed. This yields a coefficient matrix U, a singular value matrix S and a rotational matrix V. [21]

The data can be represented as an $m \times n$ matrix $\mathsf{D} = [\boldsymbol{d_1}, ..., \boldsymbol{d_n}]$ with $\boldsymbol{d_k}$ being column vectors, which contain the individual data points for each time stamp. In order to normalise the impact of each data channel, the data must first be made mean-free.

$$\tilde{\boldsymbol{d_k}} = \boldsymbol{d_k} - \frac{1}{n}\sum_i \boldsymbol{d_i} \tag{4.1}$$

Using SVD, D can be represented as:

$$\mathsf{D} = \mathsf{USV}^{\mathrm{T}} \tag{4.2}$$

with

$$\mathsf{U}^{\mathrm{T}}\mathsf{U} = \mathsf{I} \tag{4.3}$$

and

$$\mathsf{V}^{\mathrm{T}}\mathsf{V} = \mathsf{I} \tag{4.4}$$

The matrix S is an $n \times n$ diagonal matrix consisting of n scalar values σ_k, where $\sigma_k \geq \sigma_{k+1}$. The values in S are collected in vector form in:

$$\boldsymbol{s} = \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_n \end{bmatrix} \tag{4.5}$$

Having defined these parameters, the dimensional coverage of the raw data, before the application of any processing, can be calculated as follows:

$$c_i = \frac{\sum_{j=1}^{i} \sigma_j}{\sum_{j=1}^{n} \sigma_j} \tag{4.6}$$

The value of c_i represents the percentage of the data, which is covered cumulatively by all dimensions up to i. This can be visualised in the following figure.

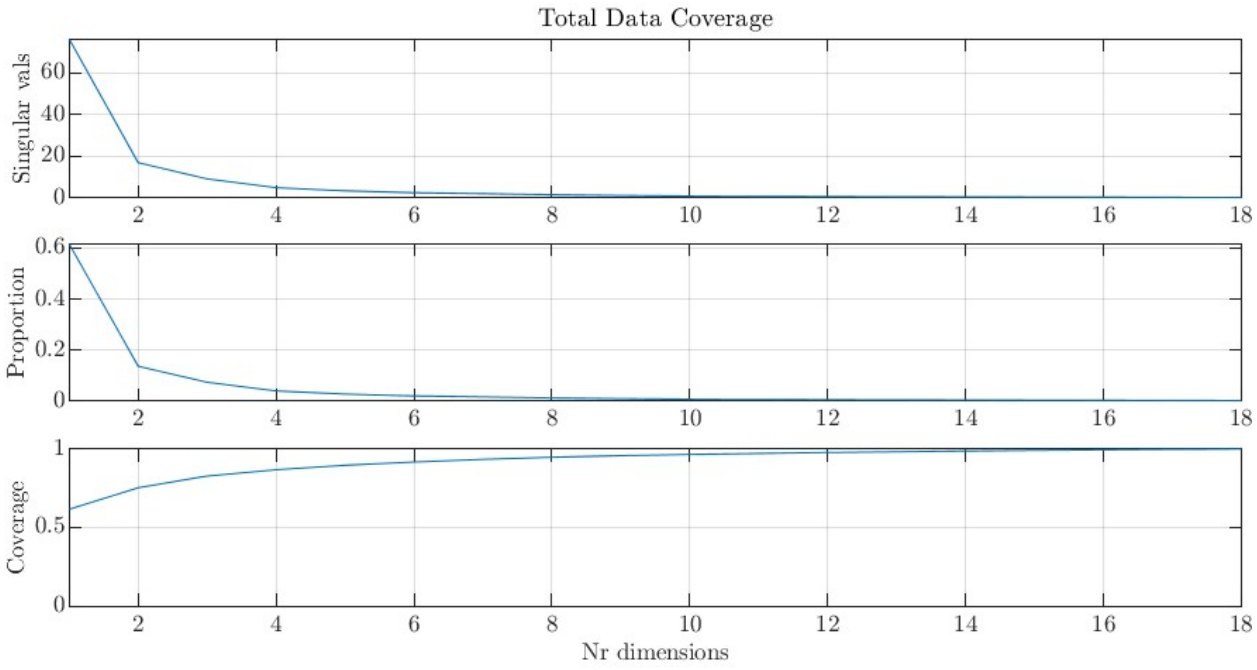


Fig. 4.2: This plot displays the dimensional coverage of the data set as a function of the number of dimensions, as described in the previous equation. If all 18 dimensions of acceleration data are considered, 100% coverage of the information within the data set is assumed.

4.5.2 Principal Component Analysis (PCA)

Next, PCA is be applied, in order to improve the dimensional coverage shown in 4.2. PCA is a commonly used tool in data science, which is used to identify the ideal base for the data, in order to gain new insights, reveal previously unseen structures in the data and to filter out noise. [22] In this case, PCA is applied to each sensor individually and the main principal component is extracted from each. The primary principal component represents the direction in which the most significant acceleration takes place. This step will yield six principal components, one from each sensor. The principal components can be computed as follows:

$$P_i = \mathsf{US} = \mathsf{DV} \tag{4.7}$$

where the columns of P represent the most significant principal components of the sensor and U and S are matrices attained from applying singular value decomposition to each sensor.

The Visualisation of the principal components suggests a correlation between data channels. This correlation gives an indication that there might be redundancies in the data and the dimensionality could be reduced without significant losses in accuracy and only minor sacrifices of certainty.

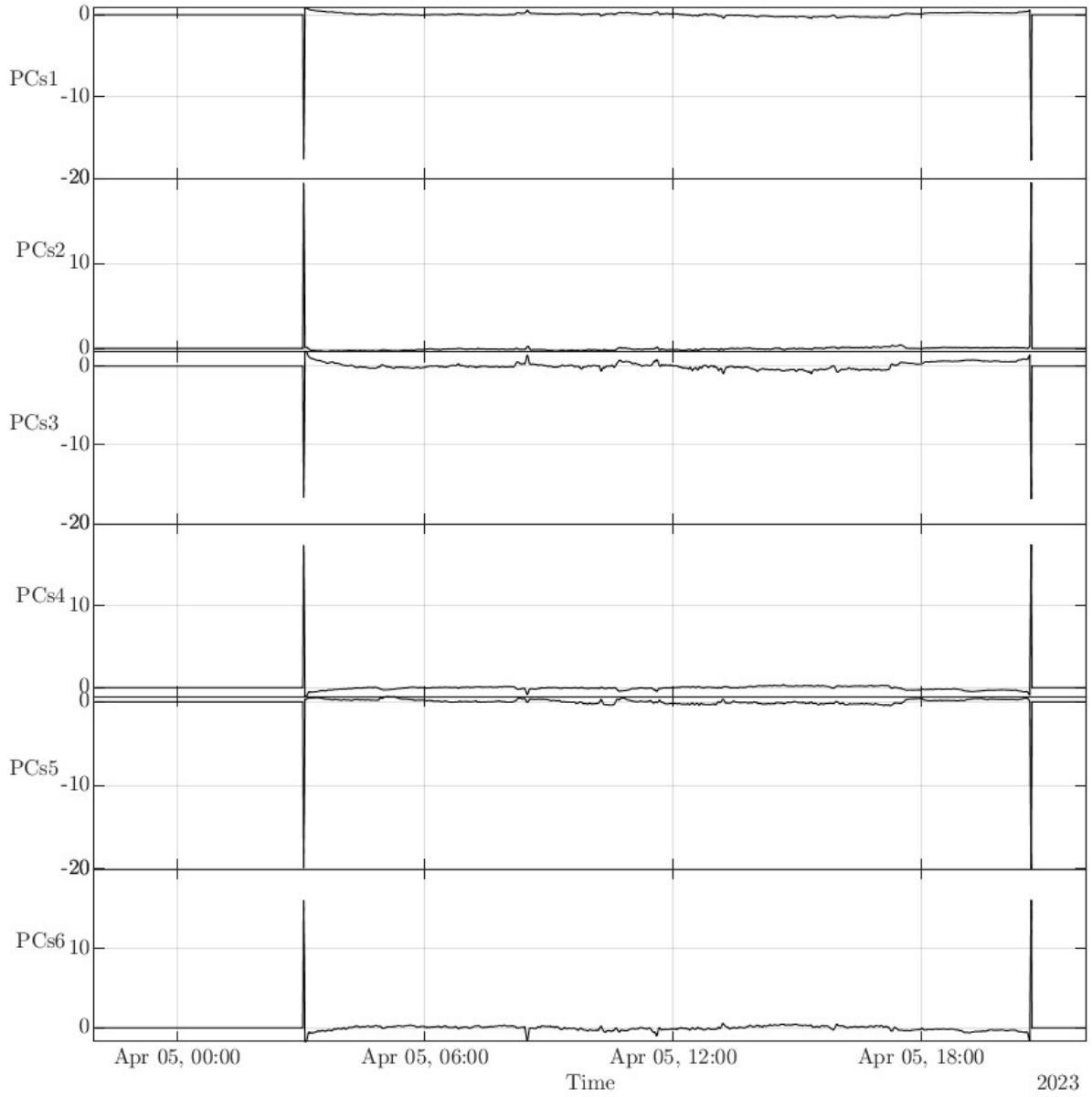


Fig. 4.3: This figure displays the primary principal component from each of the six sensor units over the time contained within an exemplary data file. The data here has been made mean-free and shows some clear correlations between the principal components.

4.5.3 Checking the dimensional coverage after PCA

The same process as in 4.6 can be applied to the principal components, in order to determine the coverage of the data set after the applied transformations.
The result of this can be seen in the following figure.

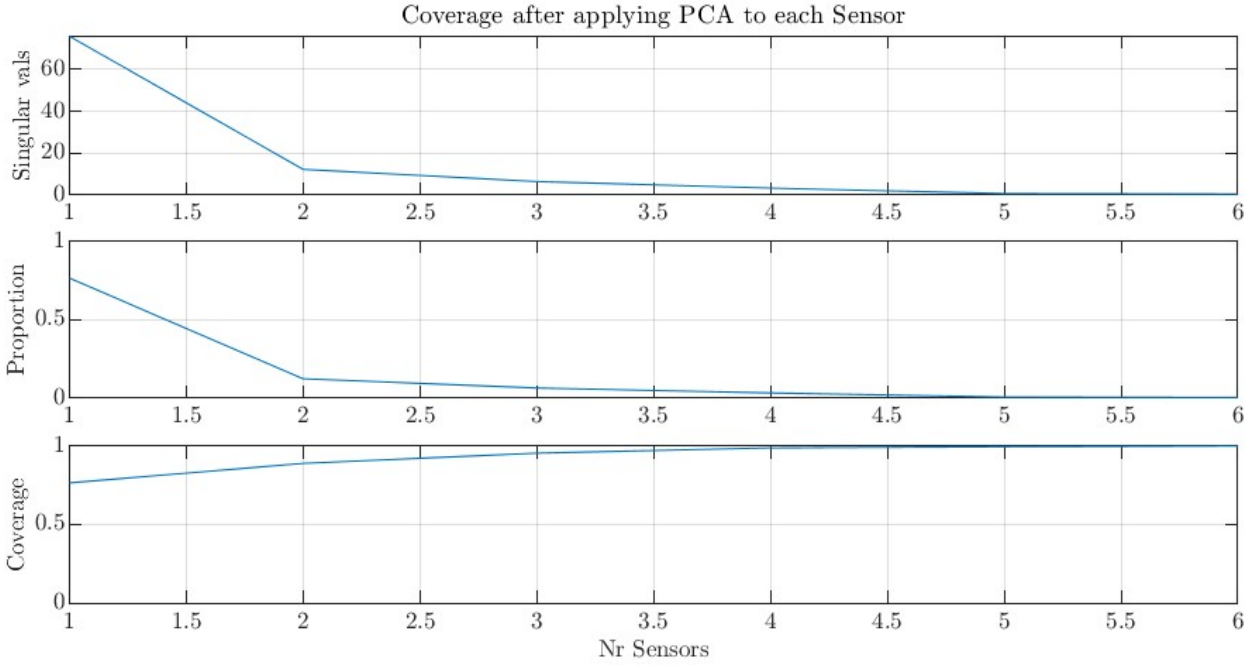


Fig. 4.4: This visualisation shows the data coverage after the application of principal component analysis plotted against the number of active sensors. This clearly shows an improvement due to the applied transformations and indicates, that the loss of one or two sensors still results in significant dimensional coverage.

The results shown in 4.4 indicate that there are redundancies in the data. The graph shows, that when only four out of the six sensors are present, a coverage of 95% can be achieved and that even with only one sensor present, 57% is already covered. The reduction in coverage caused by the loss each further sensor is can be seen in the following table:

Nr. Active Sensors	Coverage [%]
1	76.649
2	88.89
3	95.41
4	98.77
5	99.50
6	100.00

Table 4.3: This table contains the percentage of information in the data, covered when activating each sensor, one after the other, starting with a single sensor in the first line of the table until all six are active and 100 % coverage is reached.

4.6 Dimensional Coverage when Losing Sensors

In this section, we will analyse the effect of choosing any number k out of the total of n sensors and calculate the resulting dimensional coverage. This will be done for each possible combination of sensors. The number of possible combinations n_c is calculated by the following binomial:

$$c = \binom{n}{k} \tag{4.8}$$

For each possible combination, the same methods as described in 4.5 and 4.6 are applied. For quantifying the coverage in comparison to using all six sensors, a measure is created. This measure uses QR decomposition to calculate a matrix R. In QR decomposition a matrix A is decomposed into an orthogonal matrix Q and an upper-triangular matrix R. The relationship between these three matrices is $\mathsf{A} = \mathsf{QR}$. [23]
QR decomposition is applied to both the overall matrix containing all six primary components of the system will all six sensors being operational and to each combination of working sensors.

The measure r_c is then defined as the ratio between the norm of the upper-triangular matrix of the active sensors with the deactivated sensor(s) $\mathsf{R_k}$ and the upper-triangular matrix of all sensors $\mathsf{R_t}$.

$$r_c = \frac{||\mathsf{R_k}||_2}{||\mathsf{R_t}||_2} \tag{4.9}$$

4.6.1 Choosing 5 out of 6 Sensors

The first case analysed here is when only one of the six sensors is removed, leaving any combination of the remaining five sensors. The results for each of the six cases are collected in the following table:

Combination of Sensors	r_c [%]
1,2,3,4,5	91.3969
1,2,3,4,6	91.3308
1,2,3,5,6	90.9947
1,2,4,5,6	91.6765
1,3,4,5,6	91.6956
2,3,4,5,6	91.0005

Table 4.4: This table contains the coverage for each instance, when any one of the six sensors is inactive in relation to the total available information. The coverage for any permutation of the loss of a single sensor remains very consistent, indicating that it is not of great importance which of the six sensors is lost.

The results of the analysis of the dimensional coverage when one of six sensors is lost in table 4.4 shows significant consistency across each permutation of sensor loss. This indicates that the impact on the dimensional coverage achievable when one sensor is deactivated is not dependant on which of the six sensors is lost.

4.6.2 Choosing 4 out of 6 Sensors

The second case analysed in this section involves any two of the six sensors being inactive, leaving any combination of the remaining four sensors. All 15 such cases are represented in the following table:

Combination of Sensors	r_c [%]
1,2,3,4	81.696
1,2,3,5	81.480
1,2,3,6	81.396
1,2,4,5	82.362
1,3,4,6	82.069
1,2,5,6	81.797
1,3,4,5	82.144
1,3,4,6	82.304
1,3,5,6	81.796
1,4,5,6	82.298
2,3,4,5	81.454
2,3,4,6	81.395
2,3,5,6	80.992
2,4,5,6	81.789
3,4,5,6	81.854

Table 4.5: This table contains the coverage for each instance, when any two of the six sensors are inactive in relation to the total available information. The consistency among the resulting coverage across different combinations of lost sensors indicates, that the coverage does not depend on which pair of sensors is deactivated.

The results listed in table 4.5 indicate, that which pair of sensors is lost does not have a significant impact on the remaining dimensional coverage, which can be expected.

4.7 Redundancies

The examination carried out in this chapter demonstrates, that even in scenarios where one or two sensors are omitted, there remains a considerable likelihood of obtaining accurate results from

the dataset. Building on this observation, the next phase of the investigation will extend into the realm of applying machine learning algorithms. The goal is to comprehensively understand the repercussions, or lack thereof, stemming from the absence of certain sensors in regard to the classification of the machine state. By incorporating machine learning methods, the study aims to discern the robustness and adaptability of the chosen algorithms in accommodating missing sensor data, thus enhancing the overall understanding of the system's resilience to sensor failure scenarios. This strategic progression in the research design will contribute valuable insights to the broader discourse on the reliability and effectiveness of machine learning approaches in real-world applications with potential sensor irregularities.

Chapter 5
Exploration of different Machine Learning Algorithms

This chapter will explore a selection of different machine learning algorithms, all of which have the potential to be useful in the classification problem investigated in this thesis. For each method, the fundamental working mechanisms, different variations and hyperparameters, as well as the suitability for application will be discussed.

5.1 Binary Decision Trees

Decision trees represent a simple, yet powerful supervised classification approach. Decision Trees offer a clear graphical representation made up of a root, nodes, branches and leaves. Starting from the root node, the tree is generally drawn from the top down and from right to left and ending in leaf nodes at the bottom of the tree. Each node represents a certain characteristic within the tree and the nodes are connected by branches, representing a range of values, acting as partition points for the data set. In a binary decision tree, each node splits into exactly two branches. [24, 25]

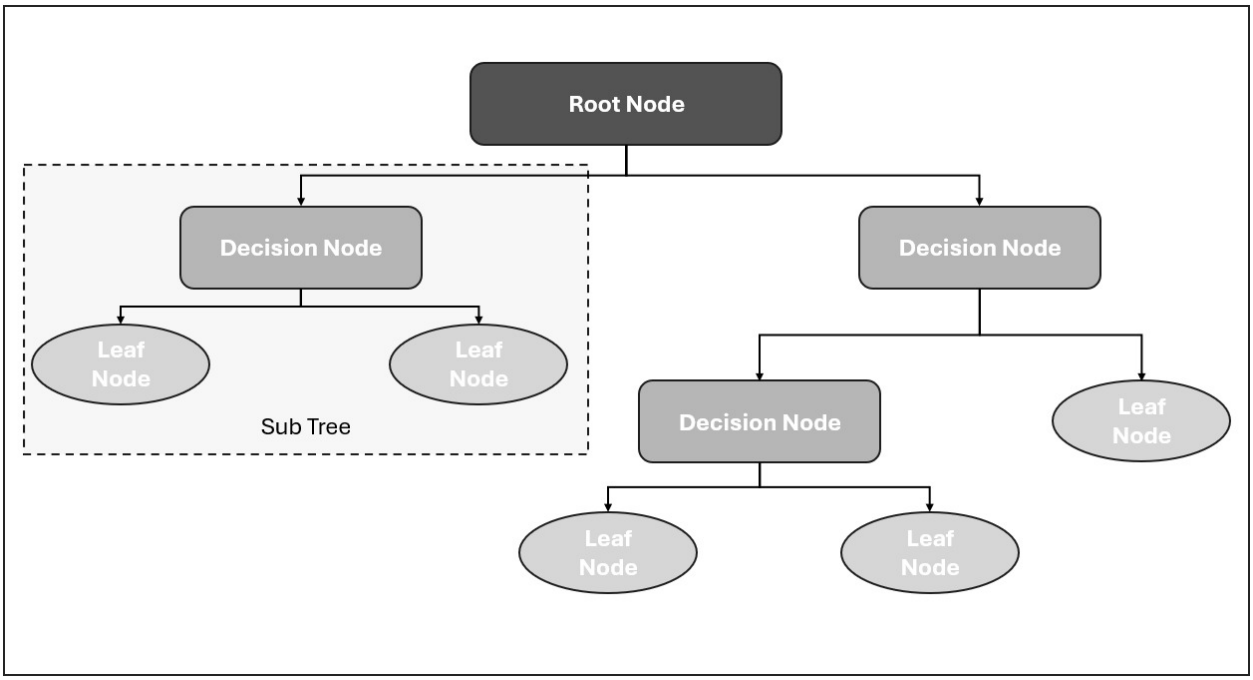


Fig. 5.1: This figure shows an example structure of a binary decision tree, starting from the root down to the leaf nodes, inspired by [25].

5.1.1 Types of Decision Trees

Decision tree algorithms are primarily used for regression or classification problems. Some of the main types of decision tree algorithms are listed here:

1. ID3 (Iterative Dichotomiser 3):
   ID3 was one of the first formulations of a decision tree algorithm, developed by Ross Quinlan in [26]. It literately creates the tree from the top down by recursively choosing the optimal attribute for splitting the data based on the gaining of information or entropy. [26].

2. C4.5 (Classification and Regression Trees) :
   C4.5 represents an improvement of the ID3 algorithm, also developed by Ross Quinlan in [27]. It again uses information gain for attribute selection but handles both discrete and continuous attributes. Moreover, C4.5 is characterised by its ability to handle missing data points well. [27]

3. CART (Classification and Regression Trees):
   CART is a decision tree algorithm, which can be both used for classification and regression. It was developed by Leo Breiman in [28] and is implemented in the binary decision tree classifier in MATLAB . CART constructs binary trees by recursively splitting the data set

into two subsets, either based on the Gini Impurity for classification tasks or mean squared error reduction for regression tasks. [28].

4. CHAID (Chi-squared Automatic Interaction Detection):
   CHAID is mainly employed for categorical data. The algorithm recursively splits the data set based on statistical testing, using tests such as the chi-squared test for independence. The main aim of CHAID is to find relevant relationships between different variables. [29]

5.1.2 Impurity Measures

Decision tree algorithms require impurity measures for their decision-making process, in order to evaluate how well particular attributes or features separate the data into classes. The impurity measure guides the process of constructing an optimised decision tree. Decision trees have the aim of maximizing the homogeneity of the resulting subsets, where impurity measures play an important role in quantifying this subset purity. [28, 25]. Moreover, impurity measures play an important role in selecting the optimal features for splitting the tree by evaluating the potential splits in terms of how they would reduce the overall impurity. The reduction in impurity is often expressed as information gain, which in turn is used to quantify the improvements achieved by a specific split. Therefore, potential splits can be ranked by the information that can be gained. Lastly, impurity measures are crucial for the tree pruning process. For classification tasks in decision trees, there are three main impurity measures: Misclassification Error, Gini Index and Cross-entropy (or Deviance) [22].

5.1.2.1 Mathematical Definition

Let m describe a node within a decision tree, which represents the region R_m, which contains N_m observations. Each observation N_i exists an input variable x_i and a corresponding response variable y_i, which for classification tasks represents the assigned class. The proportion of observations from class k within node m can be defined as:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \#\{y_i = k\} \tag{5.1}$$

The observations in node m are classified to class $k(m) = \text{argmax}_k \hat{p}_{mk}$, which describes the majority class in node m [30].

The measures for node impurity $Q(T)$ can consequently be defined as follows [30]:

1. Misclassification error:

$$\frac{1}{N_m} \sum_{x_i \in R_m} \#\{y_i \neq k(m)\} = 1 - \hat{p}_{mk}(m) \tag{5.2}$$

2. Gini Index:

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{5.3}$$

3. Cross-entropy or Deviance:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \tag{5.4}$$

How these impurity measures compare can be visualised in the following figure 5.2:

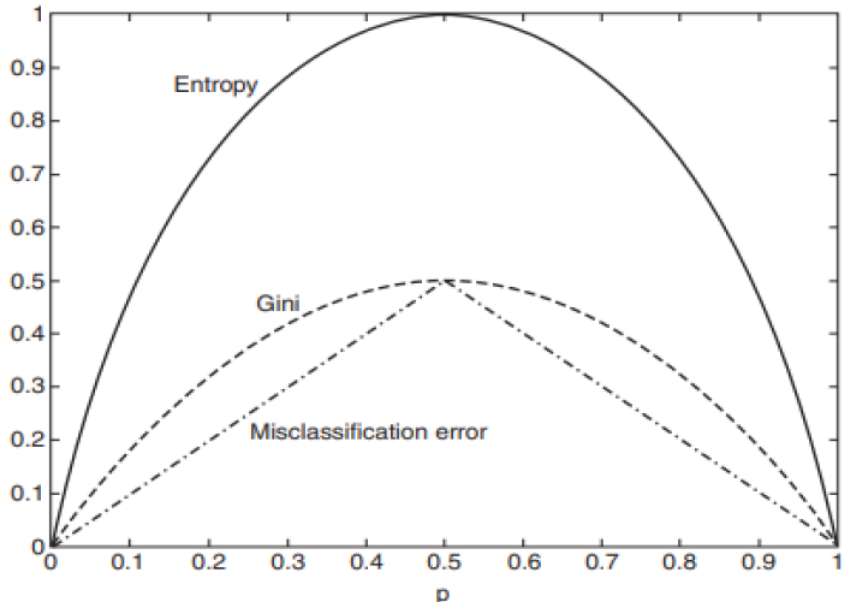


Fig. 5.2: This figure compares the three most significant impurity measures for decision trees, used in classification tasks. The figure is taken from [25]. The y-axis shows the impurity, which lies between 0 and 1. The closer this value gets to 0, the better. The x-axis displays the proportion p of a specific class in a specific node. [25]

5.1.3 Tree Pruning

An essential step in the construction of decision trees with the goal of refining the tree structure is tree pruning. This step is essential to prevent overfitting of trees. When overfitting occurs, the tree structure becomes so complex, that noise in the training data is picked up instead of general patterns in the data, which is detrimental to the trees predictive capabilities. [28, 30] The process of tree pruning can be described in the following steps:

1. Full Tree Construction
   Initially, the entire tree is constructed up to the defined stopping criterion. The stopping criterion is usually set as a maximum tree depth or making sure each leaf node includes a minimum number of instances.

2. Node Evaluation
   When the whole tree has been constructed, each leaf node can be analysed, gauging the impact each node has on the predictive prowess of the tree. This is usually quantified by applying validation techniques.

3. Tree Pruning
   The pruning process identifies the parts of the tree, which don't significantly contribute to its predictive capabilities to prevent overfitting.

4. Subtree Removal
   Once the non- or low-contribution sections of the tree have been identified, they are systematically removed.

5. Stopping Criterion
   Lastly, the pruning process terminates when a stopping criterion is reached. This can involve achieving a desired predictive accuracy or optimal tree complexity.

5.1.3.1 Cost-Complexity Pruning

One of the most commonly employed methods for tree pruning is cost-complexity pruning. This can be defined as [30]:

$$N_m = \#\{x_i \in R_m\} \tag{5.5}$$

with N_m referring to the number of observations N, at node m in the region R_m. Each single observation N_i is described by $(x_i|, y_i)$, for $i = 1, 2, ..., N$.

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i \tag{5.6}$$

$$\tag{5.7}$$

Here, index m denotes a specific node, N_m represents the number of observations at node m in region R_m. Moreover, x_i and y_i describe a specific observation N within the data set. The calculated parameter Q_m represents the impurity measure for the analysed node, which is discussed in equations 5.2, 5.3 and 5.4. [30] The complexity criterion for a tree T, with the number of terminal nodes $|T|$, can thus be defined as:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T| \tag{5.8}$$

The parameter α represents a tuning parameter, which can be adjusted to find the balance between the size of the tree and how well it fits the data. In cost-complexity pruning, the aim is, for each value of α, finding a subtree $T_\alpha \subset T_0$ to minimize the value of $C_\alpha(T)$. [30]

5.1.4 The CART Decision Tree Algorithm

The specific algorithm employed within this thesis, and implemented in MATLAB is the Classification and Regression Trees (CART) algorithm, which is capable of both regression and classification tasks. The tree construction process works by recursively splitting the data set into subsets, based on feature selection optimisation. To quantify the homogeneity of subsets in classification tasks, the algorithm uses the Gini Impurity index, previously defined in 5.3.
The CART algorithm was applied, not only due to its wide availability in various machine learning libraries, but also due to its consistent performance [31], its systematic pruning strategy, as discussed in the previous paragraph, and its robustness to outliers and noise in the data [30].

5.2 Linear Discriminant Analysis (LDA)

Discriminant Analysis describes a statistical technique employed for classification tasks and dimensionality reduction, first introduced by R.A. Fisher in [32]. In this case, linear discriminant analysis (LDA) is used as opposed to e.g. quadratic or pseudo-quadratic discriminant analysis, due to it yielding the best results in trial applications for this data set.
Generally, the aim of LDA is to find the optimal linear combination of features for separating the data set into a given number of classes. This is achieved by dimensionality reduction, through the

projection of the data onto a lower-dimensional subspace while preserving the separability of the classes. [33]

LDA works under the assumption, that the data's features are normally distributed within each class. If this holds true in the data set, this algorithm will work well. [34] Moreover, LDA handles multi class problems very efficiently and provides insights into the importance of different features for the separation of classes. [30]

Some drawbacks, which can come with the application of LDA include a significant sensitivity to outliers in the data and the inherent assumption that the covariance matrices of different classes are equal. [33, 35] In addition, LDA inherently assumes linear decision boundaries. If this is not reflected in the reality of the data, LDA may deliver suboptimal results and either quadratic discriminant analysis or other non-linear models, such as support vector machines, may be better suited. [30]

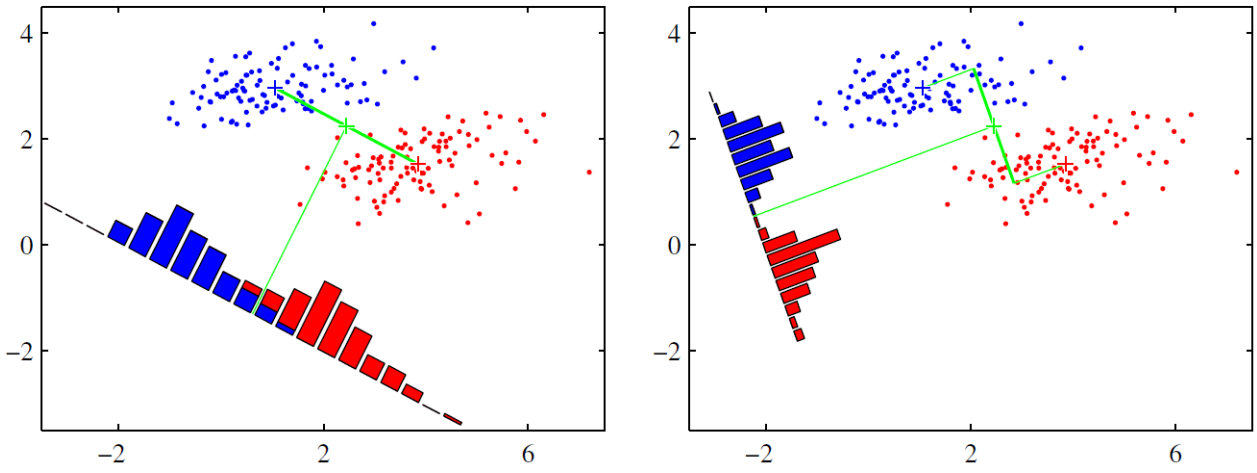


Fig. 5.3: This graph, taken from [11], illustrates the difference between the projection onto the connecting line between the class means (left) and projecting onto a line obtained from Fisher linear discriminant analysis. This comparison clearly displays some class overlap in the projected space in the left graph contrasted with greatly improved class separation due to the application of LDA on the right. [11]

5.2.1 Mathematical Definition

Let there be a data set with N samples and D different features and the aim is to classify the data set into K classes. The first step would be to create an in-class scatter matrix S_w. This is done by calculating an individual scatter matrix for each class k, which is defined as: [11, 30]

$$\mathsf{S}_\mathsf{k} = \sum_{i \in C_k} (\boldsymbol{x_i} - \boldsymbol{m_k})(\boldsymbol{x_i} - \boldsymbol{m_k})^T \tag{5.9}$$

with C_k representing the number of samples within class k, $\boldsymbol{x_i}$ being a single sample and $\boldsymbol{m_k}$ the mean vector of class k. The total in-class scatter matrix is then computed as follows:

$$\mathsf{S_w} = \sum_{k=1}^{K} \mathsf{S_k} \tag{5.10}$$

The next required parameter is the scatter matrix between classes S_b. For this calculation, one must first compute the mean vector for the entire data set $\bar{\boldsymbol{x}}$ and the mean vector for each class $\boldsymbol{m_k}$. The calculation is defined as follows:

$$\mathsf{S_b} = \sum_{k=1}^{K} N_k(\boldsymbol{m_k} - \bar{\boldsymbol{x}})(\boldsymbol{m_k} - \bar{\boldsymbol{x}})^T \tag{5.11}$$

with N_k being the number of samples in class k.

The matrices $\mathsf{S_b}$ and $\mathsf{S_w}$ can be used to formulate the following eigenvalue problem:

$$\mathsf{S_w}^{-1}\mathsf{S_b}\mathsf{V} = \boldsymbol{\lambda}\mathsf{V} \tag{5.12}$$

This needs to be solved for the matrix V of eigenvectors and $\boldsymbol{\lambda}$ of eigenvalues. Finally, the eigenvectors corresponding to the top $C-1$ eigenvalues must be chosen to form the transformation matrix W.

The linear discriminant function for classifying a new sample x into one of the K defined classes is consequently given as:

$$y(x) = \mathsf{W}^{\mathrm{T}}x \tag{5.13}$$

5.2.2 Suitability of Discriminant Analysis for Classification Tasks

Some key reasons, why linear discriminant analysis is relevant for the application discussed in this thesis, is its probabilistic framework for the approximation of classes, robustness to imbalances between different classes and its inherent feature selection. [36, 11]

5.3 k-Nearest Neighbours (kNN)

Another algorithm, which was considered for the classification task at hand is the k-nearest neighbours algorithm, first introduced by T. Cover and P. Hart in [37]. The algorithm works by setting a point x, which will represent the centre of a sphere. The radius of this sphere will be continually increased until k number of data points are contained within it. [11]

KNN offers the advantages of being quite simple and straightforward to implement and to un-

derstand and offers very good adaptability to decision boundaries that are non-linear in nature. However, its effectiveness wanes as the size of the data set increases, due to the computational cost incurred. Furthermore, kNN is quite sensitive to outliers and struggles with noise within the data, due to every data point being considered equally. Lastly, the choice of the hyperparameter k is essential. If the chosen value of k is too small, overfitting can occur and the algorithm might capture noise. If k is set too large, this can cause an oversmoothing the decision boundaries, which can lead to missing certain patterns. [11] [37]

5.3.1 Mathematical Definition

The first step for explaining the mathematical background of the kNN algorithm is to define a feature space X which is made up of individual samples $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,d})$ with $i = 1, 2, ..., N$ and D representing the training data set consisting of N samples and d features.
A key aspect of the KNN algorithm is the choice of distance measure, which can have an impact on the efficiency and applicability. Some significant distance measures will be discussed in the following list:

1. Euclidean Distance

$$d(X_i, X_j) = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2}$$

   The euclidean distance is the simplest distance metric discussed, as it describes a straight line between two points in the euclidean space. [11]

2. Hamming Distance [38]

$$d(X_i, X_j) = \frac{1}{d} \sum_{k=1}^{d} \delta(x_{ik}, x_{jk})$$

   With δ being the Kronecker delta function, which returns 0 if $x_{ik} = x_{jk}$ or otherwise 1. [38]
   In its traditional form, hamming distance is designed to work with binary data. A modified version, referred to as mismatch distance, can be used for applications with non-binary data. The mismatch distance counts the non-matching elements between two vectors. [11]

3. Mahalanobis Distance

$$d(X_i, X_j) = \sqrt{(X_i - X_j)^T \Sigma^{-1} (X_i - X_j)},$$

where Σ is the covariance matrix for the points X_i.
This distance metric takes the correlation between variables into account, making it very useful in applications with non-independent dimensions in the data. [39]

4. City Block (Manhattan) Distance

$$d(X_i, X_j) = \sum_{k=1}^{d} |x_{ik} - x_{jk}|$$

City Block or Manhattan distance is defined as the absolute sum of differences for each feature. It measures the distance akin to travelling in a city grid, only moving horizontally and vertically. [40]

5. Spearman Rank Correlation Distance

$$d(X_i, X_j) = \sum_{k=1}^{d} (\text{rank}_i(x_{ik}) - \text{rank}_j(x_{jk}))^2$$

The Spearman rank correlation distance is based on Spearman rank correlation coefficient and represents a metric for the differences in rank between the variables. It is very robust to outliers and can be useful for different distributions present in the data. [41]

6. Minkowski Distance

$$d(X_i, X_j) = \left(\sum_{k=1}^{d} |x_{ik} - x_{jk}|^p \right)^{\frac{1}{p}}$$

The Minkowski distance represents a generalization of the euclidean and Manhattan distances by including both cases. Euclidean distance can be measured by setting $p = 2$ and the Manhattan distance can be attained by setting $p = 1$. These represent special cases for the Minkowski distance. Another significant speciality concerning the Minkowski distance is given by the fact, that the greater the value of p, the closer it comes to the Chebyshev distance. For $p = \infty$ it is equal to the Chebyshev distance. [30]

7. Cosine Similarity

$$d(X_i, X_j) = \frac{\sum_{k=1}^{d} x_{ik} x_{jk}}{\sqrt{\sum_{k=1}^{d} (x_{ik})^2} \sqrt{\sum_{k=1}^{d} (x_{jk})^2}}$$

Cosine similarity measures and compares the cosine angle between two vectors in a high dimensional space. As this is a similarity measure, it returns 1 if $X_i = X_j$ and -1 if their difference is maximised. It is especially useful for high-dimensional data but can also be very

effectively applied to data of lower dimensionality. [38]

8. Correlation Distance

$$d(X_i, X_j) = 1 - c(X_i, X_j),$$

with c being the correlation coefficient of points X_i and X_j.
This metric quantifies the dissimilarity between two vectors, i.e. how vectors deviate from being prefectly correlated, by the use of the correlation coefficient. [42]

9. Chebyshev Distance

$$d(X_i, X_j) = \max_{k=1}^{d} |x_{ik} - x_{jk}|$$

The Chebyshev distance measures the maximum absolute difference between two data points. [40]

10. Jaccard Distance

$$d(X_i, X_j) = 1 - \frac{|X_i \cap X_j|}{|X_i \cup X_j|}$$

The Jaccard distance is classically applied to binary data sets. For non-binary tasks, a modified version, the generalized Jaccard similarity, can be applied. It quantifies the intersection and union of two data sets. [43]

5.3.2 Suitability of kNN for Classification Tasks

The choice of numerous different distance measures, as mentioned in the previous section, is a key reason, as to why kNN classification seems suitable to the classification task discussed in this thesis. The different distance measures make the algorithm well-suited to handle different types of data, making it especially key when dealing with several batches of data.
An additional reason, which makes kNN useful in this project is its non-parametric nature, which means that no assumptions about the underlying distribution in the data are made. This adds to the algorithm's flexibility in multi-class classification problems [11, 30].

5.4 Artificial Neural Networks (NN)

Artificial neural networks are a widely used type machine learning algorithm, the idea of which is to mimic the neural networks of biological organisms. Instead of an interconnected web of neurons, connected by nerves, an artificial neural network is made up of a number of input nodes, which feed weighted inputs into a computational output node. When training a neural network, the network receives data, along with a data label, defining what a correct prediction based on the values would be. How correctly the system is able to predict the result provides feedback as to how well-balanced the input weights were. The input weights are thereafter continuously adjusted until the prediction matches the label of the training data [44]. Generally, a neural network used in machine learning consists of a number of interconnected computational nodes, organised in layers and is thus referred to as a multilayer neural network. In a classification task, the input layer typically receives all the data features as information and the output layer delivers the classification result. Between the input and output layers, several hidden computational layers exist, which are not visible to the user. [11, 44]

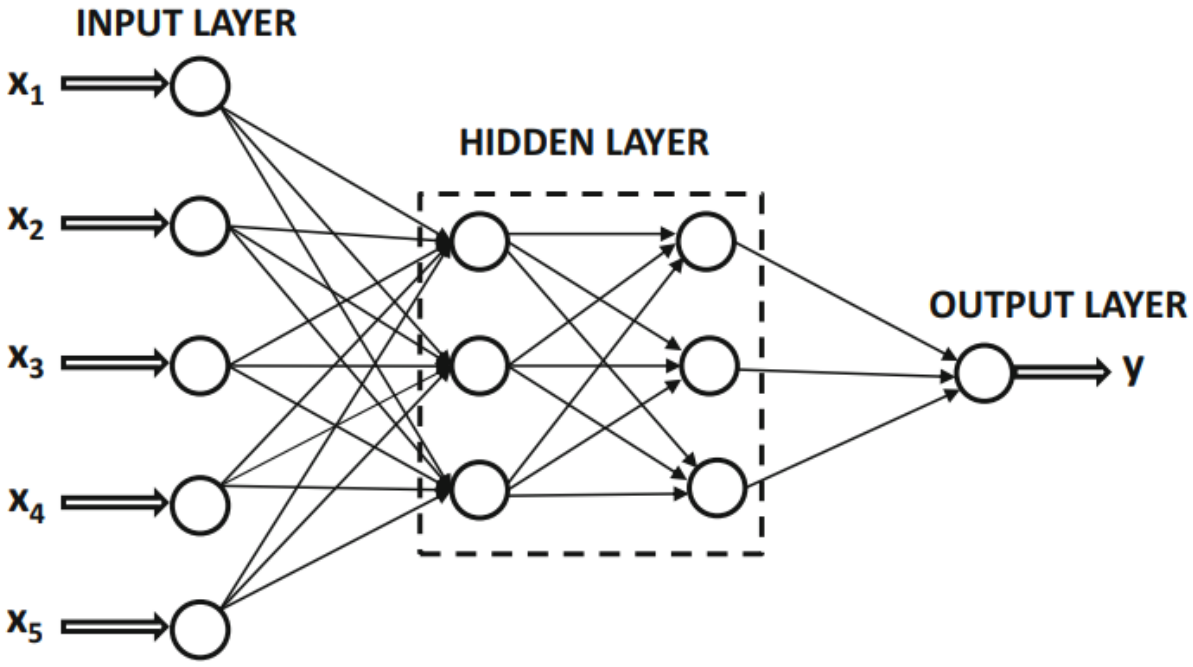


Fig. 5.4: This illustration, taken from [44], displays a general example of a multilayer neural network. The weighted inputs x_i are fed into the layer of input nodes. The input nodes, feed into two hidden layers, which perform computations that cannot be seen by the user. Finally, the hidden layers feed into the output layer, which provides the output value y [44].

5.4.1 Types of Neural Networks

Some of the most important and commonly used types of neural networks include:

1. Feed-Forward Neural Networks
   Feed-forward neural networks are the most basic type, where information travels in one direction only, from the input through to the output without any loops or cycles [44]. This type of neural network is typically used for simple classification tasks [11] and will therefore be utilised for the classification task, discussed within this work.

2. Recurrent Neural Networks
   When employing a recurrent neural network, information is able to be either fed forward or fed back to a previous network layer, allowing for a system memory. This type of neural network is commonly used in natural language processing [10, 45]. A special version of a recurrent neural network, which improves the handling of long term dependencies in data is a long short-term memory network (LSTM). LSTM networks are particularly useful for language translation tasks [45].

Some other types of artificial neural networks, which will not be discussed in more detail include: Convolutional Neural Networks, Generative Adversarial Networks, Radial Basis Function Networks, Autoencoders and Self-Organizing Maps. They are generally not used for classification tasks and therefore not applicable to the task, focussed on in this thesis.

5.4.2 Feed-Forward Network

For classification and pattern recognition tasks, the feed-forward network, or ”multilayer perceptron” is generally considered the most important model due to its architectural simplicity, making it the simplest network type to implement and train [10]. Furthermore, feed-forward networks are very effective at feature learning because the hidden layers are able to automatically learn hierarchical representations within the input data [11] and feed-forward networks are easily scaled by adding more hidden layers to work with data sets differing in complexity [10].

5.4.3 Mathematical Definition

This section will provide some mathematical insight into how the computations within a multilayer neural network can be defined.
Let L be the total number of layers present in the multilayer neural network with layers $l =$

$0, 1, 2, ..., L-1$. The input layer $X^{(0)}$ is made up of the neurons $x_1, x_2, ..., x_n$ with n being the number of input features.
The hidden layers are defined as $X^{(1)}, X^{(2)}, ..., X^{(L-1)}$, each with neurons $a_1^{(l)}, a_2^{(l)}, ..., a_{n_l}^{(l)}$ in layer l with n_l being the total number of neurons in layer l.
Finally, the output layer is defined as $X^{(L)}$, with neurons $y_1, y_2, ..., y_m$ with m being the total number of output classes.

5.4.3.1 Activation Functions

Next, let $f^{(l)}$ be the activation function for layer l. The activation function operates as a non-linear element, introduced into the network in order to enable it to learn more complex patterns in the data. It is applied to the weighted sum of the inputs for each neuron. Commonly used examples for activation functions are [10, 44]:

1. Sigmoid Activation (σ)
   A sigmoid activation function scales the input values to range between 0 and 1, making it well suited to binary classification tasks.

2. Hyperbolic Tangent Activation (tanh)
   Hyperbolic tangent activation functions work in a similar way to sigmoid functions but in the range between -1 and 1, which results in a output centred around 0.

3. (Leaky) Rectified Linear Unit ((Leaky)ReLU)
   A rectified linear unit, will directly return the received input if the input is positive and will return 0 if the input is negative. A variation on ReLU is called leaky ReLU, which allows for a slight negative slope for negative inputs, preventing non-contributing neurons.

4. Softmax Activation
   In softmax activation, the raw output values are converted to probability distributions, making it very useful for multi-class classification problems.

Having defined the activation function, the output for each neuron $a_i^{(l)}$ is calculated as follows:

$$a_i^{(l)} = f^{(l)} \left(\sum_{j=1}^{n_{l-1}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)} \right) \tag{5.14}$$

where $w_{ij}^{(l)}$ denotes the weight assigned to the connection between neuron j in layer $l-1$ and neuron i in layer l and $b_i^{(l)}$ referring to the bias term for neuron i in layer l.

5.4.4 Suitability of Neural Networks for Classification Tasks

The applicability of feed-forward neural networks is concisely described by the universal approximation theorem. The theorem states, that, due to the non-linearity introduced by the activation functions, a multilayer feed-forward neural network with at least one hidden layer can be used to approximate functions of unlimited complexity [44].

5.5 Ensemble Classification

Ensemble classification algorithms refer to techniques in machine learning, which combine aspects of different prediction models in order to achieve a better overall result than any of the individual methods. Key to this is to aim for diversity in choosing learning models to be included in the ensemble, in order to improve the models' predictive prowess and robustness. There is a wide array of different types of ensemble classification algorithms, the most important of which are bagging and boosting [31, 30]

5.5.1 Bagging (Bootstrap Aggregation)

Bagging in ensemble classification refers to the process of training multiple instances of the same base learning algorithm on different, randomly selected, subsets of the training data and combining their predictions. The randomly selected samples are referred to as bootstrap samples, which are individually trained on a base learning algorithm (e.g. a decision tree). The predictions from the individually trained models are combined through a voting system. Voting in this case, means the classification is achieved by a majority decision [46, 31]. A popular example for a bagging ensemble algorithm is a random forest, which uses decision trees as base learning algorithms [46, 30].

5.5.2 Boosting

Contrary to the parallel training in the bagging algorithm, when using a boosting ensemble classification method, basic machine learning models are trained sequentially. The reason for the sequential training procedure is, that a learning algorithm gives greater weight to the instances, which were misclassified by the learner immediately preceding it. The final model is comprised of a combination of the sequence of basic training models, with a greater weight assigned to the training models with lower error rates. Popular versions of a boosting ensemble classification method include AdaBoost (Adaptive Boosting), which refers to the general boosting method described in this

section and Gradient Boosting, which aims to minimize a loss function by adding weaker learning models in a gradient descending manner. [47, 30]

5.5.3 Suitability of Ensemble Methods for Classification Tasks

Due to the fact, that the ensemble classification methods implemented within MATLAB use decision trees, k-nearest neighbours and discriminant analysis methods as base training methods, the possibility of improving upon the result of any of the individually applied algorithms, the application of ensemble classification for this task makes sense.

5.6 Naive Bayes Classification (NB)

A very simple machine learning classification technique, which will be analysed in this section is the naive Bayes classification. The term naive Bayes refers to a group of classifiers, based on Bayes' theorem while assuming independence among the features of the data set. In spite of the fact, that this algorithm is comparatively simple, it has proven quite effective for some tasks, especially text classification problems (e.g. spam filtering or document classification) [30, 38].
The term "naive" stems from the algorithms assumptions about feature independence, which generally doesn't reflect the reality of the data, but does not stop the method from performing well in some cases. Even though the algorithm produces class density estimates, which are biased, the introduced bias might not significantly affect the posterior probabilities [11, 30].

5.6.1 Mathematical Definition

Naive Bayes classification models are generally based on Bayes' theorem, which relates the conditional and marginal probabilities of random events. When applied to a classification task, the theorem can be expressed as [11]:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)} \tag{5.15}$$

Where [11]:

- $P(C|X)$ is the probability of class C, given observation X
- $P(X|C)$ is the probability of observing X, given class C
- $P(C)$ is the probability of class C
- $P(X)$ is the probability of observation X

Another relevant definition concerns the ”naive” aspect of the naive Bayes theorem. It assumes that the all features in the data set, used to describe a specific instance within it, are conditionally independent when given the class label. This means, that the existence or non-existence of a specific feature has no bearing on any of the other features. This can be described as [11]:

$$P(X|C) = P(x_1|C)P(x_2|C)...P(x_n|C) \tag{5.16}$$

Where class X consists of features $x_1, ..., x_n$. The parameters $P(x_1|C)$ of the training are estimated using maximum-likelihood estimation. When applied to discrete functions, this involves the counting of feature-value pairs, whereas for the application in continuous functions, a distribution (e.g. Gaussian) is assumed [11].
The actual prediction is defined by [48]:

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \text{for} 1 \leq j \leq m, j \neq i \tag{5.17}$$

where m denotes the number of classes required in the classification task it is applied to.
The equation 5.17 seeks to predict the class label for feature X. The value $P(X|C_i)P(C_i)$ is calculated for each class C_i and the class C_i is predicted when the condition in equation 5.17 is met, meaning, when $P(X|C_i)P(C_i)$ is the maximum of all values.

5.6.2 Suitability of Naive Bayes Classification

Due to the simple nature of naive Bayes classification along with its ability to perform multi-class classification, it is definitely suitable to the classification task within this thesis. Whether the naive assumptions of feature independence hold true when applying the algorithm, or have a detrimental impact on the result, must be tested.

Chapter 6
Example Application and Results

In this chapter, the application of each algorithm is presented and discussed. Moreover the resulting classifications are compared and evaluated.

6.1 Approach

Of the machine learning algorithms, introduced earlier within this thesis, five relevant methods were applied to each batch of data separately. Each batch of data was partitioned randomly to create 90% training data and to reserve 10% of data points for testing. For each batch of data, the results of one application of an algorithm is a confusion matrix showing the correctly classified test data points along the diagonal and each deviation outside the matrix diagonal. Additionally, the accuracy of the procedure is calculated and the time taken for training the algorithm, normalised to the number of instances contained within a data batch, in order to make it comparable, is measured and documented. An example for a confusion matrix is visualised in the following figure 6.1:

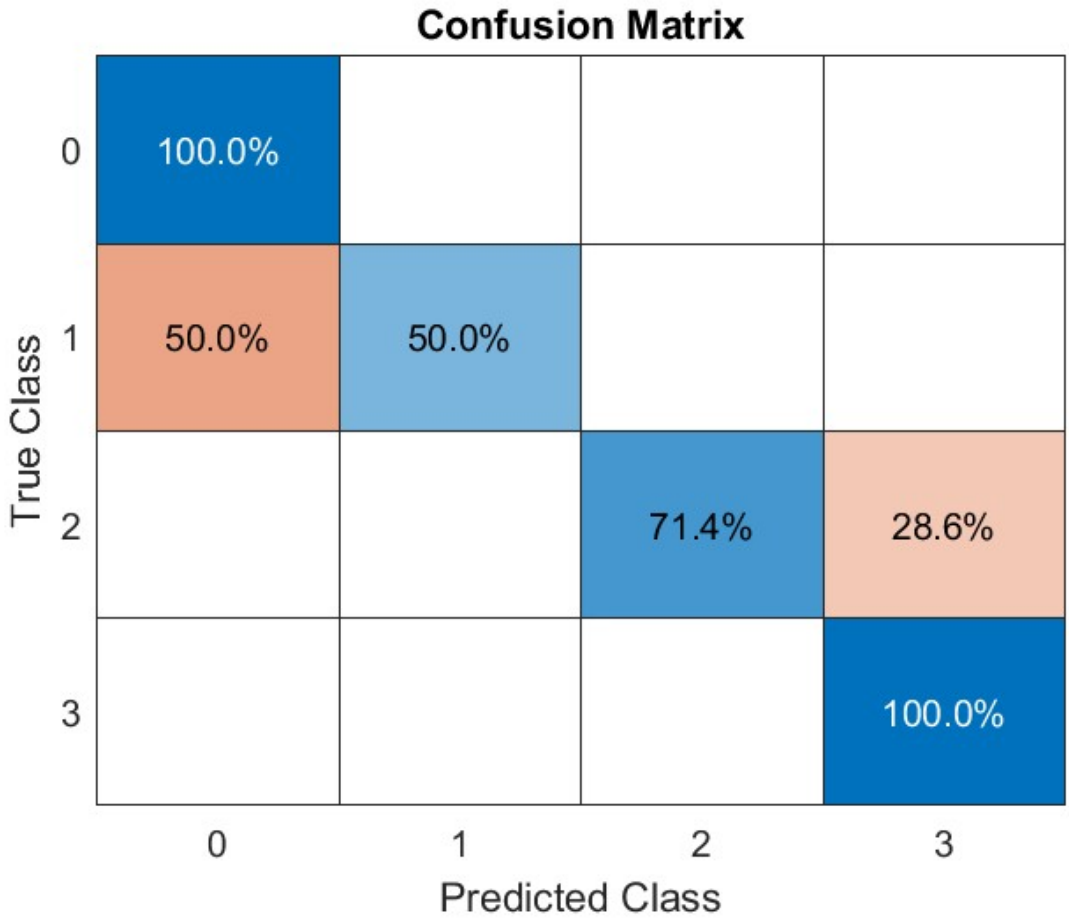


Fig. 6.1: This figure shows an example of a confusion matrix, calculated for each data batch. This example is from a decision tree classifier, applied to processed data obtained on 08.03.2023.

The accuracy for each iteration is calculated by defining the following terms. Let:

1. TP be the number of true positives (meaning correctly predicted states)
2. TN be the number of true negatives
3. FP be the number of false positives
4. FN be the number of false negatives.

The accuracy measure m_a in % can now be calculated as:

$$m_a = \frac{TP+TN}{TP+TN+FP+FN} \times 100 \tag{6.1}$$

The confusion matrix (in absolute values, unlike the example shown in 6.1) for a machine learning algorithm and for each data batch is collected in a hypermatrix, with each layer being a two dimensional confusion matrix, stacked in the third dimension. Additionally, the accuracy values and training time values for each algorithm are averaged and collected in a single value for each algorithm.

6.1.1 Hyperparameter Optimisation

For each individual application of an algorithm, the automated hyperparameter optimisation, implemented in MATLAB , is used to attain the best result per data batch. Each time an algorithm is applied, each file is evaluated 30 times in order to optimise various hyperparameters, e.g. for kNN-Classification, different values for hyperparameter k and the different types of distance measures are evaluated by the system before the best combination is selected. This makes the training process a lot more time consuming and offers a clear possibility for increasing the efficiency, if the algorithm is selected for real world application.

6.2 Computational Limitation

A caveat, which needs to be considered when evaluating the results of the application of the machine learning algorithms and the hyperparameter optimisation is the available computational capacity. For this thesis, the maximum number of iterations for the hyperparameter optimization is limited to 30. This limitation is set due to the computational power, which is privately available to process the data. For industrial applications, the optimization procedure can be expanded, which might yield an improvement in the results.

6.3 Application

This section will show the results from each of the five applied machine learning algorithms with all six initially mounted sensors being available.

6.3.1 Binary Decision Trees

The first algorithm to be applied is a binary CART decision tree with hyperparameter optimisation. The numerical results for the classification procedure are shown in figure 6.2. The measures for accuracy and time for each data batch are also displayed in table 6.1.

6.3.1.1 Optimised Hyperparameters

When applying the binary decision tree algorithm in MATLAB , the hyperparameter optimization features, incorporated in MATLAB were applied. In this case, this refers to finding the optimal minimal leaf size per file and pruning the tree to the optimal size.

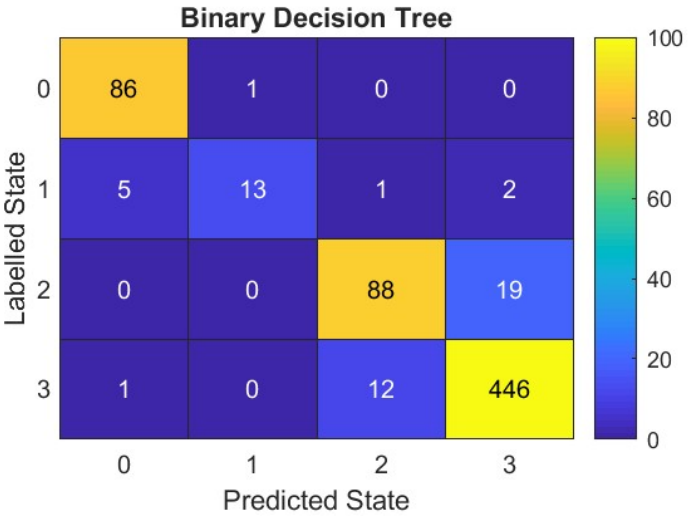


Fig. 6.2: This figure shows the cumulative classification of test data, using a binary decision tree, in all available data files with all six sensors functioning. This represents a stacking of the confusion matrices of all data files in absolute values, meaning the closer to a perfectly diagonal distribution, the better the classification result.

# File	Accuracy [%]	Training Time [s/point]
1	94.3820	0.0308
2	96.5035	0.0122
3	91.5663	0.0206
4	95.4955	0.0143
5	95.4545	0.0352
6	88.5714	0.0250
7	92.1053	0.0150
8	95.0000	0.0795
Median	94.6910	0.0228

Table 6.1: This table shows the accuracy and training time per data point when applying a decision tree classifier to each available file and optimizing its hyperparameters, with all six sensors functioning.

The results displayed in the heatmap in 6.1 and table 6.2 clearly show how well this algorithm is suited to the classification task at hand. The diagonal nature of the visualisation of the predicted system states against the labelled system states clearly illustrates what the accuracy measure tells us numerically, which is that misclassification by this algorithm is quite rare. The most common classification error occurs between states 2 and 3 i.e. the most common issue in this case is the distinction between the machine only carrying fluid and carrying fluid and material mixed together. Furthermore, the training time required for this algorithm indicates that it would be scalable to greater training data volumes without the time taken for training the system becoming a concern.

6.3.2 Linear Discriminant Analysis

The results of the application of linear discriminant analysis are shown in visual form in 6.3 and numerically, displaying the training time including hyperparameter optimization and the accuracy of the classification in 6.2.

6.3.2.1 Optimised Hyperparameters

When applying linear discriminant analysis in MATLAB , the hyperparameter optimization features, incorporated in MATLAB were applied. The hyperparameters optimized by MATLAB in this case are the parameters gamma and delta. Gamma refers to the amount of regularization used to adjust the covariance matrix. This will result in a scalar value between 0, meaning no regularization being applied and 1, where maximum regularization is applied to the covariance matrix. The parameter delta specifies the linear coefficient threshold. If a coefficient in the model is smaller than the value of delta, it is set to 0 and the corresponding predictor is eliminated from the model. The higher the value of delta is set, the more predictors are eliminated.

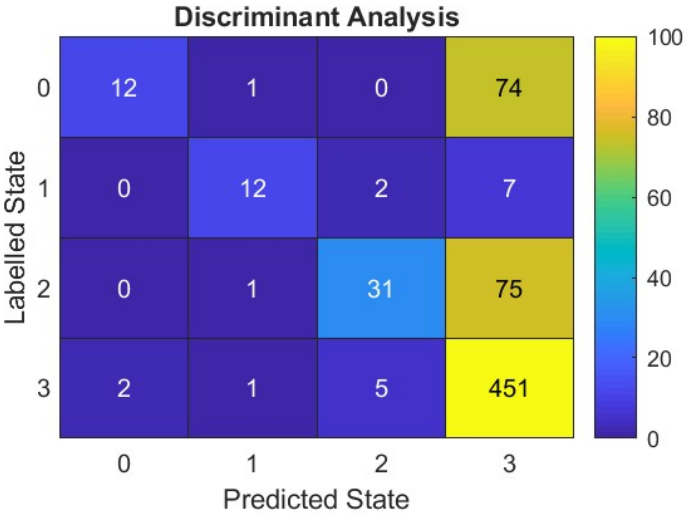


Fig. 6.3: This figure shows the cumulative classification of test data, using linear discriminant analysis, in all available data files with all six sensors functioning. This represents a stacking of the confusion matrices of all data files in absolute values, meaning the closer to a perfectly diagonal distribution, the better the classification.

# File	Accuracy [%]	Training Time [s/point]
1	84.2697	0.0245
2	65.0350	0.0131
3	74.6988	0.0263
4	79.2793	0.0162
5	86.3636	0.0427
6	80.0000	0.0254
7	69.298	0.0159
8	75.0000	0.0871
Median	77.1396	0.0249

Table 6.2: This table shows the accuracy and training time per data point when applying a linear discriminant analysis classifier to each available file and optimizing its hyperparameters, with all six sensor functioning.

The results displayed in 6.3 and 6.2 show very poor results throughout all states. Only the prediction of state 3 seems reliable, however the misclassification of most other states as being state 3 suggests that this correct classification is by coincidence and perhaps questionable.

6.3.3 k-Nearest Neighbours

The results of the application of the k-nearest neighbours algorithm are shown in visual form in 6.4 and numerically, displaying the training time including hyperparameter optimization and the accuracy of the classification in 6.3.

6.3.3.1 Optimized Hyperparameters

When applying the k-nearest neighbours classification algorithm in MATLAB , the hyperparameter optimization features, inherent to MATLAB were applied. In this case, this refers to optimizing the value for k, meaning the number of neighbours and the type of distance measure applied (e.g. Jaccard-Distance, Chebyshev-Distance, etc.).

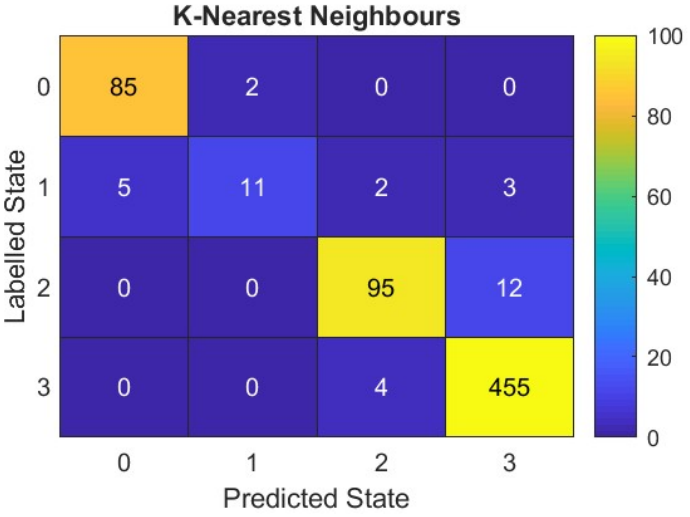


Fig. 6.4: This figure shows the cumulative classification of test data, using the k-nearest neighbours classifier, in all available data files with all six sensors functioning. This represents a stacking of the confusion matrices of all data files in absolute values, meaning the closer to a perfectly diagonal distribution, the better the classification.

# File	Accuracy [%]	Training Time [s/point]
1	96.6292	0.0210
2	98.6014	0.0143
3	93.9759	0.0232
4	97.2973	0.0182
5	95.4545	0.0422
6	95.7143	0.0277
7	92.1053	0.0173
8	95.0000	0.091
Median	95.5972	0.0221

Table 6.3: This table shows the accuracy and training time per data point when applying a k-nearest neighbours classifier to each available file and optimizing its hyperparameters, with all six sensor functioning.

The results displayed in the heatmap in figure 6.3 and the table 6.3 show both very good classification accuracy as well as short training times. This suggests, that the k-nearest neighbours algorithm is well suited to the classification task and can be upscaled to greater data volumes without the required training time becoming a hindrance to the application.

6.3.4 Neural Networks

The results of the application of artificial neural networks are shown in visual form in 6.5 and numerically, displaying the training time including hyperparameter optimization and the accuracy of the classification in 6.4.

6.3.4.1 Optimised Hyperparameters

When applying the neural network classification algorithm in MATLAB , the hyperparameter optimization features, native to MATLAB were applied. In this case, this refers to optimizing the layer sizes for each layer of the generated multi-layer network, optimizing the activation function between layers (e.g. sigmoid activation, tanh activation or no activation function) and the term λ, which refers to a regularization function, which controls the amount of regularization applied to the network. In addition, the hyperparameter optimization process assesses whether or not to standardize the predictor data. If standardization is active, then the system centres and scales the numeric predictor variable by the mean and standard deviation.

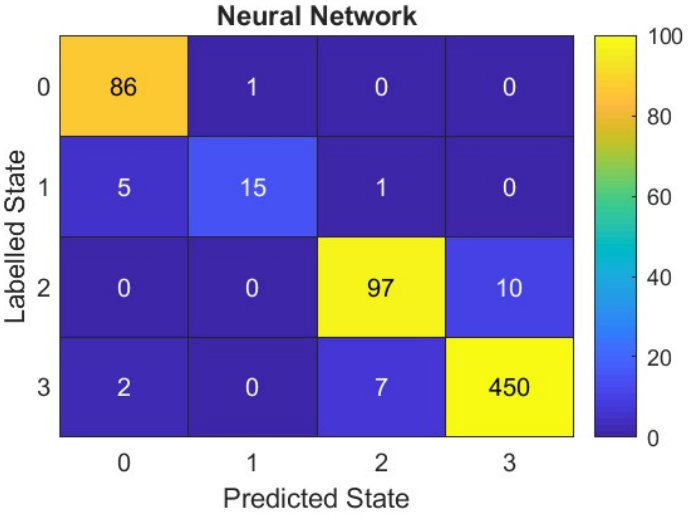


Fig. 6.5: This figure shows the cumulative classification of test data, using artificial neural networks as the classification algorithm, in all available data files with all six sensors functioning. This represents a stacking of the confusion matrices of all data files in absolute values, meaning the closer to a perfectly diagonal distribution, the better the classification.

# File	Accuracy [%]	Training Time [s/point]
1	94.3820	0.5861
2	97.9021	0.2467
3	98.7952	0.8566
4	95.4955	0.1826
5	97.7273	0.3513
6	97.1429	0.6271
7	92.1053	0.2721
8	100.000	0.5849
Median	97.4351	0.4681

Table 6.4: This table shows the accuracy and training time per data point when applying an artificial neural network classifier to each available file and optimizing its hyperparameters, with all six sensors functioning.

The results displayed in the figure 6.5 show exceptional classification accuracy, however the training times, collected in table 6.4 indicate that the amount of data provided in the batches anal-

ysed causes very long training times and significantly calls this algorithm's suitability to larger tasks into question.

6.3.5 Ensemble Classification Methods

The results of the application of ensemble classification methods are shown in visual form in 6.6 and numerically, displaying the training time including hyperparameter optimization and the accuracy of the classification in 6.5.

6.3.5.1 Optimized Hyperparameters

When optimizing the hyperparameters for ensemble classification in MATLAB , the algorithm automatically optimizes the type of ensemble used. There are several types of bagging and boosting algorithms, suited for multiclass classification implemented. A second aspect that is optimized in each iteration, which is dependant on the type of ensemble learning method used, is the type of base learning algorithm. In MATLAB this will either be discriminant analysis, k-nearest neighbours or a decision tree.

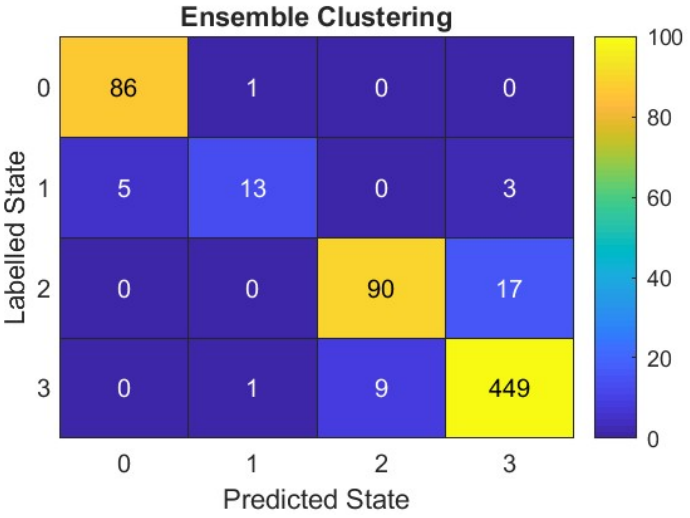


Fig. 6.6: This figure shows the cumulative classification of test data, using ensemble methods as the classification algorithm, in all available data files with all six sensors functioning. This represents a stacking of the confusion matrices of all data files in absolute values, meaning the closer to a perfectly diagonal distribution, the better the classification.

# File	Accuracy [%]	Training Time [s/point]
1	94.3820	0.0618
2	97.2028	0.0360
3	96.3855	0.0829
4	96.3964	0.0652
5	93.1818	0.1171
6	91.4286	0.0538
7	91.2281	0.1273
8	95.0000	0.4595
Median	94.6910	0.0740

Table 6.5: This table shows the accuracy and training time per data point when applying ensemble classifiers to each available file and optimizing its hyperparameters, with all six sensors functioning.

The results displayed in the figure 6.5 and the table 6.5, show very good classification capabilities for this algorithm, however the accuracy does not significantly exceed previously applied algorithms, while the training time is significantly longer. This suggests, that, in this case, the process of using several base learning algorithms does not yield better classification results, than the base algorithms on their own.

6.3.6 Naive Bayes

The results of the application naive Bayes classification methods are shown in visual form in 6.7 and numerically, displaying the training time including hyperparameter optimization and the accuracy of the classification in 6.6.

6.3.6.1 Optimized Hyperparameters

When applying the naive Bayes multiclass classification techniques, inherent to MATLAB , two hyperparameters where iteratively optimized. The first being the type of assumed distribution. The classifier can be set to assume multinomial distribution, multivariate multinomial distribution, normal (Gaussian) distribution or kernel smoothing. The second optimized hyperparameter is the assumed kernel-width.

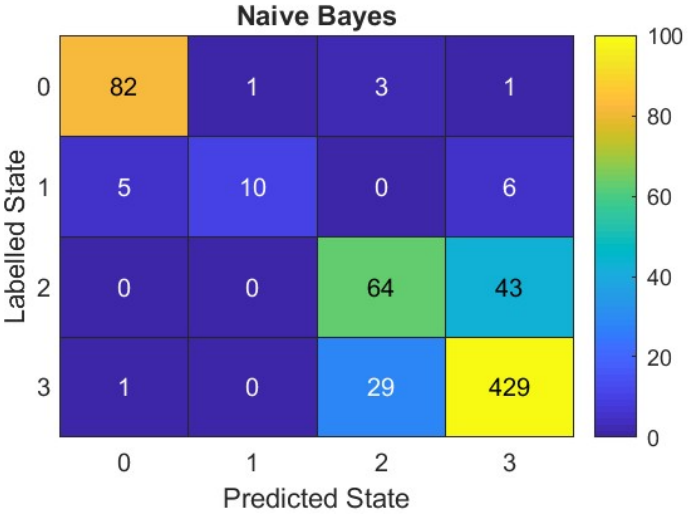


Fig. 6.7: This figure shows the cumulative classification of test data, using naive Bayes as the classification algorithm, in all available data files with all six sensors functioning. This represents a stacking of the confusion matrices of all data files in absolute values, meaning the closer to a perfectly diagonal distribution, the better the classification.

# File	Accuracy [%]	Training Time [s/point]
1	91.0112	0.0358
2	86.0140	0.0311
3	79.5181	0.0473
4	93.6937	0.0336
5	95.4545	0.0624
6	85.7143	0.0431
7	79.8246	0.0254
8	95.0000	0.1359
Median	88.5126	0.0394

Table 6.6: This table shows the accuracy and training time per data point when applying naive Bayes classification methods to each available file and optimizing its hyperparameters, with all six sensors functioning.

The results displayed in the figure 6.6 and the table 6.6, show good classification accuracy and acceptable training times. However both these measures are significantly worse than those previously discussed algorithms have yielded.

6.4 Comparison of Results

This section will compare the results from the application of each of the six different machine learning algorithms displayed in the previous section.

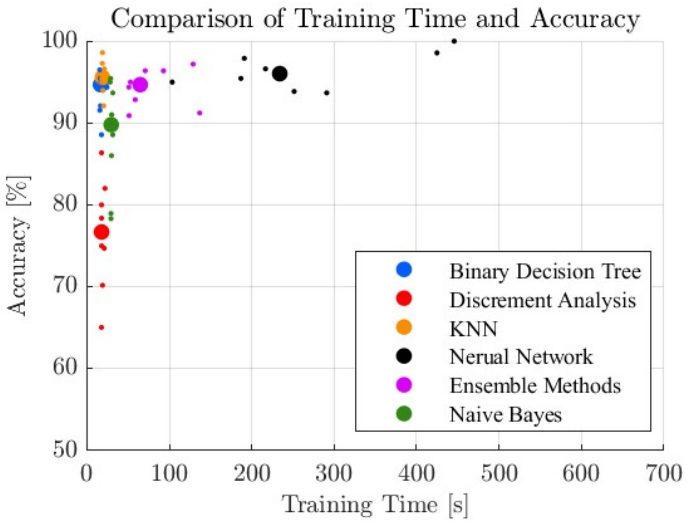


Fig. 6.8: This scatter plot compares the six applied machine learning algorithms in terms of their predictive accuracy and the training time throughout the available data files. Each small dot represents a single data batch and the larger dots show the median value for each algorithm.

Algorithm	Acc. [%]	Time [s/point]
Decision Trees	94.6910	0.0228
Discriminant Analysis	77.1396	0.0249
k-Nearest Neighbours	95.5844	0.0221
Neural Networks	97.4351	0.4681
Ensemble Methods	94.6910	0.0740
Naive Bayes	88.5126	0.0394

Table 6.7: This table lists the median predictive accuracy and median training times per data point for each of the applied machine learning classification methods across the individual data files.

The numerical results in 6.7 show, that maximum predictive accuracy for the task was achieved by using artificial neural networks. However the achievement of these exceptional high values for accuracy come at the cost of the significantly longest time required to train the algorithm.
The visual representation of the results in figure 6.8 clearly displays two methods in the upper left corner of the graph which excel in both predictive capabilities and boast short training times; Binary decision trees and k-nearest neighbours classifiers.

6.5 Results with 5 sensors

Two data files were recorded after one of the six sensor units failed. These are analysed separately in this section in order to compare whether the results are comparable to those attained from the application with all six active sensors discussed in the previous section. The resulting heatmaps from each application are collected in 6.9.

The results displayed in 6.9 show clear similarities to the results achieved with all six sensors active, presented in the previous section. The certainty, with which these results can be utilised is given by the analysis in chapter 4. The combination of this and the fact, that the same algorithms as for all six sensors show promise of performing well in data classification lead to the conclusion, that even with five sensors, the machine state can be reliably classified.

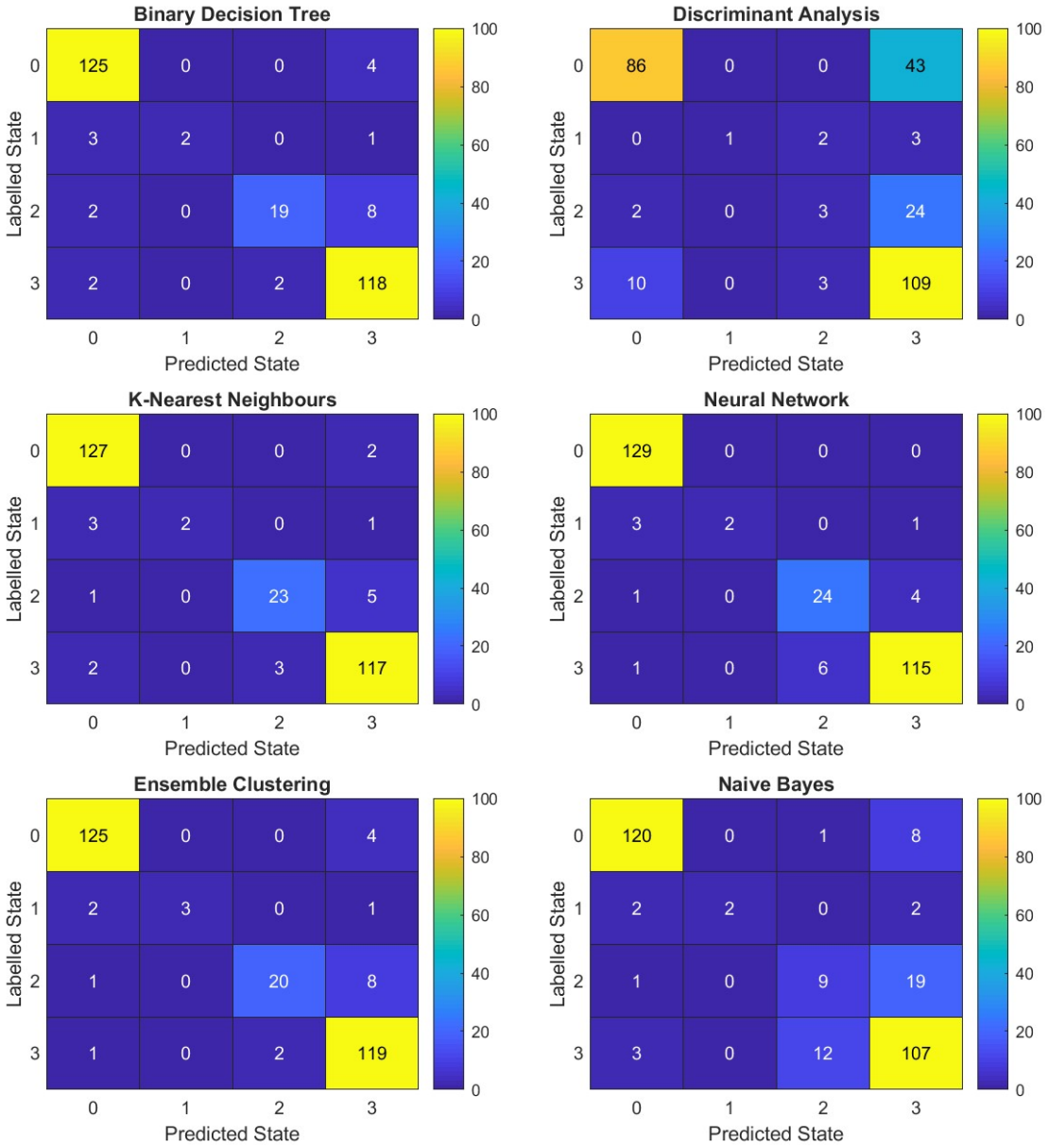


Fig. 6.9: This figure shows the results from the application of each machine learning algorithm on the two data files with five active sensor units, collected in heatmaps.

Chapter 7
Conclusion, Summary and Outlook

This thesis investigated the levels of redundancy within batches of sensor data from six sensor units, applied to an industrial machine, in order to learn about the amount of information still available when one or two of the six sensor units fail. Moreover, an analysis of the applicability of several supervised machine learning algorithms was applied to batches of labelled sensor data in order to assess their viability for condition monitoring purposes in industrial applications.
The redundancy level within the data is quantified by analysing the inherent dimensional coverage within the data. This is achieved by the application of singular value decomposition, which yields the proportional coverage of information within the data as a function of the number of considered dimensions. The resulting calculated coverage gives an indication, whether dimensionality reduction measures can viably be applied. Dimensionality reduction methods are applied in order to reduce the amount of data required to still be able to extract the necessary information and thus streamline the data processing. The chosen dimensionality reduction method applied is principal component analysis, which identifies the major vibrational axis of the system and realigns the system accordingly. Consequently, only the primary principal component of each sensor needs to be considered and the dimensionality of the data has been reduced from 18 data channels, three acceleration measurements per sensor unit, to only six principal components, one per sensor unit.
A major reason for analysing the system's redundancy is to be able to gauge the impact on the amount of information contained within the data that losing a sensor would have. This is analysed by systematically calculating the dimensional coverage of every permutation of one or two sensors being inactive. The results of this evaluation show that when one of six sensors is inactive, the system still has 91% of the original coverage and when any two sensors are inactive, about 81% coverage remains. The outcome from this evaluation shows that whatever specific combination of sensor units is inactive has little to no bearing on the remaining dimensional coverage. The conclusion, which can be drawn at this point is, that when any one or two of the six sensor units fail, the certainty with which results of analyses applied to the data can be interpreted remains very high.
The second aspect that this thesis is focused on is the applicability of machine learning algorithms for the classification of the machine state based on the provided batches of labelled sensor data. Several types of machine learning algorithms suited to classification tasks and capable of multi-

class classification are analysed and applied. Each selected algorithm is applied to each data batch separately, the results of each being collected and interpreted. Prior to each application, the data set is randomly split into 90% training data and 10% testing data. Moreover, the hyperparameters for each applied algorithm are optimized automatically, in an effort to achieve the best possible result for each. Generally, the aim here is to identify promising machine learning methods capable of classifying the machine state, purely from acceleration data. The suitability of any of the applied algorithms is quantified in two metrics: the predictive accuracy and the required average training time.

The results from the application process show several well suited algorithms for the classification task at hand. Two algorithms in particular stand out given the accuracy and training time metrics: Binary Decision Trees and k-Nearest Neighbours classification. The applicability of these algorithms represent the conclusion from the application process and comparison performed within this thesis.

Based on the work presented within this thesis, further investigations into different types of classification, such as binary classification algorithms (e.g. support vector machines), which can be applied to multi-class classification by error-corrective output codes can be conducted. Moreover, the possibility of applying unsupervised learning algorithms to continuous data streams instead of supervised learning methods to labelled data batches represents a possible avenue for further research.

References

[1] Tianjia He et al. “Exploring Inherent Sensor Redundancy for Automotive Anomaly Detection”. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)* (2020), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:221134703.

[2] Yong Gao, Kui Wu, and Fulu Li. “Analysis on the Redundancy of Wireless Sensor Networks”. In: *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*. WSNA ’03. San Diego, CA, USA: Association for Computing Machinery, 2003, pp. 108–114. ISBN: 1581137648. DOI: 10.1145/941350.941366. URL: https://doi.org/10.1145/941350.941366.

[3] Bogdan Carbunar et al. “Redundancy and Coverage Detection in Sensor Networks”. In: *TOSN* 2 (Feb. 2006), pp. 94–128. DOI: 10.1145/1138127.1138131.

[4] Karolina Kudelina et al. “Trends and challenges in intelligent condition monitoring of electrical machines using machine learning”. In: *Applied Sciences* 11.6 (2021), p. 2761.

[5] Minh-Quang Tran et al. “Machine learning and IoT-based approach for tool condition monitoring: A review and future prospects”. In: *Measurement* 207 (2023), p. 112351.

[6] Shaveta. “A review on machine learning”. In: *International Journal of Science and Research Archive (IJSRA)* (May 2023). ISSN: 2582-8185.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[8] Dan Jurafsky and James H. Martin. “Speech and Language Processing”. In: 2000. URL: https://api.semanticscholar.org/CorpusID:5073927.

[9] Ziad Obermeyer and Ezekiel J. Emanuel. “Predicting the future—big data, machine learning, and clinical medicine”. eng. In: *N Engl J Med* 375 (Sept. 2016), pp. 1216–1219. ISSN: 0028-4793. DOI: 10.1056/nejmp1606181.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[11] Christopher M.Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, 2006.

[12] Rich Caruana et al. “Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015). URL: https://api.semanticscholar.org/CorpusID:14190268.

[13] Jack E. Olson. *Data Quality: The Accuracy Dimension*. Elsevier, 2003. ISBN: 978-1-55860-891-7.

[14] Richard Y. Wang and Diane M. Strong. “Beyond Accuracy: What Data Quality Means to Data Consumers”. In: *Journal of Management Information Systems* 12.4 (1996), pp. 5–33. ISSN: 0742-1222.

[15] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 3540331727.

[16] Mark J. Embrechts, Boleslaw Karol Szymanski, and Karsten Sternickel. “Introduction to Scientific Data Mining: Direct Kernel Methods and Applications”. In: *Computationally Intelligent Hybrid Systems*. 2004. URL: https://api.semanticscholar.org/CorpusID:63656646.

[17] Sasa Baskarada and Andy Koronios. “Data, Information, Knowledge, Wisdom (DIKW): A Semiotic Theoretical and Empirical Exploration of the Hierarchy and its Quality Dimension”. In: *Australasian Journal of Information Systems* 18.1 (Nov. 2013). DOI: 10.3127/ajis.v18i1.748. URL: https://journal.acs.org.au/index.php/ajis/article/view/748.

[18] Jennifer Rowley. “The Wisdom Hierarchy: Representations of the DIKW Hierarchy”. In: *J. Inf. Sci.* 33.2 (Apr. 2007), pp. 163–180. ISSN: 0165-5515. DOI: 10.1177/0165551506070706. URL: https://doi.org/10.1177/0165551506070706.

[19] R. L. Ackoff. “From Data to Wisdom”. In: *Journal of applied systems analysis* 16 (1989), pp. 3–9. URL: https://api.semanticscholar.org/CorpusID:86409890.

[20] D. Boddy, A. Boonstra, and G. Kennedy. *Managing Information Systems: Strategy and Organisation*. Prentice Hall/Financial Times, 2008. ISBN: 9780273716815. URL: https://books.google.at/books?id=geYcjeqwb_wC.

[21] Takio Kurita. “Principal Component Analysis (PCA)”. In: *Computer Vision: A Reference Guide*. Cham: Springer International Publishing, 2019, pp. 1–4. ISBN: 978-3-030-03243-2. URL: https://doi.org/10.1007/978-3-030-03243-2_649-1.

[22] S.M. Rafizul Haque. “Singular Value Decomposition and Discrete Cosine Transform Based Image Watermarking”. MA thesis. School of Engineering, Blekinge Institute of Technology, 2008.

[23] David Fuertes Roncero. “A study of QR decomposition and Kalman filter implementations”. MA thesis. KTH Stockholm, 2014.

[24] Nasir Ahmad Jehad Ali Rehanullah Kahn and Imran Masqsood. “Random Forests and Decision Trees”. In: *International Journal of Computer Science Issues (IJCSI)* 9.3 (Sept. 2012). ISSN: 1694-0814.

[25] Adnan Mohsin Abdulazeez Bahzad Taha Jijo. “Classification Based on Decision Tree Algorithm for Machine Learning”. In: *Journal of Applied Science and Technology Trends* (2021).

[26] J.R. Quinlan. “Induction of Decision Trees”. In: *Machine Learning 1*. Centre for Advanced Computing Sciences, New South Wales Institute of Technology, Sydney. 1986, pp. 81–106.

[27] Steven L. Salzberg. “Book Review: C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993”. In: *Machine Learning* 16 (1994), pp. 235–240. URL: https://api.semanticscholar.org/CorpusID:17414427.

[28] L. Breiman et al. “Classification and Regression Trees”. In: *Biometrics* 40 (1984), p. 874. URL: https://api.semanticscholar.org/CorpusID:29458883.

[29] G. V. Kass. “An Exploratory Technique for Investigating Large Quantities of Categorical Data”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 29.2 (1980), pp. 119–127. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2986296 (visited on 12/06/2023).

[30] Jerome Friedman Trevor Hastie Eobert Tibshirani. *The Elemenets of Statistical Learning*. Springer, 2008.

[31] Leo Breiman. “Bagging Predictors”. In: *Machine Learning* 24 (1996), pp. 123–140.

[32] R.A. Fisher. “The Use of Multiple Measurements in Taxonomic Problems”. In: *The Annals of Eugenics v.7* (1936), pp. 179–188.

[33] Richard Duda, Peter Hart, and David G.Stork. “Pattern Classification”. In: vol. xx. Jan. 2001. ISBN: 0-471-05669-3.

[34] Geoffrey J. McLachlan and Thriyambakam Krishnan. “The EM algorithm and extensions”. In: 1996. URL: https://api.semanticscholar.org/CorpusID:122530182.

[35] Hubert, Rousseeuw, and Branden. “ROBPCA: A New Approach to Robust Principal Component Analysis”. In: 47 (2005). DOI: 10.1198/004017004000000563.

[36] Richard A. Johnson. *Applied Multivariate Statistical Analysis*. Ed. by Linda Mihatov Behrens Petra Recter Debbie Ryan. 6th ed. Pearson Education Inc., 2007. ISBN: 978-0-13-187715.

[37] T. Cover and P. Hart. “Nearest neighbor pattern classification”. In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[38] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 9780511573361. URL: https://books.google.at/books?id=4DgZywEACAAJ.

[39] P.C. Mahalanobis. “On the generalized distance in statistics”. In: vol. 2. National Institute of Science of India, 1936, pp. 49–55.

[40] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. “On the surprising behavior of distance metrics in high dimensional space”. In: *Lecture Notes in Computer Science*. Springer, 2001, pp. 420–434.

[41] Sidney Siegel. “Nonparametric statistics for the behavioral sciences”. In: 1956. URL: https://api.semanticscholar.org/CorpusID:146286676.

[42] S.S. Wilks. “The Large-Sample Distribution of the Likelihood Ratio for Testing Composite Hypotheses”. In: *The Annals of Mathematical Statistics* 9 (1938), pp. 60–62.

[43] Jaccard, Paul. “Étude comparative de la distribution florale dans une portion des Alpes et du Jura”. In: (1901). DOI: 10.5169/SEALS-266450. URL: https://www.e-periodica.ch/digbib/view?pid=bsv-002:1901:37::790.

[44] Charu C. Aggarwal. *Neural Networks and Deep Learning*. 2nd ed. Springer Cham, June 2023. ISBN: 978-3-031-29641-3.

[45] Christopher Olah. “Understanding lstm networks”. In: (2015).

[46] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[47] Yoav Freund and Robert E Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. In: *J. Comput. Syst. Sci.* 55.1 (Aug. 1997), pp. 119–139. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504. URL: https://doi.org/10.1006/jcss.1997.1504.

[48] J.Pei J. Han M.Kumber. *Data MiningConcepts and Techniques*. Third. Elsevier, 2012. ISBN: 978-0-12-381479-1.

Appendix A

Appendix A: Dimensionality Reduction Code

Dimensionality Analyis and Data Processing

Author: Elliot Lang

E-Mail: elliot.lang@stud.unileoben.ac.at

This script shows the steps and methods used for processing the raw sensor data. The goal of this script is to apply dimensionality reduction measures and investigate how the informational coverage behaves when sensors are lost.

```
close all;
clear;
```

Load the training data file

```
trainDir = [cd,'\TotalData\TrainingData'];
trainFile = '2023-04-05_combined_data.mat';

fullFileRef = fullfile( trainDir, trainFile );
load( fullFileRef );
```

Select only the required channels

```
requiredNames = {'State', '_Acceleration'};
stateTT = TTSelectPartialName(DataTT, {'State'});
accelTT = TTSelectPartialName(DataTT, {'_Acceleration'});

D = accelTT{:,:};
```

Deal with the NaNs by finding the Nan Range and replacing/interpolating them.

```
d = sum(D,2);
inds = find( ~isnan(d) );
range = inds(1):inds(end);
D(1:inds(1), :) = 0;
D(inds(end):end, :) = 0;
D = patchColumnNaNs(D);
```

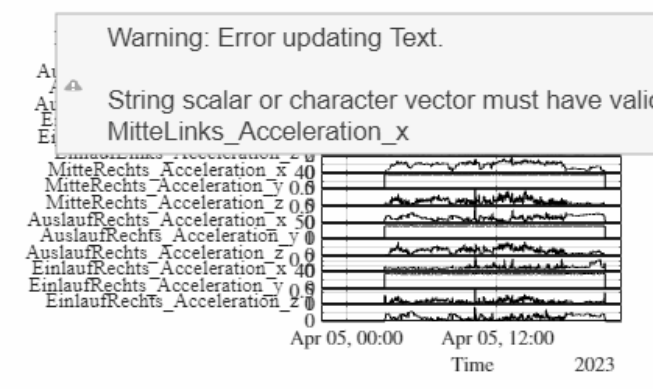
Perform linear interpolation for NaNs, which don't occur at the beginning or end of the data.

```
D = patchColumnNaNs( D );
accelTT{:,:} = D;
```

Visualise all the channels with the adjusted data.

```
figRaw = figureGen(18,30);
rawData = TTStackedPlot( figRaw, accelTT);
```

```
saveas(rawData, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\05_Data
Preparation\figures\trimmed.jpg');
```



Process data

At this point we have valid data without NaNs

Make the data mean free.

```
[m,n] = size( D );
for k=1:n
  D(inds(1):inds(end),k) = D(inds(1):inds(end),k) - mean(D(inds(1):inds(end),k));
end
```

Fillter the data as little as possible, applying a small amount of smoothing.

```
filterData = true;
ls = 3;
if filterData
  [md,nd] = size( D );
  for k=1:nd
    D(inds(1):inds(end),k) = movmean( D(inds(1):inds(end),k), ls);
  end
end
```

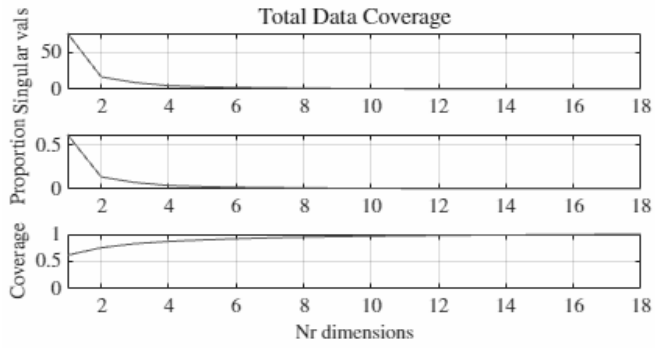
Apply PCA to the complete Data Set and calculate the dimensional coverage.

```
[Ut,St,Vt] = svd( D, 0 );
st = diag(St);
portion_t = st / sum(st) ;
Coverage_total = cumsum( portion_t );
```

Visualize the singular values attained from the SVD, the proportion and the dimensional coverage.

```
figPCA_total = figureGen(8,15);
tiles = tiledlayout( 3, 1, 'TileSpacing','tight','Padding','tight');
```

```
Ax(1) = nexttile( tiles );
plot( st );
ylabel('Singular vals');
grid on;
title('Total Data Coverage');
Ax(1).XLim = [1,18];

Ax(2) = nexttile( tiles );
plot( portion_t );
ylabel('Proportion');
grid on;
Ax(2).XLim = [1,18];

Ax(3) = nexttile( tiles );
plot( Coverage_total );
xlabel('Nr dimensions');
ylabel('Coverage');
grid on;

Ax(3).XLim = [1,18];

Ax(3).YLim = [0,1];

saveas(figPCA_total, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\05_Data
Preparation\figures\PCA_total_data.jpg');
```



Apply PCA in groups of three

The application occurs for each sensor containing three acceleration dimensions each. Begin by defining the number of groups.

```
nr = 3;
range = 1:nr;
nrGroups = round( n / nr );
```

Create space to save the resulting principal components in a matrix.

```
PCs = zeros(m,nrGroups);
```

Cycle through each sensor and apply PCA.

```
for k=1:nrGroups
  k;
  cut = nr*(k-1) + range;
  Cut = D(:,cut);

  [Us,Ss,Vs] = svd( Cut, 0 );
  ss = diag(Ss);
  ss = cumsum(ss / sum(ss));
```

Check if one channel is mirrored.

```
  detVs = det( Vs );
  if mod(k,2) == 0
    ps = Us(:,1) * Ss(1,1);
  else
    ps = -Us(:,1) * Ss(1,1);
  end

  PCs(:,k) = ps;
end
```

```
k = 1
ss = 3×1
    0.8083
    0.9591
    1.0000
k = 2
ss = 3×1
    0.8680
    0.9837
    1.0000
k = 3
ss = 3×1
    0.7139
    0.9509
    1.0000
k = 4
ss = 3×1
    0.8741
    0.9630
    1.0000
k = 5
ss = 3×1
    0.8576
    0.9755
    1.0000
k = 6
ss = 3×1
    0.8155
    0.9414
    1.0000
```

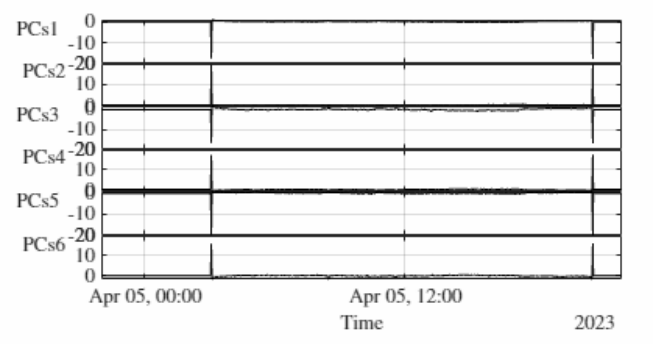
Make each principal component mean free.

```
[mp,np] = size( PCs );
```

```
for k=1:np
  PCs(inds(1):inds(end),k) = PCs(inds(1):inds(end),k) -
mean(PCs(inds(1):inds(end),k));
end
```

Convert the result back into a timetable with the corresponding row times and visualize it.

```
PCsTT = array2timetable(PCs, 'RowTimes', accelTT.Properties.RowTimes);
figPCs = figureGen(8,15);
PCA_PLOT = TTStackedPlot( figPCs, PCsTT );
saveas(PCA_PLOT, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\05_Data
Preparation\figures\PCA_Plot.jpg');
```




```
destDir = 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master Thesis\PCA Data';
fName = 'PCA_05.04.23';
dataTTFileName = fullfile( destDir, [fName,'.mat']);
save( dataTTFileName, 'PCsTT', 'stateTT');
```

Check the Coverage applying PCA to each Sensor seperately

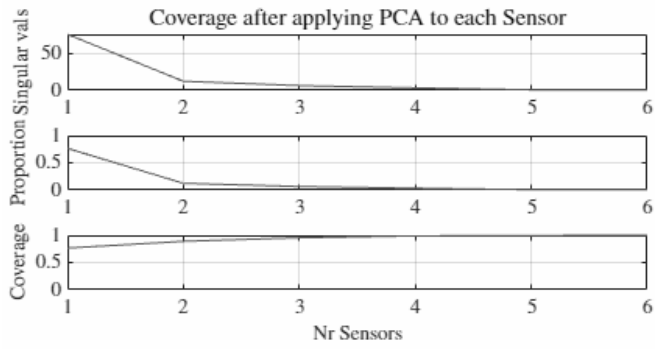
The process is the same as above.

```
[Ui,Si,Vi] = svd( PCs, 0 );
si = diag(Si);
portion_i = si / sum(si) ;
Coverage_individual = cumsum( portion_i );
```

Visualize the coverage, the singular values and the portion.

```
figPCAPCs = figureGen(8,15);
tiles = tiledlayout( 3, 1, 'TileSpacing','tight','Padding','tight');

Ax(1) = nexttile( tiles );
plot( si );
ylabel('Singular vals');
grid on;
title('Coverage after applying PCA to each Sensor');
```

```
Ax(2) = nexttile( tiles );
plot( portion_i );
ylabel('Proportion');
grid on;
Ax(2).YLim = [0,1];

Ax(3) = nexttile( tiles );
plot( Coverage_individual );
xlabel('Nr Sensors');
ylabel('Coverage');
grid on;
AX(3).XLim = [0,6];
Ax(3).YLim = [0,1];

saveas(figPCAPCs, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\05_Data
Preparation\figures\PCA_individual_sesor.jpg');
```



Define the number of sensors and create a table, which displays the coverage per sensor.

```
NrSensors = (1:6)';
coverageT = table(NrSensors, Coverage_individual*100);
```

Coverage when dropping Sensors

Now the remaining coverage when any two sensors are dropped is determined.

```
NrSensors = 6;
NrUsed = 4;
```

Here we create a matrix C, containing all the possible combinations of choosing 4 out of the six sensors.

```
C = nchoosek(1:NrSensors,NrUsed);
[mC, nC] = size( C );

Coverage = zeros( NrUsed , mC);

for k=1:mC
  inds = C(k,:);
```

```
  [Uc,Sc,Vc] = svd( PCs(:,inds), 0 );

  sc = diag(Sc);
  portionc = sc / sum(sc) ;
  Coverage_c(:,k) = cumsum( portionc );
end

SelCov = [C'; Coverage_c];
coverageT = table( SelCov );
%table2latex( coverageT );
```

Correlation Analysis and QR Decomposition

Here we can now check how strong the correlation between individual sensors is for each combination.

We start by normalizing the data.

```
[m,n] = size( PCs ) ;
for k=1:n
  PCs(:,k) = PCs(:,k) / norm( PCs(:,k ));
end
```

We then pick a specific combination from C, our combinations matrix, and extract the principal components from the PCs matrix in accordance with that combination from C and then calculate the correlation between the sensor selection C and the PCs matrix.

```
K = PCs(:, C(1,:) );
Co = K' * PCs
```

```
Co = 4×6
    1.0000   -0.9720    0.9648   -0.9955    0.9822   -0.9759
   -0.9720    1.0000   -0.8767    0.9623   -0.9714    0.9229
    0.9648   -0.8767    1.0000   -0.9677    0.9312   -0.9702
   -0.9955    0.9623   -0.9677    1.0000   -0.9840    0.9823
```

We can also generate a general complete correlation matrix by multplying the inverse of PCs with PCs.

```
Call = PCs' * PCs
```

```
Call = 6×6
    1.0000   -0.9720    0.9648   -0.9955    0.9822   -0.9759
   -0.9720    1.0000   -0.8767    0.9623   -0.9714    0.9229
    0.9648   -0.8767    1.0000   -0.9677    0.9312   -0.9702
   -0.9955    0.9623   -0.9677    1.0000   -0.9840    0.9823
    0.9822   -0.9714    0.9312   -0.9840    1.0000   -0.9350
   -0.9759    0.9229   -0.9702    0.9823   -0.9350    1.0000
```

Next we can apply QR decomposition to the total PCs matrix, which yields the orthogonal matrix Q and the upper triangular matrix R. Of this matrix R we need the norm value for comparison. Additionally the condition value is computed to check the rank deficiancy of the matrix.

```
[Qt,Rt] = qr( PCs, 0 );
Rt;
```

```
normRt = norm(Rt);
condRt = cond(Rt);
```

We can then apply QR decomposition to each of the combinations in C. A comparison between the norm of the resulting matrix R and the matrix from the analysis of the QR decomposition of the total matrix Rt gives us a measure to quantify and compare the different combinations.

```
for k=1:mC
  inds = C(k,:);

  [Q,R] = qr( PCs(:,inds), 0 );

  normR = norm(R)/norm(Rt);
  NormR(k) = normR;

  condR = cond(R)/cond(Rt);
  CondR(k) = condR;

  Combinations = zeros(mC, 1);

  theta = subspace(Qt,Q);


end
norms = table(C, 100*NormR');
```

Canonical Correlation Analysis

```
X = PCs;
for i = 1:mC
    Y = PCs(:, C(i,:) );
    [A,B,r,U,V,stats] = canoncorr(X,Y);
    cor = corr(U,V);
end
```

```
function cleanedData = replaceNaNWithZero(data)
    % Input:
    %   data: Input dataset with NaN entries
    % Output:
    %   cleanedData: Dataset with NaN entries replaced by 0

    % Find the indices of NaN entries in the data
    nanIndices = isnan(data);

    % Replace NaN entries with 0
    data(nanIndices) = 0;
```

```
    % Return the cleaned data
    cleanedData = data;
end
```

Appendix B

Appendix B: Machine Learning Application Code

Processing Time Series Data using Machine Learning

Author: Elliot Lang

E-Mail: elliot.lang@stud.unileoben.ac.at

This file aims to apply different machine learning techniques for classification to time-series sensor data and compare the outcomes of each algorithm.

```
clear all;
close all;
setLiveScriptDir;
```

Set the file directory and create the empty matrices used for collecting the results.

```
fileDir = 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master Thesis\PCA
Data_6_sensors';
files = dir(fileDir);
files = files(find(~[files.isdir]==true));
nrFiles = numel(files);
```

Define an empty hypermatrix for each algorithm

```
hyp_BT = zeros(4,4,nrFiles);
hyp_DA = zeros(4,4,nrFiles);
hyp_KNN = zeros(4,4,nrFiles);
hyp_NN = zeros(4,4,nrFiles);
hyp_En = zeros(4,4,nrFiles);
hyp_NB = zeros(4,4,nrFiles);
hyp_ECOC = zeros(4,4,nrFiles);
```

Create an empty accuracy vector for each algorithm

```
acc_BT = zeros(nrFiles, 1);
acc_DA = zeros(nrFiles, 1);
acc_KNN = zeros(nrFiles, 1);
acc_NN = zeros(nrFiles, 1);
acc_En = zeros(nrFiles, 1);
acc_NB = zeros(nrFiles, 1);
acc_ECOC = zeros(nrFiles, 1);
```

Create an empty time vector for each algorithm

```
time_BT = zeros(nrFiles,1);
time_DA = zeros(nrFiles,1);
time_KNN = zeros(nrFiles,1);
time_NN = zeros(nrFiles,1);
time_En = zeros(nrFiles,1);
time_NB = zeros(nrFiles, 1);
```

```
time_ECOC = zeros(nrFiles, 1);
```

Set the dimensions as variables.

```
[n,m,p] = size(hyp_BT);
```

Binary Decision Tree

The first method to be applied will be a binary decision tree.

This will be evaluated by cycling through all data batches.

```
for i = 1:numel(files)
    %Load file i from the folder
    fileName = fullfile(files(i).folder, files(i).name);
    load(fileName);
    %Extract the state vector
    state = stateTT.State;
    %Extract the Principal COmponent Data
    D = PCsTT{:,:};
    %Create a random 90/10 Partition for Training and Test Data
    rng('default');
    Partition_States = cvpartition(state, 'Holdout', 0.10);

    %Seperate the training and testing Ids
    trainingIds = training(Partition_States);
    DTrain = D(trainingIds, :);
    stateTrain = state(trainingIds);

    testIds = test(Partition_States);
    DTest = D(testIds, :);
    stateTest = state(testIds);
```

Begin measuring the time this algorithm will take

```
    tBT = tic;
```

train the decsion tree classifier

```
    trainedClassifier = fitctree(DTrain, stateTrain, 'OptimizeHyperparameters',
'auto');
```

end the time measurement

```
    timeBT = toc(tBT);
```

optimize and prune the tree

```
    [~, ~, ~, bestLevel] = cvLoss(trainedClassifier, 'SubTrees', 'All', 'TreeSize',
'min');
    BT = prune(trainedClassifier, 'Level', bestLevel);
```

use the tree to predict the test states

```
    TestModel_BT = predict(BT, DTest);
```

measure the accuracy

```
    accuracy_BT = sum(stateTest == TestModel_BT)/length(stateTest);
   % if i == 4
   %     conFig = figureGen(9,12);
   %     conCh = confusionchart(stateTest, TestModel_BT, 'Normalization', 'row-
normalized');
   %     title('Confusion Matrix');
   %     saveas(conCh, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\conMat.jpg');
   % end
```

Create a confusion matrix

```
    [C_BT, order] = confusionmat(stateTest, TestModel_BT);
    %titleStr_BT = strrep([fName,' Binary Tree'],'_','-');
    % title(titleStr_BT);
```

save the confusion matrix as one layer of the hypermatrix and the accuracy and measured time in vector form

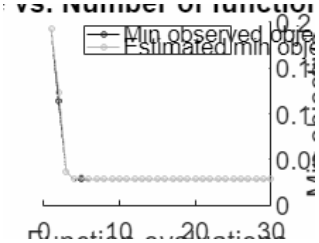
```
    hyp_BT(:,:,i) = C_BT;
    acc_BT(i) = accuracy_BT;
    time_BT(i) = timeBT/length(D);
end
```

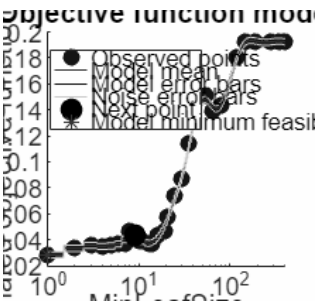
```
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
1	Best	0.19259	1.9944	0.19259	0.19259	187
2	Best	0.11358	0.24342	0.11358	0.12363	36
3	Best	0.037037	0.09817	0.037037	0.037048	7
4	Best	0.028395	0.075565	0.028395	0.028391	1
5	Accept	0.033333	0.17852	0.028395	0.030478	2
6	Accept	0.028395	0.051327	0.028395	0.028413	1
7	Accept	0.028395	0.042342	0.028395	0.028406	1
8	Accept	0.028395	0.040589	0.028395	0.028402	1
9	Accept	0.034568	0.04788	0.028395	0.028399	4
10	Accept	0.037037	0.040345	0.028395	0.028398	13
11	Accept	0.046914	0.042511	0.028395	0.028397	19
12	Accept	0.19259	0.044174	0.028395	0.028398	405
13	Accept	0.14321	0.047025	0.028395	0.028397	80
14	Accept	0.035802	0.05159	0.028395	0.028397	3
15	Accept	0.039506	0.048885	0.028395	0.028397	10
16	Accept	0.035802	0.042121	0.028395	0.028396	5
17	Accept	0.074074	0.038295	0.028395	0.028396	25

18	Accept	0.18025	0.040552	0.028395	0.028397	120
19	Accept	0.15062	0.04103	0.028395	0.028397	54
20	Accept	0.19259	0.042295	0.028395	0.028397	285

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
21	Accept	0.04321	0.041982	0.028395	0.028397	16
22	Accept	0.037037	0.04427	0.028395	0.028397	6
23	Accept	0.046914	0.041032	0.028395	0.028431	8
24	Accept	0.039506	0.042992	0.028395	0.028423	11
25	Accept	0.19259	0.040515	0.028395	0.028419	150
26	Accept	0.08642	0.037479	0.028395	0.028427	30
27	Accept	0.037037	0.038295	0.028395	0.028423	14
28	Accept	0.19259	0.034025	0.028395	0.028421	345
29	Accept	0.13951	0.046529	0.028395	0.028423	66
30	Accept	0.058025	0.03924	0.028395	0.02842	21





Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 25.8809 seconds
Total objective function evaluation time: 3.6574

Best observed feasible point:

MinLeafSize
1

Observed objective function value = 0.028395
Estimated objective function value = 0.02842
Function evaluation time = 0.075565

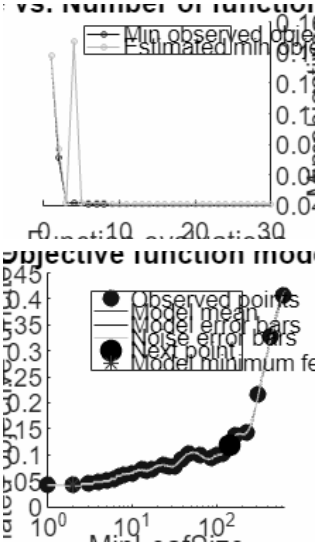
Best estimated feasible point (according to models):

MinLeafSize
1

Estimated objective function value = 0.02842
Estimated function evaluation time = 0.056303

Warning: One or more of the unique class values in GROUP is not present in one or more folds. For classification problems, either remove this class from the data or use N instead of GROUP to obtain nonstratified partitions. For regression problems with continuous response, use N.

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
1	Best	0.13812	0.064582	0.13812	0.13812	168
2	Best	0.070988	0.046815	0.070988	0.076502	13
3	Best	0.041667	0.046301	0.041667	0.041672	1
4	Accept	0.40586	0.042112	0.041667	0.14747	609
5	Accept	0.041667	0.045218	0.041667	0.041674	1
6	Best	0.040895	0.043003	0.040895	0.04167	2
7	Accept	0.10262	0.046053	0.040895	0.041671	47
8	Accept	0.049383	0.049467	0.040895	0.04167	5
9	Accept	0.043981	0.042618	0.040895	0.040907	3
10	Accept	0.040895	0.045617	0.040895	0.040901	2
11	Accept	0.040895	0.042076	0.040895	0.040899	2
12	Accept	0.040895	0.042925	0.040895	0.040898	2
13	Accept	0.092593	0.046722	0.040895	0.040895	87
14	Accept	0.080247	0.041781	0.040895	0.0409	23
15	Accept	0.0625	0.044603	0.040895	0.040904	8
16	Accept	0.21605	0.041071	0.040895	0.040898	314
17	Accept	0.04784	0.042289	0.040895	0.040887	4
18	Accept	0.078704	0.054036	0.040895	0.040891	32
19	Accept	0.10108	0.046616	0.040895	0.040891	118
20	Accept	0.072531	0.050491	0.040895	0.040892	17
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
21	Accept	0.053241	0.045651	0.040895	0.040892	6
22	Accept	0.099537	0.043925	0.040895	0.040892	64
23	Accept	0.064815	0.10413	0.040895	0.040892	10
24	Accept	0.14352	0.038637	0.040895	0.040894	231
25	Accept	0.32716	0.041454	0.040895	0.040894	443
26	Accept	0.090278	0.04657	0.040895	0.040894	38
27	Accept	0.059414	0.05815	0.040895	0.040894	7
28	Accept	0.077932	0.041962	0.040895	0.040894	27
29	Accept	0.099537	0.040031	0.040895	0.040894	101
30	Accept	0.068673	0.042788	0.040895	0.040894	15

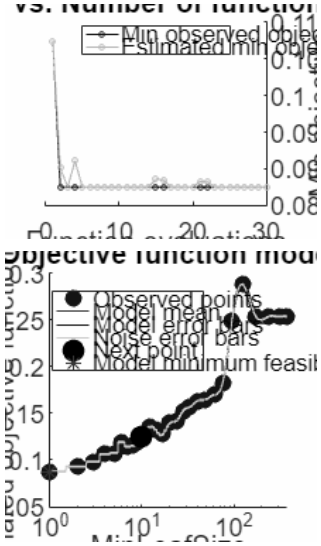


Optimization completed.
MaxObjectiveEvaluations of 30 reached.

```
Total function evaluations: 30
Total elapsed time: 16.5871 seconds
Total objective function evaluation time: 1.4277

Best observed feasible point:
    MinLeafSize
    ___________

         2

Observed objective function value = 0.040895
Estimated objective function value = 0.040894
Function evaluation time = 0.043003

Best estimated feasible point (according to models):
    MinLeafSize
    ___________

         2

Estimated objective function value = 0.040894
Estimated function evaluation time = 0.04663
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
1	Best	0.10743	0.072714	0.10743	0.10743	5
2	Best	0.087533	0.049609	0.087533	0.090084	1
3	Accept	0.1817	0.046566	0.087533	0.087538	75
4	Accept	0.25332	0.036304	0.087533	0.091072	236
5	Accept	0.087533	0.046159	0.087533	0.087537	1
6	Accept	0.092838	0.046642	0.087533	0.087526	2
7	Accept	0.087533	0.04728	0.087533	0.087529	1
8	Accept	0.087533	0.042757	0.087533	0.08753	1
9	Accept	0.12732	0.041318	0.087533	0.087525	17
10	Accept	0.25332	0.034242	0.087533	0.087535	377
11	Accept	0.1565	0.049899	0.087533	0.087535	33
12	Accept	0.11936	0.039458	0.087533	0.087536	9
13	Accept	0.098143	0.040771	0.087533	0.087536	3
14	Accept	0.1061	0.040555	0.087533	0.087537	4
15	Accept	0.2878	0.036582	0.087533	0.088592	124
16	Accept	0.092838	0.056465	0.087533	0.088459	2
17	Accept	0.16446	0.040808	0.087533	0.087535	49
18	Accept	0.13528	0.040713	0.087533	0.087535	12
19	Accept	0.14191	0.038673	0.087533	0.087535	24
20	Accept	0.11406	0.046111	0.087533	0.087535	7

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
21	Accept	0.25332	0.036117	0.087533	0.088267	174
22	Accept	0.25332	0.039434	0.087533	0.088217	310
23	Accept	0.16976	0.041636	0.087533	0.087535	61
24	Accept	0.13926	0.042711	0.087533	0.087535	20
25	Accept	0.24801	0.038653	0.087533	0.087535	95
26	Accept	0.11936	0.041521	0.087533	0.087535	6
27	Accept	0.16313	0.043252	0.087533	0.087535	40
28	Accept	0.13263	0.041119	0.087533	0.087535	14
29	Accept	0.15252	0.048086	0.087533	0.087535	28
30	Accept	0.11538	0.040195	0.087533	0.087535	8



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 16.3456 seconds
Total objective function evaluation time: 1.3063

Best observed feasible point:

MinLeafSize
1

Observed objective function value = 0.087533
Estimated objective function value = 0.087535
Function evaluation time = 0.049609

Best estimated feasible point (according to models):

MinLeafSize
1

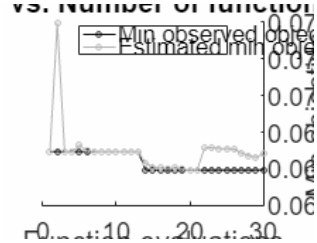
Estimated objective function value = 0.087535
Estimated function evaluation time = 0.046091

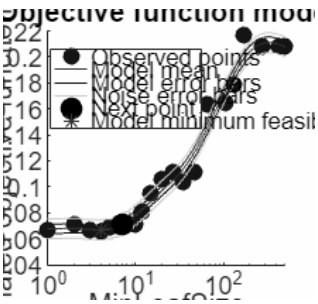
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For classification problems, either remove this class from the data or use N instead of GROUP to obtain nonstratified partitions. For regression problems with continuous response, use N.

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
1	Best	0.066933	0.069441	0.066933	0.066933	1
2	Accept	0.20779	0.036403	0.066933	0.073951	421
3	Accept	0.068931	0.041912	0.066933	0.066936	5
4	Accept	0.16384	0.043256	0.066933	0.066928	66
5	Accept	0.070929	0.042285	0.066933	0.067298	2
6	Accept	0.070929	0.041829	0.066933	0.066971	2
7	Accept	0.066933	0.042608	0.066933	0.06695	1
8	Accept	0.071928	0.041594	0.066933	0.066949	8
9	Accept	0.066933	0.053995	0.066933	0.066943	1
10	Accept	0.066933	0.044695	0.066933	0.06694	1
11	Accept	0.10589	0.042043	0.066933	0.066939	20
12	Accept	0.066933	0.04326	0.066933	0.066942	3

13	Accept	0.066933	0.042461	0.066933	0.066941	3
14	Best	0.065934	0.047021	0.065934	0.066295	4
15	Accept	0.065934	0.040919	0.065934	0.06611	4
16	Accept	0.065934	0.04799	0.065934	0.066045	4
17	Accept	0.065934	0.045737	0.065934	0.066013	4
18	Accept	0.21678	0.038498	0.065934	0.066033	172
19	Accept	0.1039	0.043842	0.065934	0.065958	35
20	Accept	0.080919	0.041235	0.065934	0.065948	12

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
21	Accept	0.20779	0.036667	0.065934	0.065952	276
22	Accept	0.16484	0.038701	0.065934	0.067138	103
23	Accept	0.06993	0.043447	0.065934	0.06715	6
24	Accept	0.066933	0.041115	0.065934	0.067106	3
25	Accept	0.20779	0.034204	0.065934	0.0671	500
26	Accept	0.11089	0.037901	0.065934	0.06708	27
27	Accept	0.11089	0.041871	0.065934	0.066866	47
28	Accept	0.094905	0.041052	0.065934	0.066721	15
29	Accept	0.070929	0.038164	0.065934	0.066629	10
30	Accept	0.17882	0.040101	0.065934	0.066806	131





Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 15.064 seconds
Total objective function evaluation time: 1.2842

Best observed feasible point:

MinLeafSize
4

Observed objective function value = 0.065934
Estimated objective function value = 0.066806
Function evaluation time = 0.047021

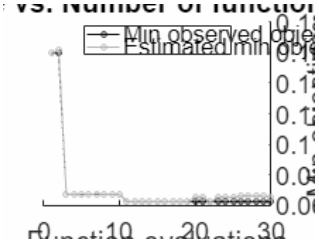
Best estimated feasible point (according to models):

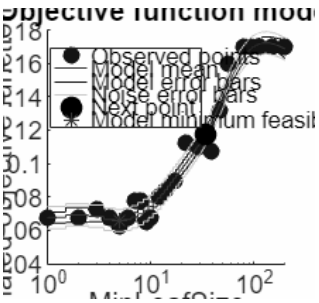
MinLeafSize
4

```
Estimated objective function value = 0.066806
Estimated function evaluation time = 0.043477
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
1	Best	0.1596	0.063424	0.1596	0.1596	56
2	Accept	0.16958	0.054618	0.1596	0.16178	198
3	Best	0.067332	0.038827	0.067332	0.067342	4
4	Accept	0.067332	0.040805	0.067332	0.067331	1
5	Accept	0.067332	0.040644	0.067332	0.067126	2
6	Accept	0.072319	0.037261	0.067332	0.067284	3
7	Accept	0.067332	0.037401	0.067332	0.067495	1
8	Accept	0.067332	0.039734	0.067332	0.067331	1
9	Accept	0.077307	0.036087	0.067332	0.067335	8
10	Accept	0.067332	0.036978	0.067332	0.067331	1
11	Best	0.062344	0.03645	0.062344	0.062422	5
12	Accept	0.062344	0.035843	0.062344	0.062381	5
13	Accept	0.062344	0.036484	0.062344	0.062368	5
14	Accept	0.062344	0.035904	0.062344	0.062361	5
15	Accept	0.067332	0.036477	0.062344	0.062382	6
16	Accept	0.089776	0.038363	0.062344	0.06238	17
17	Accept	0.10973	0.036667	0.062344	0.062384	29
18	Accept	0.16958	0.03352	0.062344	0.062399	112
19	Accept	0.0798	0.039631	0.062344	0.062387	12
20	Accept	0.10723	0.037259	0.062344	0.065448	39

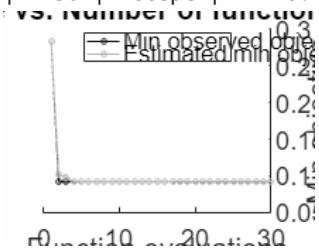
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
21	Accept	0.16958	0.035784	0.062344	0.065307	150
22	Accept	0.16958	0.03417	0.062344	0.062353	79
23	Accept	0.11222	0.03569	0.062344	0.065313	22
24	Accept	0.067332	0.035516	0.062344	0.065195	10
25	Accept	0.084788	0.040248	0.062344	0.065142	14
26	Accept	0.077307	0.035594	0.062344	0.066061	7
27	Accept	0.072319	0.036472	0.062344	0.066371	3
28	Accept	0.064838	0.035227	0.062344	0.06608	9
29	Accept	0.13217	0.036846	0.062344	0.065971	47
30	Accept	0.16958	0.035283	0.062344	0.065819	94

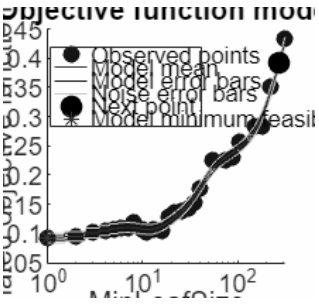




```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 14.8024 seconds
Total objective function evaluation time: 1.1532

Best observed feasible point:
    MinLeafSize
    ___________

        5

Observed objective function value = 0.062344
Estimated objective function value = 0.065819
Function evaluation time = 0.03645

Best estimated feasible point (according to models):
    MinLeafSize
    ___________

        5

Estimated objective function value = 0.065819
Estimated function evaluation time = 0.03809
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
|=====================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |  MinLeafSize |
|      | result |             | runtime     | (observed)  | (estim.)    |              |
|=====================================================================================|
|    1 | Best   |     0.28368 |    0.057932 |     0.28368 |     0.28368 |          150 |
|    2 | Best   |    0.091918 |    0.042894 |    0.091918 |      0.1016 |            1 |
|    3 | Accept |     0.10618 |    0.046544 |    0.091918 |    0.098179 |           13 |
|    4 | Accept |     0.10143 |    0.040518 |    0.091918 |     0.09193 |            3 |
|    5 | Accept |     0.10935 |    0.040417 |    0.091918 |    0.091921 |            7 |
|    6 | Accept |    0.095087 |    0.042804 |    0.091918 |    0.091916 |            2 |
|    7 | Accept |    0.091918 |    0.052379 |    0.091918 |    0.091917 |            1 |
|    8 | Accept |    0.091918 |    0.052886 |    0.091918 |    0.091917 |            1 |
|    9 | Accept |    0.091918 |    0.041309 |    0.091918 |    0.091917 |            1 |
|   10 | Accept |     0.17591 |    0.038389 |    0.091918 |    0.091918 |           40 |
|   11 | Accept |     0.43265 |    0.034213 |    0.091918 |    0.091919 |          315 |
|   12 | Accept |     0.13629 |    0.042077 |    0.091918 |     0.09192 |           22 |
|   13 | Accept |     0.22504 |    0.037295 |    0.091918 |     0.09192 |           75 |
|   14 | Accept |     0.10618 |    0.040797 |    0.091918 |    0.091917 |            5 |
|   15 | Accept |     0.10618 |    0.040728 |    0.091918 |    0.091917 |           10 |
|   16 | Accept |      0.1046 |    0.047206 |    0.091918 |    0.091917 |            4 |
|   17 | Accept |      0.1046 |    0.057899 |    0.091918 |    0.091919 |           16 |
|   18 | Accept |     0.22504 |    0.041235 |    0.091918 |    0.091967 |           54 |
|   19 | Accept |    0.095087 |    0.040511 |    0.091918 |    0.091962 |            2 |
|   20 | Accept |     0.35024 |    0.034712 |    0.091918 |    0.091951 |          225 |
|=====================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |  MinLeafSize |
|      | result |             | runtime     | (observed)  | (estim.)    |              |
|=====================================================================================|
|   21 | Accept |     0.25674 |    0.035605 |    0.091918 |    0.091946 |          106 |
|   22 | Accept |     0.14422 |     0.04006 |    0.091918 |    0.091953 |           30 |
|   23 | Accept |     0.11886 |    0.040122 |    0.091918 |    0.091974 |            8 |
|   24 | Accept |     0.10935 |    0.039462 |    0.091918 |    0.091968 |            6 |
|   25 | Accept |     0.12837 |    0.038659 |    0.091918 |    0.091971 |           19 |
|   26 | Accept |     0.10301 |    0.039823 |    0.091918 |     0.09197 |           11 |
|   27 | Accept |     0.28368 |    0.037054 |    0.091918 |    0.092005 |          183 |
|   28 | Accept |     0.13788 |    0.038697 |    0.091918 |       0.092 |           26 |
```

```
|    29 | Accept |      0.22979 |     0.036142 |     0.091918 |     0.092003 |           89 |
|    30 | Accept |      0.15372 |     0.039822 |     0.091918 |     0.092009 |           35 |
```

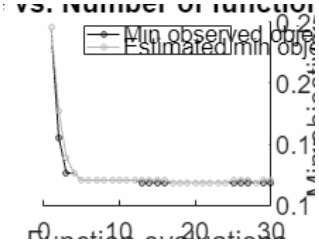


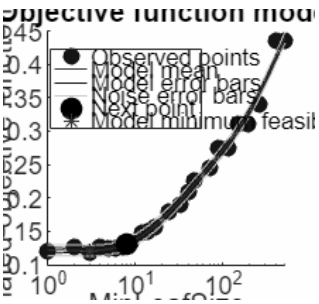


```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 16.7892 seconds
Total objective function evaluation time: 1.2582

Best observed feasible point:
    MinLeafSize
    ___________

        1

Observed objective function value = 0.091918
Estimated objective function value = 0.092009
Function evaluation time = 0.042894

Best estimated feasible point (according to models):
    MinLeafSize
    ___________

        1

Estimated objective function value = 0.092009
Estimated function evaluation time = 0.042357

|=====================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |  MinLeafSize |
|      | result |             | runtime     | (observed)  | (estim.)    |              |
|=====================================================================================|
|    1 | Best   |     0.24537 |    0.071515 |     0.24537 |     0.24537 |           70 |
|    2 | Best   |     0.15579 |    0.043157 |     0.15579 |     0.17758 |           17 |
|    3 | Best   |     0.12658 |    0.049875 |     0.12658 |      0.1394 |            2 |
|    4 | Accept |     0.43427 |    0.039494 |     0.12658 |     0.12659 |          405 |
|    5 | Best   |     0.12074 |    0.052899 |     0.12074 |     0.12081 |            1 |
|    6 | Accept |     0.12074 |    0.051064 |     0.12074 |     0.12077 |            1 |
|    7 | Accept |     0.12074 |    0.059655 |     0.12074 |     0.12076 |            1 |
|    8 | Accept |     0.12074 |    0.050578 |     0.12074 |     0.12075 |            1 |
|    9 | Accept |     0.12366 |    0.052265 |     0.12074 |     0.12076 |            6 |
|   10 | Accept |     0.12561 |    0.047452 |     0.12074 |     0.12075 |            4 |
|   11 | Accept |      0.1334 |    0.052198 |     0.12074 |     0.12075 |            9 |
|   12 | Accept |     0.19085 |    0.047018 |     0.12074 |     0.12075 |           33 |
```

13	Best	0.11879	0.049415	0.11879	0.121	3
14	Accept	0.11879	0.044853	0.11879	0.12106	3
15	Accept	0.11879	0.046139	0.11879	0.12098	3
16	Accept	0.11879	0.049097	0.11879	0.12061	3
17	Accept	0.31061	0.040861	0.11879	0.11888	191
18	Accept	0.27556	0.039171	0.11879	0.1189	114
19	Accept	0.14703	0.042722	0.11879	0.11889	12
20	Accept	0.18014	0.045612	0.11879	0.11884	24

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
21	Accept	0.43427	0.03441	0.11879	0.11884	513
22	Accept	0.22493	0.040047	0.11879	0.11883	48
23	Accept	0.12366	0.043699	0.11879	0.11884	5
24	Accept	0.12561	0.04127	0.11879	0.11885	7
25	Accept	0.3408	0.04395	0.11879	0.12049	265
26	Accept	0.12658	0.044957	0.11879	0.12116	2
27	Accept	0.14995	0.041931	0.11879	0.12116	14
28	Accept	0.27556	0.038417	0.11879	0.11881	89
29	Accept	0.31061	0.039522	0.11879	0.12087	148
30	Accept	0.20935	0.039974	0.11879	0.12092	40





Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 16.3431 seconds
Total objective function evaluation time: 1.3832

Best observed feasible point:

MinLeafSize
3

Observed objective function value = 0.11879
Estimated objective function value = 0.12092
Function evaluation time = 0.049415

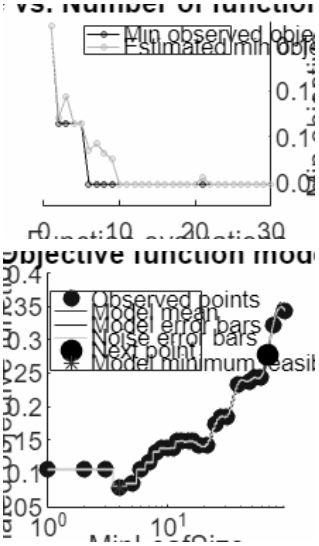
Best estimated feasible point (according to models):

MinLeafSize
3

```
Estimated objective function value = 0.12092
Estimated function evaluation time = 0.047901
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
1	Best	0.14815	0.058929	0.14815	0.14815	15
2	Best	0.10582	0.038212	0.10582	0.10829	1
3	Accept	0.32275	0.032263	0.10582	0.11773	75
4	Accept	0.10582	0.03491	0.10582	0.10582	3
5	Accept	0.10582	0.033152	0.10582	0.10576	6
6	Best	0.079365	0.03405	0.079365	0.09424	4
7	Accept	0.10582	0.034006	0.079365	0.096907	3
8	Accept	0.079365	0.035755	0.079365	0.092943	4
9	Accept	0.079365	0.035732	0.079365	0.090245	4
10	Accept	0.18519	0.032944	0.079365	0.079378	31
11	Accept	0.084656	0.035281	0.079365	0.079375	5
12	Accept	0.079365	0.042704	0.079365	0.079372	4
13	Accept	0.10582	0.038093	0.079365	0.07938	2
14	Accept	0.13757	0.037204	0.079365	0.079379	10
15	Accept	0.2381	0.033742	0.079365	0.079384	47
16	Accept	0.14286	0.038615	0.079365	0.079377	21
17	Accept	0.34392	0.035691	0.079365	0.079387	94
18	Accept	0.13228	0.044311	0.079365	0.079392	8
19	Accept	0.14815	0.033912	0.079365	0.07939	12
20	Accept	0.1746	0.034258	0.079365	0.079373	25

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize
21	Accept	0.2328	0.033098	0.079365	0.082078	38
22	Accept	0.24339	0.033019	0.079365	0.079368	59
23	Accept	0.14286	0.034193	0.079365	0.079368	18
24	Accept	0.1164	0.034074	0.079365	0.079366	7
25	Accept	0.13757	0.035514	0.079365	0.079366	9
26	Accept	0.14815	0.035077	0.079365	0.079366	13
27	Accept	0.18519	0.034679	0.079365	0.079366	28
28	Accept	0.24339	0.031725	0.079365	0.079366	53
29	Accept	0.14815	0.033987	0.079365	0.079366	16
30	Accept	0.13757	0.04023	0.079365	0.079366	11



```
_________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 15.8995 seconds
Total objective function evaluation time: 1.0894

Best observed feasible point:
    MinLeafSize
    ___________

        4

Observed objective function value = 0.079365
Estimated objective function value = 0.079366
Function evaluation time = 0.03405

Best estimated feasible point (according to models):
    MinLeafSize
    ___________

        4

Estimated objective function value = 0.079366
Estimated function evaluation time = 0.036027
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Decision Tree visualisation

The results of every layer of the hypermatrix are summed up and visualised in a heatmap. Additionally the mean accuracy and time taken is calculated and used as a comparison metric between methods.

```
BT = zeros(m,n);
for j = 1:n
    for k = 1:m
        BT(j,k) = sum(hyp_BT(j,k,:));
    end
end
figBT = figureGen(7,10);
heat_BT6 = heatmap(BT, "XDisplayLabels", [0,1,2,3], "YDisplayLabels", [0,1,2,3],
"ColorMethod", "mean", "ColorLimits", [0,100])
```

```
heat_BT6 =
  HeatmapChart with properties:

       XData: {4×1 cell}
       YData: {4×1 cell}
    ColorData: [4×4 double]

  Show all properties
```

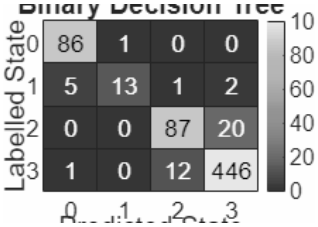
```
heat_BT6.Colormap = parula(64);
xlabel("Predicted State");
ylabel("Labelled State");
average_acc_BT = median(acc_BT)
```

```
average_acc_BT = 0.9469
```

```
avergage_time_BT = median(time_BT)
```

```
avergage_time_BT = 0.0228
```

```
heat_BT6.Title = "Binary Decision Tree";
saveas(heat_BT6, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\6_BT.jpg');
```



Discriminant Analysis

The next method to be applied will be linear Discriminant Analysis.

```
for i = 1:numel(files)
    %Load file i from the folder
    fileName = fullfile(files(i).folder, files(i).name);
    load(fileName);
    %Extract the state vector
    state = stateTT.State;
    %Extract the Principal COmponent Data
    D = PCsTT{:,:};
    %Create a random 90/10 Partition for Training and Test Data
    rng('default');
    Partition_States = cvpartition(state, 'Holdout', 0.10);

    %Seperate the training and testing Ids
    trainingIds = training(Partition_States);
    DTrain = D(trainingIds, :);
    stateTrain = state(trainingIds);

    testIds = test(Partition_States);
    DTest = D(testIds, :);
    stateTest = state(testIds);
```

Begin measuring the time this algorithm will take

```
     tDA = tic;
```

train the decsion tree classifier

```
    classifierDA = fitcdiscr(DTrain, stateTrain, 'OptimizeHyperparameters','auto');
```

end the time measurement

```
    timeDA = toc(tDA);
```

use the tree to predict the test states

```
    TestModel_DA = predict(classifierDA, DTest);
```

measure the accuracy

```
    accuracy_DA = sum(stateTest == TestModel_DA)/length(stateTest);
    %figure;
    %confusionchart(stateTest, TestModel_BT, 'Normalization', 'row-normalized');
```

Create a confusion matrix

```
    [C_DA, order] = confusionmat(stateTest, TestModel_DA);
    %titleStr_BT = strrep([fName,' Binary Tree'],'_','-');
    % title(titleStr_BT);
```

save the confusion matrix as one layer of the hypermatrix and the accuracy and measured time in vector form

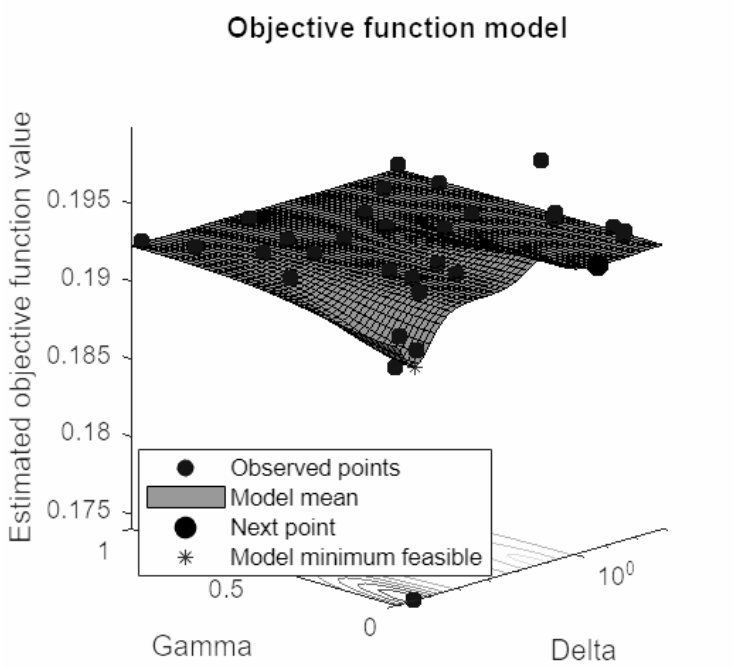
```
    hyp_DA(:,:,i) = C_DA;
    acc_DA(i) = accuracy_DA;
    time_DA(i) = timeDA/length(D);
end
```

```
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.19259	0.52463	0.19259	0.19259	0.00037316	0.31422
2	Accept	0.19259	0.11454	0.19259	0.19259	1.4362e-06	0.98178
3	Accept	0.19259	0.093996	0.19259	0.19259	23.476	0.87075
4	Accept	0.19259	0.15437	0.19259	0.19259	446.49	0.10992
5	Accept	0.19259	0.053893	0.19259	0.19259	0.0053024	0.61306
6	Accept	0.19259	0.046354	0.19259	0.19259	3.0904e-06	0.4555
7	Accept	0.19259	0.047203	0.19259	0.19259	983.22	0.99483
8	Accept	0.19259	0.048573	0.19259	0.19259	0.14184	0.61255
9	Accept	0.19259	0.045562	0.19259	0.19259	0.0065701	0.27321
10	Accept	0.2	0.058266	0.19259	0.19259	0.072013	0.00014954
11	Accept	0.19259	0.048925	0.19259	0.19259	0.0036119	0.1722
12	Accept	0.19259	0.043029	0.19259	0.19259	0.42724	0.74356
13	Accept	0.19259	0.053537	0.19259	0.19259	21.556	0.53186
14	Accept	0.19259	0.049238	0.19259	0.19259	449.13	0.80618
15	Accept	0.19259	0.045072	0.19259	0.19259	3.6769e-05	0.67684
16	Accept	0.19259	0.047843	0.19259	0.19259	945.53	0.18126
17	Accept	0.19259	0.046733	0.19259	0.19259	739.74	0.38971
18	Accept	0.19259	0.046592	0.19259	0.19259	844.8	0.13767
19	Accept	0.19259	0.051116	0.19259	0.19259	0.002001	0.92174
20	Accept	0.19259	0.040225	0.19259	0.19259	0.00036112	0.22734

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma

21	Accept	0.19259	0.043047	0.19259	0.19259	0.00022188	0.57158
22	Accept	0.19259	0.047855	0.19259	0.19258	1.0731	0.49186
23	Accept	0.19259	0.038572	0.19259	0.19258	313.06	0.35337
24	Accept	0.19259	0.051439	0.19259	0.19258	0.00047227	0.71026
25	Accept	0.19259	0.042235	0.19259	0.19258	5.2843e-06	0.83813
26	Accept	0.19259	0.040998	0.19259	0.19259	6.8228e-05	0.12291
27	Best	0.19136	0.052494	0.19136	0.19136	1.1762e-06	0.0012063
28	Best	0.17407	0.047215	0.17407	0.17408	3.9984e-06	0.0077154
29	Accept	0.19012	0.05524	0.17407	0.18959	4.7569e-06	0.0015563
30	Accept	0.18889	0.052232	0.17407	0.18906	2.6674e-06	0.055492



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 20.8123 seconds
Total objective function evaluation time: 2.131

Best observed feasible point:

Delta	Gamma
3.9984e-06	0.0077154

Observed objective function value = 0.17407
Estimated objective function value = 0.18906
Function evaluation time = 0.047215

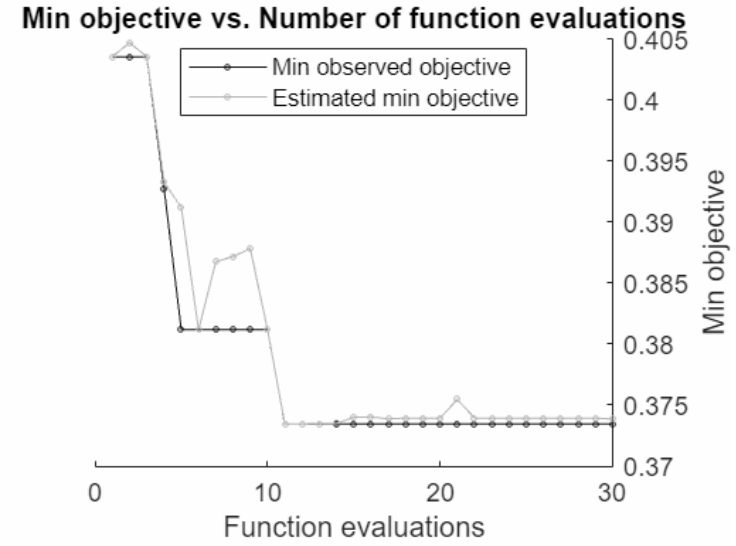
Best estimated feasible point (according to models):

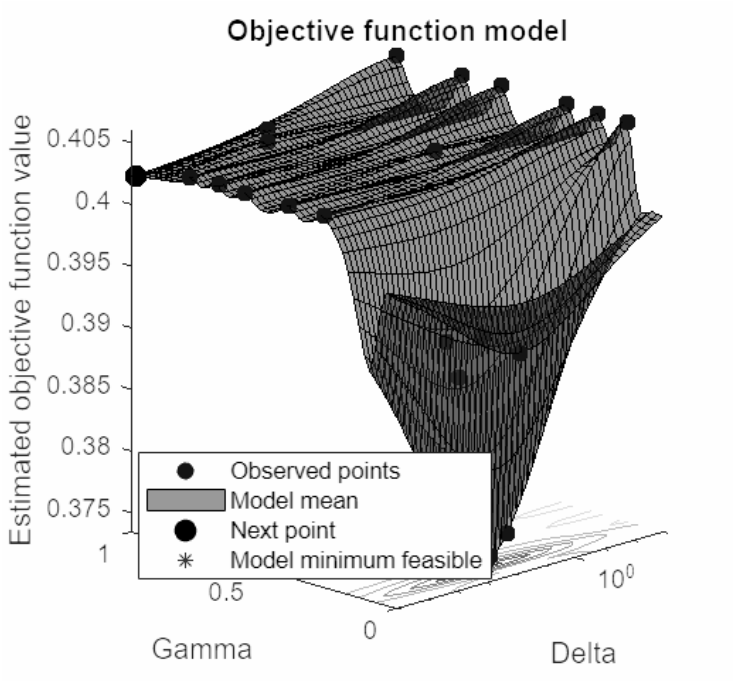
Delta	Gamma

```
    3.9984e-06    0.0077154

Estimated objective function value = 0.18906
Estimated function evaluation time = 0.056475
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.40355	0.053845	0.40355	0.40355	0.55597	0.49649
2	Accept	0.40586	0.045284	0.40355	0.40471	955.78	0.13282
3	Accept	0.40355	0.044171	0.40355	0.40355	0.013686	0.94477
4	Best	0.39275	0.061214	0.39275	0.39328	0.0024534	0.059148
5	Best	0.38117	0.054997	0.38117	0.39118	0.0026356	0.082952
6	Accept	0.38194	0.049647	0.38117	0.38118	0.0029045	0.12144
7	Accept	0.40355	0.041612	0.38117	0.38674	0.0027544	0.87097
8	Accept	0.38889	0.044509	0.38117	0.38718	0.0027259	0.14804
9	Accept	0.39198	0.048688	0.38117	0.38782	0.0012312	0.15973
10	Accept	0.38194	0.049621	0.38117	0.38118	0.0072485	0.1222
11	Best	0.37346	0.046356	0.37346	0.37347	0.0080333	0.099683
12	Accept	0.37577	0.049216	0.37346	0.37347	0.037492	0.096972
13	Accept	0.37423	0.042163	0.37346	0.37348	0.0029421	0.10221
14	Accept	0.37423	0.045013	0.37346	0.3735	0.012255	0.10292
15	Accept	0.375	0.051912	0.37346	0.374	0.004117	0.097535
16	Accept	0.37423	0.042159	0.37346	0.37403	0.0064046	0.10233
17	Accept	0.37346	0.050822	0.37346	0.37387	0.007314	0.102
18	Accept	0.37423	0.04995	0.37346	0.37393	0.0076869	0.10269
19	Accept	0.40355	0.044103	0.37346	0.37393	1.1182e-06	0.28114
20	Accept	0.40355	0.049635	0.37346	0.37393	1.2854e-06	0.68274

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
21	Accept	0.40586	0.082025	0.37346	0.37546	999.43	0.36371
22	Accept	0.3912	0.040822	0.37346	0.37393	0.015901	0.0031529
23	Accept	0.40586	0.042502	0.37346	0.37393	967.49	0.75413
24	Accept	0.40586	0.06142	0.37346	0.37393	999.95	0.60993
25	Accept	0.40355	0.083912	0.37346	0.37393	1.1044e-06	0.41271
26	Accept	0.40586	0.050123	0.37346	0.37393	995.32	0.24723
27	Accept	0.40355	0.042463	0.37346	0.37393	1.0072e-06	0.573
28	Accept	0.40355	0.04057	0.37346	0.37393	1.0407e-06	0.78156
29	Accept	0.40586	0.058239	0.37346	0.37393	904.04	0.99934
30	Accept	0.37346	0.052674	0.37346	0.37386	0.0043597	0.10295



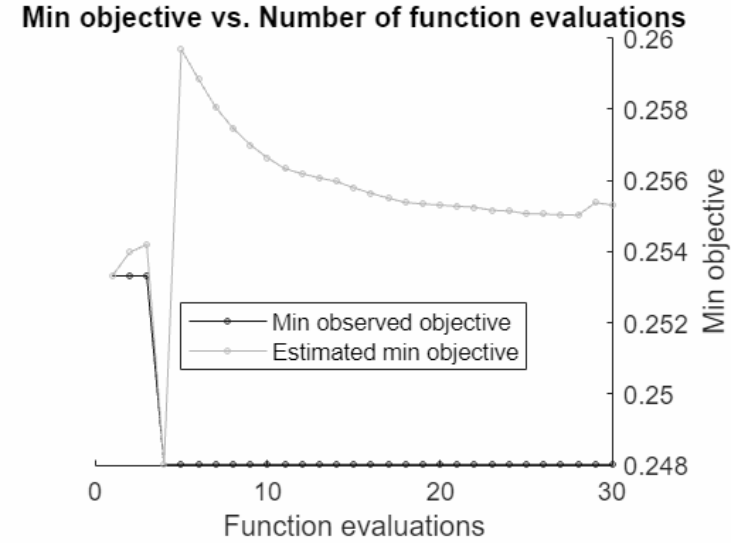


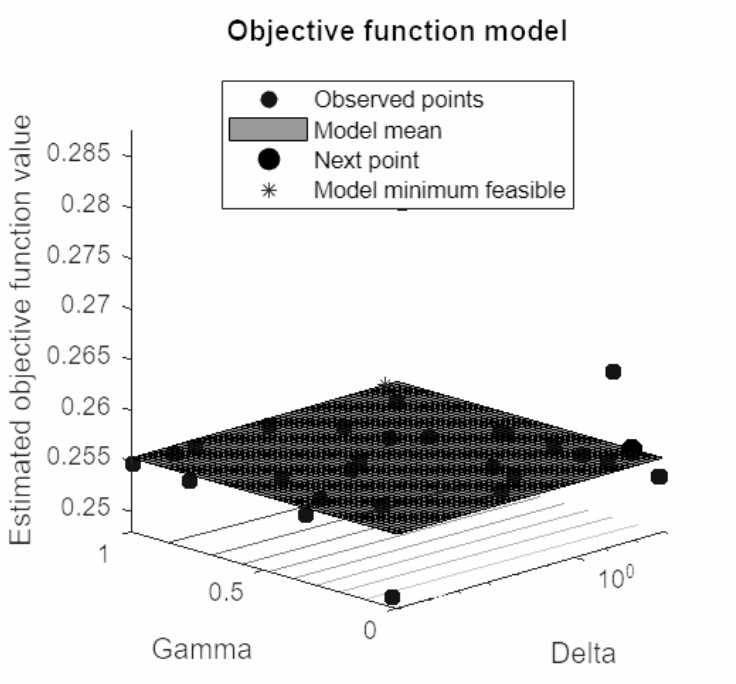
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 17.8078 seconds
Total objective function evaluation time: 1.5197

Best observed feasible point:
      Delta       Gamma
    _________    ________

    0.0080333    0.099683

Observed objective function value = 0.37346
Estimated objective function value = 0.37396
Function evaluation time = 0.046356

Best estimated feasible point (according to models):
     Delta      Gamma
    ________    _____

    0.007314    0.102

Estimated objective function value = 0.37386
Estimated function evaluation time = 0.049797
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.25332	0.05904	0.25332	0.25332	3.8247	0.93251
2	Accept	0.25464	0.039244	0.25332	0.25397	2.3298e-05	0.99245
3	Accept	0.25464	0.042758	0.25332	0.2542	1.0451	0.30776
4	Best	0.24801	0.045427	0.24801	0.24801	3.7299e-06	0.082337
5	Accept	0.2878	0.045933	0.24801	0.25968	1.8029e-06	0.0085473
6	Accept	0.25464	0.041313	0.24801	0.25884	0.85333	0.68771
7	Accept	0.25332	0.045183	0.24801	0.25805	993.56	0.9993
8	Accept	0.25332	0.042153	0.24801	0.25746	993.51	0.61299
9	Accept	0.25332	0.039942	0.24801	0.257	997.97	0.99962

10	Accept	0.25332	0.03868	0.24801	0.25663	946.78	0.010881
11	Accept	0.25332	0.041857	0.24801	0.25633	996.62	0.40952
12	Accept	0.25464	0.051721	0.24801	0.25619	1.0526e-06	0.99852
13	Accept	0.25464	0.04128	0.24801	0.25607	5.4025e-05	0.62406
14	Accept	0.25464	0.044194	0.24801	0.25597	0.044394	0.99879
15	Accept	0.25332	0.046176	0.24801	0.25579	997.72	0.30162
16	Accept	0.25332	0.038542	0.24801	0.25564	975.76	0.58415
17	Accept	0.25332	0.041739	0.24801	0.2555	970.49	0.010069
18	Accept	0.25332	0.053325	0.24801	0.25538	984.19	0.99947
19	Accept	0.25464	0.04284	0.24801	0.25534	7.5787e-05	0.26452
20	Accept	0.25464	0.040114	0.24801	0.25531	0.0025943	0.55079

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
21	Accept	0.25464	0.050415	0.24801	0.25527	1.0051e-06	0.34586
22	Accept	0.25464	0.045934	0.24801	0.25524	0.049149	0.1298
23	Accept	0.25332	0.045932	0.24801	0.25516	29.78	0.70997
24	Accept	0.25464	0.051444	0.24801	0.25514	1.0053e-06	0.78301
25	Accept	0.25332	0.048628	0.24801	0.25507	4.3429	0.29593
26	Accept	0.25464	0.071911	0.24801	0.25505	0.012399	0.59793
27	Accept	0.25464	0.051917	0.24801	0.25503	0.00013918	0.99787
28	Accept	0.25464	0.039178	0.24801	0.25502	1.2601e-05	0.41316
29	Accept	0.26525	0.049424	0.24801	0.25537	21.653	0.00029419
30	Accept	0.25332	0.047152	0.24801	0.25531	997.58	0.20937

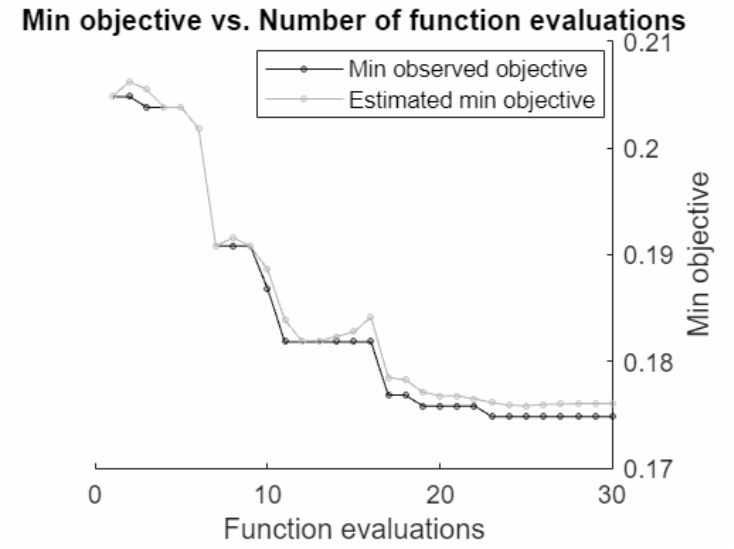


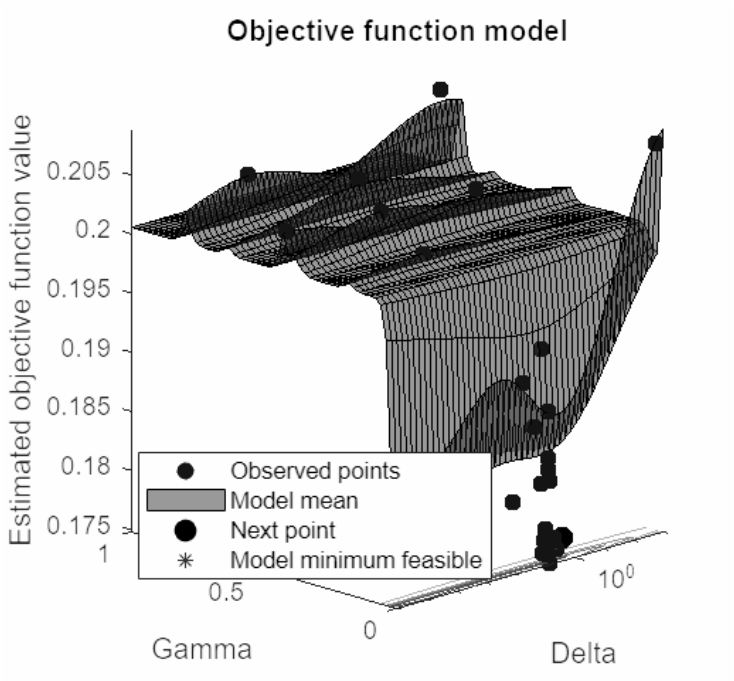


```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 21.0914 seconds
Total objective function evaluation time: 1.3834

Best observed feasible point:
      Delta         Gamma
    __________    ________

    3.7299e-06    0.082337

Observed objective function value = 0.24801
Estimated objective function value = 0.25531
Function evaluation time = 0.045427

Best estimated feasible point (according to models):
    Delta       Gamma
    ______    _______

    997.97    0.99962

Estimated objective function value = 0.25531
Estimated function evaluation time = 0.045664
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.2048	0.058713	0.2048	0.2048	0.00013458	0.7962
2	Accept	0.20779	0.047236	0.2048	0.2062	209.45	0.76078
3	Best	0.2038	0.059953	0.2038	0.20546	0.76615	0.35544
4	Accept	0.2038	0.047458	0.2038	0.2038	1.2056e-06	0.42941
5	Accept	0.2038	0.046402	0.2038	0.2038	0.00087401	0.38774
6	Best	0.2018	0.043534	0.2018	0.2018	0.00035527	0.18223
7	Best	0.19081	0.094233	0.19081	0.19081	0.01795	0.00014427
8	Accept	0.20779	0.042022	0.19081	0.19157	569.56	0.0013827
9	Accept	0.2038	0.044221	0.19081	0.19081	0.025985	0.63853

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
10	Best	0.18681	0.078076	0.18681	0.1886	0.04373	0.0014032
11	Best	0.18182	0.043313	0.18182	0.18387	0.081794	0.0018521
12	Accept	0.18182	0.044236	0.18182	0.18189	0.14855	0.0017505
13	Accept	0.19281	0.045901	0.18182	0.18193	0.16617	0.036605
14	Accept	0.18282	0.044957	0.18182	0.18232	0.14499	0.0028772
15	Accept	0.18382	0.040124	0.18182	0.18276	0.13974	0.002416
16	Accept	0.18781	0.05126	0.18182	0.18413	0.13172	0.00096085
17	Best	0.17682	0.043906	0.17682	0.17848	0.14004	0.0061366
18	Accept	0.17782	0.039895	0.17682	0.17826	0.1177	0.0077275
19	Best	0.17582	0.049868	0.17582	0.17711	0.10309	0.0089792
20	Accept	0.17682	0.041398	0.17582	0.17677	0.14032	0.010619
21	Accept	0.17682	0.04074	0.17582	0.17676	0.11681	0.010236
22	Accept	0.17582	0.049726	0.17582	0.17648	0.13749	0.0098105
23	Best	0.17483	0.037888	0.17483	0.17613	0.18696	0.0096641
24	Accept	0.17582	0.038141	0.17483	0.17587	0.30462	0.010157
25	Accept	0.17582	0.057116	0.17483	0.17584	0.26104	0.010177
26	Accept	0.17682	0.041821	0.17483	0.17595	0.26281	0.010406
27	Accept	0.17782	0.040391	0.17483	0.17599	1.159e-05	0.0073337
28	Accept	0.17782	0.047366	0.17483	0.17603	9.0319e-05	0.013411
29	Accept	0.17782	0.049842	0.17483	0.17604	4.809e-06	0.013355
30	Accept	0.18082	0.039039	0.17483	0.17603	0.011674	0.015436



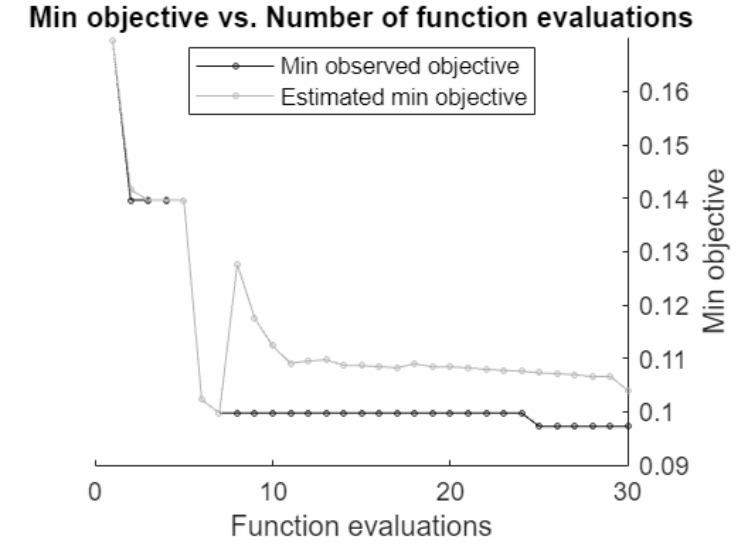


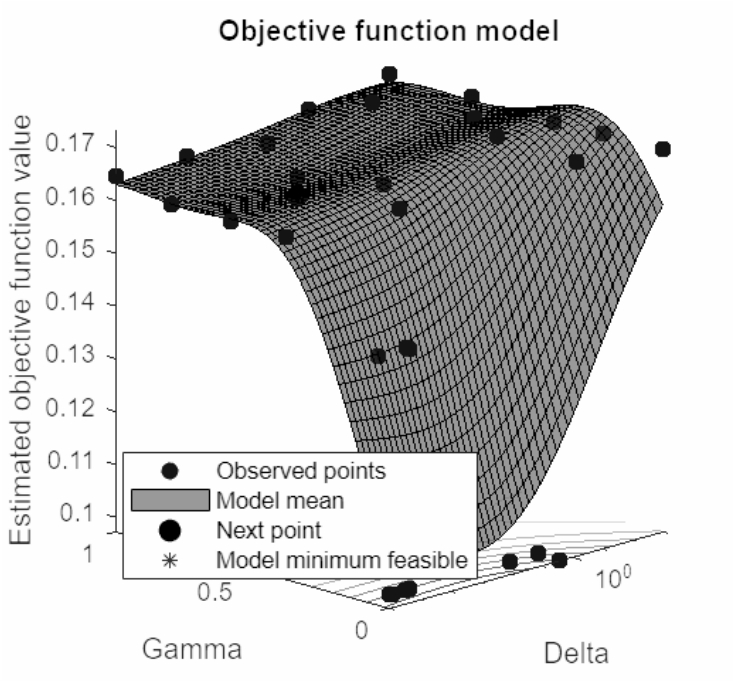
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 16.9684 seconds
Total objective function evaluation time: 1.4488

Best observed feasible point:
     Delta       Gamma
    _______    _________

    0.18696    0.0096641

Observed objective function value = 0.17483
Estimated objective function value = 0.17603
Function evaluation time = 0.037888

Best estimated feasible point (according to models):
     Delta       Gamma
    _______    _________

    0.18696    0.0096641

Estimated objective function value = 0.17603
Estimated function evaluation time = 0.047215
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.16958	0.047149	0.16958	0.16958	7.7979	0.080071
2	Best	0.13965	0.044379	0.13965	0.14159	0.00025202	0.19538
3	Accept	0.16958	0.043932	0.13965	0.13965	713	0.38354
4	Accept	0.16459	0.036387	0.13965	0.13965	0.02855	0.93936
5	Accept	0.13965	0.044981	0.13965	0.13965	2.8619e-05	0.20402
6	Best	0.10224	0.036506	0.10224	0.10225	7.4681e-05	0.001821
7	Best	0.099751	0.040716	0.099751	0.099754	4.8871e-06	0.0013396

8	Accept	0.14464	0.044859	0.099751	0.12759	7.7337e-06	0.03372
9	Accept	0.099751	0.03706	0.099751	0.11743	3.4659e-06	0.0013082
10	Accept	0.099751	0.038844	0.099751	0.11232	1.2349e-06	6.619e-05
11	Accept	0.099751	0.046687	0.099751	0.1091	1.085e-06	0.0011123
12	Accept	0.16209	0.046169	0.099751	0.10949	1.0156e-06	0.58259
13	Accept	0.16958	0.037315	0.099751	0.10974	967.11	0.99835
14	Accept	0.16459	0.043397	0.099751	0.10868	1.016e-06	0.99741
15	Accept	0.16209	0.039621	0.099751	0.10862	0.047812	0.5378
16	Accept	0.16958	0.03929	0.099751	0.10842	995.32	0.70074
17	Accept	0.16958	0.051466	0.099751	0.10824	986.88	0.000475
18	Accept	0.099751	0.037489	0.099751	0.10898	0.0092659	0.00012409
19	Accept	0.16209	0.055148	0.099751	0.10843	1.0066e-06	0.37751
20	Accept	0.16209	0.03653	0.099751	0.10839	1.0463e-06	0.80032

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
21	Accept	0.16209	0.039857	0.099751	0.10824	0.004429	0.7426
22	Accept	0.16209	0.043929	0.099751	0.10792	0.0031856	0.35411
23	Accept	0.16958	0.040983	0.099751	0.10774	10.096	0.8374
24	Accept	0.16958	0.045316	0.099751	0.10759	54.551	0.55061
25	Best	0.097257	0.04114	0.097257	0.10726	0.37808	0.00011606
26	Accept	0.16708	0.046616	0.097257	0.1071	2.1095	0.99987
27	Accept	0.16459	0.040125	0.097257	0.10694	0.00020879	0.99966
28	Accept	0.17207	0.037544	0.097257	0.10656	1.8478	0.30043
29	Accept	0.16958	0.049555	0.097257	0.10661	999.93	0.21598
30	Accept	0.099751	0.039783	0.097257	0.10394	0.079909	0.00026921



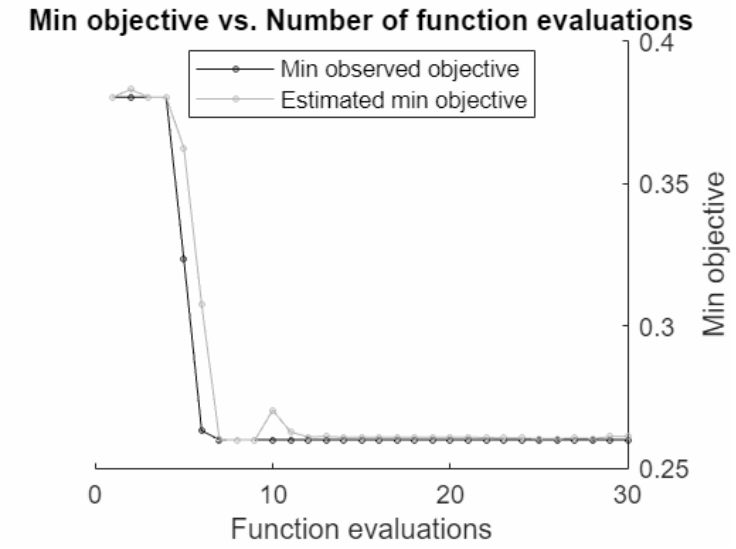


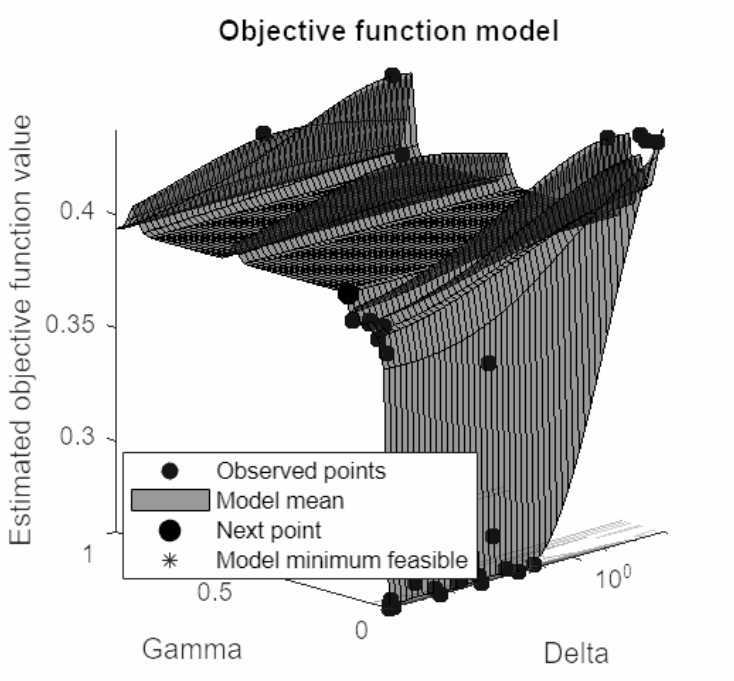
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 18.2712 seconds
Total objective function evaluation time: 1.2728

Best observed feasible point:
     Delta       Gamma
    _______    _________

    0.37808    0.00011606

Observed objective function value = 0.097257
Estimated objective function value = 0.1188
Function evaluation time = 0.04114

Best estimated feasible point (according to models):
      Delta        Gamma
    __________    _______

    7.4681e-05    0.001821

Estimated objective function value = 0.10394
Estimated function evaluation time = 0.039587
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.38035	0.051885	0.38035	0.38035	2.5188e-06	0.065374
2	Accept	0.43265	0.052146	0.38035	0.383	231.04	0.91873
3	Accept	0.41997	0.042331	0.38035	0.38035	0.3466	0.5673
4	Accept	0.42155	0.04523	0.38035	0.38035	0.023209	0.94379
5	Best	0.3233	0.040942	0.3233	0.3624	1.0141e-06	0.00655
6	Best	0.26307	0.039812	0.26307	0.30729	0.00019929	0.00012093
7	Best	0.2599	0.045569	0.2599	0.26009	0.01795	0.00014427
8	Accept	0.43265	0.038901	0.2599	0.25993	657.55	0.00015686
9	Accept	0.35499	0.038682	0.2599	0.25993	0.0024998	0.013569

10	Accept	0.27892	0.060405	0.2599	0.27034	0.002505	0.0014879
11	Accept	0.2599	0.04371	0.2599	0.26268	0.0010491	0.00017245
12	Accept	0.26307	0.045747	0.2599	0.26093	0.0008762	6.5107e-05
13	Accept	0.2599	0.052633	0.2599	0.26109	1.0139e-06	0.0001651
14	Accept	0.26783	0.037213	0.2599	0.26078	6.7592e-06	0.00042201
15	Accept	0.37084	0.049853	0.2599	0.26079	1.3832e-06	0.027812
16	Accept	0.43265	0.056832	0.2599	0.2608	501.69	0.028388
17	Accept	0.38193	0.042451	0.2599	0.26081	1.3115e-06	0.085849
18	Accept	0.43265	0.051082	0.2599	0.26082	822.22	0.075029
19	Accept	0.37718	0.042272	0.2599	0.26083	1.1081e-06	0.047797
20	Accept	0.38193	0.041101	0.2599	0.26083	1.9844e-06	0.10443

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
21	Accept	0.27892	0.038032	0.2599	0.26087	1.0183e-06	0.0016473
22	Accept	0.26307	0.049418	0.2599	0.2607	0.0063956	1.0119e-05
23	Accept	0.26149	0.040215	0.2599	0.26059	1.0546e-06	5.8133e-05
24	Accept	0.38193	0.037346	0.2599	0.26055	1.0255e-06	0.13498
25	Accept	0.2599	0.050003	0.2599	0.26029	1.3574e-06	0.0001758
26	Accept	0.43265	0.037697	0.2599	0.26031	191.34	0.12367
27	Accept	0.2599	0.043236	0.2599	0.26052	4.6121e-05	0.00018415
28	Accept	0.26149	0.047259	0.2599	0.26037	0.056895	5.589e-05
29	Accept	0.26307	0.041712	0.2599	0.26105	1.0729e-06	4.2521e-05
30	Accept	0.26307	0.044052	0.2599	0.26103	3.257e-05	6.5448e-05



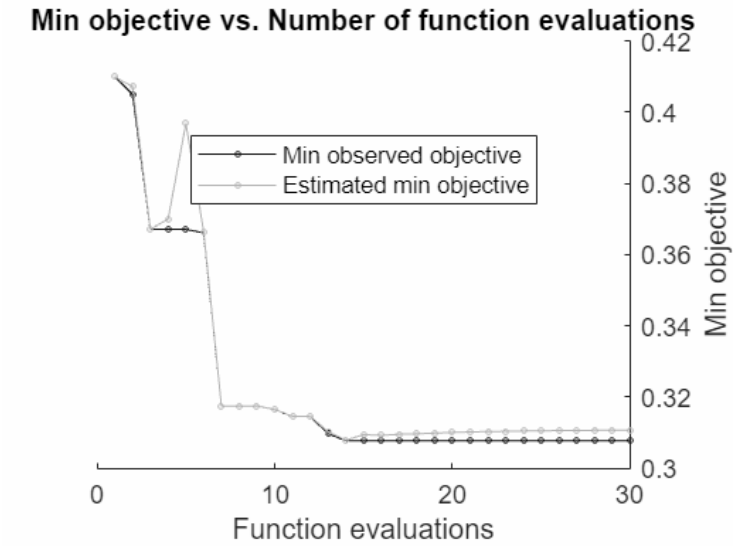


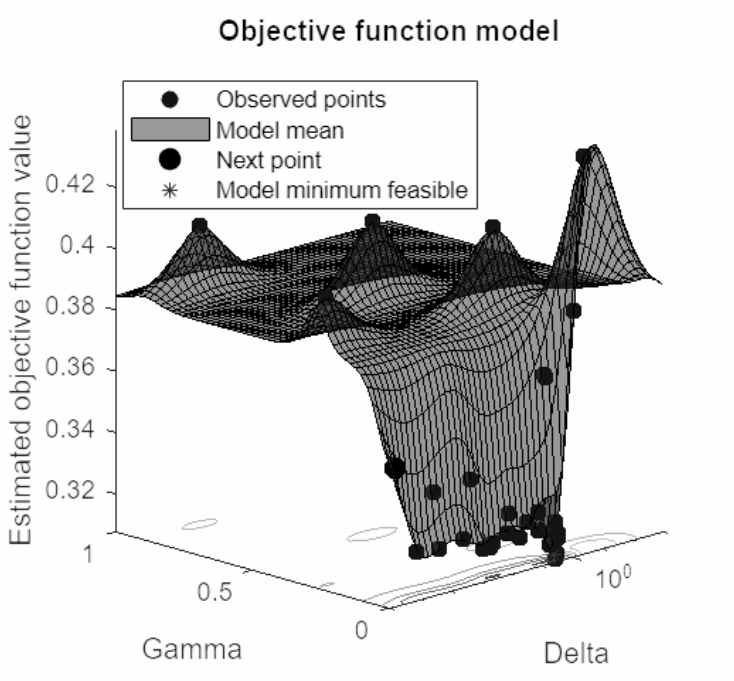
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 16.9835 seconds
Total objective function evaluation time: 1.3478

Best observed feasible point:
     Delta       Gamma
    _______    _________

    0.01795    0.00014427

Observed objective function value = 0.2599
Estimated objective function value = 0.26122
Function evaluation time = 0.045569

Best estimated feasible point (according to models):
      Delta         Gamma
    _________    _________

    1.0729e-06    4.2521e-05

Estimated objective function value = 0.26103
Estimated function evaluation time = 0.04455
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.40993	0.052685	0.40993	0.40993	0.011344	0.51472
2	Best	0.40506	0.042611	0.40506	0.40719	6.6266e-05	0.89526
3	Best	0.36709	0.038914	0.36709	0.3671	0.36144	0.052903
4	Accept	0.40117	0.044464	0.36709	0.37001	2.2178e-06	0.26921
5	Accept	0.43427	0.047332	0.36709	0.39683	8.0904	0.057235
6	Best	0.36611	0.045039	0.36611	0.36649	0.36022	0.04957
7	Best	0.31743	0.053739	0.31743	0.31744	0.32556	0.010707
8	Accept	0.41091	0.04206	0.31743	0.31744	0.30909	0.23282
9	Accept	0.32035	0.043911	0.31743	0.31746	0.31404	0.00577

10	Best	0.31646	0.048948	0.31646	0.31647	0.40518	0.0078749
11	Best	0.31451	0.03896	0.31451	0.31455	0.48469	0.013205
12	Accept	0.38754	0.044421	0.31451	0.31456	1.2895	0.0068434
13	Best	0.30964	0.050084	0.30964	0.31018	0.42374	0.015953
14	Best	0.30769	0.041438	0.30769	0.30785	0.41879	0.018619
15	Accept	0.31743	0.048083	0.30769	0.30939	0.47266	0.022711
16	Accept	0.31256	0.047818	0.30769	0.30919	0.30988	0.022333
17	Accept	0.3184	0.042505	0.30769	0.30948	0.11593	0.019158
18	Accept	0.32522	0.042337	0.30769	0.30964	0.081522	0.0026678
19	Accept	0.3223	0.051077	0.30769	0.30982	0.053134	0.024103
20	Accept	0.3184	0.05594	0.30769	0.31001	0.023231	0.0064547

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
21	Accept	0.32619	0.052076	0.30769	0.31016	0.014334	0.025623
22	Accept	0.32132	0.051302	0.30769	0.31027	0.0080446	0.0016286
23	Accept	0.31743	0.062769	0.30769	0.31033	0.003095	0.020141
24	Accept	0.3184	0.046287	0.30769	0.31043	0.0013095	0.0053539
25	Accept	0.3408	0.05733	0.30769	0.31047	0.00088282	0.031719
26	Accept	0.31938	0.038398	0.30769	0.31054	0.0026934	0.0049827
27	Accept	0.32327	0.044654	0.30769	0.31057	0.00028624	0.0040369
28	Accept	0.3223	0.046422	0.30769	0.31062	4.6013e-05	0.0024003
29	Accept	0.33982	0.047354	0.30769	0.31067	5.4067e-05	0.03021
30	Accept	0.32327	0.037043	0.30769	0.31072	8.5269e-06	0.0042039



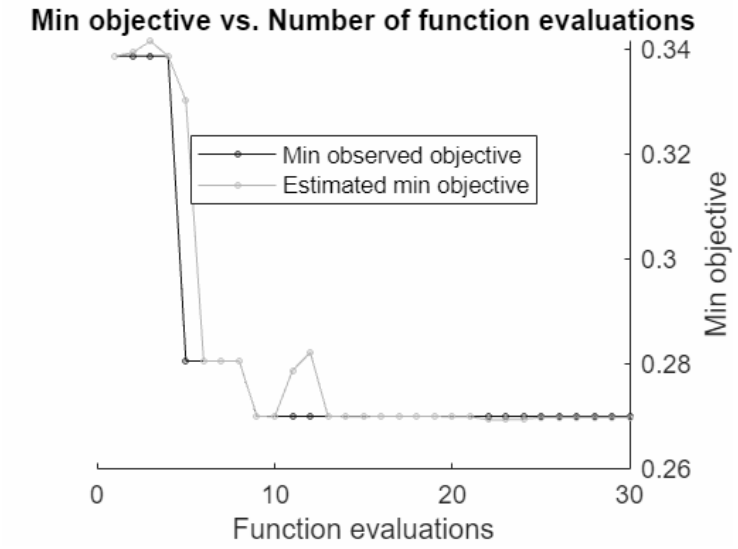


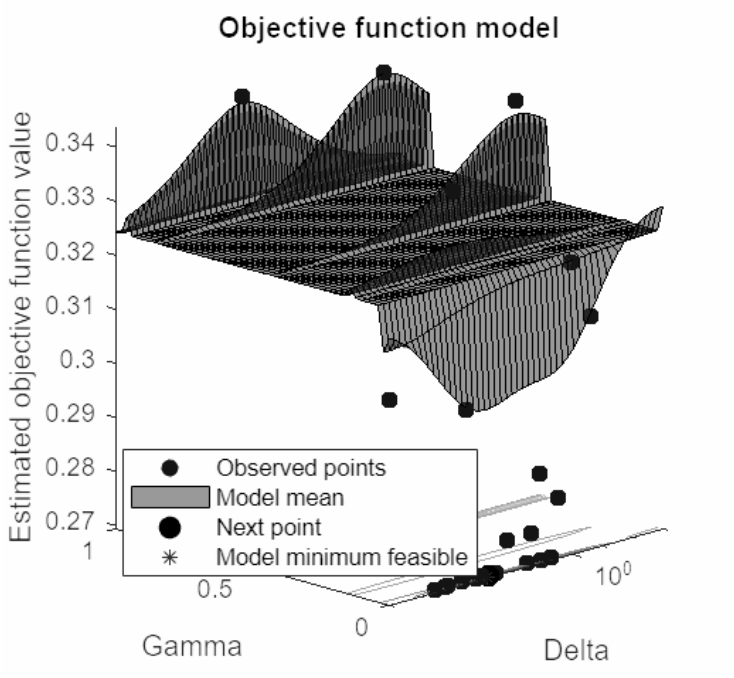
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 17.1307 seconds
Total objective function evaluation time: 1.406

Best observed feasible point:
     Delta      Gamma
    _______    _______

    0.41879    0.018619

Observed objective function value = 0.30769
Estimated objective function value = 0.31072
Function evaluation time = 0.041438

Best estimated feasible point (according to models):
     Delta      Gamma
    _______    _______

    0.41879    0.018619

Estimated objective function value = 0.31072
Estimated function evaluation time = 0.046508
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
1	Best	0.33862	0.041928	0.33862	0.33862	0.0022289	0.13592
2	Accept	0.34392	0.039416	0.33862	0.33925	0.0060645	0.96052
3	Accept	0.34392	0.035584	0.33862	0.34148	108.95	0.43057
4	Accept	0.34392	0.043906	0.33862	0.33862	33.883	0.85661
5	Best	0.28042	0.046799	0.28042	0.33016	0.00040883	0.0074982
6	Accept	0.28042	0.048077	0.28042	0.28043	0.00038293	0.0075241
7	Accept	0.32275	0.041306	0.28042	0.28043	1.5291	0.024516
8	Accept	0.28042	0.037126	0.28042	0.28042	0.39212	0.0032281
9	Best	0.26984	0.044257	0.26984	0.26987	3.5191e-05	0.0054045

10	Accept	0.26984	0.04447	0.26984	0.26982	0.22374	0.0050596
11	Accept	0.31217	0.037361	0.26984	0.27861	4.9415	0.0053231
12	Accept	0.30159	0.06543	0.26984	0.28208	0.00034528	5.3442e-05
13	Accept	0.27513	0.039905	0.26984	0.26985	5.023e-06	0.0063372
14	Accept	0.26984	0.0574	0.26984	0.26983	0.039401	0.0059593
15	Accept	0.28571	0.050285	0.26984	0.26983	0.10819	0.0080731
16	Accept	0.27513	0.033946	0.26984	0.26985	0.052218	0.0041905
17	Accept	0.28571	0.040821	0.26984	0.26985	9.8796e-06	0.0038047
18	Accept	0.27513	0.040641	0.26984	0.26985	2.5251e-05	0.0066652
19	Accept	0.26984	0.042949	0.26984	0.26984	0.00025739	0.0050902
20	Accept	0.26984	0.039673	0.26984	0.26983	0.0026429	0.0054412

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Delta	Gamma
21	Accept	0.26984	0.037275	0.26984	0.26978	0.11066	0.0055036
22	Accept	0.27513	0.044432	0.26984	0.26921	0.0089434	0.0062126
23	Accept	0.26984	0.037216	0.26984	0.26923	0.001118	0.0048185
24	Accept	0.26984	0.036096	0.26984	0.26928	0.00077956	0.0054664
25	Accept	0.26984	0.043384	0.26984	0.26969	0.0012192	0.0052386
26	Accept	0.26984	0.037084	0.26984	0.26969	8.8847e-05	0.0057142
27	Accept	0.26984	0.037903	0.26984	0.26969	7.1723e-05	0.00516
28	Accept	0.26984	0.047102	0.26984	0.26968	0.0035445	0.0047984
29	Accept	0.29101	0.040053	0.26984	0.26968	1.1472e-06	0.010461
30	Accept	0.30688	0.036899	0.26984	0.26969	1.3825e-06	0.016468






```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 17.2267 seconds
Total objective function evaluation time: 1.2687

Best observed feasible point:
      Delta          Gamma
    _________    _________

    3.5191e-05    0.0054045

Observed objective function value = 0.26984
Estimated objective function value = 0.2699
Function evaluation time = 0.044257

Best estimated feasible point (according to models):
      Delta         Gamma
    _________    _________

    0.0012192    0.0052386

Estimated objective function value = 0.26969
Estimated function evaluation time = 0.041849
```

Discriminant Analysis Visualisation

The results of every layer of the hypermatrix are summed up and visualised in a heatmap. Additionally the mean accuracy and time taken is calculated and used as a comparison metric between methods.

```
DA = zeros(m,n);
for j = 1:n
    for k = 1:m
        DA(j,k) = sum(hyp_DA(j,k,:));
    end
```

```
end
figDA = figureGen(7,10);
heat_DA6 = heatmap(DA, "XDisplayLabels", [0,1,2,3], "YDisplayLabels", [0,1,2,3],
"ColorMethod", "mean", "ColorLimits", [0,100])
```

```
heat_DA6 =
  HeatmapChart with properties:

       XData: {4×1 cell}
       YData: {4×1 cell}
    ColorData: [4×4 double]

  Show all properties
```

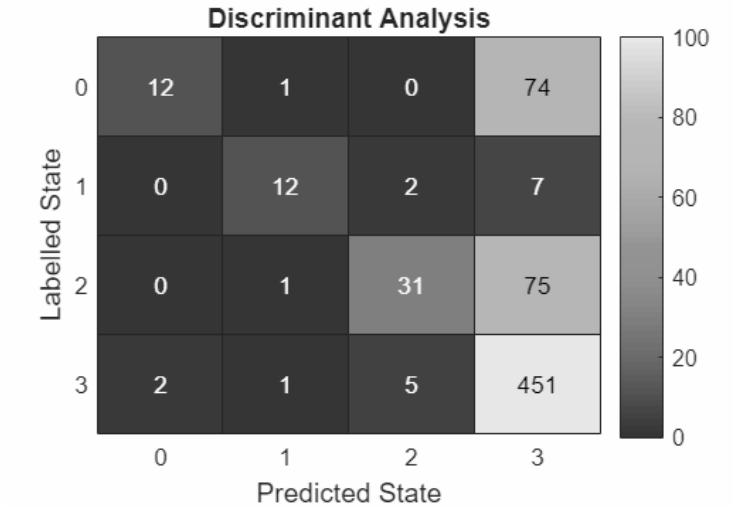
```
heat_DA6.Colormap = parula(64);
xlabel("Predicted State");
ylabel("Labelled State");
average_acc_DA = median(acc_DA)
```

```
average_acc_DA = 0.7714
```

```
avergage_time_DA = median(time_DA)
```

```
avergage_time_DA = 0.0249
```

```
heat_DA6.Title = "Discriminant Analysis";
saveas(heat_DA6, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\6_DA.jpg');
```



K-Nearest Neighbors (KNN)

The next method to be applied will be KNN.

```
for i = 1:numel(files)
    %Load file i from the folder
    fileName = fullfile(files(i).folder, files(i).name);
    load(fileName);
```

```
%Extract the state vector
state = stateTT.State;
%Extract the Principal COmponent Data
D = PCsTT{:,:};
%Create a random 90/10 Partition for Training and Test Data
rng('default');
Partition_States = cvpartition(state, 'Holdout', 0.10);

%Seperate the training and testing Ids
trainingIds = training(Partition_States);
DTrain = D(trainingIds, :);
stateTrain = state(trainingIds);

testIds = test(Partition_States);
DTest = D(testIds, :);
stateTest = state(testIds);
```

Begin measuring the time this algorithm will take

```
 tKNN = tic;
```

train the decsion tree classifier

```
classifierKNN = fitcknn(DTrain, stateTrain, 'OptimizeHyperparameters','auto',...
'HyperparameterOptimizationOptions',...
struct('AcquisitionFunctionName','expected-improvement-plus'));
```

end the time measurement

```
timeKNN = toc(tKNN);
```

use the tree to predict the test states

```
TestModel_KNN = predict(classifierKNN, DTest);
```

measure the accuracy

```
accuracy_KNN = sum(stateTest == TestModel_KNN)/length(stateTest);
%figure;
%confusionchart(stateTest, TestModel_BT, 'Normalization', 'row-normalized');
```

Create a confusion matrix

```
[C_KNN, order] = confusionmat(stateTest, TestModel_KNN);
%titleStr_BT = strrep([fName,' Binary Tree'],'_','-');
% title(titleStr_BT);
```

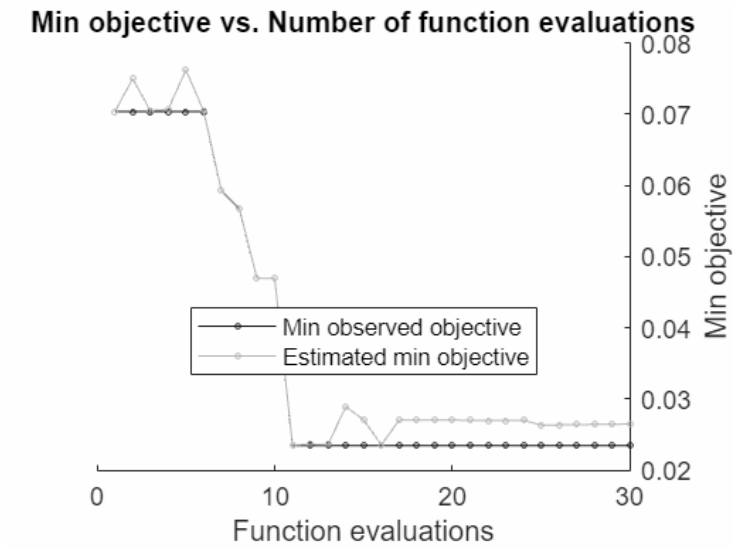
save the confusion matrix as one layer of the hypermatrix and the accuracy and measured time in vector form

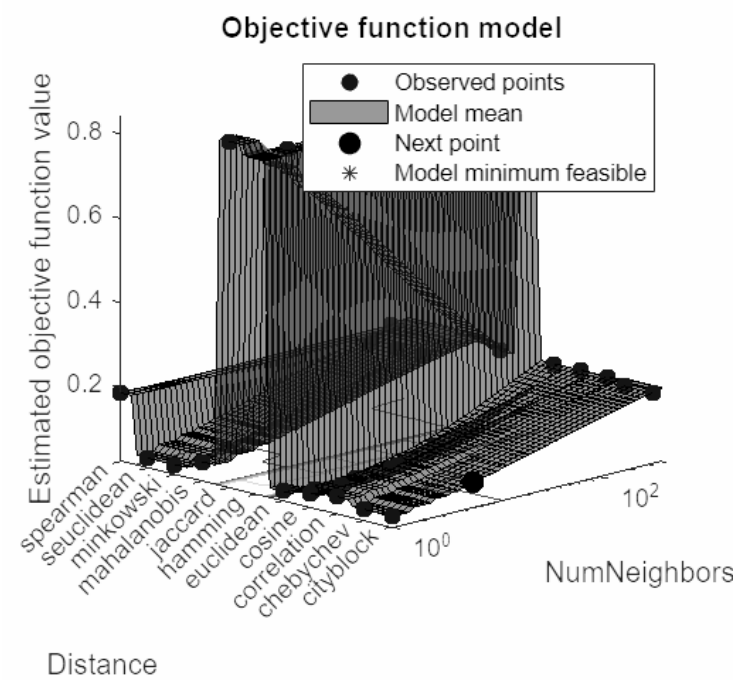
```
    hyp_KNN(:,:,i) = C_KNN;
    acc_KNN(i) = accuracy_KNN;
    time_KNN(i) = timeKNN/length(D);
end
```

```
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.07037	0.075873	0.07037	0.07037	6	cosine
2	Accept	0.18642	0.11671	0.07037	0.074985	1	spearman
3	Accept	0.19259	0.057523	0.07037	0.070504	137	seuclidean
4	Accept	0.19259	0.072936	0.07037	0.070673	321	chebychev
5	Accept	0.19259	0.077503	0.07037	0.076231	403	cosine
6	Accept	0.07284	0.09767	0.07037	0.07042	7	cosine
7	Best	0.059259	0.056069	0.059259	0.059281	2	cosine
8	Best	0.05679	0.05973	0.05679	0.056836	1	cosine
9	Best	0.046914	0.051057	0.046914	0.046923	1	euclidean
10	Accept	0.066667	0.050546	0.046914	0.046923	29	euclidean
11	Best	0.023457	0.058626	0.023457	0.023514	3	euclidean
12	Accept	0.034568	0.045444	0.023457	0.023619	5	euclidean
13	Accept	0.19259	0.069966	0.023457	0.023594	403	euclidean
14	Accept	0.033333	0.05245	0.023457	0.028914	2	euclidean
15	Accept	0.023457	0.049425	0.023457	0.027065	3	euclidean
16	Accept	0.023457	0.045351	0.023457	0.023542	3	euclidean
17	Accept	0.82346	0.055979	0.023457	0.027027	2	hamming
18	Accept	0.064198	0.055954	0.023457	0.027011	1	correlation
19	Accept	0.19259	0.065963	0.023457	0.027025	398	correlation
20	Accept	0.19259	0.063021	0.023457	0.027005	137	minkowski

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.19383	0.059932	0.023457	0.026986	137	mahalanobis
22	Accept	0.19259	0.075177	0.023457	0.026968	323	cityblock
23	Accept	0.19259	0.071871	0.023457	0.026951	404	jaccard
24	Accept	0.19259	0.12262	0.023457	0.026997	401	spearman
25	Accept	0.84444	0.054395	0.023457	0.026314	1	jaccard
26	Accept	0.048148	0.057132	0.023457	0.026351	1	cityblock
27	Accept	0.049383	0.047401	0.023457	0.026384	1	chebychev
28	Accept	0.046914	0.063418	0.023457	0.02641	1	seuclidean
29	Accept	0.046914	0.051088	0.023457	0.026432	1	minkowski
30	Accept	0.069136	0.052189	0.023457	0.026461	1	mahalanobis



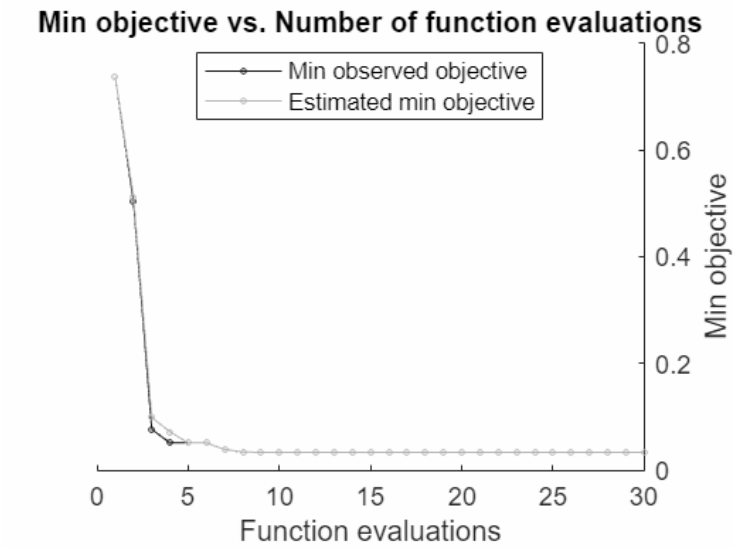


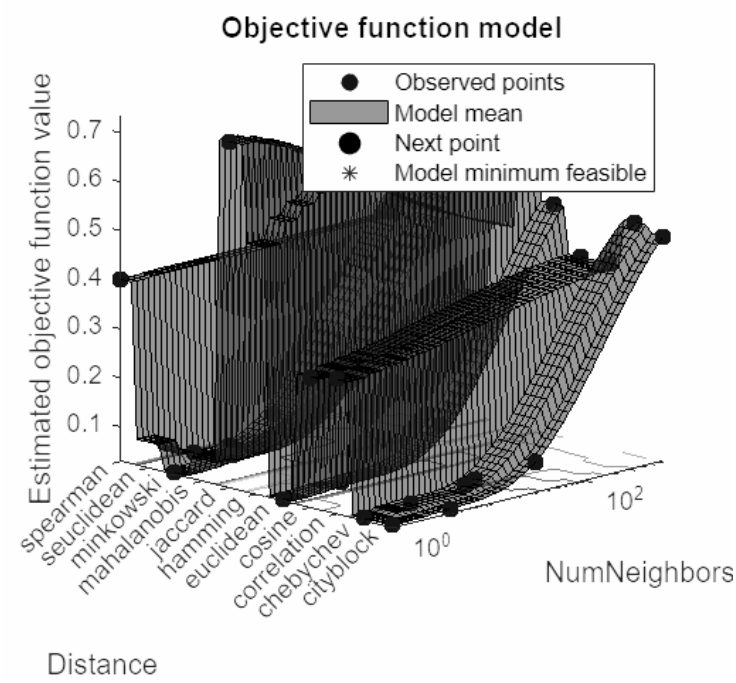
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 17.8376 seconds
Total objective function evaluation time: 1.933

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         3          euclidean

Observed objective function value = 0.023457
Estimated objective function value = 0.026461
Function evaluation time = 0.058626

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ____________    _________

         3          euclidean

Estimated objective function value = 0.026461
Estimated function evaluation time = 0.051361
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.73611	0.082437	0.73611	0.73611	62	hamming
2	Best	0.50231	0.10319	0.50231	0.51161	639	chebychev
3	Best	0.076389	0.070171	0.076389	0.098449	20	seuclidean
4	Best	0.050926	0.063759	0.050926	0.071681	11	chebychev
5	Accept	0.060185	0.065211	0.050926	0.050817	14	chebychev
6	Accept	0.51235	0.090115	0.050926	0.050964	614	seuclidean
7	Best	0.03858	0.054509	0.03858	0.038588	3	seuclidean
8	Best	0.033951	0.058668	0.033951	0.033985	1	chebychev
9	Best	0.033179	0.060254	0.033179	0.033204	1	cityblock
10	Accept	0.091821	0.058439	0.033179	0.033199	31	cityblock
11	Accept	0.30401	0.063148	0.033179	0.0332	1	correlation
12	Accept	0.037809	0.056498	0.033179	0.033198	4	cityblock
13	Accept	0.40123	0.13941	0.033179	0.033202	1	spearman
14	Accept	0.033179	0.062436	0.033179	0.033196	1	minkowski
15	Accept	0.084877	0.057015	0.033179	0.033195	29	minkowski
16	Accept	0.068673	0.065421	0.033179	0.033193	2	mahalanobis
17	Accept	0.23071	0.08685	0.033179	0.033194	131	mahalanobis
18	Accept	0.40586	0.11356	0.033179	0.033196	641	cosine
19	Accept	0.039352	0.054725	0.033179	0.033194	4	minkowski
20	Accept	0.73611	0.058283	0.033179	0.033197	1	jaccard

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.033179	0.053829	0.033179	0.033193	1	euclidean
22	Accept	0.10494	0.054553	0.033179	0.033192	42	euclidean
23	Accept	0.40586	0.11459	0.033179	0.033194	646	correlation
24	Accept	0.039352	0.064929	0.033179	0.033191	4	euclidean
25	Accept	0.40586	0.21453	0.033179	0.033192	627	spearman
26	Accept	0.29552	0.057237	0.033179	0.033193	1	cosine
27	Accept	0.5	0.10224	0.033179	0.033201	636	minkowski
28	Accept	0.48688	0.097134	0.033179	0.033204	641	cityblock
29	Accept	0.042438	0.054122	0.033179	0.033203	3	chebychev
30	Accept	0.50077	0.23486	0.033179	0.033226	644	euclidean



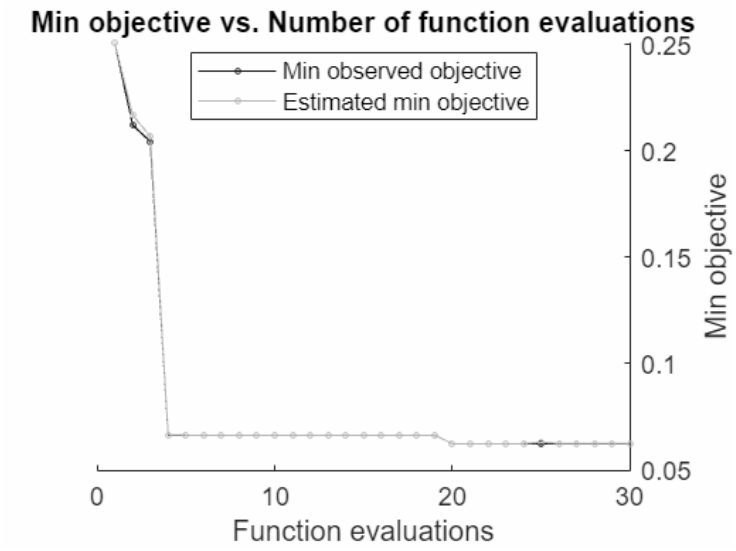


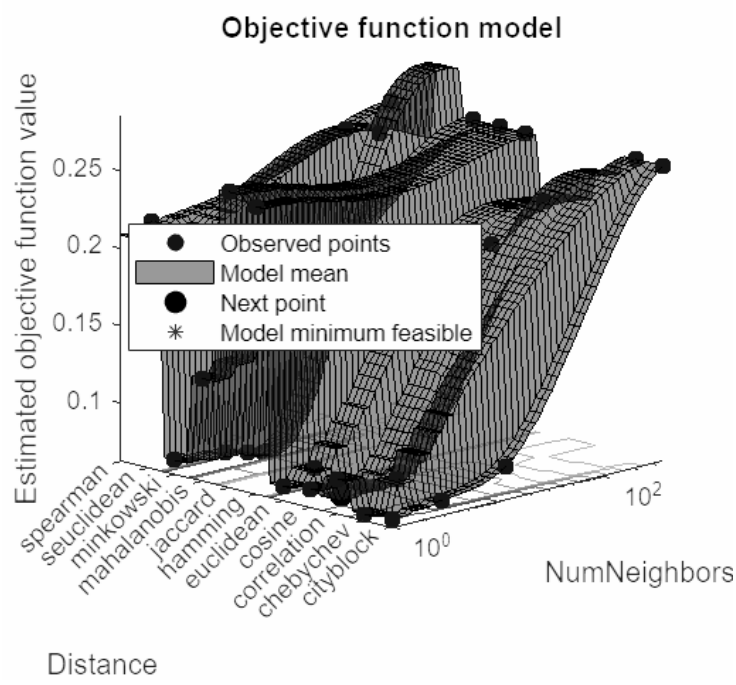
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 19.2149 seconds
Total objective function evaluation time: 2.5121

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         1          cityblock

Observed objective function value = 0.033179
Estimated objective function value = 0.033226
Function evaluation time = 0.060254

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ___________    _________

         1          cityblock

Estimated objective function value = 0.033226
Estimated function evaluation time = 0.060255
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.25066	0.097298	0.25066	0.25066	77	seuclidean
2	Best	0.2122	0.12398	0.2122	0.21685	2	spearman
3	Best	0.20424	0.056849	0.20424	0.20668	53	cosine
4	Best	0.066313	0.049112	0.066313	0.066484	1	chebychev
5	Accept	0.067639	0.047256	0.066313	0.066402	1	cityblock
6	Accept	0.084881	0.052149	0.066313	0.066354	1	correlation
7	Accept	0.25332	0.059772	0.066313	0.066381	376	chebychev
8	Accept	0.083554	0.044898	0.066313	0.066338	2	chebychev
9	Accept	0.083554	0.0502	0.066313	0.066342	12	cityblock
10	Accept	0.2321	0.059005	0.066313	0.066337	91	correlation
11	Accept	0.071618	0.05909	0.066313	0.066342	3	cityblock
12	Accept	0.25332	0.049092	0.066313	0.066346	1	jaccard
13	Accept	0.071618	0.051558	0.066313	0.066342	1	euclidean
14	Accept	0.20557	0.051841	0.066313	0.066338	42	euclidean
15	Accept	0.25332	0.061228	0.066313	0.066334	376	cityblock
16	Accept	0.24801	0.03985	0.066313	0.066335	1	hamming
17	Accept	0.078249	0.052912	0.066313	0.066338	2	euclidean
18	Accept	0.25332	0.063999	0.066313	0.066339	376	mahalanobis
19	Accept	0.071618	0.048972	0.066313	0.066337	1	minkowski
20	Best	0.062334	0.044369	0.062334	0.062344	7	minkowski

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.25066	0.049076	0.062334	0.062356	87	minkowski
22	Accept	0.068966	0.043249	0.062334	0.062385	3	minkowski
23	Accept	0.079576	0.063609	0.062334	0.062389	4	correlation
24	Accept	0.25332	0.11109	0.062334	0.0624	373	spearman
25	Accept	0.25332	0.051415	0.062334	0.063145	367	hamming
26	Accept	0.064987	0.052777	0.062334	0.062511	5	minkowski
27	Accept	0.12865	0.050229	0.062334	0.062498	1	mahalanobis
28	Accept	0.25332	0.064425	0.062334	0.062504	377	jaccard
29	Accept	0.074271	0.060362	0.062334	0.062496	1	cosine
30	Accept	0.067639	0.058106	0.062334	0.062503	3	cosine



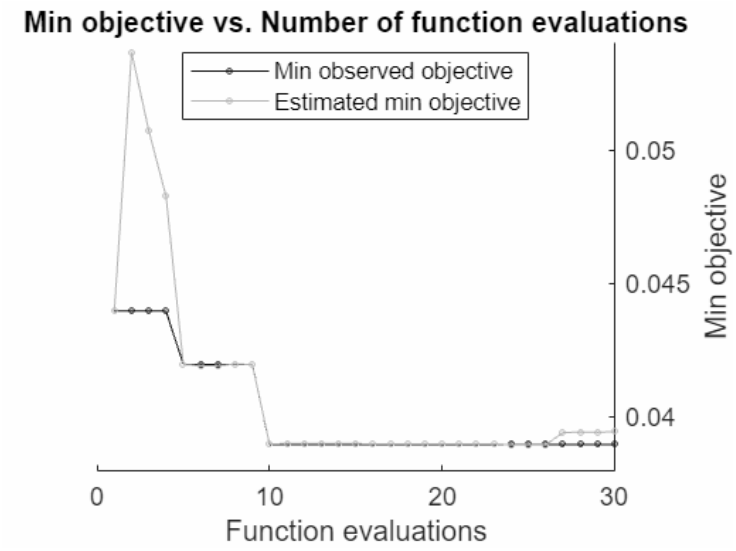


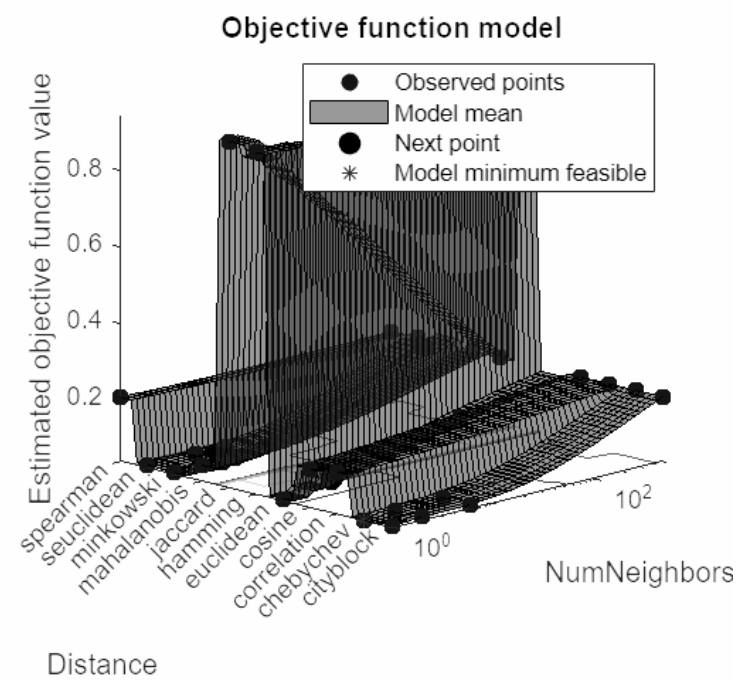
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 18.4849 seconds
Total objective function evaluation time: 1.7678

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         7          minkowski

Observed objective function value = 0.062334
Estimated objective function value = 0.062503
Function evaluation time = 0.044369

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ____________    _________

         7          minkowski

Estimated objective function value = 0.062503
Estimated function evaluation time = 0.047683
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.043956	0.074626	0.043956	0.043956	4	minkowski
2	Accept	0.20779	0.07001	0.043956	0.053649	313	minkowski
3	Accept	0.14486	0.06512	0.043956	0.050716	58	euclidean
4	Accept	0.043956	0.048928	0.043956	0.048254	1	euclidean
5	Best	0.041958	0.060107	0.041958	0.04196	3	minkowski
6	Accept	0.043956	0.044661	0.041958	0.041905	1	minkowski
7	Accept	0.13886	0.055377	0.041958	0.041909	1	cosine
8	Accept	0.93307	0.055783	0.041958	0.041976	1	hamming
9	Accept	0.14885	0.057752	0.041958	0.04195	1	correlation
10	Best	0.038961	0.059211	0.038961	0.03898	1	chebychev
11	Accept	0.20779	0.078694	0.038961	0.038996	500	chebychev
12	Accept	0.042957	0.049971	0.038961	0.038993	1	cityblock
13	Accept	0.20779	0.072716	0.038961	0.038993	499	cityblock
14	Accept	0.045954	0.049701	0.038961	0.038991	1	seuclidean
15	Accept	0.20779	0.070897	0.038961	0.03899	496	seuclidean
16	Accept	0.20679	0.11513	0.038961	0.038989	1	spearman
17	Accept	0.080919	0.080533	0.038961	0.038988	1	mahalanobis
18	Accept	0.20779	0.081174	0.038961	0.038988	501	mahalanobis
19	Accept	0.20779	0.066186	0.038961	0.038987	499	jaccard
20	Accept	0.20779	0.088443	0.038961	0.038988	499	cosine

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.20779	0.081046	0.038961	0.038988	498	correlation
22	Accept	0.20779	0.14711	0.038961	0.038988	500	spearman
23	Accept	0.94505	0.049229	0.038961	0.038984	1	jaccard
24	Accept	0.048951	0.051628	0.038961	0.038946	6	chebychev
25	Accept	0.047952	0.053904	0.038961	0.038941	6	cityblock
26	Accept	0.043956	0.051185	0.038961	0.038939	6	seuclidean
27	Accept	0.043956	0.056161	0.038961	0.039402	2	chebychev
28	Accept	0.043956	0.065102	0.038961	0.039427	3	seuclidean
29	Accept	0.046953	0.070386	0.038961	0.039414	2	cityblock
30	Accept	0.098901	0.070167	0.038961	0.03944	9	mahalanobis



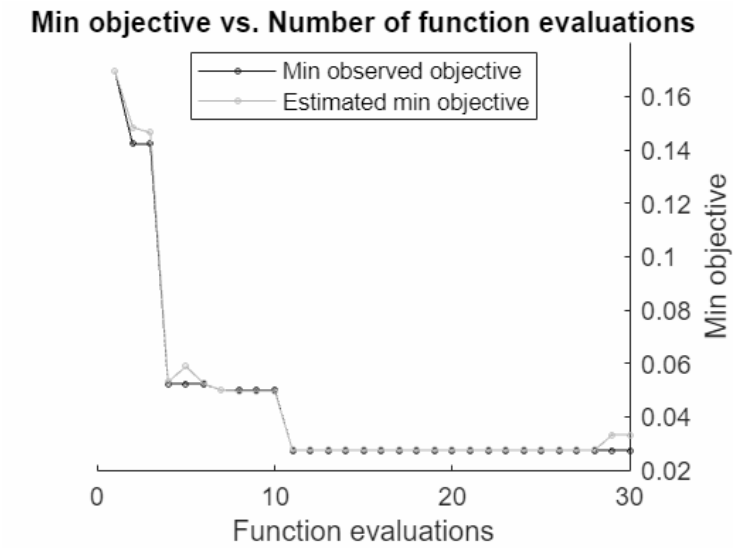


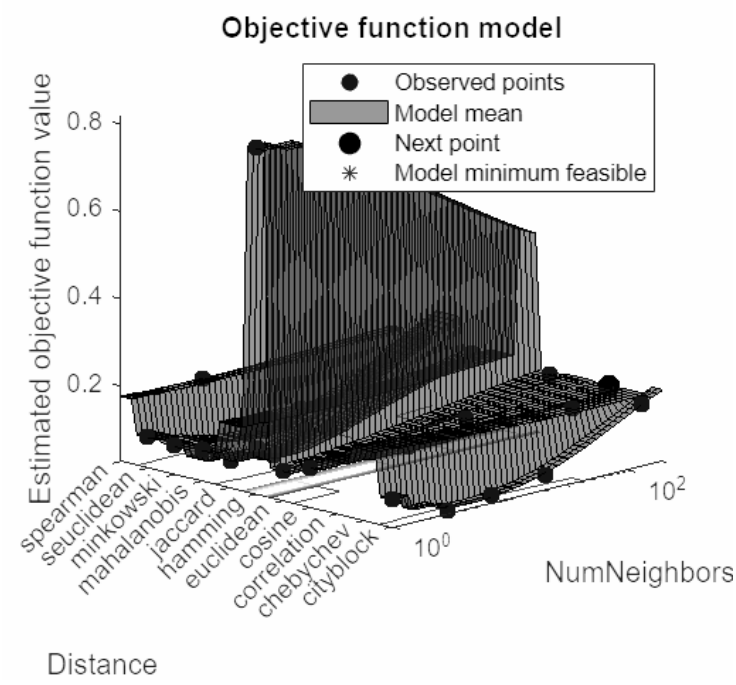
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 19.2263 seconds
Total objective function evaluation time: 2.0409

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         1          chebychev

Observed objective function value = 0.038961
Estimated objective function value = 0.03944
Function evaluation time = 0.059211

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ____________    _________

        1           chebychev

Estimated objective function value = 0.03944
Estimated function evaluation time = 0.058911
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.16958	0.066126	0.16958	0.16958	58	chebychev
2	Best	0.14214	0.062467	0.14214	0.14847	4	correlation
3	Accept	0.16958	0.054734	0.14214	0.14639	184	euclidean
4	Best	0.052369	0.052775	0.052369	0.053011	14	seuclidean
5	Accept	0.16958	0.064799	0.052369	0.058834	201	seuclidean
6	Accept	0.054863	0.04886	0.052369	0.052452	15	seuclidean
7	Best	0.049875	0.047861	0.049875	0.049869	7	seuclidean
8	Accept	0.049875	0.044897	0.049875	0.049618	10	seuclidean
9	Accept	0.097257	0.04119	0.049875	0.049569	1	seuclidean
10	Accept	0.049875	0.049547	0.049875	0.049584	9	seuclidean
11	Best	0.027431	0.047785	0.027431	0.027444	3	minkowski
12	Accept	0.054863	0.050054	0.027431	0.027449	13	minkowski
13	Accept	0.094763	0.045036	0.027431	0.027452	1	minkowski
14	Accept	0.042394	0.044557	0.027431	0.027474	5	minkowski
15	Accept	0.16958	0.047984	0.027431	0.027469	60	minkowski
16	Accept	0.047382	0.073037	0.027431	0.027466	4	mahalanobis
17	Accept	0.077307	0.049083	0.027431	0.027466	12	mahalanobis
18	Accept	0.099751	0.053975	0.027431	0.027464	1	mahalanobis
19	Accept	0.16958	0.055776	0.027431	0.027468	195	mahalanobis
20	Accept	0.17207	0.079613	0.027431	0.027471	5	spearman

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.16958	0.050977	0.027431	0.027474	4	jaccard
22	Accept	0.092269	0.047934	0.027431	0.027471	1	cityblock
23	Accept	0.047382	0.048094	0.027431	0.02747	7	cityblock
24	Accept	0.062344	0.046358	0.027431	0.027472	20	cityblock
25	Accept	0.16958	0.046779	0.027431	0.027472	133	cityblock
26	Accept	0.11721	0.05174	0.027431	0.027471	1	cosine
27	Accept	0.034913	0.044981	0.027431	0.02747	3	cityblock
28	Accept	0.14464	0.053652	0.027431	0.02747	21	cosine
29	Accept	0.82045	0.044647	0.027431	0.033238	1	hamming
30	Accept	0.094763	0.041554	0.027431	0.033239	1	euclidean



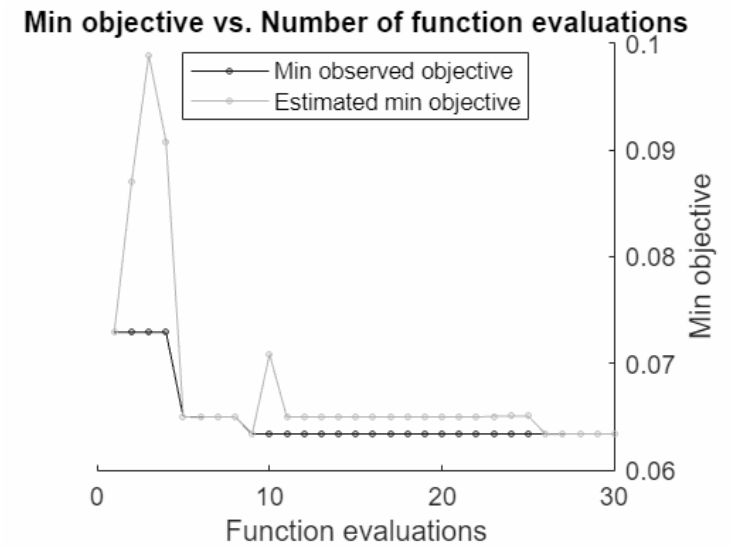


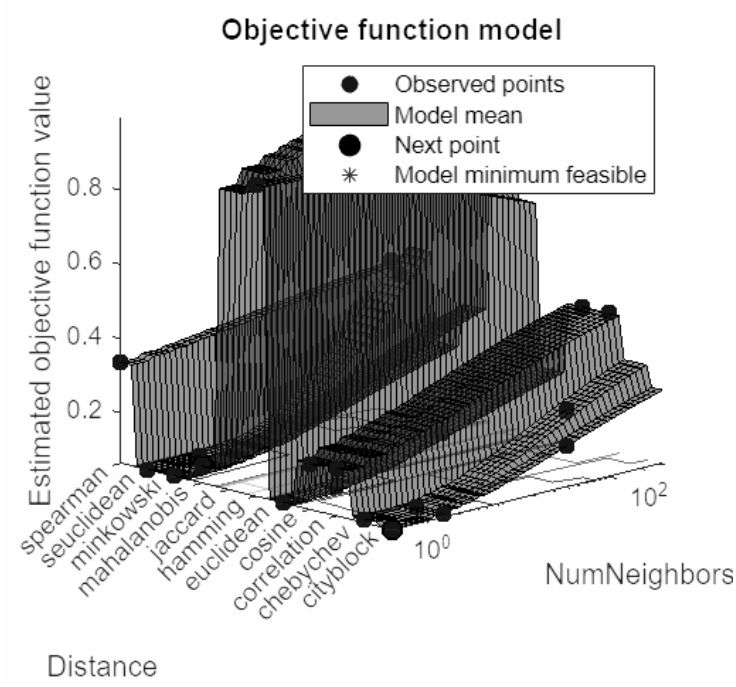
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 17.8658 seconds
Total objective function evaluation time: 1.5569

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         3          minkowski

Observed objective function value = 0.027431
Estimated objective function value = 0.037622
Function evaluation time = 0.047785

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ____________    _________

         5          minkowski

Estimated objective function value = 0.033239
Estimated function evaluation time = 0.047235
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.0729	0.067449	0.0729	0.0729	1	chebychev
2	Accept	0.42631	0.059711	0.0729	0.086952	210	seuclidean
3	Accept	0.99683	0.04988	0.0729	0.098855	35	jaccard
4	Accept	0.10935	0.056417	0.0729	0.090753	16	seuclidean
5	Best	0.064976	0.054858	0.064976	0.064965	1	seuclidean
6	Accept	0.23455	0.050395	0.064976	0.064962	73	chebychev
7	Accept	0.19176	0.056303	0.064976	0.064968	1	correlation
8	Accept	0.43265	0.062958	0.064976	0.064972	316	correlation
9	Best	0.063391	0.048093	0.063391	0.06342	1	cityblock
10	Accept	0.17274	0.049023	0.063391	0.070852	41	cityblock
11	Accept	0.064976	0.045875	0.063391	0.064972	1	seuclidean
12	Accept	0.33597	0.097434	0.063391	0.064974	1	spearman
13	Accept	0.064976	0.050401	0.063391	0.064975	1	minkowski
14	Accept	0.15848	0.064571	0.063391	0.064973	36	minkowski
15	Accept	0.18542	0.056991	0.063391	0.064974	1	cosine
16	Accept	0.43265	0.065876	0.063391	0.064975	316	cosine
17	Accept	0.064976	0.05017	0.063391	0.064976	1	euclidean
18	Accept	0.14897	0.051477	0.063391	0.064974	29	euclidean
19	Accept	0.90333	0.047983	0.063391	0.064966	1	hamming
20	Accept	0.10935	0.051249	0.063391	0.064967	1	mahalanobis

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.33597	0.065403	0.063391	0.064968	169	mahalanobis
22	Accept	0.43265	0.10983	0.063391	0.064968	315	spearman
23	Accept	0.068146	0.052975	0.063391	0.065035	3	seuclidean
24	Accept	0.0729	0.057374	0.063391	0.065048	3	minkowski
25	Accept	0.071315	0.082725	0.063391	0.065058	3	cityblock
26	Accept	0.0729	0.054846	0.063391	0.063367	3	euclidean
27	Accept	0.071315	0.051078	0.063391	0.063368	3	chebychev
28	Accept	0.063391	0.05017	0.063391	0.06338	1	cityblock
29	Accept	0.063391	0.045313	0.063391	0.063384	1	cityblock
30	Accept	0.15214	0.051224	0.063391	0.063382	5	mahalanobis



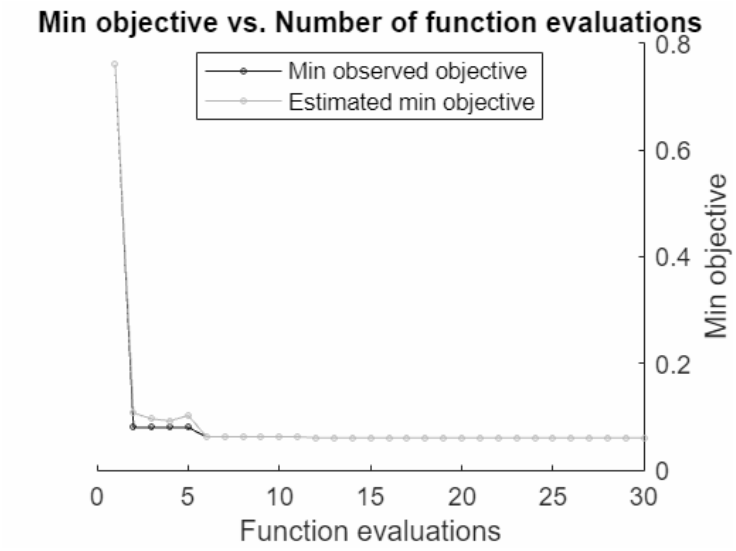


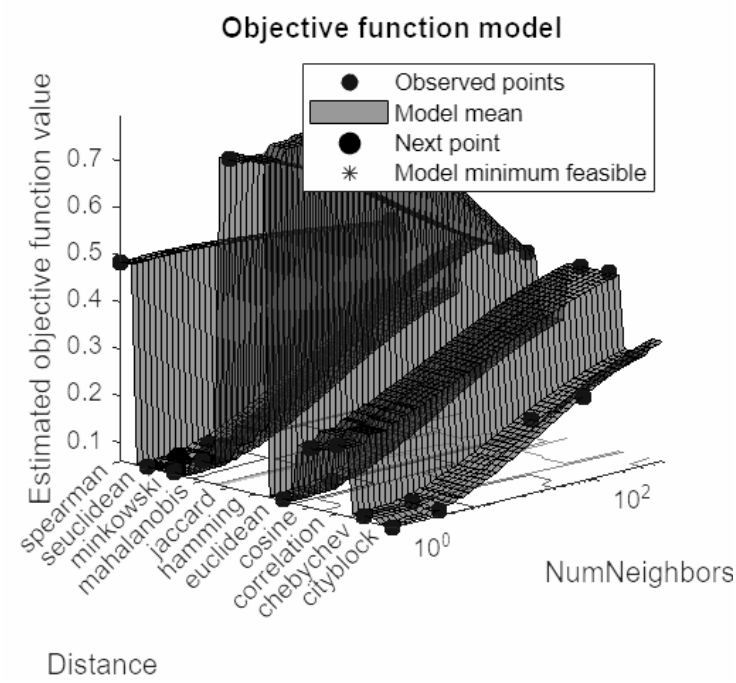
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 18.4623 seconds
Total objective function evaluation time: 1.758

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         1          cityblock

Observed objective function value = 0.063391
Estimated objective function value = 0.063382
Function evaluation time = 0.048093

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ___________    _________

        1          cityblock

Estimated objective function value = 0.063382
Estimated function evaluation time = 0.05431
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.75949	0.069123	0.75949	0.75949	17	hamming
2	Best	0.079844	0.060435	0.079844	0.10687	4	seuclidean
3	Accept	0.18987	0.061658	0.079844	0.09589	47	chebychev
4	Accept	0.18598	0.06064	0.079844	0.092078	1	cosine
5	Accept	0.20351	0.057146	0.079844	0.10234	49	seuclidean
6	Best	0.062317	0.055345	0.062317	0.062221	1	seuclidean
7	Accept	0.43427	0.079917	0.062317	0.062253	513	cosine
8	Accept	0.20643	0.059579	0.062317	0.062268	1	correlation
9	Accept	0.43427	0.079727	0.062317	0.062287	513	correlation
10	Accept	0.48588	0.11857	0.062317	0.062295	1	spearman
11	Accept	0.069133	0.048857	0.062317	0.062279	1	chebychev
12	Best	0.06037	0.053622	0.06037	0.060395	1	cityblock
13	Accept	0.23953	0.066152	0.06037	0.060408	82	cityblock
14	Accept	0.064265	0.048658	0.06037	0.060405	1	minkowski
15	Accept	0.24051	0.062185	0.06037	0.060403	86	minkowski
16	Accept	0.10321	0.056088	0.06037	0.060401	1	mahalanobis
17	Accept	0.40798	0.089333	0.06037	0.060399	221	mahalanobis
18	Accept	0.75949	0.047775	0.06037	0.060407	1	jaccard
19	Accept	0.064265	0.055804	0.06037	0.060405	1	euclidean
20	Accept	0.23856	0.060604	0.06037	0.060404	85	euclidean

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.43427	0.17357	0.06037	0.060405	511	spearman
22	Accept	0.071081	0.057961	0.06037	0.060322	3	cityblock
23	Accept	0.074002	0.051561	0.06037	0.060317	3	euclidean
24	Accept	0.074002	0.047602	0.06037	0.060315	3	minkowski
25	Accept	0.075949	0.053629	0.06037	0.060313	3	chebychev
26	Accept	0.06037	0.046762	0.06037	0.060342	1	cityblock
27	Accept	0.06037	0.05167	0.06037	0.060351	1	cityblock
28	Accept	0.06037	0.049267	0.06037	0.060356	1	cityblock
29	Accept	0.43427	0.068818	0.06037	0.060358	514	jaccard
30	Accept	0.43427	0.062365	0.06037	0.060364	514	hamming



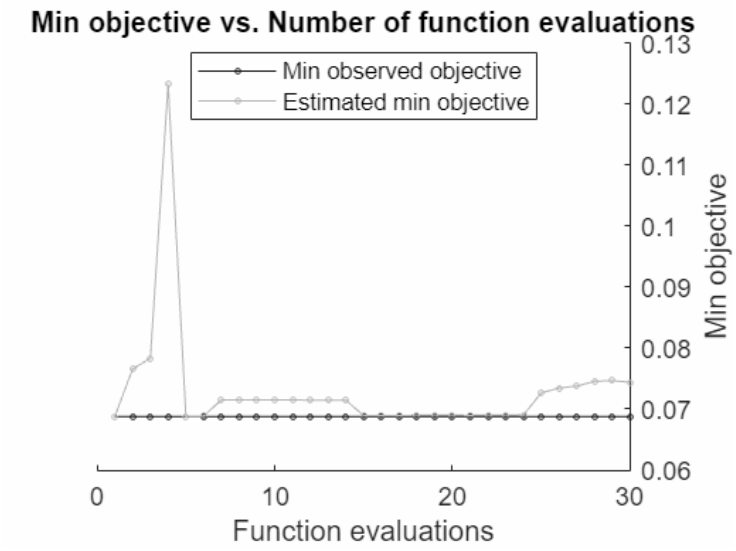


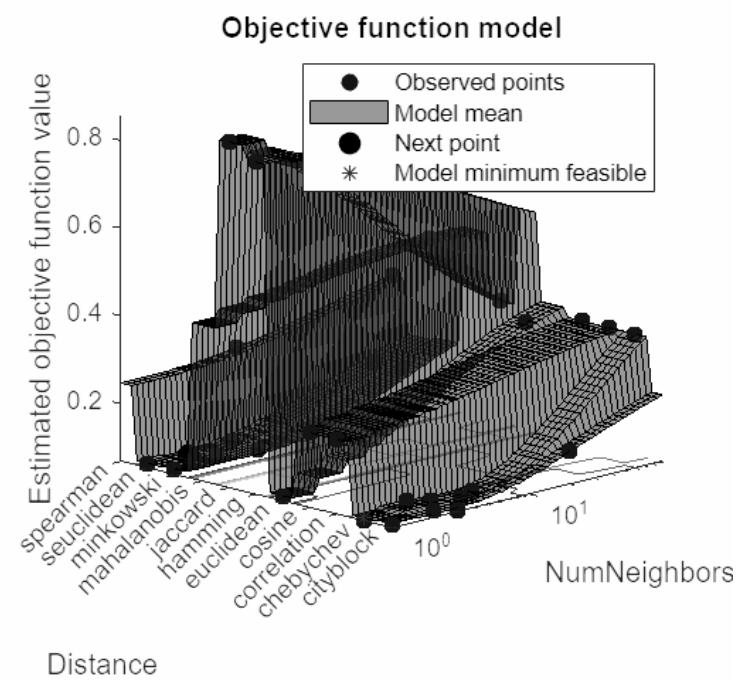
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 18.7807 seconds
Total objective function evaluation time: 1.9544

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         1          cityblock

Observed objective function value = 0.06037
Estimated objective function value = 0.060364
Function evaluation time = 0.053622

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ____________    _________

         1          cityblock

Estimated objective function value = 0.060364
Estimated function evaluation time = 0.051003
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
1	Best	0.068783	0.063802	0.068783	0.068783	5	chebychev
2	Accept	0.26455	0.081923	0.068783	0.076567	7	spearman
3	Accept	0.34392	0.045817	0.068783	0.078351	58	euclidean
4	Accept	0.18519	0.050805	0.068783	0.12332	45	seuclidean
5	Accept	0.068783	0.043495	0.068783	0.068793	5	chebychev
6	Accept	0.079365	0.054392	0.068783	0.068852	6	chebychev
7	Accept	0.068783	0.039395	0.068783	0.071528	1	chebychev
8	Accept	0.074074	0.042147	0.068783	0.0715	1	cityblock
9	Accept	0.2381	0.051541	0.068783	0.071518	1	correlation
10	Accept	0.18519	0.068829	0.068783	0.071531	46	minkowski
11	Accept	0.49735	0.069736	0.068783	0.071493	46	mahalanobis
12	Accept	0.8254	0.047703	0.068783	0.07146	1	hamming
13	Accept	0.2381	0.052165	0.068783	0.071463	1	cosine
14	Accept	0.34392	0.042238	0.068783	0.071467	95	jaccard
15	Accept	0.34392	0.048162	0.068783	0.068921	94	chebychev
16	Accept	0.14286	0.046826	0.068783	0.068946	20	cityblock
17	Accept	0.079365	0.047182	0.068783	0.068965	1	minkowski
18	Accept	0.079365	0.047842	0.068783	0.068982	1	seuclidean
19	Accept	0.34392	0.054129	0.068783	0.068984	95	cosine
20	Accept	0.34392	0.068589	0.068783	0.068986	95	correlation

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumNeighbors	Distance
21	Accept	0.85714	0.043258	0.068783	0.068971	1	jaccard
22	Accept	0.079365	0.057236	0.068783	0.069023	4	minkowski
23	Accept	0.079365	0.046803	0.068783	0.068999	1	euclidean
24	Accept	0.084656	0.046451	0.068783	0.069047	4	seuclidean
25	Accept	0.089947	0.054265	0.068783	0.072655	2	chebychev
26	Accept	0.068783	0.047749	0.068783	0.073442	3	cityblock
27	Accept	0.34392	0.067842	0.068783	0.073761	95	spearman
28	Accept	0.084656	0.047541	0.068783	0.074557	2	cityblock
29	Accept	0.089947	0.046317	0.068783	0.074665	2	minkowski
30	Accept	0.079365	0.044392	0.068783	0.074396	3	chebychev





```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 18.2943 seconds
Total objective function evaluation time: 1.5686

Best observed feasible point:
    NumNeighbors    Distance
    ____________    _________

         5          chebychev

Observed objective function value = 0.068783
Estimated objective function value = 0.074396
Function evaluation time = 0.063802

Best estimated feasible point (according to models):
    NumNeighbors    Distance
```

```
    ___________    _________

         5          chebychev

Estimated objective function value = 0.074396
Estimated function evaluation time = 0.049582
```

K Nearest Neighbours (KNN) Visualisation

The results of every layer of the hypermatrix are summed up and visualised in a heatmap. Additionally the mean accuracy and time taken is calculated and used as a comparison metric between methods.

```
KNN = zeros(m,n);
for j = 1:n
    for k = 1:m
        KNN(j,k) = sum(hyp_KNN(j,k,:));
    end
end
figKNN = figureGen(7,10);
heat_KNN6 = heatmap(KNN, "XDisplayLabels", [0,1,2,3], "YDisplayLabels", [0,1,2,3],
"ColorMethod", "mean", "ColorLimits", [0,100])
```

```
heat_KNN6 =
  HeatmapChart with properties:

        XData: {4×1 cell}
        YData: {4×1 cell}
    ColorData: [4×4 double]

  Show all properties
```

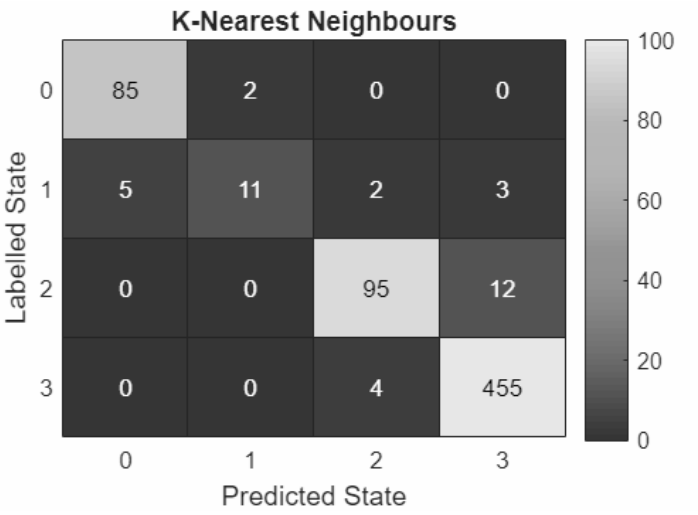
```
heat_KNN6.Colormap = parula(64);
xlabel("Predicted State");
ylabel("Labelled State");
average_acc_KNN = median(acc_KNN)
```

```
average_acc_KNN = 0.9558
```

```
avergage_time_KNN = median(time_KNN)
```

```
avergage_time_KNN = 0.0221
```

```
heat_KNN6.Title = "K-Nearest Neighbours";
saveas(heat_KNN6, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\6_KNN.jpg');
```



Neural Network (NN)

The next method to be applied will be NN.

```
for i = 1:numel(files)
    %Load file i from the folder
    fileName = fullfile(files(i).folder, files(i).name);
    load(fileName);
    %Extract the state vector
    state = stateTT.State;
    %Extract the Principal COmponent Data
    D = PCsTT{:,:};
    %Create a random 90/10 Partition for Training and Test Data
    rng('default');
    Partition_States = cvpartition(state, 'Holdout', 0.10);

    %Seperate the training and testing Ids
    trainingIds = training(Partition_States);
    DTrain = D(trainingIds, :);
    stateTrain = state(trainingIds);

    testIds = test(Partition_States);
    DTest = D(testIds, :);
    stateTest = state(testIds);
```

Begin measuring the time this algorithm will take

```
    tNN = tic;
```

train the decsion tree classifier

```
    classifierNN = fitcnet(DTrain, stateTrain, 'OptimizeHyperparameters', 'auto');
```

end the time measurement

```
    timeNN = toc(tNN);
```

use the tree to predict the test states

```
    TestModel_NN = predict(classifierNN, DTest);
```

measure the accuracy

```
    accuracy_NN = sum(stateTest == TestModel_NN)/length(stateTest);
    %figure;
    %confusionchart(stateTest, TestModel_BT, 'Normalization', 'row-normalized');
```

Create a confusion matrix

```
    [C_NN, order] = confusionmat(stateTest, TestModel_NN);
    %titleStr_BT = strrep([fName,' Binary Tree'],'_','-');
    % title(titleStr_BT);
```

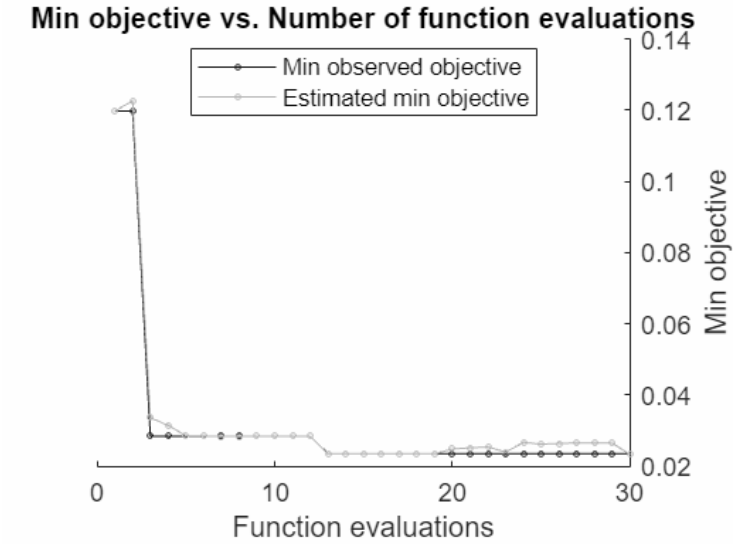
save the confusion matrix as one layer of the hypermatrix and the accuracy and measured time in vector form

```
    hyp_NN(:,:,i) = C_NN;
    acc_NN(i) = accuracy_NN;
    time_NN(i) = timeNN/length(D);
end
```

```
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
1	Best	0.11975	7.9942	0.11975	0.11975	relu	false	5.3123e-08
2	Accept	0.19259	0.23739	0.11975	0.12265	none	false	23.901
3	Best	0.028395	18.868	0.028395	0.033588	tanh	true	1.2517e-05
4	Accept	0.15185	30.909	0.028395	0.031436	none	false	0.00017
5	Accept	0.19259	0.1329	0.028395	0.028535	tanh	true	73.028
6	Accept	0.041975	4.1734	0.028395	0.028426	tanh	true	2.8677e-06
7	Best	0.028395	7.7141	0.028395	0.028361	tanh	true	4.0709e-05
8	Accept	0.19259	0.41766	0.028395	0.028365	relu	false	0.6763
9	Accept	0.11728	10.153	0.028395	0.028382	tanh	true	0.003366
10	Accept	0.037037	5.1763	0.028395	0.028391	relu	true	1.2475e-07
11	Accept	0.02963	75.697	0.028395	0.028395	relu	true	2.3609e-05
12	Accept	0.19259	0.10471	0.028395	0.028389	relu	true	0.171
13	Best	0.023457	29.837	0.023457	0.023447	relu	true	2.3613e-06
14	Accept	0.033333	4.7453	0.023457	0.023449	sigmoid	true	1.4727e-06
15	Accept	0.10741	35.329	0.023457	0.02345	sigmoid	true	0.0002223
16	Accept	0.039506	14.354	0.023457	0.023449	sigmoid	true	1.4239e-08
17	Accept	0.10988	0.88318	0.023457	0.023458	sigmoid	true	2.1181e-07
18	Accept	0.082716	8.7304	0.023457	0.023459	sigmoid	true	4.9696e-07
19	Accept	0.10617	8.978	0.023457	0.023458	relu	true	0.0005893
20	Accept	0.19259	0.12778	0.023457	0.0248	relu	true	38.66

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.030864	28.822	0.023457	0.025136	tanh	true	2.8086e-07
22	Accept	0.030864	48.081	0.023457	0.025364	relu	true	1.4889e-08
23	Accept	0.11852	1.3476	0.023457	0.023918	relu	true	2.9387e-08
24	Accept	0.028395	5.6723	0.023457	0.026666	tanh	true	4.6041e-06
25	Accept	0.19259	0.15763	0.023457	0.026156	sigmoid	true	0.0013988
26	Accept	0.067901	32.223	0.023457	0.026355	sigmoid	true	1.5146e-07
27	Accept	0.059259	4.6058	0.023457	0.026497	tanh	true	1.3805e-08
28	Accept	0.035802	69.953	0.023457	0.026459	relu	false	4.8368e-08
29	Accept	0.15556	1.4279	0.023457	0.026439	none	false	1.7883e-08
30	Accept	0.02716	40.867	0.023457	0.023232	relu	true	5.3696e-07



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 516.294 seconds
Total objective function evaluation time: 497.7204

Best observed feasible point:

Activations	Standardize	Lambda	LayerSizes
relu	true	2.3613e-06	46 229

Observed objective function value = 0.023457
Estimated objective function value = 0.025199
Function evaluation time = 29.8372

Best estimated feasible point (according to models):

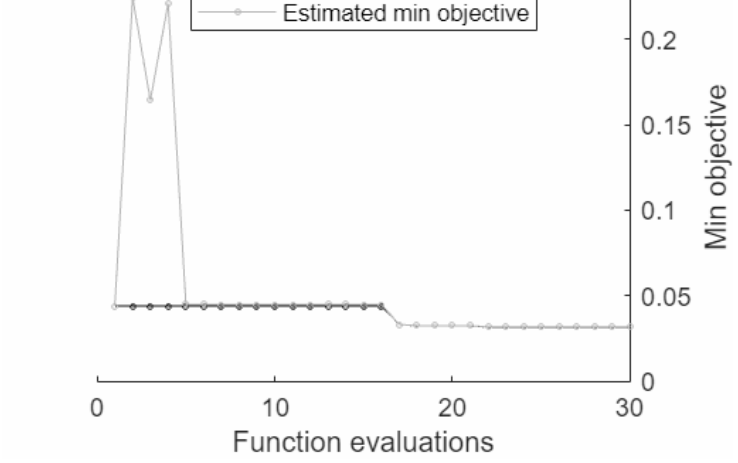
Activations	Standardize	Lambda	LayerSizes
relu	true	5.3696e-07	130 48 240

Estimated objective function value = 0.023232
Estimated function evaluation time = 40.8768

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
1	Best	0.043981	5.2411	0.043981	0.043981	sigmoid	true	1.6294e-05

2	Accept	0.40509	1.1021	0.043981	0.22449	none	true	0.0003835
3	Accept	0.044753	13.839	0.043981	0.16448	tanh	true	3.8369e-08
4	Accept	0.40586	0.094129	0.043981	0.22111	tanh	false	42.388
5	Accept	0.048611	4.4723	0.043981	0.044831	sigmoid	true	1.646e-05
6	Accept	0.40586	0.1368	0.043981	0.044831	tanh	false	44.698
7	Accept	0.40586	0.12889	0.043981	0.044774	tanh	false	37.052
8	Accept	0.40586	0.11831	0.043981	0.044778	sigmoid	false	76.899
9	Accept	0.40586	0.10457	0.043981	0.04478	relu	false	70.761
10	Accept	0.40586	0.10917	0.043981	0.044788	relu	true	0.19905
11	Accept	0.40586	0.23979	0.043981	0.044793	none	false	76.677
12	Accept	0.40586	0.11358	0.043981	0.044813	tanh	true	75.484
13	Accept	0.068673	13.582	0.043981	0.044817	sigmoid	true	7.7514e-09
14	Accept	0.40586	0.18141	0.043981	0.044815	sigmoid	true	65.348
15	Accept	0.091049	6.5249	0.043981	0.044805	relu	false	7.9864e-09
16	Accept	0.083333	5.0183	0.043981	0.044798	sigmoid	false	7.8107e-09
17	Best	0.033179	10.394	0.033179	0.033172	sigmoid	true	5.9359e-07
18	Best	0.032407	27.314	0.032407	0.032439	tanh	false	8.1589e-09
19	Accept	0.3696	0.76678	0.032407	0.03244	none	false	7.7489e-09
20	Accept	0.059414	28.101	0.032407	0.032447	relu	true	7.8267e-09

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.054784	86.896	0.032407	0.032435	sigmoid	false	0.0001022
22	Best	0.031636	51.98	0.031636	0.031667	relu	false	0.0001025
23	Accept	0.054784	8.0809	0.031636	0.031679	tanh	true	0.0002420
24	Accept	0.044753	11.612	0.031636	0.03168	tanh	false	4.9035e-05
25	Accept	0.40586	0.33883	0.031636	0.031682	none	true	76.892
26	Accept	0.1088	41.312	0.031636	0.031674	tanh	true	3.9041e-06
27	Accept	0.40201	0.44635	0.031636	0.031668	tanh	true	0.022602
28	Accept	0.3804	1.9471	0.031636	0.031669	none	false	0.002555
29	Accept	0.40586	1.4299	0.031636	0.031658	sigmoid	true	0.01296
30	Accept	0.40586	0.16673	0.031636	0.031668	relu	true	75.166



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 339.4879 seconds
Total objective function evaluation time: 321.7917

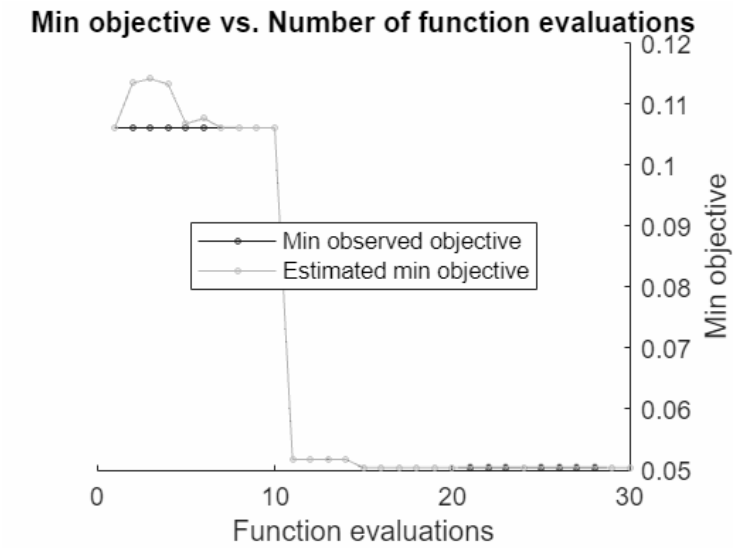
Best observed feasible point:

Activations	Standardize	Lambda	LayerSizes

```
       relu           false       0.00010254    166    121     56

Observed objective function value = 0.031636
Estimated objective function value = 0.031668
Function evaluation time = 51.98

Best estimated feasible point (according to models):
    Activations    Standardize      Lambda         LayerSizes
    ___________    ___________    _________    ________________

       relu           false       0.00010254    166    121     56

Estimated objective function value = 0.031668
Estimated function evaluation time = 51.4712
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
1	Best	0.1061	21.236	0.1061	0.1061	tanh	true	2.992e-05
2	Accept	0.25332	0.15785	0.1061	0.11345	relu	false	128.7
3	Accept	0.25332	2.8684	0.1061	0.11414	sigmoid	false	0.005035
4	Accept	0.21485	4.0572	0.1061	0.11325	none	true	2.643e-08
5	Accept	0.11671	20.244	0.1061	0.10677	tanh	true	6.9377e-07
6	Accept	0.27056	0.76765	0.1061	0.10764	relu	false	1.773e-08
7	Accept	0.25332	3.9697	0.1061	0.10616	tanh	true	0.0102
8	Accept	0.25332	0.36761	0.1061	0.10613	tanh	true	118.5
9	Accept	0.25332	0.26378	0.1061	0.10612	tanh	true	89.29
10	Accept	0.14324	23.645	0.1061	0.10613	tanh	true	8.7342e-06
11	Best	0.051724	53.565	0.051724	0.051764	tanh	true	8.1897e-06
12	Accept	0.25332	0.11628	0.051724	0.051764	relu	false	84.014
13	Accept	0.25332	11.43	0.051724	0.051776	relu	false	0.01014
14	Accept	0.12865	14.43	0.051724	0.051757	tanh	true	4.6424e-06
15	Best	0.050398	74.874	0.050398	0.050421	tanh	true	8.1107e-05
16	Accept	0.066313	67.464	0.050398	0.050422	tanh	true	1.2377e-06
17	Accept	0.071618	94.309	0.050398	0.050412	sigmoid	true	1.5534e-07
18	Accept	0.25332	0.49149	0.050398	0.050417	sigmoid	true	13.319
19	Accept	0.25332	0.32393	0.050398	0.050418	relu	true	118.01
20	Accept	0.21751	2.6397	0.050398	0.050419	relu	true	4.0989e-07

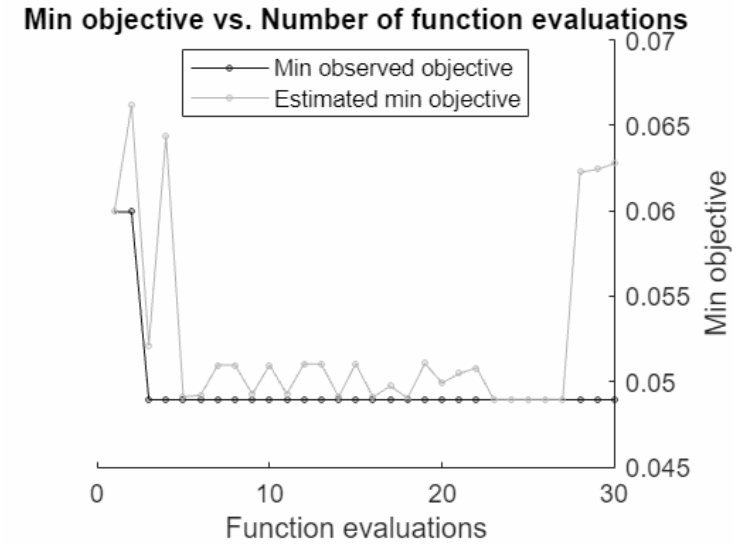
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.05305	72.472	0.050398	0.050478	tanh	true	2.0142e-05
22	Accept	0.25332	0.29038	0.050398	0.050476	tanh	false	85.642
23	Accept	0.2122	49.166	0.050398	0.050473	none	false	1.3557e-08
24	Accept	0.24934	4.1606	0.050398	0.050449	sigmoid	true	7.3623e-05
25	Accept	0.059682	53.647	0.050398	0.050469	tanh	true	2.4842e-06
26	Accept	0.27851	0.55856	0.050398	0.050469	tanh	false	2.4656e-06
27	Accept	0.25332	0.092422	0.050398	0.050468	none	true	132.57
28	Accept	0.070292	25.357	0.050398	0.050478	sigmoid	true	1.3455e-08
29	Accept	0.17905	37.404	0.050398	0.050422	sigmoid	true	1.6862e-08
30	Accept	0.06366	36.618	0.050398	0.050423	sigmoid	true	1.5014e-08




```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 697.7894 seconds
Total objective function evaluation time: 676.9864

Best observed feasible point:
    Activations    Standardize      Lambda         LayerSizes
    ___________    ___________    _________    ________________

       tanh           true        8.1107e-05    158    243    196

Observed objective function value = 0.050398
Estimated objective function value = 0.050423
Function evaluation time = 74.8745

Best estimated feasible point (according to models):
    Activations    Standardize      Lambda         LayerSizes
    ___________    ___________    _________    ________________

       tanh           true        8.1107e-05    158    243    196

Estimated objective function value = 0.050423
Estimated function evaluation time = 74.3266

|=====================================================================================================================
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |  Activations |  Standardize |       Lambda
|      | result |             | runtime     | (observed)  | (estim.)    |              |              |
|=====================================================================================================================
|    1 | Best   |     0.05994 |      23.402 |     0.05994 |     0.05994 |         tanh |         true |   2.4649e-0
|    2 | Accept |     0.18581 |     0.51216 |     0.05994 |    0.066176 |      sigmoid |        false |   1.2693e-0
|    3 | Best   |    0.048951 |      10.168 |    0.048951 |     0.05212 |         relu |        false |    1.487e-0
|    4 | Accept |     0.20779 |     0.20815 |    0.048951 |    0.064396 |         relu |         true |      0.6870
|    5 | Accept |    0.052947 |      7.7139 |    0.048951 |    0.049112 |         relu |        false |   2.1285e-0
|    6 | Accept |     0.20779 |     0.11135 |    0.048951 |    0.049207 |         relu |         true |      0.6044
|    7 | Accept |     0.20779 |      0.1008 |    0.048951 |    0.050969 |         relu |         true |      0.1799
|    8 | Accept |     0.20779 |     0.10557 |    0.048951 |    0.050973 |         tanh |        false |       95.08
|    9 | Accept |     0.21179 |     0.81048 |    0.048951 |    0.049274 |         none |         true |   7.8425e-0
|   10 | Accept |     0.21479 |      1.0249 |    0.048951 |    0.050973 |         tanh |        false |   1.3546e-0
|   11 | Accept |     0.20779 |     0.12699 |    0.048951 |    0.049284 |      sigmoid |         true |       93.43
|   12 | Accept |     0.15485 |      7.8519 |    0.048951 |    0.051036 |         relu |        false |   1.0236e-0
|   13 | Accept |     0.20779 |     0.34331 |    0.048951 |    0.051045 |         tanh |        false |       96.51
|   14 | Accept |     0.20779 |     0.19951 |    0.048951 |    0.049102 |         relu |         true |       4.286
```

15	Accept	0.18282	0.99214	0.048951	0.051048	relu	true	9.8202e-08
16	Accept	0.20779	0.10409	0.048951	0.049085	relu	false	97.67
17	Accept	0.10889	32.174	0.048951	0.049744	relu	false	4.0346e-06
18	Accept	0.073926	2.6732	0.048951	0.049043	tanh	true	0.0001351
19	Accept	0.20779	0.12222	0.048951	0.051074	tanh	false	0.6098
20	Accept	0.20779	0.71495	0.048951	0.049919	tanh	true	0.1008

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.050949	9.4044	0.048951	0.050481	tanh	true	5.3273e-06
22	Accept	0.068931	7.2585	0.048951	0.050804	tanh	true	8.7674e-06
23	Accept	0.19081	0.95891	0.048951	0.04897	tanh	true	1.0582e-06
24	Accept	0.085914	5.3374	0.048951	0.048973	tanh	true	1.3821e-05
25	Accept	0.061938	12.904	0.048951	0.048973	tanh	true	2.9488e-05
26	Accept	0.16484	14.473	0.048951	0.048966	relu	false	0.001065
27	Accept	0.18182	0.474	0.048951	0.04897	tanh	true	5.7331e-05
28	Accept	0.11988	5.5051	0.048951	0.062259	tanh	true	0.0002490
29	Accept	0.052947	6.8483	0.048951	0.062433	relu	false	2.0321e-06
30	Accept	0.062937	30.897	0.048951	0.062762	tanh	true	1.079e-05



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 200.4643 seconds
Total objective function evaluation time: 183.5202

Best observed feasible point:

Activations	Standardize	Lambda	LayerSizes
relu	false	1.487e-05	160

Observed objective function value = 0.048951
Estimated objective function value = 0.062342
Function evaluation time = 10.1676

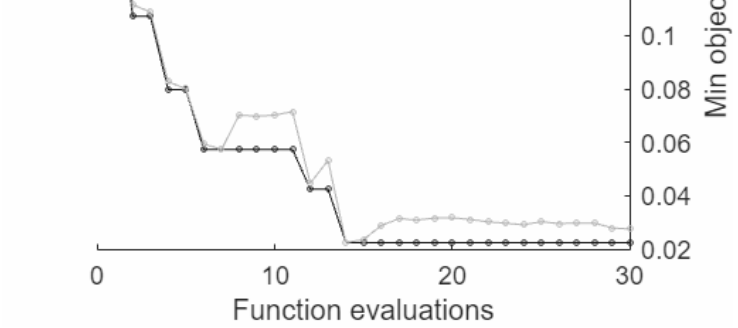
Best estimated feasible point (according to models):

Activations	Standardize	Lambda	LayerSizes
relu	false	2.1285e-05	66

```
Estimated objective function value = 0.062762
Estimated function evaluation time = 7.7318
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
1	Best	0.16958	0.1282	0.16958	0.16958	sigmoid	true	4.820
2	Best	0.10723	2.7631	0.10723	0.11152	none	true	4.4757e-0
3	Accept	0.16708	8.0677	0.10723	0.10885	tanh	false	0.008629
4	Best	0.0798	4.269	0.0798	0.082684	relu	false	7.0581e-0
5	Accept	0.11222	14.892	0.0798	0.080101	sigmoid	true	2.5913e-0
6	Best	0.057357	5.1362	0.057357	0.059403	relu	false	8.4524e-0
7	Accept	0.16958	0.093941	0.057357	0.057381	sigmoid	true	5.359
8	Accept	0.16958	0.095268	0.057357	0.070184	sigmoid	true	16.01
9	Accept	0.16958	0.085448	0.057357	0.06975	sigmoid	true	9.194
10	Accept	0.10723	2.0976	0.057357	0.070228	relu	false	0.002597
11	Accept	0.089776	7.7525	0.057357	0.071474	relu	false	2.5914e-0
12	Best	0.042394	4.5566	0.042394	0.044693	relu	false	1.2324e-0
13	Accept	0.097257	1.129	0.042394	0.053213	relu	false	2.7912e-0
14	Best	0.022444	4.3491	0.022444	0.022442	relu	false	8.7706e-0
15	Accept	0.16958	0.14405	0.022444	0.023579	relu	false	35.34
16	Accept	0.037406	4.583	0.022444	0.028855	relu	false	3.4409e-0
17	Accept	0.064838	8.6558	0.022444	0.0314	relu	false	3.2149e-0
18	Accept	0.032419	4.1103	0.022444	0.03086	relu	false	2.0271e-0
19	Accept	0.034913	4.3639	0.022444	0.031604	relu	false	7.2988e-0
20	Accept	0.044888	4.3893	0.022444	0.031699	relu	false	6.775e-0

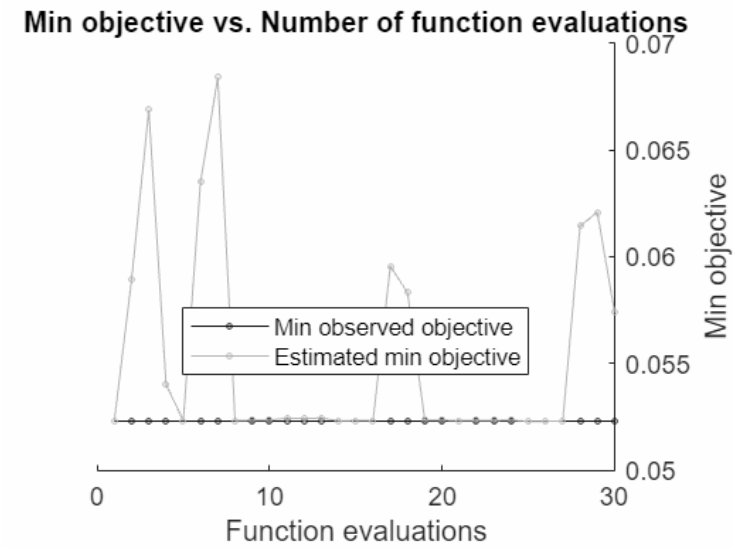
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.029925	4.3063	0.022444	0.031108	relu	false	1.2738e-0
22	Accept	0.027431	4.0777	0.022444	0.030208	relu	false	1.7131e-0
23	Accept	0.032419	4.1803	0.022444	0.029699	relu	false	3.5835e-0
24	Accept	0.027431	4.5089	0.022444	0.029084	relu	false	1.8825e-0
25	Accept	0.042394	4.2473	0.022444	0.030275	relu	false	6.9636e-0
26	Accept	0.027431	4.3127	0.022444	0.029561	relu	false	6.7775e-0
27	Accept	0.029925	4.2564	0.022444	0.029755	relu	false	3.8084e-0
28	Accept	0.029925	4.3835	0.022444	0.029729	relu	false	2.8348e-0
29	Accept	0.029925	16.574	0.022444	0.027893	relu	false	4.9995e-0
30	Accept	0.027431	5.0674	0.022444	0.02752	relu	false	1.5799e-0



```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 154.6454 seconds
Total objective function evaluation time: 137.5771

Best observed feasible point:
    Activations    Standardize      Lambda      LayerSizes
    ___________    ___________    __________    __________

       relu           false       8.7706e-06        9

Observed objective function value = 0.022444
Estimated objective function value = 0.030292
Function evaluation time = 4.3491

Best estimated feasible point (according to models):
    Activations    Standardize      Lambda      LayerSizes
    ___________    ___________    __________    __________

       relu           false       1.5799e-05        17

Estimated objective function value = 0.02752
Estimated function evaluation time = 4.7994
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
1	Best	0.052298	154.35	0.052298	0.052298	tanh	false	7.0048e-05
2	Accept	0.2187	5.4838	0.052298	0.058914	none	true	1.8692e-06
3	Accept	0.43265	0.11203	0.052298	0.066872	relu	true	0.24538
4	Accept	0.28526	6.0682	0.052298	0.054014	none	true	6.9046e-06
5	Accept	0.077655	76.675	0.052298	0.052314	relu	true	4.8415e-08
6	Accept	0.1141	2.9428	0.052298	0.063503	tanh	false	5.0685e-06
7	Accept	0.24089	2.0885	0.052298	0.068392	tanh	false	0.0023595
8	Accept	0.43265	0.12469	0.052298	0.052315	relu	true	0.40960
9	Accept	0.099842	5.313	0.052298	0.052342	relu	true	1.5993e-08
10	Accept	0.18225	3.902	0.052298	0.052339	relu	true	8.812e-05
11	Accept	0.063391	4.8509	0.052298	0.052433	tanh	false	3.4042e-05
12	Accept	0.11252	11.954	0.052298	0.052416	tanh	false	3.0609e-08
13	Accept	0.43265	0.24279	0.052298	0.052438	relu	true	157.17
14	Accept	0.29952	5.6862	0.052298	0.05231	relu	true	7.5281e-07
15	Accept	0.31062	0.70533	0.052298	0.052318	relu	true	1.5968e-08
16	Accept	0.22821	7.6473	0.052298	0.052317	tanh	false	0.0010384
17	Accept	0.43265	0.085792	0.052298	0.05955	relu	true	4.27
18	Accept	0.063391	5.5687	0.052298	0.058312	tanh	false	1.0536e-05
19	Accept	0.066561	5.8606	0.052298	0.052336	tanh	false	3.5992e-05
20	Accept	0.43265	0.10542	0.052298	0.052336	tanh	false	93.109

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.21712	10.706	0.052298	0.052323	tanh	false	1.16e-05
22	Accept	0.43265	0.14217	0.052298	0.052328	tanh	false	116.29
23	Accept	0.066561	5.9603	0.052298	0.052328	tanh	false	3.715e-06
24	Accept	0.34548	6.6595	0.052298	0.052339	relu	true	2.564e-08
25	Accept	0.052298	36.446	0.052298	0.052281	tanh	false	3.1848e-05
26	Accept	0.07607	17.79	0.052298	0.052281	relu	true	2.4448e-08
27	Accept	0.060222	4.291	0.052298	0.052295	tanh	false	1.0232e-05
28	Accept	0.12203	5.3399	0.052298	0.061427	tanh	false	3.8167e-05
29	Accept	0.063391	13.881	0.052298	0.062078	tanh	false	6.7913e-07
30	Accept	0.068146	19.049	0.052298	0.057408	tanh	false	1.1282e-05

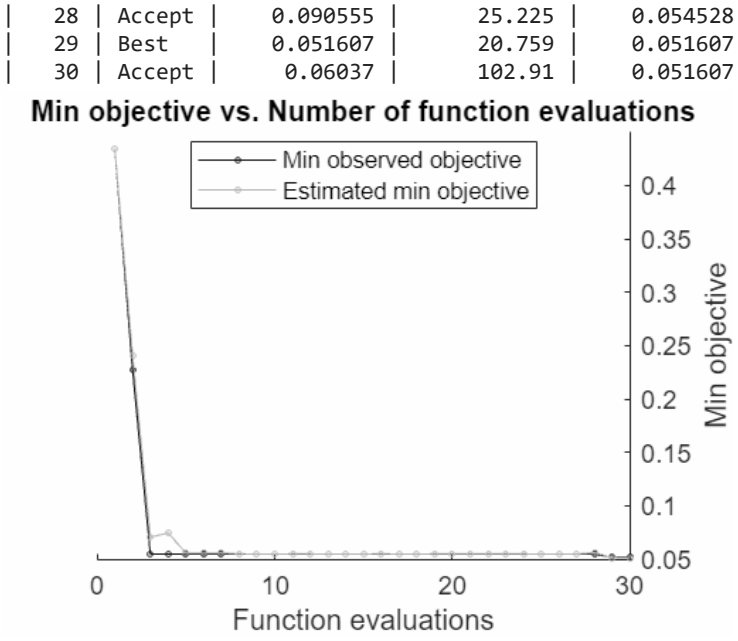



```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 437.1436 seconds
Total objective function evaluation time: 420.0343

Best observed feasible point:
    Activations    Standardize      Lambda         LayerSizes
    ___________    ___________    _________    ________________

       tanh           false       7.0048e-05    233    277    297

Observed objective function value = 0.052298
Estimated objective function value = 0.05847
Function evaluation time = 154.3522

Best estimated feasible point (according to models):
    Activations    Standardize      Lambda      LayerSizes
    ___________    ___________    _________    __________

       tanh           false       1.0536e-05        50

Estimated objective function value = 0.057408
Estimated function evaluation time = 5.8397
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
1	Best	0.43427	0.14667	0.43427	0.43427	sigmoid	false	19.984
2	Best	0.22687	5.4307	0.22687	0.2405	tanh	true	1.998e-08
3	Best	0.054528	8.5394	0.054528	0.070484	sigmoid	false	1.5826e-05
4	Accept	0.43427	2.0316	0.054528	0.074426	sigmoid	true	0.003741
5	Accept	0.057449	8.7897	0.054528	0.055777	sigmoid	false	2.025e-05
6	Accept	0.43427	0.79555	0.054528	0.055743	sigmoid	false	0.001329
7	Accept	0.087634	26.651	0.054528	0.055834	sigmoid	false	7.0685e-06
8	Accept	0.25706	5.0229	0.054528	0.054987	sigmoid	false	8.5049e-08
9	Accept	0.15969	5.3281	0.054528	0.054563	sigmoid	false	9.8337e-09
10	Accept	0.12074	7.319	0.054528	0.054642	sigmoid	false	1.6783e-08
11	Accept	0.43427	0.30059	0.054528	0.054887	sigmoid	false	26.68
12	Accept	0.43427	0.14408	0.054528	0.054836	sigmoid	false	45.291
13	Accept	0.27945	0.53704	0.054528	0.054672	sigmoid	false	1.6418e-05
14	Accept	0.43427	0.53863	0.054528	0.05472	sigmoid	false	0.001958

15	Accept	0.43427	0.20275	0.054528	0.054712	sigmoid	false	51.574
16	Accept	0.40993	6.2262	0.054528	0.054964	sigmoid	false	0.0009889
17	Accept	0.24245	5.093	0.054528	0.054672	sigmoid	false	2.295e-05
18	Accept	0.43427	0.34071	0.054528	0.054674	sigmoid	false	41.31
19	Accept	0.088608	7.6048	0.054528	0.05489	sigmoid	false	3.3518e-07
20	Accept	0.063291	9.2953	0.054528	0.054934	sigmoid	false	1.2363e-05

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.43427	0.30286	0.054528	0.054927	sigmoid	false	95.360
22	Accept	0.058423	8.0428	0.054528	0.054947	sigmoid	false	3.5324e-06
23	Accept	0.43427	0.22262	0.054528	0.054865	sigmoid	false	85.245
24	Accept	0.063291	10.232	0.054528	0.054917	sigmoid	false	4.1303e-06
25	Accept	0.057449	7.896	0.054528	0.054567	sigmoid	false	6.7716e-06
26	Accept	0.06816	9.3084	0.054528	0.054794	sigmoid	false	3.3336e-06
27	Accept	0.28724	0.94717	0.054528	0.054706	sigmoid	false	5.8824e-07
28	Accept	0.090555	25.225	0.054528	0.056128	sigmoid	false	1.0978e-08
29	Best	0.051607	20.759	0.051607	0.051222	sigmoid	false	7.2686e-06
30	Accept	0.06037	102.91	0.051607	0.051081	tanh	true	2.5943e-08



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 305.0343 seconds
Total objective function evaluation time: 286.1818

Best observed feasible point:

Activations	Standardize	Lambda	LayerSizes		
sigmoid	false	7.2686e-06	27	124	24

Observed objective function value = 0.051607
Estimated objective function value = 0.051081
Function evaluation time = 20.7586

Best estimated feasible point (according to models):

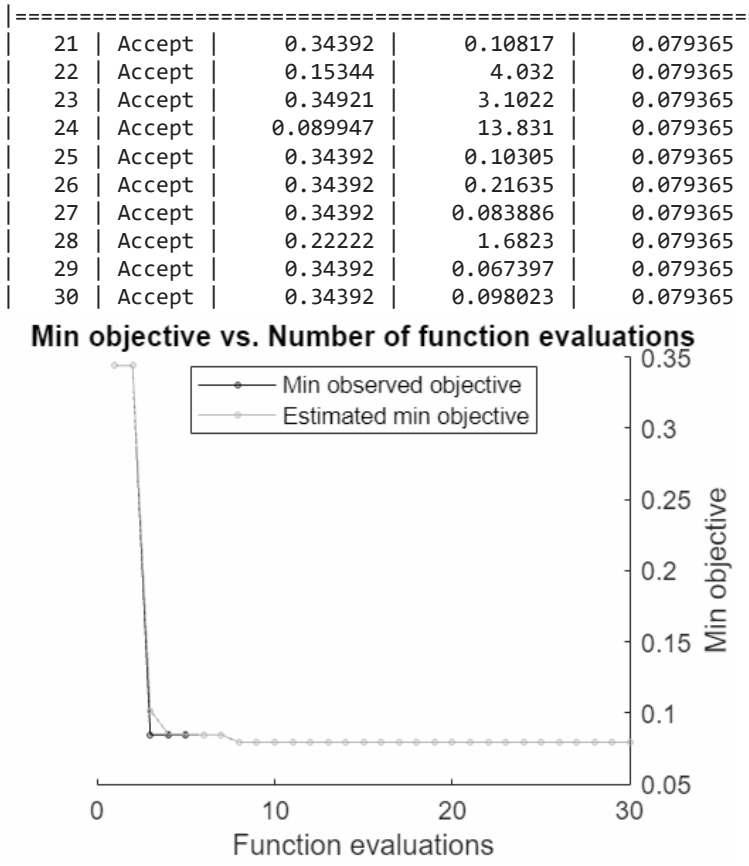
Activations	Standardize	Lambda	LayerSizes		
sigmoid	false	7.2686e-06	27	124	24

Estimated objective function value = 0.051081

```
Estimated function evaluation time = 20.7432
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
1	Best	0.34392	0.22621	0.34392	0.34392	tanh	true	0.4876
2	Accept	0.34392	0.22172	0.34392	0.34392	none	false	0.08246
3	Best	0.084656	8.6852	0.084656	0.10177	sigmoid	false	5.8137e-05
4	Accept	0.34392	0.10752	0.084656	0.085229	tanh	false	49.82
5	Accept	0.34392	0.26968	0.084656	0.084905	sigmoid	false	5.667
6	Accept	0.34392	0.11258	0.084656	0.084682	sigmoid	false	461.5
7	Accept	0.095238	4.2901	0.084656	0.084679	tanh	false	0.0001278
8	Best	0.079365	11.012	0.079365	0.079379	tanh	false	6.0495e-08
9	Accept	0.084656	8.5845	0.079365	0.079376	relu	false	4.3108e-06
10	Accept	0.079365	13.006	0.079365	0.079376	tanh	true	5.3652e-08
11	Accept	0.24339	2.3386	0.079365	0.079376	none	false	5.3417e-08
12	Accept	0.10053	4.2001	0.079365	0.079375	relu	true	5.4407e-08
13	Accept	0.14815	5.3506	0.079365	0.079374	relu	true	0.005327
14	Accept	0.084656	11.002	0.079365	0.079373	sigmoid	true	5.3208e-08
15	Accept	0.34392	0.13788	0.079365	0.079374	none	true	500.3
16	Accept	0.095238	4.1461	0.079365	0.079374	none	true	4.8739e-05
17	Accept	0.34392	0.11927	0.079365	0.079375	relu	false	522.2
18	Accept	0.10053	3.7092	0.079365	0.079374	none	true	5.5189e-08
19	Accept	0.34392	0.11039	0.079365	0.079375	sigmoid	true	522.0
20	Accept	0.34392	0.68274	0.079365	0.079375	sigmoid	true	0.004161

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Activations	Standardize	Lambda
21	Accept	0.34392	0.10817	0.079365	0.079376	relu	true	528.7
22	Accept	0.15344	4.032	0.079365	0.079376	sigmoid	false	5.3981e-08
23	Accept	0.34921	3.1022	0.079365	0.079377	relu	false	0.01747
24	Accept	0.089947	13.831	0.079365	0.079376	relu	false	5.2995e-08
25	Accept	0.34392	0.10305	0.079365	0.079376	none	false	527.2
26	Accept	0.34392	0.21635	0.079365	0.079376	none	true	0.0352
27	Accept	0.34392	0.083886	0.079365	0.079377	tanh	true	516.5
28	Accept	0.22222	1.6823	0.079365	0.079378	none	true	1.0305e-06
29	Accept	0.34392	0.067397	0.079365	0.079378	none	true	0.8055
30	Accept	0.34392	0.098023	0.079365	0.079376	tanh	false	0.3375




```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
```

```
Total elapsed time: 119.3387 seconds
Total objective function evaluation time: 101.6366

Best observed feasible point:
    Activations    Standardize      Lambda      LayerSizes
    ___________    ___________    __________    __________

       tanh           false       6.0495e-08    15    161

Observed objective function value = 0.079365
Estimated objective function value = 0.079376
Function evaluation time = 11.0122

Best estimated feasible point (according to models):
    Activations    Standardize      Lambda      LayerSizes
    ___________    ___________    __________    __________

       tanh           false       6.0495e-08    15    161

Estimated objective function value = 0.079376
Estimated function evaluation time = 11.011
```

Neural Network Visualisation

The results of every layer of the hypermatrix are summed up and visualised in a heatmap. Additionally the mean accuracy and time taken is calculated and used as a comparison metric between methods.

```
NN = zeros(m,n);
for j = 1:n
    for k = 1:m
        NN(j,k) = sum(hyp_NN(j,k,:));
    end
end
figNN = figureGen(7,10);
heat_NN6 = heatmap(NN, "XDisplayLabels", [0,1,2,3], "YDisplayLabels", [0,1,2,3],
"ColorMethod", "mean", "ColorLimits", [0,100])
```

```
heat_NN6 =
  HeatmapChart with properties:

       XData: {4×1 cell}
       YData: {4×1 cell}
    ColorData: [4×4 double]

  Show all properties
```

```
heat_NN6.Colormap = parula(64);
xlabel("Predicted State");
ylabel("Labelled State");
average_acc_NN = median(acc_NN)
```

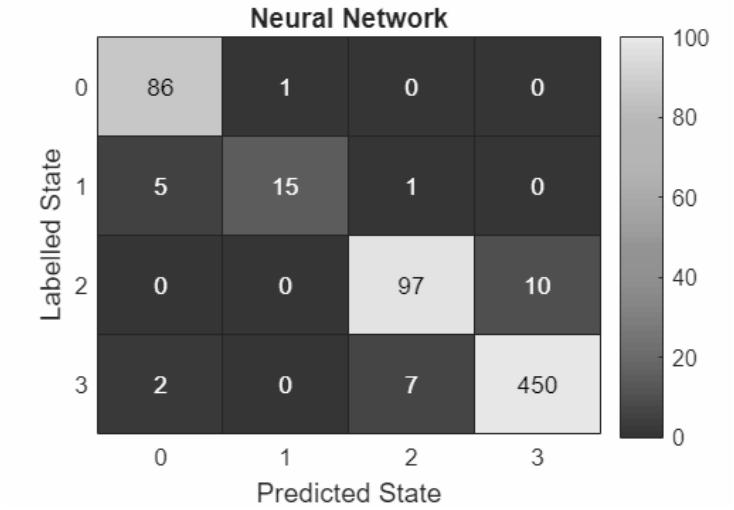
```
average_acc_NN = 0.9744
```

```
average_time_NN = median(time_NN)
```

```
average_time_NN = 0.4681
```

```
heat_NN6.Title = "Neural Network";
```

```
saveas(heat_NN6, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\6_NN.jpg');
```



Ensemble Classification

The next method to be applied will be Ensemble Classification.

```
for i = 1:numel(files)
    %Load file i from the folder
    fileName = fullfile(files(i).folder, files(i).name);
    load(fileName);
    %Extract the state vector
    state = stateTT.State;
    %Extract the Principal COmponent Data
    D = PCsTT{:,:};
    %Create a random 90/10 Partition for Training and Test Data
    rng('default');
    Partition_States = cvpartition(state, 'Holdout', 0.10);

    %Seperate the training and testing Ids
    trainingIds = training(Partition_States);
    DTrain = D(trainingIds, :);
    stateTrain = state(trainingIds);

    testIds = test(Partition_States);
    DTest = D(testIds, :);
    stateTest = state(testIds);
```

Begin measuring the time this algorithm will take

```
    tEn = tic;
```

train the decsion tree classifier

```
    %template = templateTree('MinLeafSize', 2);
    classifierEn = fitcensemble(DTrain, stateTrain, 'OptimizeHyperparameters',
'auto');
```

end the time measurement

```
    timeEn = toc(tEn);
```

use the tree to predict the test states

```
    TestModel_En = predict(classifierEn, DTest);
```

measure the accuracy

```
    accuracy_En = sum(stateTest == TestModel_En)/length(stateTest);
    %figure;
    %confusionchart(stateTest, TestModel_BT, 'Normalization', 'row-normalized');
```

Create a confusion matrix

```
    [C_En, order] = confusionmat(stateTest, TestModel_En);
    %titleStr_BT = strrep([fName,' Binary Tree'],'_','-');
    % title(titleStr_BT);
```

save the confusion matrix as one layer of the hypermatrix and the accuracy and measured time in vector form

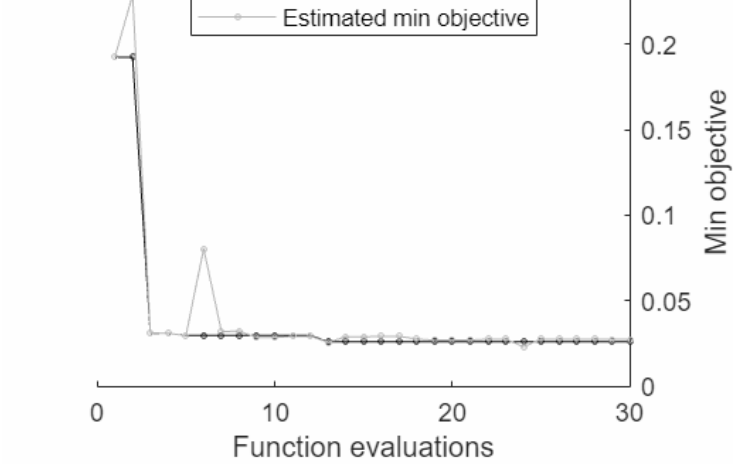
```
    hyp_En(:,:,i) = C_En;
    acc_En(i) = accuracy_En;
    time_En(i) = timeEn/length(D);
end
```

```
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
1	Best	0.19259	11.476	0.19259	0.19259	AdaBoostM2	487	0.2155
2	Accept	0.73457	1.0189	0.19259	0.2288	RUSBoost	19	0.007133
3	Best	0.030864	13.029	0.030864	0.031108	Bag	421	
4	Accept	0.11111	0.59359	0.030864	0.030939	AdaBoostM2	18	0.3947
5	Best	0.02963	1.0498	0.02963	0.029562	AdaBoostM2	35	0.9998
6	Accept	0.066667	0.42642	0.02963	0.080423	AdaBoostM2	10	0.001051
7	Accept	0.058025	0.39116	0.02963	0.031962	AdaBoostM2	10	0.8346
8	Accept	0.030864	0.43992	0.02963	0.032172	AdaBoostM2	10	0.7614
9	Accept	0.030864	0.36674	0.02963	0.028404	AdaBoostM2	10	0.9729
10	Accept	0.96667	0.68402	0.02963	0.028394	RUSBoost	22	0.999
11	Accept	0.066667	0.44582	0.02963	0.029094	AdaBoostM2	10	0.004038
12	Accept	0.19259	0.39721	0.02963	0.029129	Bag	10	
13	Best	0.025926	0.45475	0.025926	0.02561	Bag	10	
14	Accept	0.032099	0.44534	0.025926	0.028828	Bag	10	
15	Accept	0.02963	0.68163	0.025926	0.028855	AdaBoostM2	22	0.9693
16	Accept	0.030864	0.44873	0.025926	0.029073	Bag	10	

17	Accept	0.028395	0.45248	0.025926	0.029081	AdaBoostM2	10	0.142
18	Accept	0.02716	0.40802	0.025926	0.027502	AdaBoostM2	11	0.2745
19	Accept	0.025926	0.4543	0.025926	0.026848	AdaBoostM2	12	0.3125
20	Accept	0.02716	0.39274	0.025926	0.026961	AdaBoostM2	12	0.2454

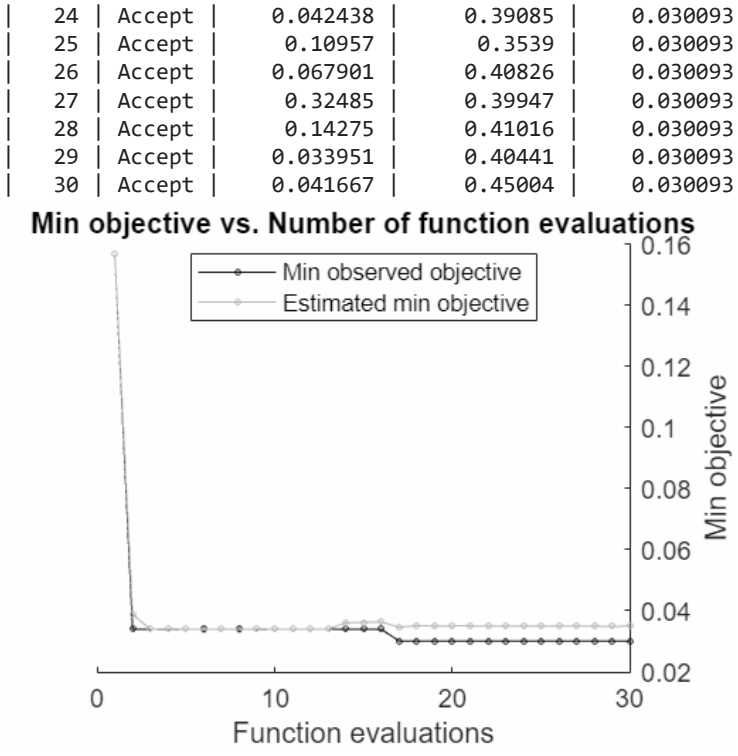
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
21	Accept	0.02716	0.36658	0.025926	0.026971	AdaBoostM2	11	0.2733
22	Accept	0.02963	0.33758	0.025926	0.027417	AdaBoostM2	10	0.3340
23	Accept	0.96667	2.2546	0.025926	0.02744	RUSBoost	80	0.001047
24	Accept	0.19259	0.32936	0.025926	0.023065	AdaBoostM2	10	0.001002
25	Accept	0.02963	0.42985	0.025926	0.027423	AdaBoostM2	11	0.9153
26	Accept	0.030864	0.35155	0.025926	0.027431	AdaBoostM2	10	0.1076
27	Accept	0.79753	0.36197	0.025926	0.027453	RUSBoost	10	0.9767
28	Accept	0.04321	0.39176	0.025926	0.027456	AdaBoostM2	10	0.001001
29	Accept	0.030864	0.38954	0.025926	0.02739	AdaBoostM2	10	0.2421
30	Accept	0.060494	0.3905	0.025926	0.027375	AdaBoostM2	12	0.001014




```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 54.5319 seconds
Total objective function evaluation time: 39.6599

Best observed feasible point:
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag              10               NaN            1

Observed objective function value = 0.025926
Estimated objective function value = 0.029185
Function evaluation time = 0.45475

Best estimated feasible point (according to models):
      Method      NumLearningCycles    LearnRate    MinLeafSize
    _________    _________________    _________    ___________

    AdaBoostM2           12              0.2454           1

Estimated objective function value = 0.027375
Estimated function evaluation time = 0.42826
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
1	Best	0.15664	7.8008	0.15664	0.15664	RUSBoost	254	0.02146
2	Best	0.033951	10.924	0.033951	0.038829	Bag	358	
3	Accept	0.15355	0.48845	0.033951	0.033975	RUSBoost	12	0.04523
4	Accept	0.08642	2.9071	0.033951	0.034052	AdaBoostM2	109	0.006226
5	Accept	0.40586	0.39772	0.033951	0.033969	Bag	11	
6	Accept	0.033951	0.42142	0.033951	0.033935	Bag	10	
7	Accept	0.73611	0.4215	0.033951	0.033956	RUSBoost	12	0.04521
8	Accept	0.051698	0.55641	0.033951	0.033872	Bag	14	
9	Accept	0.03858	0.43436	0.033951	0.034026	Bag	10	
10	Accept	0.1466	3.1244	0.033951	0.033981	RUSBoost	96	0.001974
11	Accept	0.40586	0.34009	0.033951	0.033964	AdaBoostM2	10	0.5732
12	Accept	0.064043	0.35418	0.033951	0.033962	AdaBoostM2	10	0.001123
13	Accept	0.046296	0.39505	0.033951	0.033974	AdaBoostM2	10	0.7243
14	Accept	0.03858	0.43567	0.033951	0.036163	Bag	10	
15	Accept	0.042438	0.47587	0.033951	0.036226	Bag	11	
16	Accept	0.037037	0.44655	0.033951	0.03636	Bag	10	
17	Best	0.030093	0.47874	0.030093	0.034634	Bag	10	
18	Accept	0.037037	0.44334	0.030093	0.035144	Bag	10	
19	Accept	0.10494	0.42136	0.030093	0.034998	Bag	10	
20	Accept	0.074846	0.41288	0.030093	0.035046	Bag	10	

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
21	Accept	0.064043	0.37745	0.030093	0.035053	AdaBoostM2	10	0.008901
22	Accept	0.73611	0.36782	0.030093	0.034966	RUSBoost	10	0.08250
23	Accept	0.073302	0.37693	0.030093	0.034969	AdaBoostM2	11	0.001617
24	Accept	0.042438	0.39085	0.030093	0.034973	AdaBoostM2	11	0.7061
25	Accept	0.10957	0.3539	0.030093	0.034963	AdaBoostM2	10	0.001038
26	Accept	0.067901	0.40826	0.030093	0.034969	AdaBoostM2	11	0.8838
27	Accept	0.32485	0.39947	0.030093	0.03496	RUSBoost	10	0.7500
28	Accept	0.14275	0.41016	0.030093	0.034947	Bag	10	
29	Accept	0.033951	0.40441	0.030093	0.034896	Bag	10	
30	Accept	0.041667	0.45004	0.030093	0.034976	Bag	10	

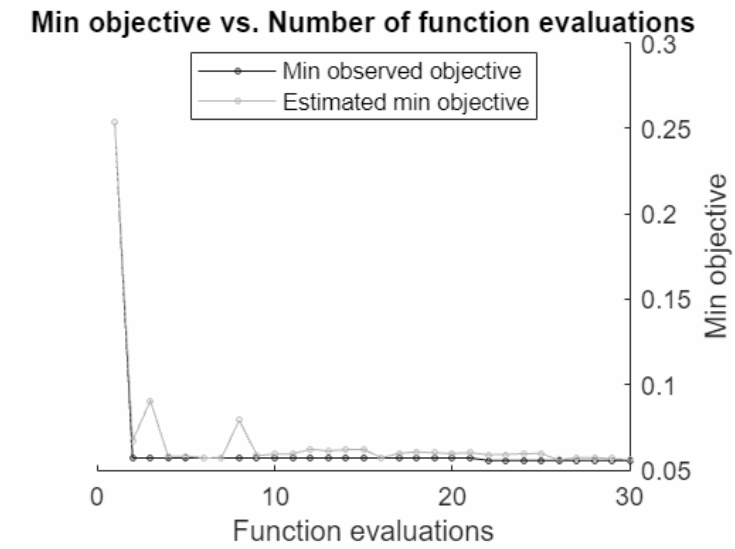


Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 50.9581 seconds
Total objective function evaluation time: 35.6195

```
Best observed feasible point:
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag              10               NaN            2

Observed objective function value = 0.030093
Estimated objective function value = 0.034976
Function evaluation time = 0.47874

Best estimated feasible point (according to models):
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag              10               NaN            2

Estimated objective function value = 0.034976
Estimated function evaluation time = 0.43985
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
1	Best	0.25332	3.5018	0.25332	0.25332	Bag	133	
2	Best	0.057029	13.611	0.057029	0.067182	Bag	462	
3	Accept	0.87931	2.6591	0.057029	0.090659	RUSBoost	95	0.9103
4	Accept	0.98408	5.7604	0.057029	0.057993	RUSBoost	226	0.00254
5	Accept	0.087533	0.37156	0.057029	0.058236	AdaBoostM2	11	0.5810
6	Accept	0.14191	0.36518	0.057029	0.05715	AdaBoostM2	11	0.001053
7	Accept	0.14324	8.5234	0.057029	0.057194	AdaBoostM2	358	0.005162
8	Accept	0.25332	0.34561	0.057029	0.079121	AdaBoostM2	11	0.8538
9	Accept	0.061008	0.43039	0.057029	0.05861	Bag	11	
10	Accept	0.071618	0.41833	0.057029	0.059224	Bag	11	
11	Accept	0.086207	0.42789	0.057029	0.059453	AdaBoostM2	11	0.7697
12	Accept	0.070292	0.44414	0.057029	0.062177	Bag	11	
13	Accept	0.062334	0.4393	0.057029	0.061202	Bag	11	
14	Accept	0.058355	0.47593	0.057029	0.061916	Bag	12	
15	Accept	0.25332	0.34442	0.057029	0.062051	AdaBoostM2	10	0.001072
16	Accept	0.30239	0.76464	0.057029	0.057197	RUSBoost	22	0.001014
17	Accept	0.067639	0.80451	0.057029	0.059728	AdaBoostM2	29	0.9942
18	Accept	0.14058	0.47271	0.057029	0.060679	AdaBoostM2	15	0.001293
19	Accept	0.058355	0.41063	0.057029	0.06003	Bag	10	
20	Accept	0.98408	0.40574	0.057029	0.059751	RUSBoost	11	0.500

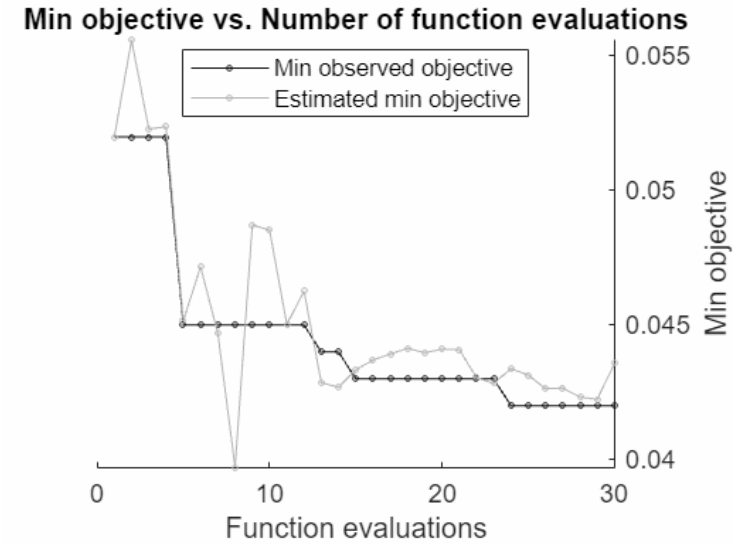
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
21	Accept	0.082228	0.38432	0.057029	0.060086	AdaBoostM2	10	0.8895
22	Best	0.055703	1.1379	0.055703	0.058985	Bag	34	
23	Accept	0.16446	0.35054	0.055703	0.059135	AdaBoostM2	10	0.9180
24	Accept	0.1313	0.3843	0.055703	0.059624	Bag	10	
25	Accept	0.26923	0.42391	0.055703	0.05969	RUSBoost	10	0.5907
26	Accept	0.32891	0.39982	0.055703	0.055888	RUSBoost	10	0.00100
27	Accept	0.058355	2.4565	0.055703	0.057303	Bag	77	
28	Accept	0.067639	1.7986	0.055703	0.057374	AdaBoostM2	70	0.9223
29	Accept	0.057029	1.8075	0.055703	0.056655	Bag	53	
30	Best	0.055703	2.1885	0.055703	0.056054	Bag	69	




```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 68.0393 seconds
Total objective function evaluation time: 52.3085

Best observed feasible point:
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag              69               NaN            2

Observed objective function value = 0.055703
Estimated objective function value = 0.056054
Function evaluation time = 2.1885

Best estimated feasible point (according to models):
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag              69               NaN            2

Estimated objective function value = 0.056054
Estimated function evaluation time = 2.1903

|=====================================================================================================================
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |       Method | NumLearningC-|    LearnRate
|      | result |             | runtime     | (observed)  | (estim.)    |              | ycles        |
|=====================================================================================================================
|    1 | Best   |    0.051948 |      7.6845 |    0.051948 |    0.051948 |   AdaBoostM2 |          277 |     0.043029
|    2 | Accept |      0.1019 |      4.3292 |    0.051948 |    0.055545 |   AdaBoostM2 |          184 |     0.095882
|    3 | Accept |     0.20779 |     0.66799 |    0.051948 |    0.052239 |          Bag |           21 |            -
|    4 | Accept |     0.93307 |     0.45174 |    0.051948 |    0.052328 |     RUSBoost |           12 |    0.0018271
|    5 | Best   |    0.044955 |      14.079 |    0.044955 |    0.045121 |          Bag |          500 |            -
|    6 | Accept |    0.068931 |     0.38727 |    0.044955 |    0.047123 |   AdaBoostM2 |           10 |     0.027098
|    7 | Accept |    0.044955 |      0.3969 |    0.044955 |    0.044657 |          Bag |           10 |            -
|    8 | Accept |    0.048951 |     0.38783 |    0.044955 |    0.039683 |   AdaBoostM2 |           11 |      0.94355
|    9 | Accept |    0.053946 |     0.39346 |    0.044955 |    0.048678 |          Bag |           10 |            -
|   10 | Accept |    0.047952 |     0.38857 |    0.044955 |    0.048484 |   AdaBoostM2 |           11 |      0.95205
|   11 | Accept |    0.050949 |     0.39336 |    0.044955 |    0.044991 |          Bag |           10 |            -
|   12 | Accept |    0.048951 |     0.77472 |    0.044955 |    0.046212 |   AdaBoostM2 |           28 |      0.91372
|   13 | Best   |    0.043956 |      1.6731 |    0.043956 |    0.042818 |   AdaBoostM2 |           66 |      0.97859
|   14 | Accept |    0.043956 |      4.8913 |    0.043956 |     0.04265 |   AdaBoostM2 |          206 |      0.98071
```

15	Best	0.042957	0.41852	0.042957	0.043301	AdaBoostM2	12	0.8831
16	Accept	0.047952	0.40622	0.042957	0.043659	AdaBoostM2	13	0.964
17	Accept	0.043956	5.4939	0.042957	0.043884	AdaBoostM2	234	0.9711
18	Accept	0.12787	0.4487	0.042957	0.044094	RUSBoost	12	0.8792
19	Accept	0.053946	0.37944	0.042957	0.043932	AdaBoostM2	11	0.1927
20	Accept	0.054945	0.38827	0.042957	0.044061	Bag	10	

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
21	Accept	0.21379	0.43491	0.042957	0.044044	RUSBoost	12	0.001037
22	Accept	0.091908	0.3945	0.042957	0.043004	AdaBoostM2	12	0.001025
23	Accept	0.20779	0.37851	0.042957	0.042793	AdaBoostM2	11	0.5098
24	Best	0.041958	0.36263	0.041958	0.043338	AdaBoostM2	11	0.8896
25	Accept	0.055944	0.34048	0.041958	0.043099	AdaBoostM2	10	0.91
26	Accept	0.056943	0.38726	0.041958	0.042594	Bag	10	
27	Accept	0.088911	0.42268	0.041958	0.04262	Bag	11	
28	Accept	0.041958	3.2578	0.041958	0.042282	AdaBoostM2	135	0.9708
29	Accept	0.05994	0.41655	0.041958	0.042199	AdaBoostM2	12	0.8934
30	Accept	0.048951	5.2768	0.041958	0.043538	AdaBoostM2	222	0.9913



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 70.8801 seconds
Total objective function evaluation time: 56.1067

Best observed feasible point:

Method	NumLearningCycles	LearnRate	MinLeafSize
AdaBoostM2	11	0.88961	2

Observed objective function value = 0.041958
Estimated objective function value = 0.047047
Function evaluation time = 0.36263

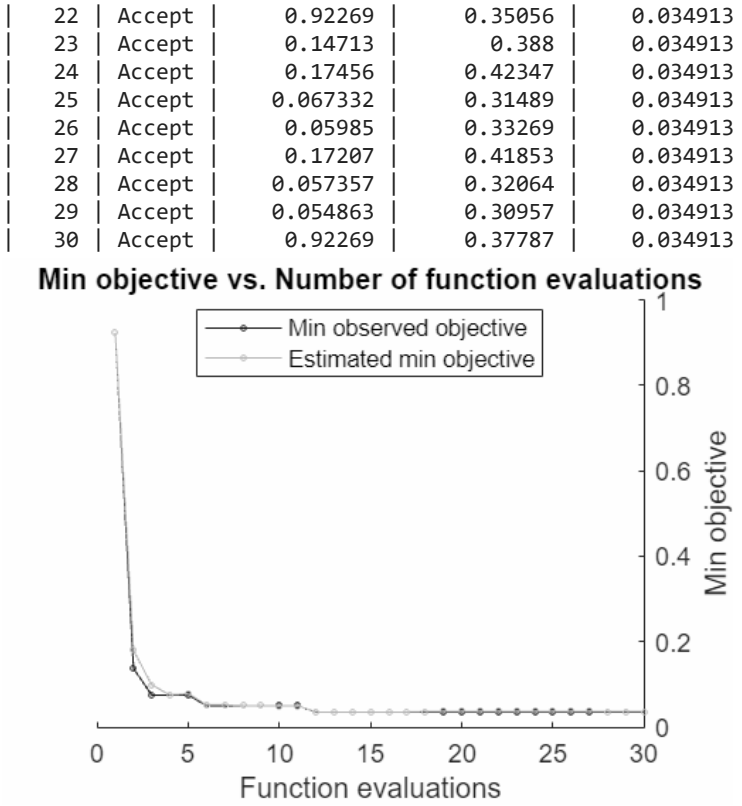
Best estimated feasible point (according to models):

Method	NumLearningCycles	LearnRate	MinLeafSize
AdaBoostM2	206	0.98071	3

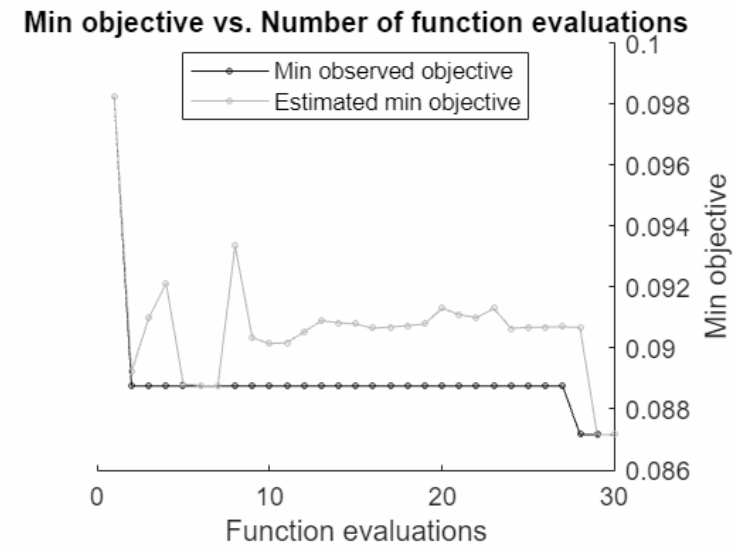
```
Estimated objective function value = 0.043538
Estimated function evaluation time = 4.9583
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
1	Best	0.92269	6.957	0.92269	0.92269	RUSBoost	298	0.7165
2	Best	0.13716	3.6551	0.13716	0.1793	RUSBoost	145	0.06136
3	Best	0.074813	7.5026	0.074813	0.098513	Bag	325	
4	Accept	0.10973	1.0301	0.074813	0.075263	AdaBoostM2	43	0.002799
5	Accept	0.094763	0.32057	0.074813	0.07838	Bag	10	
6	Best	0.049875	0.37564	0.049875	0.052006	Bag	10	
7	Accept	0.16958	0.94096	0.049875	0.051993	AdaBoostM2	43	0.001894
8	Accept	0.16958	0.94672	0.049875	0.049878	AdaBoostM2	43	0.01435
9	Accept	0.05985	0.9767	0.049875	0.049877	AdaBoostM2	43	0.001365
10	Accept	0.057357	0.35458	0.049875	0.049777	Bag	10	
11	Accept	0.16958	0.32621	0.049875	0.049811	Bag	11	
12	Best	0.034913	0.80992	0.034913	0.035375	AdaBoostM2	35	0.9168
13	Accept	0.037406	0.50816	0.034913	0.034924	AdaBoostM2	19	0.5817
14	Accept	0.16958	0.30855	0.034913	0.034937	AdaBoostM2	11	0.9414
15	Accept	0.057357	0.3011	0.034913	0.035169	AdaBoostM2	10	0.009189
16	Accept	0.0399	0.3024	0.034913	0.035118	AdaBoostM2	10	0.8329
17	Accept	0.0399	0.49268	0.034913	0.035211	AdaBoostM2	19	0.9129
18	Accept	0.037406	6.0309	0.034913	0.036456	AdaBoostM2	290	0.9460
19	Accept	0.037406	0.34689	0.034913	0.03666	AdaBoostM2	12	0.9444
20	Accept	0.037406	0.51331	0.034913	0.036742	AdaBoostM2	19	0.9633

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
21	Accept	0.042394	0.3838	0.034913	0.036733	AdaBoostM2	12	0.947
22	Accept	0.92269	0.35056	0.034913	0.036795	RUSBoost	11	0.01566
23	Accept	0.14713	0.388	0.034913	0.036845	RUSBoost	10	0.001856
24	Accept	0.17456	0.42347	0.034913	0.036638	RUSBoost	11	0.001012
25	Accept	0.067332	0.31489	0.034913	0.036642	AdaBoostM2	10	0.001375
26	Accept	0.05985	0.33269	0.034913	0.036638	Bag	10	
27	Accept	0.17207	0.41853	0.034913	0.03674	RUSBoost	12	0.7947
28	Accept	0.057357	0.32064	0.034913	0.036487	AdaBoostM2	10	0.1172
29	Accept	0.054863	0.30957	0.034913	0.036515	AdaBoostM2	10	0.897
30	Accept	0.92269	0.37787	0.034913	0.036391	RUSBoost	10	0.001013



```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 51.316 seconds
Total objective function evaluation time: 36.6201

Best observed feasible point:
      Method      NumLearningCycles    LearnRate    MinLeafSize
    __________    _________________    _________    ___________

    AdaBoostM2           35             0.91686          1

Observed objective function value = 0.034913
Estimated objective function value = 0.036391
Function evaluation time = 0.80992

Best estimated feasible point (according to models):
      Method      NumLearningCycles    LearnRate    MinLeafSize
    __________    _________________    _________    ___________

    AdaBoostM2           35             0.91686          1

Estimated objective function value = 0.036391
Estimated function evaluation time = 0.8254

|=====================================================================================================================
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |       Method | NumLearningC-|    LearnRate
|      | result |             | runtime     | (observed)  | (estim.)    |              | ycles        |
|=====================================================================================================================
|    1 | Best   |    0.098257 |      3.8028 |    0.098257 |    0.098257 |   AdaBoostM2 |          165 |    0.001675
|    2 | Best   |    0.088748 |     0.44006 |    0.088748 |    0.089228 |   AdaBoostM2 |           15 |        0.81
|    3 | Accept |     0.13788 |      3.7011 |    0.088748 |    0.090994 |   AdaBoostM2 |          174 |      0.6780
|    4 | Accept |     0.16323 |     0.31037 |    0.088748 |    0.092119 |   AdaBoostM2 |           10 |    0.008892
|    5 | Accept |    0.095087 |     0.38937 |    0.088748 |    0.088819 |   AdaBoostM2 |           13 |      0.8351
|    6 | Accept |    0.095087 |     0.39159 |    0.088748 |    0.088771 |   AdaBoostM2 |           13 |      0.6640
|    7 | Accept |    0.098257 |     0.67042 |    0.088748 |    0.088767 |   AdaBoostM2 |           26 |    0.003178
|    8 | Accept |     0.10777 |     0.49573 |    0.088748 |    0.093359 |   AdaBoostM2 |           18 |    0.005134
|    9 | Accept |    0.090333 |     0.36104 |    0.088748 |    0.090338 |   AdaBoostM2 |           10 |      0.9536
|   10 | Accept |     0.10618 |     0.33646 |    0.088748 |    0.090143 |   AdaBoostM2 |           10 |     0.07133
|   11 | Accept |     0.10618 |     0.33412 |    0.088748 |    0.090178 |   AdaBoostM2 |           10 |    0.001244
|   12 | Accept |    0.091918 |     0.41348 |    0.088748 |    0.090527 |   AdaBoostM2 |           13 |      0.6613
|   13 | Accept |    0.091918 |     0.31808 |    0.088748 |    0.090895 |   AdaBoostM2 |           10 |      0.7034
|   14 | Accept |    0.090333 |     0.33169 |    0.088748 |    0.090832 |   AdaBoostM2 |           10 |      0.9721
|   15 | Accept |    0.091918 |     0.38171 |    0.088748 |    0.090788 |   AdaBoostM2 |           11 |      0.7401
|   16 | Accept |      0.1046 |      0.3169 |    0.088748 |    0.090648 |   AdaBoostM2 |           10 |    0.002347
|   17 | Accept |     0.10935 |     0.43009 |    0.088748 |     0.09069 |     RUSBoost |           11 |      0.5756
|   18 | Accept |      0.1458 |     0.53147 |    0.088748 |    0.090735 |     RUSBoost |           12 |      0.2911
|   19 | Accept |     0.12678 |     0.38051 |    0.088748 |     0.09079 |     RUSBoost |           10 |     0.00151
|   20 | Accept |    0.096672 |      0.3701 |    0.088748 |    0.091307 |   AdaBoostM2 |           12 |      0.8146
|=====================================================================================================================
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |       Method | NumLearningC-|    LearnRate
|      | result |             | runtime     | (observed)  | (estim.)    |              | ycles        |
|=====================================================================================================================
|   21 | Accept |    0.090333 |      0.4294 |    0.088748 |    0.091098 |   AdaBoostM2 |           15 |      0.9055
|   22 | Accept |    0.090333 |     0.36462 |    0.088748 |    0.090994 |   AdaBoostM2 |           11 |      0.6797
|   23 | Accept |    0.093502 |     0.35037 |    0.088748 |    0.091302 |   AdaBoostM2 |           10 |      0.9893
|   24 | Accept |     0.90333 |     0.33513 |    0.088748 |    0.090647 |     RUSBoost |           10 |      0.0534
|   25 | Accept |     0.13788 |     0.39614 |    0.088748 |     0.09068 |          Bag |           10 |
|   26 | Accept |    0.093502 |     0.36722 |    0.088748 |    0.090689 |          Bag |           10 |
|   27 | Accept |     0.43265 |     0.32024 |    0.088748 |    0.090695 |          Bag |           10 |
|   28 | Best   |    0.087163 |      4.2735 |    0.087163 |    0.090673 |          Bag |          160 |
|   29 | Accept |     0.43265 |     0.32695 |    0.087163 |    0.087151 |   AdaBoostM2 |           10 |      0.7281
|   30 | Accept |     0.10143 |     0.35715 |    0.087163 |    0.087174 |          Bag |           10 |
```

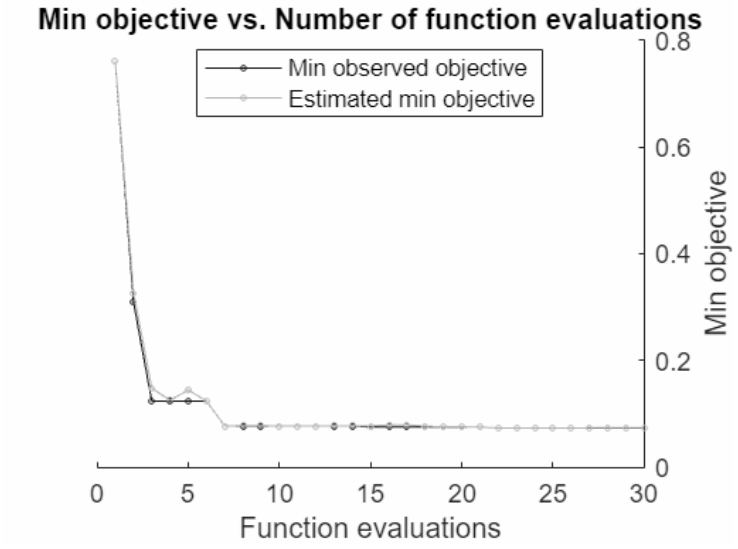



```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 36.1439 seconds
Total objective function evaluation time: 22.2278

Best observed feasible point:
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag             160               NaN            6

Observed objective function value = 0.087163
Estimated objective function value = 0.087174
Function evaluation time = 4.2735

Best estimated feasible point (according to models):
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag             160               NaN            6

Estimated objective function value = 0.087174
Estimated function evaluation time = 4.2632
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
1	Best	0.75949	0.53045	0.75949	0.75949	RUSBoost	16	0.005060
2	Best	0.30867	0.36572	0.30867	0.32659	Bag	10	
3	Best	0.12366	3.4997	0.12366	0.14808	AdaBoostM2	141	0.009513
4	Accept	0.13924	6.6316	0.12366	0.12493	AdaBoostM2	275	0.2617
5	Accept	0.16845	0.3601	0.12366	0.14445	AdaBoostM2	11	0.001388
6	Accept	0.27556	0.31434	0.12366	0.12354	AdaBoostM2	10	0.7801
7	Best	0.076923	15.072	0.076923	0.07697	Bag	496	
8	Accept	0.083739	0.43595	0.076923	0.077678	Bag	11	
9	Accept	0.111	0.37207	0.076923	0.077606	AdaBoostM2	10	0.9454
10	Accept	0.15579	0.48036	0.076923	0.076954	AdaBoostM2	16	0.02543
11	Accept	0.092502	0.38899	0.076923	0.076957	Bag	10	
12	Accept	0.085686	11.616	0.076923	0.077	AdaBoostM2	484	0.9978
13	Accept	0.10516	0.43395	0.076923	0.077268	AdaBoostM2	12	0.951
14	Accept	0.088608	0.41986	0.076923	0.077191	Bag	10	

15	Best	0.074976	3.5435	0.074976	0.076997	Bag	114	
16	Accept	0.082765	9.1567	0.074976	0.079112	Bag	308	
17	Accept	0.30574	0.55259	0.074976	0.079402	RUSBoost	16	0.9691
18	Accept	0.089581	9.4589	0.074976	0.076381	AdaBoostM2	382	0.9829
19	Accept	0.074976	12.368	0.074976	0.075431	Bag	360	
20	Accept	0.075949	15.909	0.074976	0.075552	Bag	487	

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
21	Accept	0.32619	0.53235	0.074976	0.074869	RUSBoost	14	0.001087
22	Best	0.073028	2.5377	0.073028	0.073044	Bag	74	
23	Accept	0.073028	3.1464	0.073028	0.072803	Bag	94	
24	Accept	0.074976	3.4623	0.073028	0.073365	Bag	103	
25	Accept	0.073028	3.5976	0.073028	0.073255	Bag	109	
26	Accept	0.43427	0.34868	0.073028	0.073312	AdaBoostM2	10	0.001231
27	Accept	0.75949	12.532	0.073028	0.073397	RUSBoost	493	0.9530
28	Accept	0.75949	0.3554	0.073028	0.073581	RUSBoost	10	0.001028
29	Accept	0.13632	0.41447	0.073028	0.073548	Bag	10	
30	Accept	0.089581	8.7446	0.073028	0.073528	AdaBoostM2	341	0.997



Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 143.6949 seconds
Total objective function evaluation time: 127.5805

Best observed feasible point:

Method	NumLearningCycles	LearnRate	MinLeafSize
Bag	74	NaN	1

Observed objective function value = 0.073028
Estimated objective function value = 0.074069
Function evaluation time = 2.5377

Best estimated feasible point (according to models):

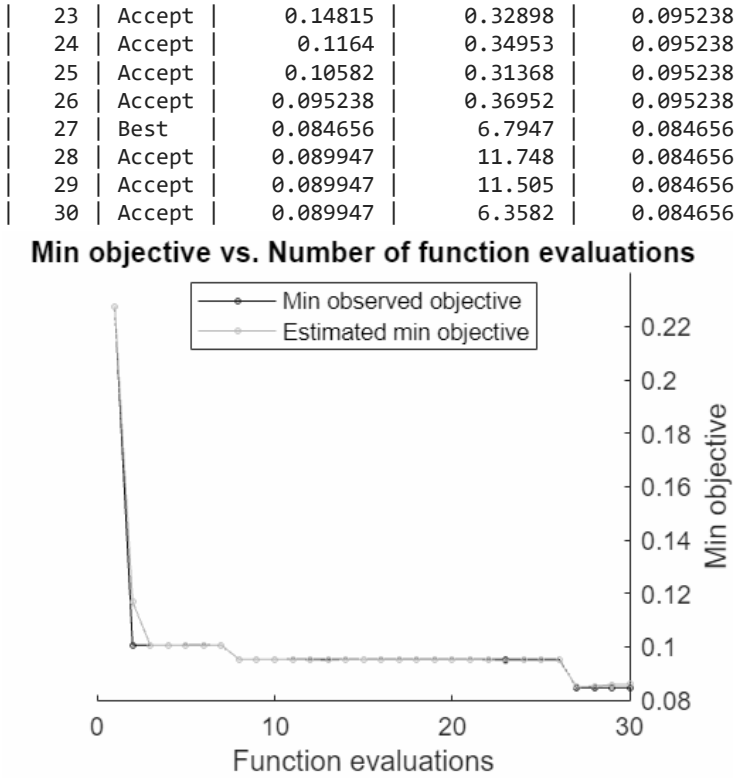
Method	NumLearningCycles	LearnRate	MinLeafSize
Bag	109	NaN	1

Estimated objective function value = 0.073528

```
Estimated function evaluation time = 3.5955
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
1	Best	0.22751	0.35078	0.22751	0.22751	AdaBoostM2	10	0.1228
2	Best	0.10053	2.4634	0.10053	0.11676	RUSBoost	83	0.07208
3	Accept	0.24339	1.769	0.10053	0.10054	AdaBoostM2	84	0.00817
4	Accept	0.10582	3.5043	0.10053	0.10054	AdaBoostM2	169	0.4922
5	Accept	0.10053	0.33263	0.10053	0.10051	AdaBoostM2	10	0.1758
6	Accept	0.8254	0.34747	0.10053	0.1005	RUSBoost	10	0.567
7	Accept	0.1164	0.40319	0.10053	0.10055	RUSBoost	10	0.001599
8	Best	0.095238	0.36652	0.095238	0.095254	Bag	10	
9	Accept	0.1164	0.33304	0.095238	0.095249	Bag	10	
10	Accept	0.34392	0.31597	0.095238	0.095244	Bag	10	
11	Accept	0.095238	0.35154	0.095238	0.095191	Bag	10	
12	Accept	0.095238	0.36737	0.095238	0.095021	Bag	10	
13	Accept	0.8254	0.37402	0.095238	0.094885	RUSBoost	10	0.03223
14	Accept	0.10582	0.33983	0.095238	0.09517	AdaBoostM2	10	0.009500
15	Accept	0.34392	0.34711	0.095238	0.095219	Bag	11	
16	Accept	0.095238	0.33665	0.095238	0.095119	AdaBoostM2	10	0.4418
17	Accept	0.10582	12.924	0.095238	0.095133	RUSBoost	499	0.00396
18	Accept	0.095238	1.1396	0.095238	0.095141	RUSBoost	40	0.4490
19	Accept	0.095238	0.57315	0.095238	0.095208	AdaBoostM2	23	0.8116
20	Accept	0.10053	0.35622	0.095238	0.09519	RUSBoost	10	0.9697

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles	LearnRate
21	Accept	0.095238	11.389	0.095238	0.095199	Bag	489	
22	Accept	0.095238	0.34651	0.095238	0.095027	AdaBoostM2	10	0.6330
23	Accept	0.14815	0.32898	0.095238	0.094814	AdaBoostM2	11	0.001249
24	Accept	0.1164	0.34953	0.095238	0.094876	Bag	10	
25	Accept	0.10582	0.31368	0.095238	0.094939	AdaBoostM2	10	0.001041
26	Accept	0.095238	0.36952	0.095238	0.094907	Bag	11	
27	Best	0.084656	6.7947	0.084656	0.084864	Bag	278	
28	Accept	0.089947	11.748	0.084656	0.08527	Bag	498	
29	Accept	0.089947	11.505	0.084656	0.085794	Bag	493	
30	Accept	0.089947	6.3582	0.084656	0.085951	AdaBoostM2	298	0.9754




```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
```

```
Total elapsed time: 94.0259 seconds
Total objective function evaluation time: 76.7993

Best observed feasible point:
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag             278              NaN            4

Observed objective function value = 0.084656
Estimated objective function value = 0.085951
Function evaluation time = 6.7947

Best estimated feasible point (according to models):
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag             278              NaN            4

Estimated objective function value = 0.085951
Estimated function evaluation time = 6.7967
```

Ensemble Clustering Visualisation

The results of every layer of the hypermatrix are summed up and visualised in a heatmap. Additionally the mean accuracy and time taken is calculated and used as a comparison metric between methods.

```
En = zeros(m,n);
for j = 1:n
    for k = 1:m
        En(j,k) = sum(hyp_En(j,k,:));
    end
end
figEn = figureGen(7,10);
heat_En6 = heatmap(En, "XDisplayLabels", [0,1,2,3], "YDisplayLabels", [0,1,2,3],
"ColorMethod", "mean", "ColorLimits", [0,100])
```

```
heat_En6 =
  HeatmapChart with properties:

       XData: {4×1 cell}
       YData: {4×1 cell}
    ColorData: [4×4 double]

  Show all properties
```

```
heat_En6.Colormap = parula(64);
xlabel("Predicted State");
ylabel("Labelled State");
average_acc_En = median(acc_En)
```

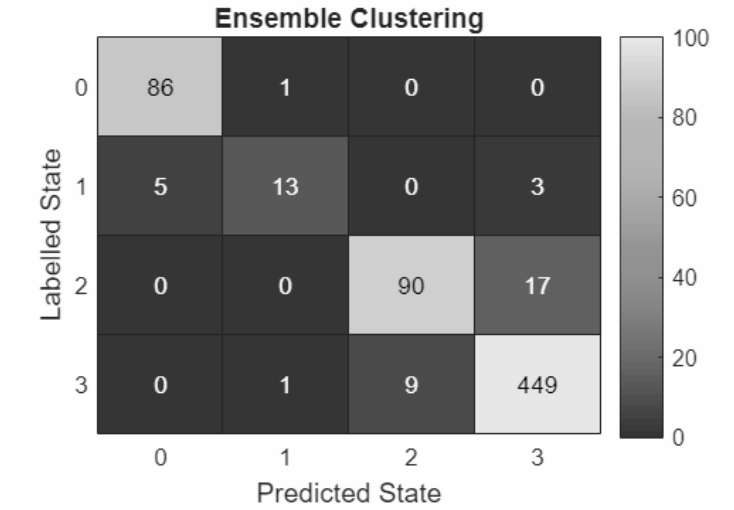
```
average_acc_En = 0.9469
```

```
average_time_En = median(time_En)
```

```
average_time_En = 0.0740
```

```
heat_En6.Title = "Ensemble Clustering";
```

```
saveas(heat_En6, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\6_En.jpg');
```



Naive Bayes (NB)

The next method to be applied will be NB.

```
for i = 1:numel(files)
    %Load file i from the folder
    fileName = fullfile(files(i).folder, files(i).name);
    load(fileName);
    %Extract the state vector
    state = stateTT.State;
    %Extract the Principal COmponent Data
    D = PCsTT{:,:};
    %Create a random 90/10 Partition for Training and Test Data
    rng('default');
    Partition_States = cvpartition(state, 'Holdout', 0.10);

    %Seperate the training and testing Ids
    trainingIds = training(Partition_States);
    DTrain = D(trainingIds, :);
    stateTrain = state(trainingIds);

    testIds = test(Partition_States);
    DTest = D(testIds, :);
    stateTest = state(testIds);
```

Begin measuring the time this algorithm will take

```
    tNB = tic;
```

train the decsion tree classifier

```
    classifierNB = fitcnb(DTrain, stateTrain, 'OptimizeHyperparameters', 'auto');
```

end the time measurement

```
    timeNB = toc(tNB);
```

use the tree to predict the test states

```
    TestModel_NB = predict(classifierNB, DTest);
```

measure the accuracy

```
    accuracy_NB = sum(stateTest == TestModel_NB)/length(stateTest);
    %figure;
    %confusionchart(stateTest, TestModel_BT, 'Normalization', 'row-normalized');
```

Create a confusion matrix

```
    [C_NB, order] = confusionmat(stateTest, TestModel_NB);
    %titleStr_BT = strrep([fName,' Binary Tree'],'_','-');
    % title(titleStr_BT);
```

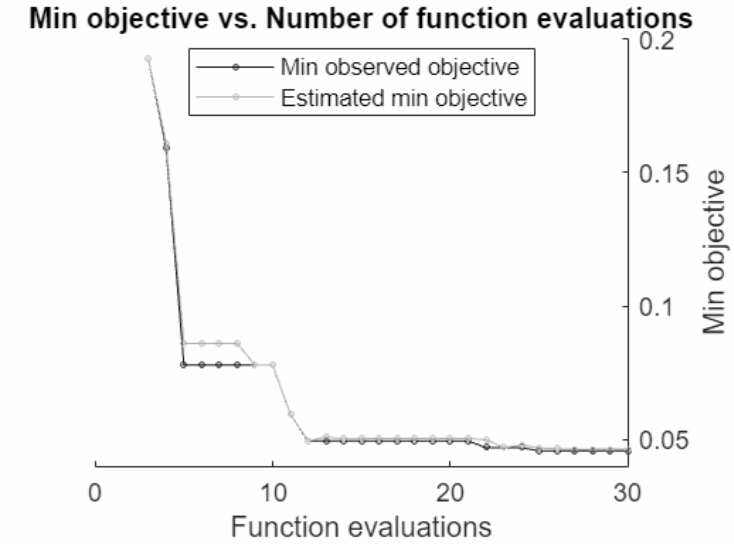
save the confusion matrix as one layer of the hypermatrix and the accuracy and measured time in vector form

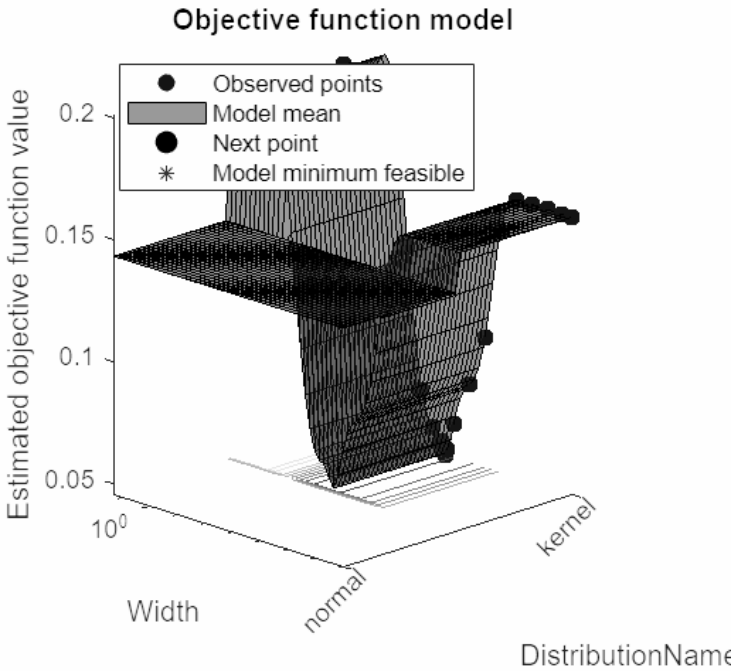
```
    hyp_NB(:,:,i) = C_NB;
    acc_NB(i) = accuracy_NB;
    time_NB(i) = timeNB/length(D);
end
```

```
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Distribution-Names	Width
1	Error	NaN	0.33295	NaN	NaN	normal	-
2	Error	NaN	0.049535	NaN	NaN	normal	-
3	Best	0.19259	1.1775	0.19259	0.19259	kernel	1.7088
4	Best	0.15926	0.34816	0.15926	0.16101	kernel	7.9523e-07
5	Best	0.077778	0.33735	0.077778	0.08596	kernel	0.00048114
6	Error	NaN	0.065728	0.077778	0.08596	normal	-
7	Error	NaN	0.051445	0.077778	0.08596	normal	-
8	Error	NaN	0.044473	0.077778	0.08596	normal	-
9	Accept	0.15926	0.31474	0.077778	0.077787	kernel	2.2959e-07
10	Accept	0.098765	0.37116	0.077778	0.077797	kernel	0.00014123
11	Best	0.059259	0.3356	0.059259	0.05929	kernel	0.001946
12	Best	0.049383	0.33924	0.049383	0.049452	kernel	0.0052596
13	Accept	0.055556	0.35022	0.049383	0.050874	kernel	0.010814
14	Accept	0.050617	0.37425	0.049383	0.050219	kernel	0.0053264
15	Accept	0.050617	0.41049	0.049383	0.050298	kernel	0.0054211
16	Accept	0.050617	0.40885	0.049383	0.05036	kernel	0.0054392
17	Accept	0.12593	0.38595	0.049383	0.05032	kernel	0.091821

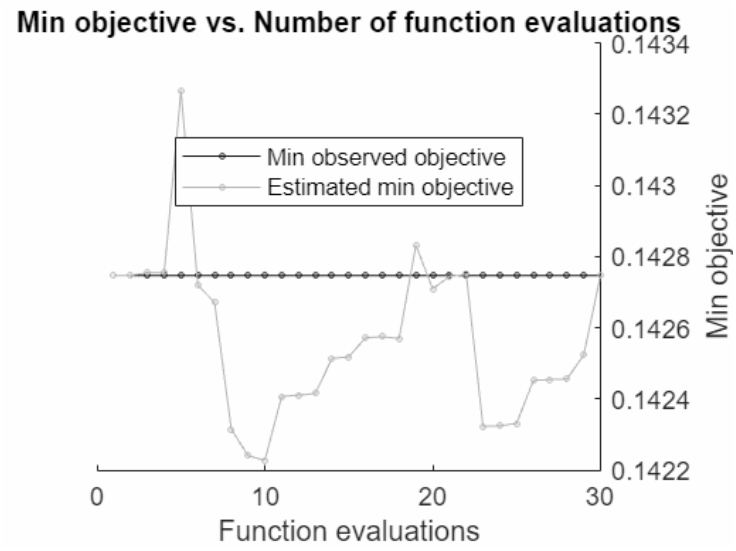
```
|   18 | Accept |     0.19259 |     0.34959 |    0.049383 |    0.050322 |       kernel |       20.332 |
|   19 | Accept |     0.15926 |     0.36952 |    0.049383 |    0.050315 |       kernel |    1.074e-05 |
|   20 | Accept |     0.15926 |     0.33407 |    0.049383 |    0.050317 |       kernel |   9.6766e-08 |
|======|========|=============|=============|=============|=============|==============|==============|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|======|========|=============|=============|=============|=============|==============|==============|
|   21 | Accept |    0.069136 |     0.34635 |    0.049383 |    0.050302 |       kernel |     0.027783 |
|   22 | Best   |    0.046914 |     0.31562 |    0.046914 |    0.049827 |       kernel |    0.0044556 |
|   23 | Accept |    0.046914 |     0.34145 |    0.046914 |     0.04697 |       kernel |    0.0045559 |
|   24 | Accept |    0.048148 |     0.35392 |    0.046914 |    0.047878 |       kernel |    0.0035726 |
|   25 | Best   |    0.045679 |      0.3441 |    0.045679 |    0.046823 |       kernel |    0.0038925 |
|   26 | Accept |    0.045679 |     0.31975 |    0.045679 |    0.046482 |       kernel |    0.0038409 |
|   27 | Accept |    0.045679 |     0.32014 |    0.045679 |    0.046277 |       kernel |    0.0038526 |
|   28 | Accept |     0.14444 |     0.30182 |    0.045679 |    0.046309 |       kernel |   3.8557e-05 |
|   29 | Accept |     0.15926 |     0.30499 |    0.045679 |    0.046355 |       kernel |   2.8851e-06 |
|   30 | Accept |     0.19136 |     0.41287 |    0.045679 |     0.04636 |       kernel |      0.30725 |
```

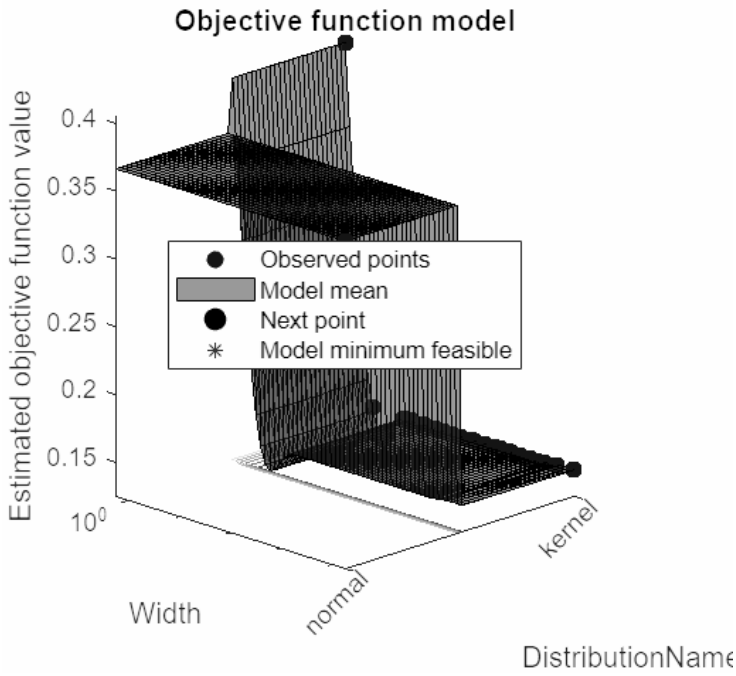




```
Optimization completed.
```

```
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 30.8638 seconds
Total objective function evaluation time: 10.1118

Best observed feasible point:
    DistributionNames      Width
    _________________    _________

         kernel          0.0038925

Observed objective function value = 0.045679
Estimated objective function value = 0.04636
Function evaluation time = 0.3441

Best estimated feasible point (according to models):
    DistributionNames      Width
    _________________    _________

         kernel          0.0038925

Estimated objective function value = 0.04636
Estimated function evaluation time = 0.3389
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|    1 | Best   |     0.14275 |     0.36547 |     0.14275 |     0.14275 |       kernel |   8.4941e-11 |
|    2 | Accept |     0.14275 |      0.3514 |     0.14275 |     0.14275 |       kernel |   5.0721e-19 |
|    3 | Accept |     0.36651 |     0.08854 |     0.14275 |     0.14275 |       normal |            - |
|    4 | Accept |     0.36651 |    0.066177 |     0.14275 |     0.14275 |       normal |            - |
|    5 | Accept |     0.14429 |     0.36157 |     0.14275 |     0.14327 |       kernel |   5.0348e-22 |
|    6 | Accept |     0.40586 |     0.54823 |     0.14275 |     0.14272 |       kernel |       20.473 |
|    7 | Accept |     0.14275 |     0.33036 |     0.14275 |     0.14267 |       kernel |   4.1677e-14 |
|    8 | Accept |     0.14275 |     0.32944 |     0.14275 |     0.14231 |       kernel |   3.0381e-12 |
|    9 | Accept |     0.14275 |     0.33659 |     0.14275 |     0.14224 |       kernel |   1.0815e-16 |
|   10 | Accept |     0.14429 |      0.3132 |     0.14275 |     0.14223 |       kernel |   2.4898e-20 |
|   11 | Accept |     0.14275 |     0.32566 |     0.14275 |     0.14241 |       kernel |   6.8019e-12 |
|   12 | Accept |     0.14275 |     0.35659 |     0.14275 |     0.14241 |       kernel |   1.1761e-17 |
|   13 | Accept |     0.14275 |     0.33449 |     0.14275 |     0.14242 |       kernel |   1.3255e-15 |
|   14 | Accept |     0.14275 |     0.32613 |     0.14275 |     0.14251 |       kernel |    8.208e-12 |
|   15 | Accept |     0.14275 |     0.31843 |     0.14275 |     0.14252 |       kernel |   1.6295e-17 |
|   16 | Accept |     0.14275 |     0.31471 |     0.14275 |     0.14257 |       kernel |   1.0106e-11 |
|   17 | Accept |     0.14275 |     0.35108 |     0.14275 |     0.14257 |       kernel |    8.239e-16 |
|   18 | Accept |     0.14275 |     0.33615 |     0.14275 |     0.14257 |       kernel |   3.6308e-13 |
|   19 | Accept |     0.14275 |     0.37931 |     0.14275 |     0.14283 |       kernel |   3.5943e-05 |
|   20 | Accept |     0.14275 |     0.49232 |     0.14275 |     0.14271 |       kernel |   3.6218e-07 |
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|   21 | Accept |     0.14275 |     0.40199 |     0.14275 |     0.14274 |       kernel |   4.9039e-06 |
|   22 | Accept |     0.14275 |     0.37741 |     0.14275 |     0.14275 |       kernel |    4.405e-09 |
|   23 | Accept |     0.14275 |     0.34858 |     0.14275 |     0.14232 |       kernel |   7.6667e-06 |
|   24 | Accept |     0.14275 |     0.41595 |     0.14275 |     0.14232 |       kernel |   5.7028e-10 |
|   25 | Accept |     0.14275 |     0.40177 |     0.14275 |     0.14233 |       kernel |   2.8365e-08 |
|   26 | Accept |     0.14275 |     0.34348 |     0.14275 |     0.14245 |       kernel |   8.6637e-06 |
|   27 | Accept |     0.14275 |     0.56162 |     0.14275 |     0.14245 |       kernel |   2.2304e-18 |
|   28 | Accept |     0.14275 |     0.97247 |     0.14275 |     0.14246 |       kernel |    9.439e-15 |
|   29 | Accept |     0.14275 |     0.46253 |     0.14275 |     0.14252 |       kernel |   9.2713e-06 |
|   30 | Accept |     0.14352 |     0.59072 |     0.14275 |     0.14275 |       kernel |     0.041037 |
```



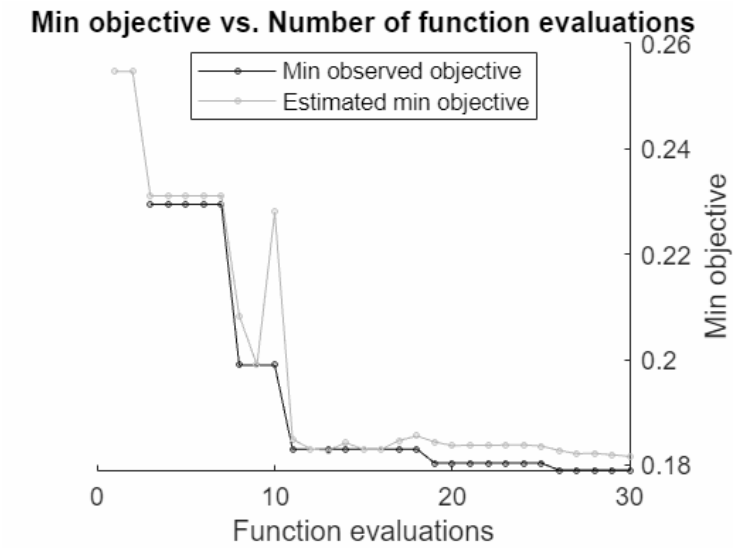


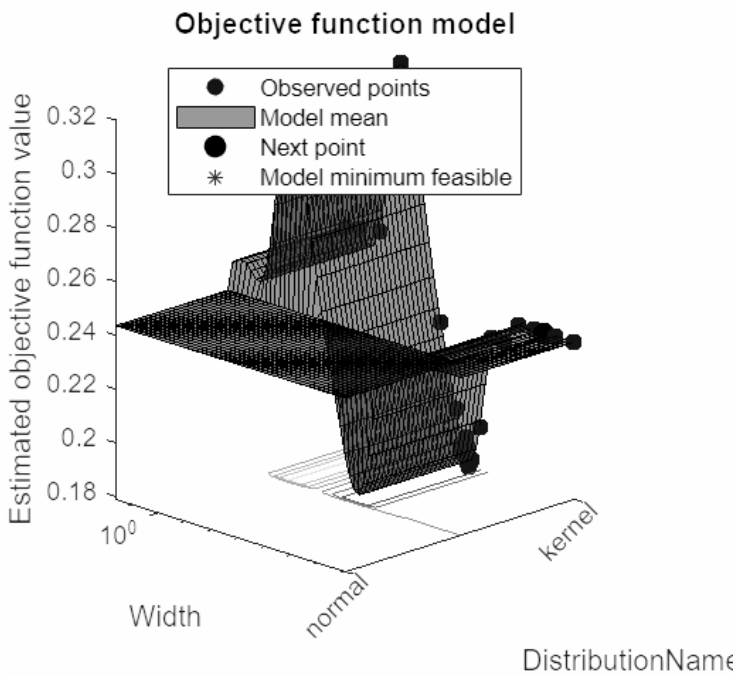
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 34.4256 seconds
Total objective function evaluation time: 11.5024

Best observed feasible point:
    DistributionNames      Width
    _________________    __________

         kernel          8.4941e-11

Observed objective function value = 0.14275
Estimated objective function value = 0.14275
Function evaluation time = 0.36547

Best estimated feasible point (according to models):
    DistributionNames      Width
```

```
    _________________    __________

          kernel          6.8019e-12

Estimated objective function value = 0.14275
Estimated function evaluation time = 0.3895
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Distribution-Names	Width
1	Best	0.25464	0.50896	0.25464	0.25464	kernel	10.748
2	Error	NaN	0.065932	NaN	0.25464	normal	-
3	Best	0.22944	0.40502	0.22944	0.23106	kernel	4.3182e-05
4	Error	NaN	0.057881	0.22944	0.23106	normal	-
5	Error	NaN	0.05263	0.22944	0.23106	normal	-
6	Error	NaN	0.047102	0.22944	0.23106	normal	-
7	Error	NaN	0.056093	0.22944	0.23106	normal	-
8	Best	0.19894	0.44106	0.19894	0.20833	kernel	0.002637
9	Accept	0.2374	0.45076	0.19894	0.19894	kernel	3.2299e-06
10	Accept	0.22944	0.47631	0.19894	0.22801	kernel	0.010168
11	Best	0.18302	0.4164	0.18302	0.18493	kernel	0.00163
12	Accept	0.18568	0.43809	0.18302	0.18304	kernel	0.0010132
13	Best	0.18302	0.41833	0.18302	0.1828	kernel	0.0013536
14	Accept	0.19496	0.35466	0.18302	0.1842	kernel	0.00033291
15	Accept	0.18568	0.45658	0.18302	0.18297	kernel	0.0010184
16	Accept	0.2374	0.34323	0.18302	0.18297	kernel	9.4924e-08
17	Accept	0.187	0.52794	0.18302	0.18465	kernel	0.0014872
18	Accept	0.18966	0.37164	0.18302	0.18551	kernel	0.0011808
19	Best	0.18037	0.34418	0.18037	0.18437	kernel	0.00097689
20	Accept	0.18037	0.35085	0.18037	0.18369	kernel	0.00090058

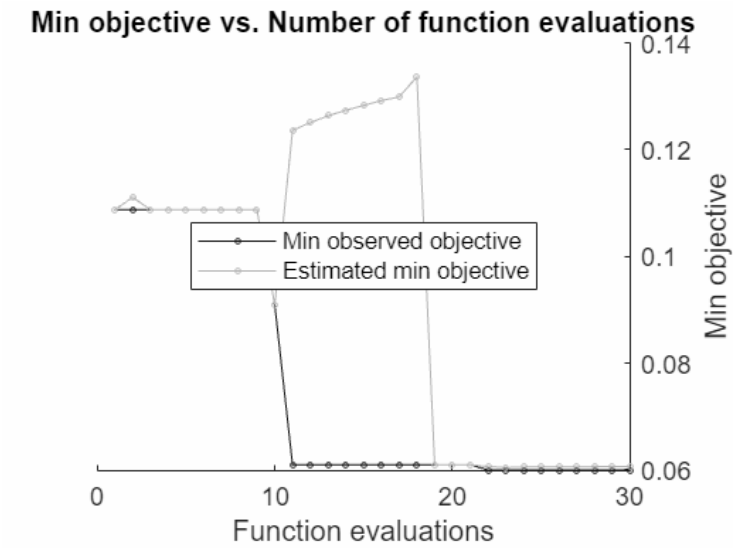
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Distribution-Names	Width
21	Accept	0.32095	0.4357	0.18037	0.18375	kernel	0.32661
22	Accept	0.25332	0.43465	0.18037	0.18374	kernel	41.088
23	Accept	0.2374	0.32674	0.18037	0.1838	kernel	5.135e-07
24	Accept	0.28117	0.3711	0.18037	0.18381	kernel	0.054948
25	Accept	0.25597	0.40938	0.18037	0.1836	kernel	2.0719
26	Best	0.17905	0.37489	0.17905	0.18277	kernel	0.00082651
27	Accept	0.18037	0.36364	0.17905	0.18216	kernel	0.00078698
28	Accept	0.2374	0.34541	0.17905	0.1822	kernel	1.1803e-05
29	Accept	0.22944	0.36109	0.17905	0.18194	kernel	0.00011906
30	Accept	0.1817	0.36983	0.17905	0.18164	kernel	0.00072613

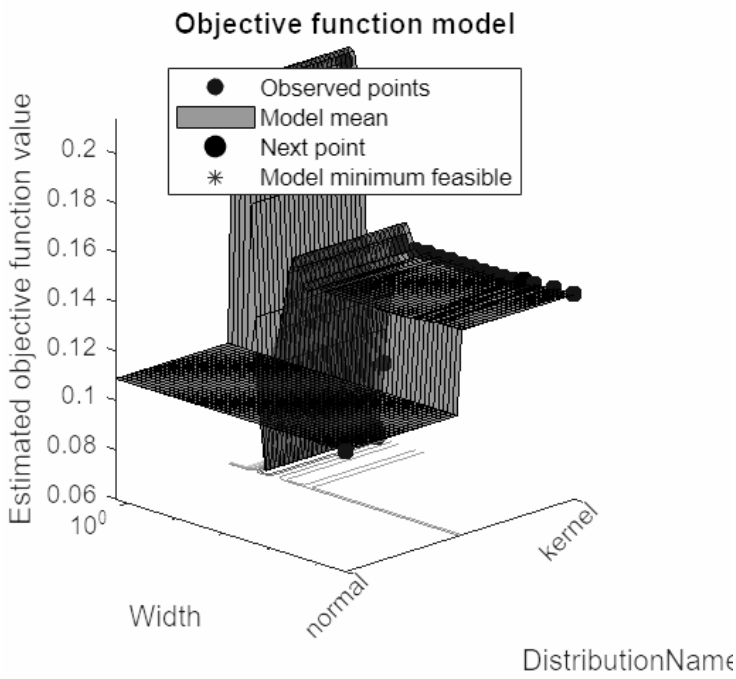





```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 38.5691 seconds
Total objective function evaluation time: 10.3761

Best observed feasible point:
    DistributionNames      Width
    _________________    _________

         kernel          0.00082651

Observed objective function value = 0.17905
Estimated objective function value = 0.18164
Function evaluation time = 0.37489

Best estimated feasible point (according to models):
    DistributionNames      Width
```

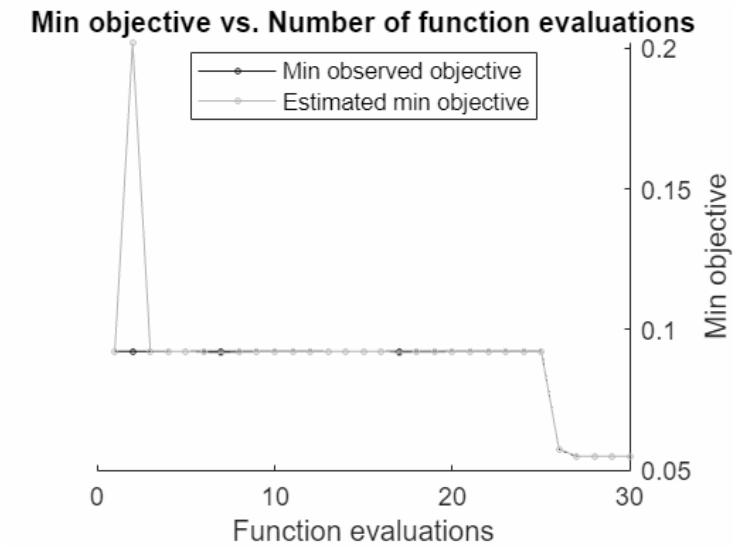
```
    _________________    _________

          kernel          0.00082651

Estimated objective function value = 0.18164
Estimated function evaluation time = 0.3989
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|    1 | Best   |     0.10875 |     0.11319 |     0.10875 |     0.10875 |       normal |            - |
|    2 | Accept |     0.13986 |     0.47838 |     0.10875 |     0.11108 |       kernel |   3.6134e-05 |
|    3 | Accept |     0.14086 |     0.42708 |     0.10875 |     0.10878 |       kernel |   6.3222e-15 |
|    4 | Accept |     0.10875 |    0.085533 |     0.10875 |     0.10875 |       normal |            - |
|    5 | Accept |     0.10875 |    0.068815 |     0.10875 |     0.10875 |       normal |            - |
|    6 | Accept |     0.10875 |    0.071951 |     0.10875 |     0.10875 |       normal |            - |
|    7 | Accept |     0.14286 |     0.63422 |     0.10875 |     0.10875 |       kernel |   1.7882e-23 |
|    8 | Accept |     0.20779 |     0.50761 |     0.10875 |     0.10875 |       kernel |       20.585 |
|    9 | Accept |     0.14186 |     0.37426 |     0.10875 |     0.10875 |       kernel |   3.1424e-05 |
|   10 | Best   |    0.090909 |     0.42417 |    0.090909 |    0.090914 |       kernel |    0.0015524 |
|   11 | Best   |    0.060939 |     0.51573 |    0.060939 |     0.12364 |       kernel |    0.0074296 |
|   12 | Accept |     0.14086 |     0.44963 |    0.060939 |     0.12508 |       kernel |    1.737e-10 |
|   13 | Accept |     0.14186 |     0.44186 |    0.060939 |     0.12637 |       kernel |   2.9023e-19 |
|   14 | Accept |     0.14086 |     0.35681 |    0.060939 |      0.1274 |       kernel |   4.4068e-08 |
|   15 | Accept |     0.14086 |     0.47524 |    0.060939 |      0.1283 |       kernel |   8.9683e-13 |
|   16 | Accept |     0.14286 |     0.38471 |    0.060939 |     0.12921 |       kernel |   2.1277e-21 |
|   17 | Accept |     0.14086 |     0.36889 |    0.060939 |      0.1299 |       kernel |   4.2605e-17 |
|   18 | Accept |      0.1978 |     0.50393 |    0.060939 |     0.13367 |       kernel |      0.63624 |
|   19 | Accept |     0.10875 |    0.079992 |    0.060939 |    0.060951 |       normal |            - |
|   20 | Accept |    0.063936 |     0.47566 |    0.060939 |    0.060956 |       kernel |     0.012267 |
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|   21 | Accept |    0.060939 |     0.73482 |    0.060939 |     0.06092 |       kernel |    0.0082645 |
|   22 | Best   |     0.05994 |     0.54244 |     0.05994 |    0.060575 |       kernel |    0.0080064 |
|   23 | Accept |     0.05994 |     0.54048 |     0.05994 |    0.060414 |       kernel |     0.008108 |
|   24 | Accept |    0.060939 |     0.43643 |     0.05994 |    0.060517 |       kernel |    0.0082866 |
|   25 | Accept |     0.14086 |     0.36764 |     0.05994 |    0.060517 |       kernel |   8.2607e-07 |
|   26 | Accept |     0.14086 |     0.42056 |     0.05994 |    0.060519 |       kernel |   2.6649e-09 |
|   27 | Accept |     0.14086 |     0.35668 |     0.05994 |     0.06052 |       kernel |   1.1878e-11 |
|   28 | Accept |    0.060939 |     0.42011 |     0.05994 |    0.060586 |       kernel |    0.0076825 |
|   29 | Accept |     0.14086 |      0.2901 |     0.05994 |    0.060588 |       kernel |   5.2708e-16 |
|   30 | Accept |     0.14086 |     0.33316 |     0.05994 |     0.06059 |       kernel |   7.6013e-14 |
```

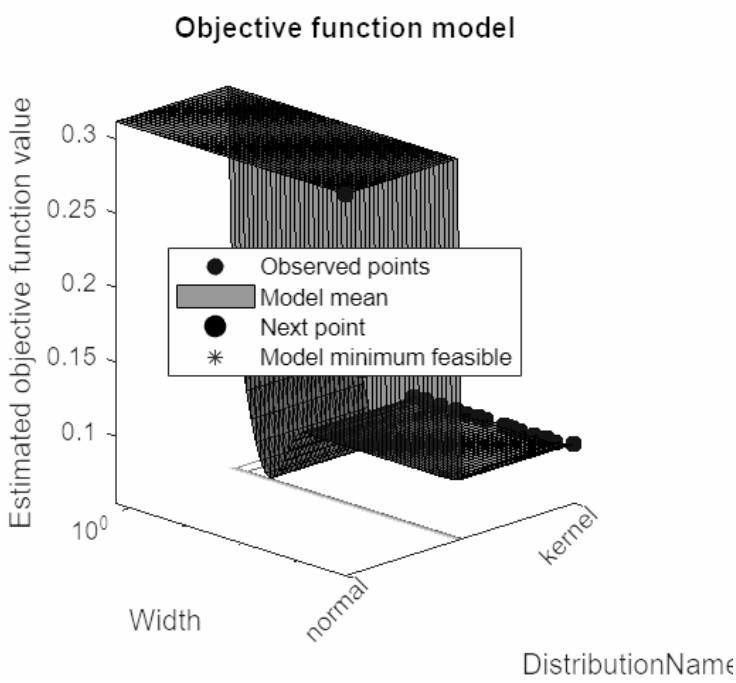





```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 36.3922 seconds
Total objective function evaluation time: 11.6801

Best observed feasible point:
    DistributionNames      Width
    _________________    _________

         kernel          0.0080064

Observed objective function value = 0.05994
Estimated objective function value = 0.06059
Function evaluation time = 0.54244

Best estimated feasible point (according to models):
    DistributionNames      Width
```

```
    _________________    _________

        kernel           0.0080064

Estimated objective function value = 0.06059
Estimated function evaluation time = 0.48346
Warning: One or more of the unique class values in GROUP is not present in one or more folds. For
classification problems, either remove this class from the data or use N instead of GROUP to obtain
nonstratified partitions. For regression problems with continuous response, use N.
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|    1 | Best   |    0.092269 |     0.35928 |    0.092269 |    0.092269 |       kernel |   2.7842e-18 |
|    2 | Accept |     0.31153 |    0.070311 |    0.092269 |     0.20184 |       normal |            - |
|    3 | Accept |    0.092269 |     0.33347 |    0.092269 |    0.092277 |       kernel |   4.6755e-12 |
|    4 | Accept |     0.31153 |    0.069698 |    0.092269 |    0.092277 |       normal |            - |
|    5 | Accept |     0.16958 |     0.39223 |    0.092269 |    0.092252 |       kernel |       20.574 |
|    6 | Accept |    0.092269 |     0.29506 |    0.092269 |    0.091843 |       kernel |   4.2778e-15 |
|    7 | Accept |    0.094763 |     0.31388 |    0.092269 |    0.091603 |       kernel |   6.3441e-20 |
|    8 | Accept |    0.092269 |     0.37719 |    0.092269 |    0.091912 |       kernel |   1.7643e-14 |
|    9 | Accept |    0.092269 |     0.32098 |    0.092269 |    0.092038 |       kernel |   4.2551e-14 |
|   10 | Accept |    0.092269 |     0.31244 |    0.092269 |     0.09206 |       kernel |   2.1169e-16 |
|   11 | Accept |    0.092269 |     0.29794 |    0.092269 |    0.092112 |       kernel |    1.082e-13 |
|   12 | Accept |    0.092269 |     0.36126 |    0.092269 |    0.092128 |       kernel |   2.5372e-16 |
|   13 | Accept |    0.092269 |     0.29318 |    0.092269 |    0.092154 |       kernel |   1.2612e-13 |
|   14 | Accept |    0.092269 |     0.28677 |    0.092269 |    0.092169 |       kernel |    2.257e-16 |
|   15 | Accept |    0.092269 |     0.28822 |    0.092269 |    0.092184 |       kernel |   1.5082e-05 |
|   16 | Accept |    0.092269 |     0.28872 |    0.092269 |    0.092264 |       kernel |   5.5484e-08 |
|   17 | Accept |    0.092269 |     0.30138 |    0.092269 |    0.091763 |       kernel |   1.1275e-06 |
|   18 | Accept |    0.092269 |     0.29565 |    0.092269 |    0.091954 |       kernel |   1.2661e-06 |
|   19 | Accept |    0.092269 |     0.29238 |    0.092269 |    0.091966 |       kernel |    3.057e-10 |
|   20 | Accept |    0.092269 |     0.31391 |    0.092269 |    0.092051 |       kernel |   1.3293e-06 |
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|   21 | Accept |    0.092269 |     0.31408 |    0.092269 |    0.092053 |       kernel |   1.6248e-17 |
|   22 | Accept |    0.092269 |     0.35191 |    0.092269 |    0.092056 |       kernel |   2.0584e-11 |
|   23 | Accept |    0.092269 |     0.29142 |    0.092269 |    0.092102 |       kernel |   1.5588e-06 |
|   24 | Accept |    0.092269 |      0.3614 |    0.092269 |    0.092106 |       kernel |   2.1439e-09 |
|   25 | Accept |    0.092269 |     0.31427 |    0.092269 |    0.092108 |       kernel |   2.0104e-17 |
|   26 | Best   |    0.057357 |      0.3513 |    0.057357 |    0.057437 |       kernel |     0.023599 |
|   27 | Best   |    0.054863 |     0.29875 |    0.054863 |    0.054985 |       kernel |     0.008381 |
|   28 | Accept |    0.054863 |     0.31494 |    0.054863 |    0.054919 |       kernel |    0.0080243 |
|   29 | Accept |    0.054863 |     0.30308 |    0.054863 |    0.054899 |       kernel |    0.0082181 |
|   30 | Accept |    0.054863 |     0.31964 |    0.054863 |    0.054889 |       kernel |    0.0082078 |
```



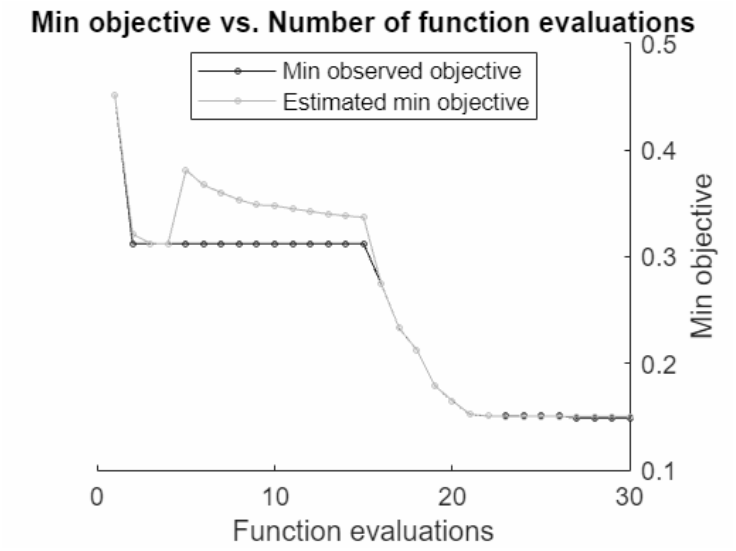


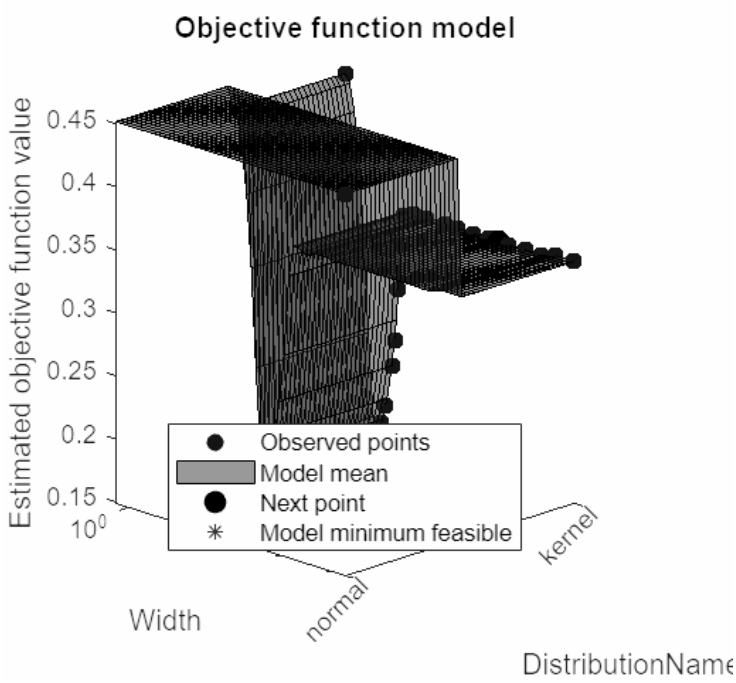

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 26.7465 seconds
Total objective function evaluation time: 9.0847

Best observed feasible point:
    DistributionNames     Width
    _________________    ________

         kernel          0.008381

Observed objective function value = 0.054863
Estimated objective function value = 0.054889
Function evaluation time = 0.29875

Best estimated feasible point (according to models):
    DistributionNames     Width
```

```
    _________________    ________

         kernel          0.008381

Estimated objective function value = 0.054889
Estimated function evaluation time = 0.32103
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Distribution-Names	Width
1	Best	0.45166	0.091019	0.45166	0.45166	normal	-
2	Best	0.3122	0.42133	0.3122	0.32101	kernel	0.29069
3	Accept	0.33756	0.38779	0.3122	0.31226	kernel	2.9322e-09
4	Accept	0.45166	0.070794	0.3122	0.31223	normal	-
5	Accept	0.43265	0.44432	0.3122	0.38095	kernel	20.374
6	Accept	0.34073	0.3366	0.3122	0.36692	kernel	3.6561e-22
7	Accept	0.34073	0.32401	0.3122	0.3598	kernel	3.6802e-22
8	Accept	0.34073	0.33545	0.3122	0.35312	kernel	3.6555e-22
9	Accept	0.34073	0.30005	0.3122	0.34841	kernel	3.6511e-22
10	Accept	0.33756	0.30449	0.3122	0.34756	kernel	1.0511e-15
11	Accept	0.33756	0.36037	0.3122	0.34488	kernel	9.5736e-14
12	Accept	0.33756	0.30389	0.3122	0.34226	kernel	1.2554e-10
13	Accept	0.33756	0.30612	0.3122	0.33992	kernel	1.5919e-07
14	Accept	0.33756	0.34532	0.3122	0.33812	kernel	3.6006e-06
15	Accept	0.33439	0.3051	0.3122	0.33679	kernel	2.6813e-05
16	Best	0.27417	0.31705	0.27417	0.27418	kernel	0.00011114
17	Best	0.23296	0.33074	0.23296	0.23299	kernel	0.00021708
18	Best	0.21236	0.37073	0.21236	0.21236	kernel	0.0004719
19	Best	0.17908	0.32695	0.17908	0.17912	kernel	0.0017581
20	Best	0.16482	0.33428	0.16482	0.16484	kernel	0.0048222

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Distribution-Names	Width
21	Best	0.15214	0.38138	0.15214	0.15223	kernel	0.010469
22	Best	0.15055	0.33929	0.15055	0.15062	kernel	0.017688
23	Accept	0.15055	0.33819	0.15055	0.15022	kernel	0.014497
24	Accept	0.15055	0.32161	0.15055	0.15032	kernel	0.014748
25	Accept	0.15055	0.33521	0.15055	0.15037	kernel	0.014871
26	Accept	0.33756	0.36792	0.15055	0.15037	kernel	7.8801e-19
27	Best	0.14897	0.32924	0.14897	0.15009	kernel	0.014445
28	Accept	0.33756	0.34896	0.14897	0.15009	kernel	3.3867e-12
29	Accept	0.33756	0.30959	0.14897	0.15009	kernel	2.8127e-17
30	Accept	0.34073	0.31627	0.14897	0.15009	kernel	2.4212e-20



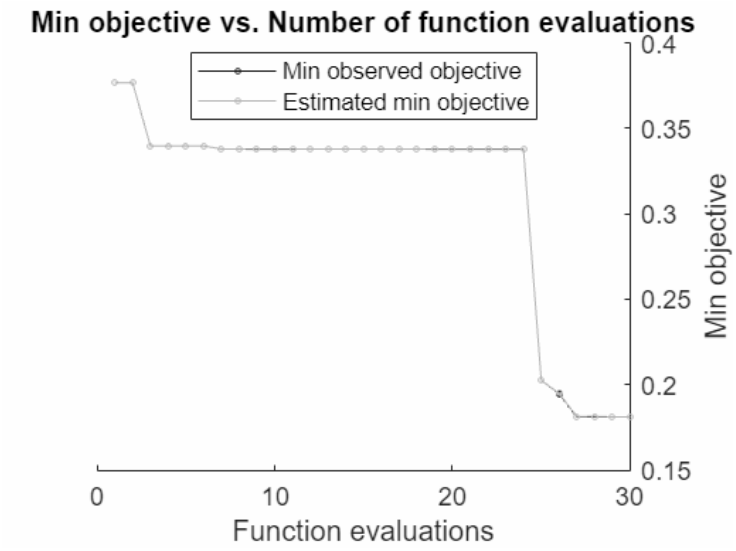


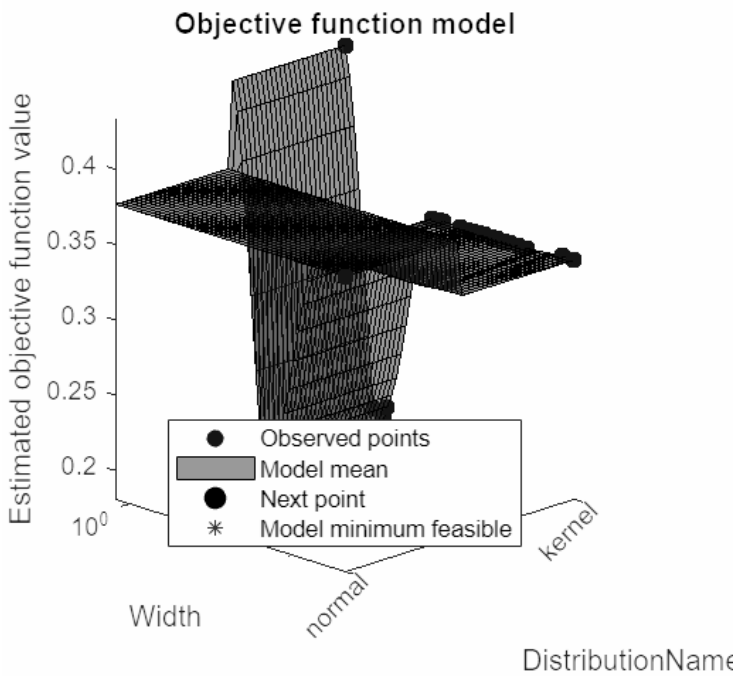
```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 29.2026 seconds
Total objective function evaluation time: 9.704

Best observed feasible point:
    DistributionNames     Width
    _________________    ________

         kernel          0.014445

Observed objective function value = 0.14897
Estimated objective function value = 0.1501
Function evaluation time = 0.32924

Best estimated feasible point (according to models):
    DistributionNames     Width
```

```
    _________________    ________

          kernel          0.014748

Estimated objective function value = 0.15009
Estimated function evaluation time = 0.34627
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Distribution-Names	Width
1	Best	0.37683	0.089353	0.37683	0.37683	normal	-
2	Accept	0.37683	0.06456	0.37683	0.37683	normal	-
3	Best	0.33982	0.35098	0.33982	0.33983	kernel	1.5517e-19
4	Accept	0.37683	0.073196	0.33982	0.33983	normal	-
5	Accept	0.33982	0.36372	0.33982	0.33983	kernel	1.176e-20
6	Accept	0.43427	0.50104	0.33982	0.33982	kernel	20.694
7	Best	0.33788	0.31392	0.33788	0.33789	kernel	9.0705e-15
8	Accept	0.33788	0.31251	0.33788	0.33787	kernel	2.9787e-16
9	Accept	0.33788	0.32013	0.33788	0.33777	kernel	1.6093e-15
10	Accept	0.33788	0.3368	0.33788	0.3378	kernel	1.7104e-15
11	Accept	0.33788	0.32091	0.33788	0.33782	kernel	1.7815e-15
12	Accept	0.33788	0.36865	0.33788	0.33788	kernel	1.916e-07
13	Accept	0.33788	0.36474	0.33788	0.33785	kernel	4.714e-10
14	Accept	0.33788	0.32693	0.33788	0.33786	kernel	1.414e-08
15	Accept	0.33788	0.33062	0.33788	0.33786	kernel	2.5214e-12
16	Accept	0.33788	0.30902	0.33788	0.33786	kernel	4.0364e-08
17	Accept	0.33788	0.35067	0.33788	0.33786	kernel	3.0706e-11
18	Accept	0.33788	0.32302	0.33788	0.33786	kernel	1.5729e-13
19	Accept	0.33788	0.35489	0.33788	0.3378	kernel	4.04e-08
20	Accept	0.33788	0.37683	0.33788	0.3378	kernel	8.7066e-12

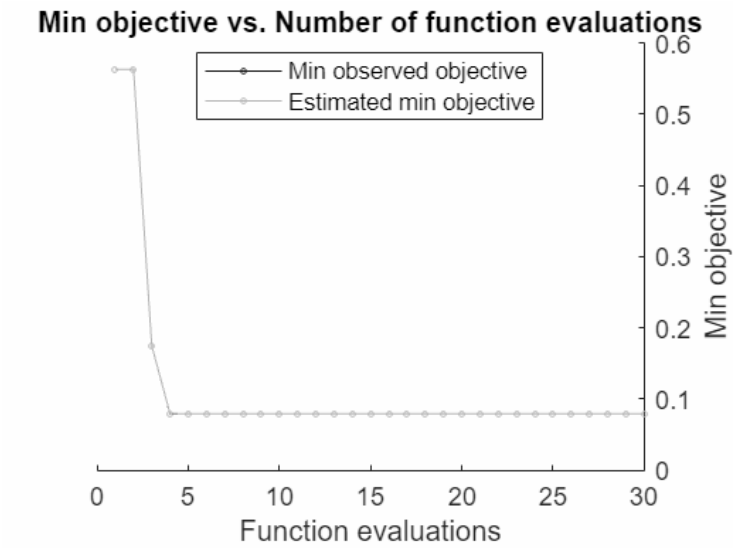
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Distribution-Names	Width
21	Accept	0.33788	0.32226	0.33788	0.33782	kernel	4.3957e-08
22	Accept	0.33788	0.30009	0.33788	0.33782	kernel	4.8155e-13
23	Accept	0.33788	0.31157	0.33788	0.33782	kernel	3.2083e-14
24	Accept	0.33788	0.29445	0.33788	0.33782	kernel	8.9693e-11
25	Best	0.20253	0.35151	0.20253	0.20254	kernel	0.0027181
26	Best	0.19474	0.44938	0.19474	0.19485	kernel	0.005611
27	Best	0.18111	0.3807	0.18111	0.18138	kernel	0.01394
28	Accept	0.19961	0.43557	0.18111	0.18148	kernel	0.04606
29	Best	0.18111	0.44785	0.18111	0.18111	kernel	0.017032
30	Accept	0.18111	0.3915	0.18111	0.18112	kernel	0.016734

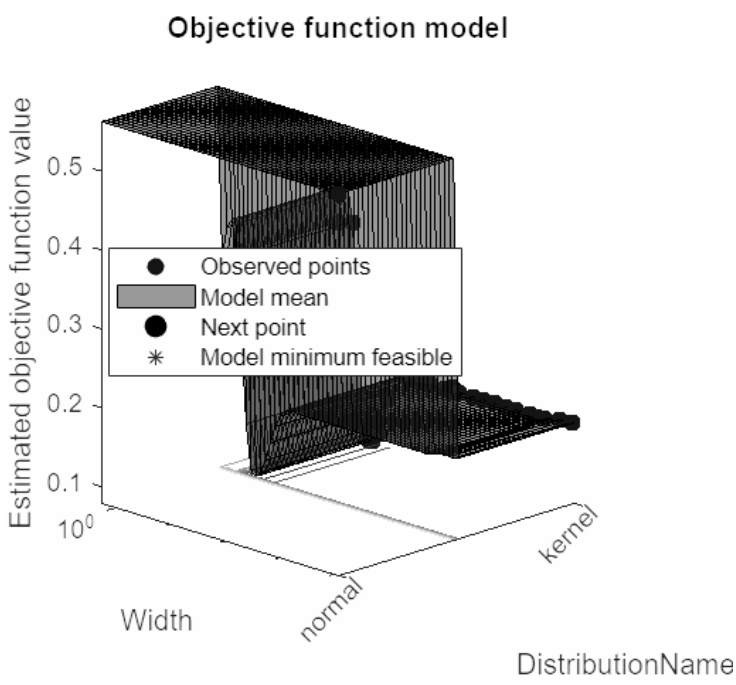




```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 27.8803 seconds
Total objective function evaluation time: 9.8374

Best observed feasible point:
    DistributionNames     Width
    _________________    ________

         kernel          0.017032

Observed objective function value = 0.18111
Estimated objective function value = 0.18114
Function evaluation time = 0.44785

Best estimated feasible point (according to models):
    DistributionNames     Width
```

```
    _________________    ________

         kernel          0.016734

Estimated objective function value = 0.18112
Estimated function evaluation time = 0.41282
Warning: It is recommended that you first standardize all numeric predictors when optimizing the Naive Bayes
'Width' parameter. Ignore this warning if you have done that.
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|    1 | Best   |     0.56291 |    0.074793 |     0.56291 |     0.56291 |       normal |            - |
|    2 | Accept |     0.56291 |    0.063593 |     0.56291 |     0.56291 |       normal |            - |
|    3 | Best   |      0.1746 |     0.35421 |      0.1746 |     0.17463 |       kernel |  9.2214e-12 |
|    4 | Best   |    0.079365 |     0.32959 |    0.079365 |    0.079448 |       kernel |     0.016104 |
|    5 | Accept |     0.34392 |     0.30224 |    0.079365 |    0.079395 |       kernel |       20.608 |
|    6 | Accept |    0.079365 |     0.30089 |    0.079365 |    0.079381 |       kernel |     0.016081 |
|    7 | Accept |    0.084656 |     0.27907 |    0.079365 |     0.07934 |       kernel |     0.046042 |
|    8 | Accept |      0.1746 |     0.37763 |    0.079365 |    0.079312 |       kernel |  1.3795e-06 |
|    9 | Accept |      0.1746 |      0.2878 |    0.079365 |    0.079311 |       kernel |  1.0305e-15 |
|   10 | Accept |     0.17989 |     0.28688 |    0.079365 |     0.07932 |       kernel |  6.3649e-21 |
|   11 | Accept |      0.1746 |     0.36127 |    0.079365 |    0.079285 |       kernel |   2.155e-18 |
|   12 | Accept |     0.13757 |     0.28063 |    0.079365 |    0.079377 |       kernel |  0.00050848 |
|   13 | Accept |      0.1746 |     0.28311 |    0.079365 |    0.079392 |       kernel |  3.1841e-09 |
|   14 | Accept |    0.079365 |     0.31302 |    0.079365 |    0.079206 |       kernel |     0.019665 |
|   15 | Accept |    0.079365 |     0.30086 |    0.079365 |    0.079243 |       kernel |     0.019874 |
|   16 | Accept |      0.1746 |     0.33087 |    0.079365 |    0.079259 |       kernel |  9.3401e-14 |
|   17 | Accept |     0.17989 |     0.29663 |    0.079365 |    0.079271 |       kernel |  9.6916e-20 |
|   18 | Accept |      0.1746 |     0.28836 |    0.079365 |    0.079283 |       kernel |  6.5641e-08 |
|   19 | Accept |      0.1746 |     0.29814 |    0.079365 |    0.079296 |       kernel |  5.2849e-17 |
|   20 | Accept |      0.1746 |     0.31828 |    0.079365 |    0.079308 |       kernel |  1.5218e-10 |
|=====================================================================================================|
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | Distribution-|        Width |
|      | result |             | runtime     | (observed)  | (estim.)    | Names        |              |
|=====================================================================================================|
|   21 | Accept |     0.34921 |     0.33185 |    0.079365 |    0.078934 |       kernel |      0.95913 |
|   22 | Accept |    0.089947 |       0.308 |    0.079365 |    0.078977 |       kernel |     0.004486 |
|   23 | Accept |    0.079365 |     0.33671 |    0.079365 |    0.079102 |       kernel |     0.026302 |
|   24 | Accept |      0.1746 |     0.30914 |    0.079365 |    0.079089 |       kernel |  2.6497e-05 |
|   25 | Accept |    0.079365 |      0.2969 |    0.079365 |    0.079156 |       kernel |      0.02493 |
|   26 | Accept |    0.079365 |     0.31932 |    0.079365 |    0.079197 |       kernel |     0.024127 |
|   27 | Accept |      0.1746 |     0.31138 |    0.079365 |    0.079189 |       kernel |  8.6366e-13 |
|   28 | Accept |      0.1746 |     0.33084 |    0.079365 |    0.079181 |       kernel |  9.0971e-15 |
|   29 | Accept |      0.1746 |     0.30949 |    0.079365 |    0.079173 |       kernel |  1.0512e-17 |
|   30 | Accept |      0.1746 |     0.29542 |    0.079365 |    0.079166 |       kernel |  2.9603e-07 |
```





```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 27.4206 seconds
Total objective function evaluation time: 8.8769

Best observed feasible point:
    DistributionNames     Width
    _________________    ________

         kernel          0.016104

Observed objective function value = 0.079365
Estimated objective function value = 0.079613
Function evaluation time = 0.32959

Best estimated feasible point (according to models):
    DistributionNames     Width
```

```
    _________________    ________

         kernel          0.019874

Estimated objective function value = 0.079166
Estimated function evaluation time = 0.31108
```

Naive Bayes Visualisation

The results of every layer of the hypermatrix are summed up and visualised in a heatmap. Additionally the mean accuracy and time taken is calculated and used as a comparison metric between methods.

```
NB = zeros(m,n);
for j = 1:n
    for k = 1:m
        NB(j,k) = sum(hyp_NB(j,k,:));
    end
end
figNB = figureGen(7,10);
heat_NB6 = heatmap(NB, "XDisplayLabels", [0,1,2,3], "YDisplayLabels", [0,1,2,3],
"ColorMethod", "mean", "ColorLimits", [0,100])
```

```
heat_NB6 =
  HeatmapChart with properties:

        XData: {4×1 cell}
        YData: {4×1 cell}
    ColorData: [4×4 double]

  Show all properties
```

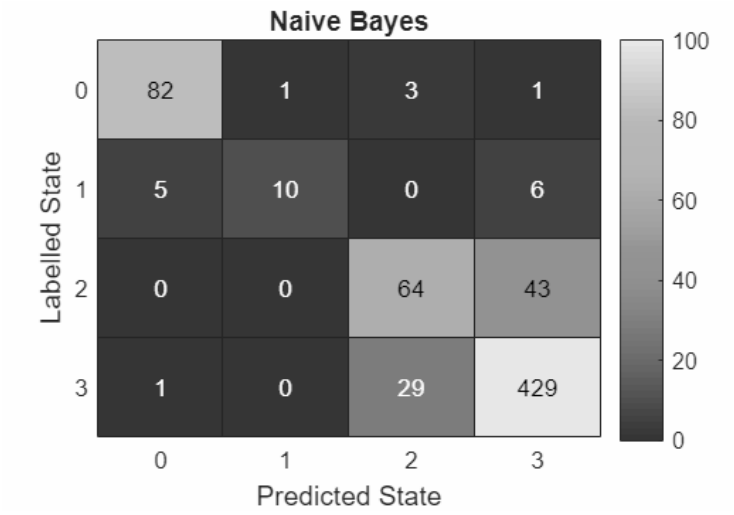
```
heat_NB6.Colormap = parula(64);
xlabel("Predicted State");
ylabel("Labelled State");
average_acc_NB = median(acc_NB)
```

```
average_acc_NB = 0.8851
```

```
average_time_NB = median(time_NB)
```

```
average_time_NB = 0.0394
```

```
heat_NB6.Title = "Naive Bayes";
saveas(heat_NB6, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\6_NB.jpg');
```

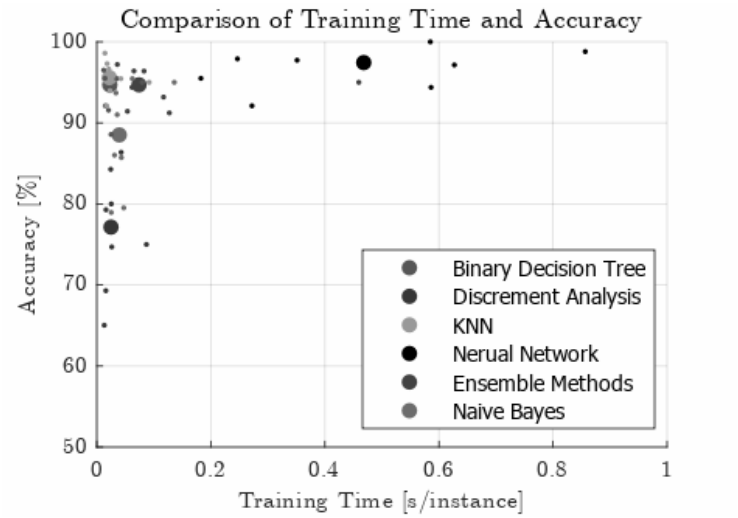


Comparison Accuracy and Time

Here we will now compare the average time and accuracy of each method.

```
accuracy = [average_acc_BT,  average_acc_DA, average_acc_KNN, average_acc_NN,
average_acc_En, average_acc_NB];
trainingTime = [avergage_time_BT, avergage_time_DA, avergage_time_KNN,
average_time_NN, average_time_En, average_time_NB];
classifierMethods = categorical(["Binary Decision Tree", "Discrement Analysis",
"KNN","Nerual Network","Ensemble Methods", "Naive Bayes"]);
fig = figureGen(7,10);
% gscatter(trainingTime,accuracy*100,classifierMethods)
% hold on;

scatter(trainingTime(1), accuracy(1)*100, [], [2/255 93/255 249/255], 'filled');
hold on;
scatter(trainingTime(2), accuracy(2)*100, [], [249/255 2/255 6/255], 'filled');
hold on;
scatter(trainingTime(3), accuracy(3)*100, [], [249/255 142/255 2/255], 'filled');
hold on;
scatter(trainingTime(4), accuracy(4)*100, [], [0 0 0], 'filled');
hold on;
scatter(trainingTime(5), accuracy(5)*100, [], [216/255 2/255 249/255], 'filled');
hold on;
scatter(trainingTime(6), accuracy(6)*100, [], [51/255 135/255 22/255], 'filled');
hold on;
scatter(time_BT, acc_BT*100, [], [2/255 93/255 249/255], '.');
hold on;
scatter(time_DA, acc_DA*100, [], [249/255 2/255 6/255], '.');
hold on;
scatter(time_KNN, acc_KNN*100, [], [249/255 142/255 2/255], '.');
```

```
hold on;
scatter(time_NN, acc_NN*100, [], [0 0 0], '.');
hold on;
scatter(time_En, acc_En*100, [], [216/255 2/255 249/255], '.');
hold on;
scatter(time_NB, acc_NB*100, [], [51/255 135/255 22/255], '.');
hold on;

legend("Binary Decision Tree", "Discrement Analysis", "KNN","Nerual
Network","Ensemble Methods", "Naive Bayes")
legend("Position", [0.48997,0.17366,0.39797,0.34595])


ylabel('Accuracy [\%]');
xlabel('Training Time [s/instance]');
ylim([50,100]);
xlim('auto');
grid on;

title('Comparison of Training Time and Accuracy');
```




```
saveas(fig, 'C:\Users\ellio\OneDrive\Dokumente\Uni\Master\Master
Thesis\Writing\LatexVorlageMA_v2.0\LatexVorlageMA_v2.0\chapters\Chapters\07_Results\
figures\6_Comparison.jpg');
```