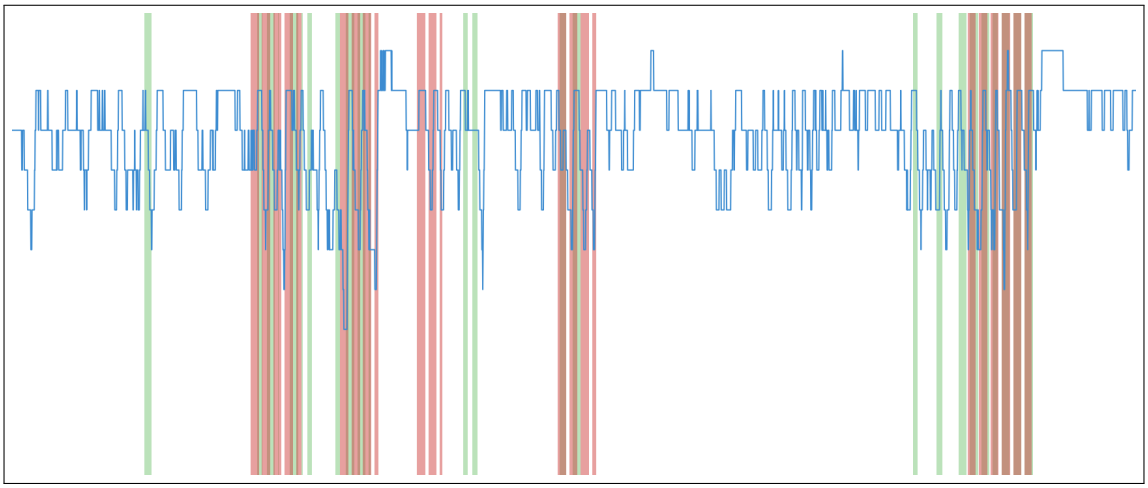
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Time Series Analysis for Sleep Apnea Detection Using Machine Learning

Bachelor's thesis in Computer science and engineering

Bavell Abdulla
Lucas Ekstener
Jacob Hedengran
Rasmus Jakobsson
Joakim Lindström
Isak Ulin

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Bachelor's thesis 2026

Time Series Analysis for Sleep Apnea Detection Using Machine Learning

Bavell Abdulla
Lucas Ekstener
Jacob Hedengran
Rasmus Jakobsson
Joakim Lindström
Isak Ulin

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Time Series Analysis for Sleep Apnea Detection Using Machine Learning
Bavell Abdulla Lucas Ekstener Jacob Hedengran Rasmus Jakobsson Joakim Lindström Isak Ulin

Supervisor (Handledare): Oana Geman, Department of Computer Science and Engineering
Examiner: Patrik Jansson, Department of Computer Science and Engineering
Graded by teacher (Medrättande lärare): Birgit Grohe, Department of Computer Science and Engineering

Bachelor's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization constructed in Python showing predicted sleep apnea events on patient data.

Typeset in LaTeX
Gothenburg, Sweden 2026

Time Series Analysis for Sleep Apnea Detection Using Machine Learning
Bavell Abdulla, Lucas Ekstener, Jacob Hedengran, Rasmus Jakobsson, Joakim Lindström, Isak Ulin

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates the applicability of various machine learning models on sleep apnea diagnosis using blood oxygen saturation measured with SpO_2. Specifically, it examines the machine learning models k-NN, SVM, random forest and fully connected neural networks. The main focus is to determine whether this approach is a realistic and reliable diagnostic tool for sleep apnea detection. Additionally, the thesis aims to identify which machine learning model is best suited for this task. To evaluate this, the classification metrics precision, recall and F1-macro will be used. The utilized dataset contained 994 subjects, from which nine features were extracted after preprocessing. By evaluating the classification metrics of the developed models, the general conclusion is that fully connected neural networks are the most suitable for diagnosing sleep apnea, while k-NN models are the least suitable. In order to establish a proof of concept, a wearable device capable of measuring the oxygen saturation called EmotiBit was utilized to simulate the diagnosis.

Sammandrag

Denna studie undersöker tillämpligheten av olika maskininlärningsmodeller för diagnostisering av sömnapné genom analys av syremättnad i blodet mätt med SpO_2. Specifikt undersöks maskininlärningsmodellerna k-NN, SVM, random forest och fullt sammankopplade neurala nätverk. Huvudfokus är att avgöra om detta tillvägagångssätt utgör ett realistiskt och tillförlitligt diagnostiskt verktyg för detektion av sömnapné, samt att fastställa vilken typ av modell som är bäst lämpad för detta ändamål. För att utvärdera detta används vanliga prestandamått för maskininlärningsmodeller, nämligen precision, recall och F1-macro. Den valda datamängden innehöll 994 patienter, och efter preprocessing extraherades nio features som användes för modellträning. Utifrån utvärdering av prestandamåtten är den övergripande slutsatsen att fullt sammankopplade neurala nätverksmodeller är mest lämpade för diagnostisering av sömnapné, medan k-NN-modeller är minst lämpade. Som konceptvalidering användes den bärbara enheten EmotiBit, som kan mäta syremättnad, för att simulera diagnostiken.

Keywords: Sleep apnea, machine learning, SpO_2, oxygen saturation, classification, fully connected neural networks, random forest, support vector machine, k-nearest neighbors, wearable device

Acknowledgements

We would like to thank our supervisor, Oana Geman, for her continuous support during this thesis project. Her guidance, advice and constructive feedback have been valuable throughout our work.

We are also grateful to our co-examiner, Birgit Grohe, for answering our questions when needed.

Special thanks are also directed to Dr. M. Brandon Westover for his assistance with questions related to the dataset used in this project.

Bavell Abdulla, Lucas Ekstener, Jacob Hedengran, Rasmus Jakobsson, Joakim Lindström, Isak Ulin, Gothenburg, May 2026

Contents

List of Figures

List of Tables

1
Introduction

Sleep apnea is a serious health disorder characterized by repeated temporary pauses in breathing during sleep. According to global data, it is estimated that nearly 936 million adults between the ages of 30 and 69 years old are affected by some form of sleep apnea [1]. Furthermore, studies show that more than 80% of sleep apnea cases remain undiagnosed due to the often subtle and predominantly nocturnal symptoms [2]. The condition severely impairs sleep quality and is associated with an elevated risk of metabolic and cardiovascular diseases, as well as cognitive dysfunction, including memory deficits and impaired attention [3].

There are three major forms of sleep apnea, namely obstructive sleep apnea (OSA), central sleep apnea, and mixed sleep apnea [4]. The most prevalent form, OSA, occurs when the upper airway partially or completely collapses during sleep. This results in repeated sleep fragmentation and intermittent hypoxia, defined as periods of reduced blood oxygen saturation. Central sleep apnea is a neurological condition characterized by the temporary cessation of respiratory effort due to impaired signaling from the nervous system. These disturbances in breathing during sleep are collectively referred to as respiratory events. Mixed sleep apnea is characterized by features of both central sleep apnea and OSA, in which a respiratory event initially presents as central sleep apnea and subsequently develops obstructive components.

Sleep apnea severity is classified using the apnea-hypopnea index (AHI), which represents the number of apnea and hypopnea events per hour of sleep [5]. Hypopnea refers to a partial reduction in airflow during sleep, typically associated with a drop in oxygen saturation or arousal from sleep, but not a complete cessation of breathing. For an event to be considered apnea or hypopnea, it has to persist for a minimum of 10 seconds. See Table 1.1 for the severity classification.

Table 1.1: Sleep apnea severity classification based on AHI [5].

Severity	**AHI (events/hour)**
Normal	$\text{AHI} < 5$
Mild	$5 \leq \text{AHI} < 15$
Moderate	$15 \leq \text{AHI} < 30$
Severe	$\text{AHI} \geq 30$

Polysomnography (PSG) is considered the clinical gold standard for diagnosing sleep apnea and typically involves simultaneous recording of several physiological signals during sleep [6]. By combining these signals, PSG enables comprehensive monitoring of both cardiovascular and neurological activity. This allows for highly accurate detection of sleep-related breathing disorders, including sleep apnea. However, PSG is often expensive and time-consuming [7], which may restrict access for economically disadvantaged populations. Due to these limitations, there has been increasing interest in developing automated and more accessible systems that rely on a reduced number of physiological signals while maintaining high diagnostic performance. In particular, approaches utilizing machine learning (ML), together with fewer and less invasive signals, offer a promising alternative for scalable and cost-effective sleep apnea assessment [8]. These approaches represent the most commonly used alternative screening method to conventional PSG [9]. Several studies have reported strong performance in sleep apnea detection using machine learning, but the field remains heterogeneous, with studies employing a wide range of objectives and methodologies [10]. Addi-

tionally, relatively few studies incorporate wearable devices, despite some reporting near-perfect correlations in AHI estimation [11].

Commonly used signals for apnea detection include electroencephalography (EEG), electrocardiogram (ECG), and blood oxygen saturation [12]. EEG records the electrical activity of the brain and is particularly useful for identifying sleep stages and detecting arousals associated with respiratory disturbances [13]. ECG captures the electrical activity of the heart across cardiac cycles and can provide indirect indicators of sleep apnea through heart rate variability. Blood oxygen saturation can be measured using two distinct measures, namely SpO_2 and SaO_2, both of which are directly related to sleep apnea, due to oxygen saturation decreasing following an apnea event [14].

1.1 Purpose

The purpose of this thesis is to investigate and compare the performance of various machine learning models for detection of sleep apnea. The project aims to contribute to research on efficient and accurate detection methods that are less invasive and more cost-effective, potentially improving accessibility for a wider population. This may lead to improved awareness, diagnosis, and treatment of the condition. The proposed approach is both data-driven and model-driven, relying on real physiological data while developing machine learning models to detect patterns associated with sleep apnea.

1.2 Goals and milestones

The overarching goal of this project is to design and implement a pipeline for detecting sleep apnea events using machine learning. The objective is to assess the performance of different models rather than to develop a clinically deployable system.

To break down the primary goal into achievable milestones, the following four sub-tasks are used as guidance throughout the project:

1. Decide on a dataset and what physiological signal will be analyzed. Split data into train/-validation/test sets.
2. Implement a preprocessing solution and extract features.
3. Implement, train and evaluate machine learning models.
4. Perform a real case study with an EmotiBit sensor.

1.3 Scope

To ensure that this thesis is feasible within the 20-week timeframe and remains appropriate for the complexity of a bachelor's thesis, the following scope limitations apply.

The project is limited to the use of existing, publicly available datasets for training the implemented machine learning models. No new physiological data is collected for training purposes; however, small amounts of new data are recorded using an EmotiBit for testing purposes. Furthermore, this thesis exclusively focuses on the use of SpO_2 as the physiological signal under analysis. The signal is selected due to its non-invasive measurement and suitability for wearable devices, such as the EmotiBit. It also provides a clinically meaningful indicator of blood oxygen saturation, which can reflect respiratory disturbances during sleep, making it especially relevant for sleep apnea detection.

The primary focus of this project is the application of classical machine learning methods for apnea detection. In particular, the study examines the performance of support vector machines (SVM), random forests (RF), and k-nearest neighbors (k-NN). A deep learning approach is also implemented in the form of a fully connected neural network (FCNN).

The project will not aim to explore all possible machine learning architectures, as this would require substantial computational resources and time. Instead, the limited set of representative models will be trained and evaluated with different parameters to compare their performances.

1.4 Ethics

This project aligns directly with the United Nations Sustainable Development Goal 3: Good Health and Well-Being, defined as "Ensure healthy lives and promote well-being for all at all ages" [15]. OSA is a contributing factor to several serious health complications such as hypertension, cardiovascular diseases, and even cancer mortality [16][17]. By evaluating and identifying optimal ML models for detecting OSA, this research aims to lay a framework for easier diagnosis and therefore, treatment of affected individuals.

Current methods for the diagnosis of OSA are obtrusive and expensive and might discourage lower-income populations from seeking medical treatment. A study conducted in 2025 also found that the majority of studies on the economic burden of OSA have been conducted in high-income countries [7]. These countries have established healthcare systems, and failing to collect data from lower-income countries can lead to critical information being lost.

Detecting apnea with ML models has a good chance of being cheaper than traditional analysis, since the amount of time needed by medical professionals is reduced. Lowering costs could allow wider amounts of the public to get diagnosed which would make it possible for more to get the treatment they might need. However, it is crucial that ML as a diagnostic tool is accurate enough at identifying sleep apnea to compete with traditional methods, thereby enabling the possibility that everyone can get a reliable diagnosis regardless of income.

Another ethical aspect that needs to be considered is bias in machine learning models. Physiological signals and sleep patterns can vary across sexes, age groups and ethnicities [18]. It is therefore important that training data is sufficiently representative, since models may perform unevenly across different populations.

From a data privacy view, the used dataset in the project follows the General Data Protection Regulation (GDPR) as the data is anonymized.

2

Data-handling and machine learning theory

This chapter presents the theoretical background underlying the methods and concepts used in this project. It outlines the signal used, dataset structure, signal preprocessing techniques, feature engineering approaches, machine learning algorithms, and evaluation metrics employed.

2.1 Blood oxygen saturation

As previously discussed in the Introduction, oxygen saturation is a clinically relevant indicator for detecting sleep apnea. The two primary measurements of oxygen saturation are, SaO_2 and SpO_2.

SaO_2 is obtained through arterial blood gas (ABG) analysis, an invasive procedure in which blood is sampled from an artery to determine parameters such as pH, electrolyte levels, and oxygen saturation [19]. In contrast, SpO_2 provides a more practical and non-invasive alternative. It is measured using a pulse oximeter, often placed on the index finger, which employs sensors that detect red and infrared light to estimate oxygen saturation. A key difference between the two approaches is that ABG is a single momentary measurement, whereas SpO_2 monitoring can be performed continuously. Additionally, SpO_2 offers advantages in terms of accessibility and minimal invasiveness, making it well suited for sleep apnea screening applications.

A study conducted in 2020 compared SpO_2 with the standard ABG method and found that SpO_2 remains a valuable tool for assessing oxygen saturation, provided its limitations are taken into account [19]. Factors such as old age were found to affect accuracy, with reduced reliability observed in patients older than 65 years. However, the study also reported that the difference between SaO_2 and SpO_2 measurements was less than 3%, indicating that SpO_2 can be considered sufficiently accurate for clinical use.

2.2 Dataset

The dataset used for this project was the "You snooze, You win: the PhysioNet/Computing in Cardiology Challenge 2018" dataset [20] provided by Physionet [21]. The dataset contains 266.6 GB of PSG recordings of 1985 subjects, monitored in a Massachusetts General Hospital (MGH) laboratory. Of the 1985 recordings, 994 entries were annotated by certified sleep technologists at MGH. The annotations included, but were not limited to, central-, obstructive- and mixed-apnea and hypopnea events. During the recordings, various physiological signals were collected. Among these were oxygen saturation, the chosen feature to be analyzed for this project. All collected signals were sampled at 200 Hz, apart from SpO_2 which was resampled to 200 Hz after recording. The physical data for each patient are saved in binary data containers, MATLAB files, with a corresponding annotation file, with *.arousal* as the file extension which is a Waveform Database file that is developed by PhysioNet.

Statistical measurements for physiological features of the patients, as well as the reason for their visit to MGH, can be viewed in Table 2.1 and their impact on the project is discussed in sec-

tion 4.3. Notably, the age of most patients is below 65, which suggests that SpO_2 remains an accurate signal as discussed in section 2.1. Patients are reported to have visited for one of three reasons: a diagnostic study, a split night continuous positive airway pressure (CPAP) session or an all night CPAP session. Sleep apnea is commonly treated with CPAP, where a breathing mask is worn over the nose [22]. The pressure exerted by the mask prevents the upper airway soft tissue from collapsing, thereby reducing apnea events.

Table 2.1: Clinical characteristics of the MGH dataset. [20]

Clinical Feature	**Total**	**Training**	**Test**
Sample size	1,983	994	989
Age	55 (14.4)	55 (14.3)	55 (14.4)
Gender (% male)	65	67	63
BMI	33 (7.6)	33 (7.8)	33 (7.5)
AHI	19 (14.4)	19 (14.6)	18.9 (14.4)
ESS	8.6 (5.3)	8.5 (5.3)	8.7 (5.3)
Recording time (h)	7.7 (0.7)	7.7 (0.7)	7.7 (0.7)
Time in bed (h)	7.5 (0.7)	7.5 (0.7)	7.5 (0.7)
Sleep time (h)	6.2 (1.2)	6.2 (1.1)	6.1 (1.2)
Drug Use (%)			
Hypertension	40.9	41.0	40.6
Sleeping aids	28.3	29.0	27.8
Antidepressant	26.1	25.7	26.5
Neuroactive	19.1	20.8	17.5
Benzodiazepine	16.1	16.9	15.4
Diabetic	11.7	11.9	11.5
Opiate	7.4	8.1	6.7
Antihistamine	4.8	4.8	4.8
Stimulant	4.7	3.9	5.5
Neuroleptic	4.2	4.5	3.8
Herbal	4.2	4.3	4.0
Reason for visit (%)			
Diagnostic	41.8	41.16	42.47
Split night CPAP	38.35	37.95	39.03
All night CPAP	19.85	20.88	18.5

2.3 Signal preprocessing for physiological signals

Physiological signals, or biosignals, are susceptible to contamination by noise and artifacts, i.e. sudden unrealistic changes in signals, and are therefore preprocessed prior to analysis [23]. This interference arises from both internal sources, caused by body activities, and external sources, caused by unwanted disturbances. Common internal artifacts for biosignals are muscle activity and movement of electrodes from breathing or sweating. Similarly, examples of external artifacts are loose electrodes or the power line interference common for physiological signals.

A vital part of signal preprocessing is artifact detection [23]. The detection method itself is influenced by the availability of a reference artifact source, the number of recorded signal channels, and whether artifact removal is performed after detection. One approach to artifact detection is to define an amplitude threshold and regard any signal whose amplitude exceeds this threshold as an artifact. Once the artifact is detected, the contaminated segment can either be rejected or processed to remove the artifact without altering the desired signal [23].

2.4 Feature engineering

For classical machine learning, it is essential that the training data is well characterized, as otherwise the model may struggle to detect patterns in practice [24]. Therefore feature engineering is a necessary step in the development process. Feature engineering for physiological signals consists of extracting meaningful physiologically grounded attributes, commonly known as features, that reflect underlying biological mechanisms. Examples of such features are the mean or minimum value of a signal. It is important that extracted features are discriminative, meaning they can be used to distinguish between different classes. Examples of such classes could be a healthy person or a person diagnosed with sleep apnea. Furthermore, physiological signals are inherently non-stationary, meaning their attributes vary over time. Therefore, dividing the signals into smaller time segments, called windows, is essential for effective analysis [24]. For biosignals, multiple methods can be used for feature extraction and they are categorized into time-domain, frequency-domain, and time-frequency-domain.

The time-domain features describe variation in the signal amplitude, waveform shape and duration. In addition to the methods described in section 2.3, feature normalization is typically performed before further processing [25]. Since features can have different numerical ranges, scaling is performed to ensure that features with larger values do not disproportionately influence the model. One such method is the min-max normalization [26], which rescales values to a fixed range, usually with an interval between 0 and 1, but a custom range can be applied using the following formula [26]:

$$x' = a + \frac{(x - x_{\min})(b - a)}{x_{\max} - x_{\min}}, \tag{2.1}$$

where a and b are the new bounds and $x_{\min}$ and $x_{\max}$ are the current bounds. Another normalization method is z-score, which standardizes each feature to have a mean of zero and variance of one [25].

Time-domain statistical features provide basic descriptive information about the signal. Several commonly used statistical features characterize different properties of the signal distribution within each window. The mean represents the average value within the window, providing an overall level of oxygen saturation. The standard deviation quantifies the spread or variability of the signal, which captures fluctuations that may indicate transient oxygen desaturation events [26]. The skewness describes the asymmetry of the signal distribution [27], which may point out windows with a sharp downward desaturation typical of sleep apnea events. The kurtosis measures the tailedness of the distribution [27], which may highlight windows with extreme deviations from the mean. The minimum value captures the smallest value within the window, which may indicate an apnea event within the window.

Proportion-based features categorize the oxygen saturation values according to clinically relevant thresholds. These features describe how the signal is distributed across different saturation ranges, which can help capture patterns associated with sleep apnea events.

2.5 Machine learning theory

Machine learning (ML) is a field of computer science concerned with developing algorithms that create patterns from data to make decisions or predictions [28]. In supervised ML, the training data consists of input samples together with their corresponding output labels. For classification problems, these labels represent predefined output classes, and the trained model is used to predict the class to which previously unseen data belongs. The input is represented as a set of elements, where each element is a vector whose components are called features.

ML algorithms can generally be understood as combinations of four core components: a dataset, a model, a cost function, and an optimization procedure [29]. The dataset, consisting of the input features of the training data $\mathbf{X}$ and the corresponding output values $\mathbf{y}$, provide the observations

used for learning. The model, which can generally be expressed as $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$, represents the mathematical relationship between an input $\mathbf{x}$ and a predicted output $\hat{y}$ based on the model parameters $\boldsymbol{\theta}$. These parameters typically include weights and biases. Weights determine the strength and direction of the relationship between input features and the output, while biases act as adjustable constants that allow the model to shift its predictions independently of the input values. Furthermore, the cost function, denoted by $J(\boldsymbol{\theta})$, is computed by aggregating the discrepancies between the model predictions $f(\mathbf{x}; \boldsymbol{\theta})$ and the actual target value y over all training examples. Lastly, the optimization procedure determines how the model parameters $\boldsymbol{\theta}$ are adjusted in order to minimize the cost. Ideally, the optimization process would locate the global minimum for the cost function. However, for complex functions, especially those with several local minima, finding the global minimum may not be feasible. Therefore, it is often acceptable to stop at a local minimum with a sufficiently low cost value.

Classical ML models rely on manually chosen features that have been extracted using domain knowledge [28]. Consequently, their performance is highly dependent on the feature extraction. Examples of classical ML models are support vector machines (SVMs), random forests and k-nearest neighbors (k-NN). Contrary to classical ML models, deep learning (DL) models are multi-layered neural networks capable of learning feature representation from data, reducing the need for manual feature extraction [30]. A neural network consists of interconnected neurons that compute a weighted sum of their inputs, which is then passed through a function to produce an output. An example of a DL model is the fully connected neural network (FCNN) [29].

The ability of DL models to learn from virtually unprocessed data, combined with their frequently superior performance relative to classical ML models, might make them appear to be an obvious choice [31]. However, a DL approach also presents drawbacks. For a DL model to achieve good results, substantially more data is needed compared to a classical ML model. The training and deployment of a DL model is also more resource intensive, often requiring the use of high end GPUs, while classical ML models are lighter and more attractive for use in low-power and on-device scenarios [32]. Neural networks are often considered "black boxes", meaning that the complexity of the model makes it difficult for a human to comprehend the decision behind the result. In the medical field, this is an important aspect to consider as it directly affects the viability of the result, since medical experts do not accept a diagnosis without explanation [33].

2.5.1 Gradient descent

In many cases, the optimization procedure for ML algorithms consist of finding where the gradient of the cost function is zero. The gradient is given by a function's partial derivatives $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. For functions that depend on multiple variables, a partial derivative measures how the function changes when only one variable is changed while all other variables are kept fixed. For a model with parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)$, the partial derivative $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_i}$, describes how the cost function $J(\boldsymbol{\theta})$ changes with respect to one parameter θ_i. The partial derivatives with respect to all parameters are collected into a vector called the gradient [29]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix}.$$

The gradient gives information about the direction in which the cost function increases most rapidly. A point where every component of the gradient is equal to zero is called a critical point and can correspond to a minimum.

To minimize the cost function, the parameters should be updated in the direction that decreases the cost most rapidly. Since the gradient points in the direction where the cost function increases most rapidly, the parameters are updated in the opposite direction. This leads to the method of gradient descent, which iteratively proposes a new point $\boldsymbol{\theta}'$, meaning new parameter values, by

[29]:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}),$$

where ϵ is the learning rate. The learning rate is a positive scalar that controls the size of each update step. This process is repeated until the gradient becomes zero or sufficiently close to zero by using the gradient of the full cost function $J(\boldsymbol{\theta})$. However when computed over a large training set, this becomes computationally expensive. Stochastic gradient descent (SGD) addresses this by estimating the gradient using a subset of the training data instead of the entire dataset at each update step, and consequently forms the basis of the optimization procedure used in most deep learning models [29].

2.5.2 Underfitting and overfitting

A central goal in machine learning is to develop models that perform well not only on the data used during training, but also on new and previously unseen data [29]. This ability is called generalization. During training, a model is optimized using a training set. The error measured on this dataset is called the training error, and the aim of the learning algorithm is to minimize it. However, low training error alone is not sufficient. In machine learning, an additional objective is to achieve low generalization error, also called test error, which measures the expected performance of the model on new inputs following the same distribution as the data used for training. The generalization error is estimated using a test set, which is collected separately from the training set.

Since both the training set and the test set are sampled from the same distribution, the training error and generalization error could be thought of as being equal [29]. However, since the model parameters are chosen after observing the training set, the expected test error is greater than or equal to the training error. The performance of a machine learning algorithm therefore depends on two main factors: its ability to make the training error small, and its ability to keep the gap between training error and test error small. These factors correspond to two central challenges in machine learning: underfitting and overfitting. Underfitting occurs when the model is unable to obtain a sufficiently low error on the training set. Overfitting occurs when the gap between training error and test error becomes too large, meaning that the model has memorized the training data rather than learned the underlying patterns, and as a result generalizes poorly to new data.

A way to influence whether a model is more likely to underfit or overfit is to alter its capacity [29]. Model capacity refers to a model's ability to fit a wide range of functions. A model with too little capacity may be unable to fit the training data well, while a model with too much capacity may fit details of the training data that do not generalize to the test data. One way to control capacity is by changing the model's hypothesis space, which is the set of functions the learning algorithm is allowed to choose from. A model usually performs best when its capacity matches the complexity of the task and the amount of available training data. If the capacity is too low, the model may underfit because it cannot represent the underlying structure of the data. If the capacity is too high, the model may overfit because many different functions can fit the training data but fail to generalize well to new data.

2.5.3 Regularization

As stated above, one way to control a model is through its hypothesis space. If this space only contains simple functions, the model may be easier to train and less likely to overfit, but it may also fail to capture complex patterns in the data [29]. If the hypothesis space contains more flexible functions, the model can represent more complex relationships, but it may also become more prone to overfitting. Therefore, model performance depends not only on the size of the hypothesis space, but also on the specific types of functions included in it.

Regularization provides another way to guide learning by making some solutions preferable to others [29]. Instead of removing complex solutions completely, regularization allows them but penalizes them unless they improve the fit substantially. More generally, this can be written as

[29]:

$$J(\boldsymbol{\theta}) = J_{\text{train}}(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta}).$$

Here $J(\boldsymbol{\theta})$ is the total cost function, $J_{\text{train}}(\boldsymbol{\theta})$ is the training cost, $\Omega(\boldsymbol{\theta})$ is the regularization term, and λ controls the strength of the regularization. For instance, the regularization term could increase the whole cost by adding penalties for too large parameter values θ, expressing a preference for solutions with smaller parameter values. A larger value of λ gives more importance to the penalty, which can encourage simpler or less extreme solutions and reduce the risk of overfitting. However, if the regularization is too strong, the model may become too restricted and underfit the data. The intention of regularization is thus to reduce the generalization error, but not the training error.

Two examples of regularization are LASSO regression, also known as L_1, and ridge regression, also called L_2 [34]. In ridge regression, the regularization term is the sum of squared coefficients, or the weights, written as $\sum_{j=1}^{m} w_j^2$. Ridge regression usually shrinks weights toward zero, but does not remove variables completely. LASSO regression uses a penalty based on the absolute values of the weights, $\sum_{j=1}^{m} |w_j| \leq t$, where t controls how large the total weight values are allowed to be. A smaller value of t creates a stronger constraint. Unlike ridge regression, LASSO can set some weights exactly to zero, meaning it can remove less useful variables from the model completely.

2.5.4 Hyperparameters and validation sets

Most machine learning algorithms include hyperparameters, which are settings used to control the behavior of the learning algorithm [29]. Unlike model parameters, hyperparameters are not learned through the training process, but rather configured prior to it. An example of a hyperparameter is λ described in subsection 2.5.3 and which controls the strength of regularization.

Some settings are treated as hyperparameters because they are too difficult for the learning algorithm to optimize [29]. More importantly, some settings must be hyperparameters since they are not appropriate to learn on the training set. If they were selected only based on training performance, the algorithm would tend to choose the highest possible model capacity, since this reduces training error. However, this leads to overfitting.

In order to avoid this problem, a validation set is used [29]. The validation set consists of examples that are not used to learn the model parameters. Instead, it is used to estimate how well the model generalizes during or after training, and to guide the choice of hyperparameters. The original training data is therefore split into two separate parts: one part is used for learning the model parameters, and the other part is used for validation. A common split is to use around 80% of the training data for parameter learning and 20% for validation. After training and hyperparameter selection are complete, the test set is used to estimate the generalization error.

2.5.5 Bias and variance

The relationship between model complexity and performance is often described through the bias-variance trade-off [34]. Bias and variance describe two sources of error that affect how well a model generalizes to unseen data.

Bias refers to error caused by assumptions built into the model [34]. If these assumptions are too restrictive, the model may fail to capture the true relationship in the data. For example, if the underlying pattern is nonlinear, whilst the model is only able to represent a simple linear relationship, the model may produce systematically inaccurate predictions. High bias is therefore associated with underfitting, where the model performs poorly because it is not flexible enough to represent the data well.

Variance, on the other hand, refers to error caused by sensitivity to fluctuations in the training set [34]. A model with high variance may adapt too closely to the specific examples it was trained on, including noise from sampling or measurement errors. As a result, the model may fit

the training data very well, but fail to generalize to new data. High variance is therefore associated with overfitting.

The bias-variance trade-off shows that increasing model complexity can reduce bias, since a more flexible model can represent more complex relationships. However, this can also increase variance, because the model becomes more sensitive to noise in the training data. Conversely, a simpler model may have lower variance but higher bias. The goal is therefore to find a balance where the model is complex enough to capture the relevant structure in the data, but not so complex that it overfits.

2.6 Machine learning models

This section presents the theoretical background of the machine learning models used in this project. Specifically, it covers k-nearest neighbors (k-NN), support vector machines (SVM), random forests, and fully connected neural (FCNN).

2.6.1 k-nearest neighbors

The k-nearest neighbors (k-NN) classifies unlabeled data by assigning it to the class of the most similar labeled examples with respect to their feature values [35]. By representing each data point as a vector in a multidimensional feature space, unlabeled data can be interpreted as points in this space. The k-NN then examines a fixed number of nearest neighbors, k, and assigns the unlabeled data point to the class to which the majority of these neighbors belong. Figure 2.1 illustrates this classification process in a two-dimensional feature space. The distance to the k-nearest-neighbors

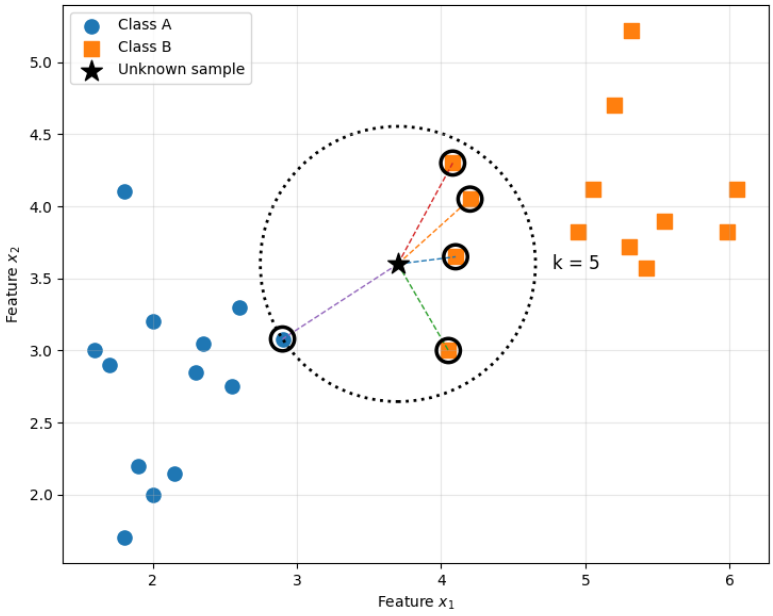


Figure 2.1: Illustration of binary k-NN classification with k = 5 in a two-dimensional feature space. The axes represent two input features, x_1 and x_2. The black star represents the unknown sample, while the surrounding labeled points indicate training samples from two classes. The dotted circle marks the neighborhood containing the five nearest neighbors. The sample will be assigned to the majority class among these neighbors, which in this case is class B.

can be calculated in numerous ways, but a standard way is using Euclidean distance [35]. This is shown in Equation 2.2, where $\mathbf{x}$ and $\mathbf{y}$ are points, n denotes the number of features, and i indexes a specific feature. Since features with larger numerical ranges may dominate the distance calculation, normalization is applied as mentioned in section 2.4.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.2}$$

As stated, the parameter k determines the number of nearest neighbors considered and affects the performance of k-NN. A larger value of k reduces the model's sensitivity to random fluctuations in the data, but could lead to the neglect of small yet important patterns [35]. One suggested approach is to set k equal to the square root of the number of observations in the training dataset.

2.6.2 Support vector machines

Support vector machines (SVM) are algorithms commonly used for binary classification that separate two classes by finding an optimal separating hyperplane, a generalization of a plane in three dimensions, in the feature space [34]. The hyperplane is chosen to maximize the margin, defined as the distance to the nearest training data points from each class, thereby achieving maximum margin separation. These nearest points are referred to as support vectors, as they determine the position of the hyperplane and define the margin boundaries. The equation of the hyperplane can be expressed as [34]:

$$\mathbf{w} \cdot \mathbf{x} - b = 0,$$

where $\mathbf{w}$ is a vector of weights, $\mathbf{x}$ is a feature vector and b is the bias determining the position of the hyperplane in the feature space. As mentioned above, $\mathbf{w}$ and b are determined by maximizing the margin, which is defined by the support vectors. The region between the margin boundaries is referred to as the separating channel and is defined by two hyperplanes parallel to and equidistant from the separating hyperplane. These two planes are given by the following equations [34]:

$$\mathbf{w} \cdot \mathbf{x} - b = 1,$$
$$\mathbf{w} \cdot \mathbf{x} - b = -1.$$

Figure 2.2 illustrates the separating hyperplane, the two margin boundaries, the support vectors, and the separating channel.
The distance between the two planes is given by $\frac{2}{\|\mathbf{w}\|}$ [34]. Thus, a smaller value of $\|\mathbf{w}\|$ corresponds to a greater margin. To ensure the margin is free of training data points, two constraints are imposed. If we denote the two output classes as class 1 and class -1, then for class 1 each training data point $\mathbf{x}$ must satisfy [34]:

$$\mathbf{w} \cdot \mathbf{x} - b \geq 1,$$

and similarly for class -1, each training data point $\mathbf{x}$ satisfies:

$$\mathbf{w} \cdot \mathbf{x} - b \leq -1.$$

Additionally, if we let $y_i \in \{-1, 1\}$ denote the class of $\mathbf{x}_i$, and n as the number of training data points, then the optimization problem solved by the SVM can be formulated as [34]:

$$\min \|\mathbf{w}\|, \quad \text{where } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 \text{ for all } 1 \leq i \leq n.$$

If there exists no separating hyperplane for a given set of training data points, then a projection of the points to a higher dimension is done [34]. In this higher-dimensional feature space, the existence of a separation might be possible. This is efficiently achieved using functions known as kernels.

2.6.3 Random forests

A random forest is an ensemble of decision trees [36]. A decision tree is a directed graph in which all edges are directed away from a root node and extend outward toward terminal nodes (also known as leaf nodes). The tree is composed of internal nodes, which have one or more children,

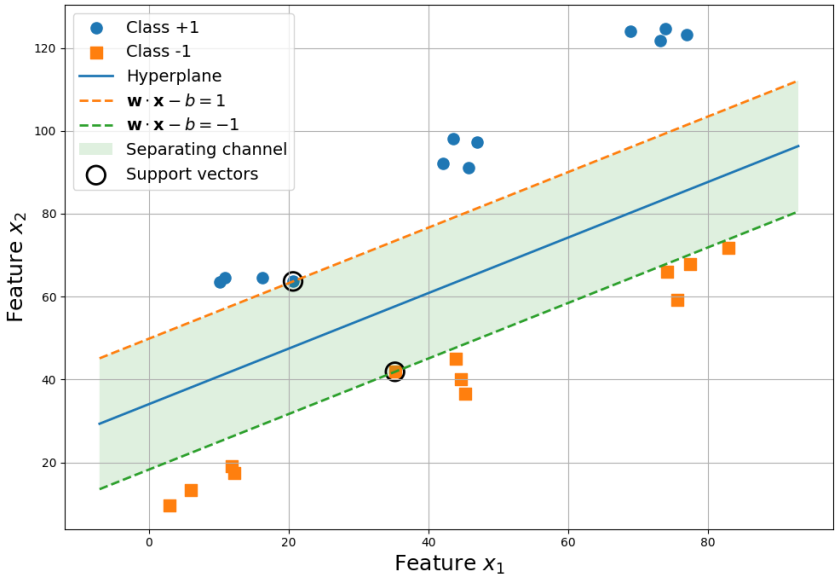


Figure 2.2: Visualization of a linear support vector machine in a two-dimensional feature space consisting of features x_1 and x_2. The solid line represents the separating hyperplane, the dashed lines represent the margin boundaries, and the circled points are the support vectors. The shaded area indicates the separating channel between the two margin boundaries.

and terminal nodes, which have no children. Each node corresponds to a subset of the input feature space, with the root node representing the entire feature space. At each internal node, a split based on one of the input features partitions the data into disjoint subsets, which are passed to the corresponding child nodes. This recursive partitioning continues until terminal nodes are reached, where each leaf is assigned a predicted output value, corresponding to a class label in classification tasks.

The goal of learning a decision tree is to construct a model that partitions the input feature space in a way that approximates the true relationship between the input variables and the output variable [36]. There may exist multiple decision tree models that achieve similar performance on the training data. Therefore, simpler trees are preferred.

Decision trees are constructed by recursively partitioning the training data into increasingly homogeneous subsets as the depth of the tree increases [36]. At each node, a split is selected that maximizes the reduction in impurity for the child nodes, where impurity measures the heterogeneity of the output variable within the node. This procedure continues until no further reduction in impurity can be achieved at the terminal nodes. See Figure 2.3 for a visualization of this partitioning.

As stated before, a random forest is an ensemble of decision trees [36]. The reason for combining multiple decision trees is that individual trees have low bias, but high variance. By aggregating their predictions, the variance is reduced, which lowers the expected prediction error on unseen data. Examples of aggregating functions for classification tasks are majority voting, in which the class most frequently predicted by the individual trees is selected, and soft voting, where each tree provides a probability distribution over the output classes, and the final prediction corresponds to the class with the highest average probability. In order for the aggregation to be beneficial, randomness is introduced to ensure diversity among the individual trees. Otherwise, if the models are highly correlated, their prediction error will not cancel out. One method introducing randomness is bagging, where instead of creating one tree from the training dataset, multiple datasets are generated from the original using bootstrap sampling (sampling with replacement), and separate trees are trained on each of these datasets [36].

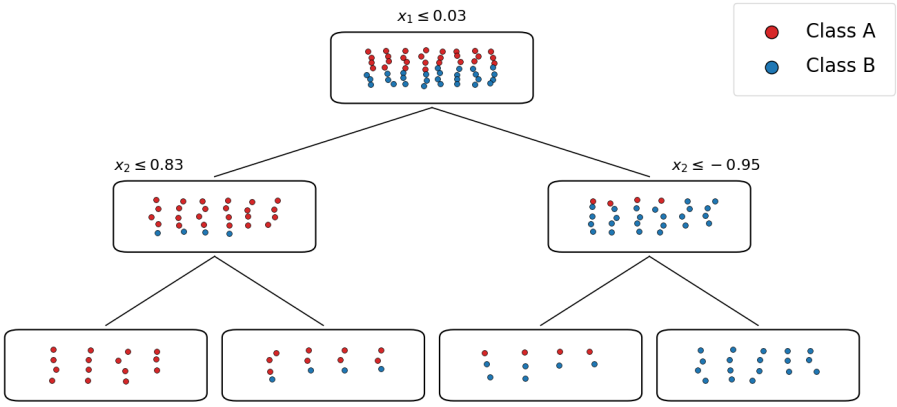


Figure 2.3: Illustration of a binary decision tree classifier using two features, x_1 and x_2, to recursively partition the training data into increasingly homogeneous subsets. Each node contains the training data assigned to that node, with red and blue points representing the two classes. The split condition displayed above each internal node determines the partition of the data. Observations satisfying the condition are sent to the left child node, while the remaining observations are sent to the right child node. As the tree depth increases the resulting child nodes become progressively purer.

2.6.4 Fully connected neural network

A fully connected neural network (FCNN), commonly referred to as a multilayer perceptron, is a type of feedforward neural network that represents a function that maps input variables to output variables [29]. Such a model is constructed by composing several functions in a sequence, where each function corresponds to a layer in the network.

For classification, a feedforward neural network defines a parametric function $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$, where $\mathbf{x}$ denotes the input, $\hat{y}$ the predicted output, and $\boldsymbol{\theta}$ the set of parameters [29]. During training of the model, parameter values are chosen such that $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$ approximates $y = f^*(\mathbf{x})$, which describes the true relationship between input and output variables. These neural networks are termed feedforward due to the unidirectional flow of information through the model. Specifically, data propagates from the input $\mathbf{x}$, through a series of intermediate computations that define the function f, and finally to the output $\hat{y}$. In this architecture, no feedback connections are present, meaning that outputs of the model are not reused as inputs. Feedforward neural networks are referred to as networks because they can be interpreted as compositions of multiple functions in a sequence [29]:

$$f(\mathbf{x}) = f^{(L)}(f^{(L-1)}(\dots f^{(1)}(\mathbf{x}))).$$

Each function corresponds to a layer in the network, where $f^{(1)}$ denotes the first layer, $f^{(2)}$ the second layer, and so on. The number of such layers, L, determines the depth of the model, which gives rise to the term "deep learning", and the final layer $f^{(L)}$ is called the output layer. As mentioned above, during training, the model is optimized such that the function $f(\mathbf{x})$ approximates an unknown target function $f^*(\mathbf{x})$. The output layer is constrained by the training data, since it is required to produce outputs that closely match the corresponding target values for each input. In contrast, the intermediate layers are not explicitly supervised. Instead, the learning algorithm determines how to adjust them to best approximate the target function, and therefore the training data do not directly constrain them. For this reason, these intermediate layers are commonly referred to as hidden layers [29]. Figure 2.4 illustrates this layered structure for an FCNN used in binary classification.

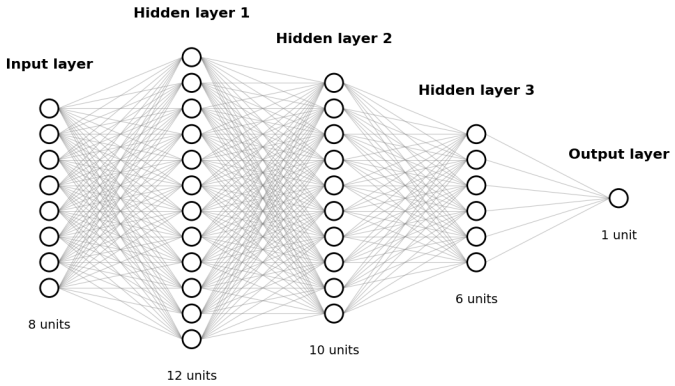


Figure 2.4: Illustration of a fully connected feedforward neural network for binary classification. The input layer represents the feature vector $\mathbf{x}$, the intermediate layers correspond to hidden layers, and the final output layer consists of a single unit, which can be used to produce a binary classification output.

Finally, these models are referred to as neural networks due to their conceptual inspiration from biological neural systems [29]. Each hidden layer can be seen as a vector, whose dimensionality corresponds to the number of units in that layer, also known as the width of the network. The individual components of these vectors can be interpreted as units, each performing a computation analogous to that of a biological neuron. Rather than viewing a layer as a single transformation from one vector to another, it can be understood as a collection of such units operating in parallel, where each unit computes a scalar output based on its inputs. These units aggregate information from multiple preceding units and generate activation values.

An FCNN is capable of modeling nonlinear relationships and capturing complex interactions between input variables [29]. This is achieved by transforming the input through a nonlinear mapping, denoted by $\phi(\mathbf{x})$, and subsequently applying a linear model to the transformed representation in order to get the model output. In this context, $\phi(\mathbf{x})$ can be interpreted as a feature mapping. A central challenge in this approach is the selection of an appropriate transformation ϕ. Deep learning solves this by learning the feature representation directly from the data. In this setting, ϕ is parameterized by $\boldsymbol{\theta}_\phi$, which contains the trainable parameters of the hidden layers. For a model with only one output, the complete set of model parameters is denoted by $\boldsymbol{\theta} = (\boldsymbol{\theta}_\phi, \mathbf{w}, b)$, where $\mathbf{w}$ and b are the weight vector and scalar bias of the final output layer. An example of a binary classification model that produces a scalar output z before the final prediction is obtained can be written as [29]:

$$z = \phi(\mathbf{x}; \boldsymbol{\theta}_\phi)^\top \mathbf{w} + b, \tag{2.3}$$

where $\phi(\mathbf{x}; \boldsymbol{\theta}_\phi)$ represents a parameterized transformation of the input, corresponding to the hidden layers of the network, and $\mathbf{w}$ maps this representation to the output. The parameters $\boldsymbol{\theta}_\phi$ are learned during training to obtain a representation that facilitates accurate prediction.

To make the feature mapping $\phi(\mathbf{x}; \boldsymbol{\theta}_\phi)$ concrete, an FCNN with one hidden layer can be considered [29]. The hidden representation can then be written as:

$$\mathbf{h} = g(\mathbf{W}^\top \mathbf{x} + \mathbf{c}),$$

where $\mathbf{W}$ is the weight matrix, $\mathbf{c}$ is the bias vector and g denotes a nonlinear activation function. The weight matrix $\mathbf{W}$ determines how the input variables are combined to form the hidden representation. In particular, its entries specify how strongly each input feature contributes to each hidden unit. The bias vector provides an additive offset for each hidden unit before the activation function is applied. A common choice for the activation function in modern neural networks is the rectified linear unit (ReLU), defined as $g(a) = \max(0, a)$, which is applied element-wise. The scalar output of the network is subsequently obtained as a linear combination of the hidden units:

$$z = \mathbf{w}^\top \mathbf{h} + b.$$

Combining these expressions yields the scalar output that is used to produce the final prediction $\hat{y}$ for this example network shown in Equation 2.4. For a network with multiple layers, each hidden layer would have its own weight matrix and bias vector.

$$z = \mathbf{w}^\top \max(0, \mathbf{W}^\top \mathbf{x} + \mathbf{c}) + b. \tag{2.4}$$

As mentioned in subsection 2.5.1, most DL models are optimized using SGD. Since neural networks include nonlinear components, their cost functions become nonconvex, meaning that they may contain several local minima, saddle points and flat regions [29]. As a result, SGD can not guarantee to reach the global minimum as for linear models, and the result is therefore dependent on the initial parameter values. In feedforward neural networks, the weights are thus usually initialized with small random values, while the biases can be initialized either to zero or small positive values. The specific optimization methods for these networks are generally extensions or refinements of SGD.

In many neural networks, the model represents a probability distribution $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ [29]. Here, the cost function used is cross-entropy, which measures how different the model's predicted probabilities are from the true labels. For binary classification, the target variable y has two possible values. This can be modeled using a Bernoulli distribution, which only requires the model to predict one probability $P(y = 1 \mid \mathbf{x})$, whose value must be within the interval $[0, 1]$ to represent a valid probability. One possible method would be to use a linear unit and force its output into the valid probability range [29]:

$$P(y = 1 \mid \mathbf{x}) = \max(0, \min(1, \mathbf{w}^\top \mathbf{h} + b)).$$

Although this produces a valid conditional distribution, it is not well suited for training with gradient descent [29]. If $\mathbf{w}^\top \mathbf{h} + b$ falls outside the interval $[0, 1]$, the gradient of the model output with respect to its parameters becomes zero. This is problematic because the learning algorithm no longer receives useful information about how the parameters should be changed.

A better approach is to transform the scalar value using a sigmoid output unit [29]. For the general model in Equation 2.3 the final predicted probability is given by:

$$\hat{y} = \sigma(z).$$

For the example network with one hidden layer, where $z = \mathbf{w}^\top \mathbf{h} + b$, it is given by:

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{h} + b).$$

Here σ is the logistic sigmoid function that gives a valid probability value. The logistic sigmoid function maps any real-valued input to a value between zero and one [29]:

$$\sigma(a) = \frac{1}{1 + e^{-a}}.$$

This makes it suitable for binary classification, where the output can be interpreted as the probability $P(y = 1 \mid \mathbf{x})$. In addition, when combined with cross-entropy, it provides useful gradients when the model makes incorrect predictions.

Lastly, in order to reduce overfitting in neural networks, dropout is used [29], a regularization method that during training, randomly deactivates a fraction of all units in the networks. Every unit is deactivated with a probability of p, known as the dropout rate. This technique forces the network to learn more robust representations and preventing units from co-adapting [37].

2.7 Evaluation of ML models

Performance of ML models is assessed using several classification metrics, based on demands within the relevant domain. This section provides background on classification metrics and presents those most relevant to sleep apnea detection.

All classification metrics are fundamentally based on four statistical measurements: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) [34]. A true positive is defined as an instance where the model correctly identifies the occurrence of a condition or event belonging to the positive class, while a false positive is a case where the model incorrectly assigns a positive label. Conversely, a true negative means the model correctly identified a negative instance, whereas a false negative occurs when the model fails to identify a positive instance. With these statistical measurements, different classification metrics are formed.

2.7.1 Recall

Recall (sensitivity) is one such metric that reflects the proportion of actual positive events correctly detected by the model. In the context of sleep apnea detection, particular emphasis is placed on recall, as failing to identify apnea events can have clinical consequences such as contributing to underdiagnosis [38]. Recall is defined as follows:

$$recall = \frac{TP}{TP + FN}.$$

2.7.2 Precision

Precision is another metric that measures the proportion of true positives among all instances predicted as positive and is defined as follows [34]:

$$precision = \frac{TP}{TP + FP}.$$

2.7.3 F1-score

F1-score is defined as the harmonic mean of recall and precision and is another frequently used classification metric [38]. It is defined as follows:

$$F1 = 2 \cdot \frac{recall \cdot precision}{recall + precision} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}.$$

2.7.4 F1-macro

If the dataset is imbalanced, the standard F1-score can be misleading, as it only evaluates performance for the positive class and does not account for performance across all classes [39]. This means that models might perform well in correctly labeling positive cases, but not negative ones. The macro-averaged F1-score, or F1-macro, addresses this by computing the F1-score for each class independently and taking their unweighted average. Consequently, the model's performance in all classes is valued with the same weight, and the F1-macro obtained indicates the performance across all classes. It is defined as follows, where $F1_i$ denotes the i-th class F1-score and N is the number of classes [39]:

$$F1_{\text{macro}} = \frac{1}{N} \sum_{i=1}^{N} F1_i.$$

2.7.5 Confusion matrix

A confusion matrix is a tool used for systematic comparison and visualization of a classification model's performance [34], see Figure 2.5. It displays the number of true positives, true negatives,

false positives, and false negatives, providing a clear overview of the model's performance and potential errors. By examining the confusion matrices of different models, their relative strengths and weaknesses in identifying respiratory events can be systematically compared.

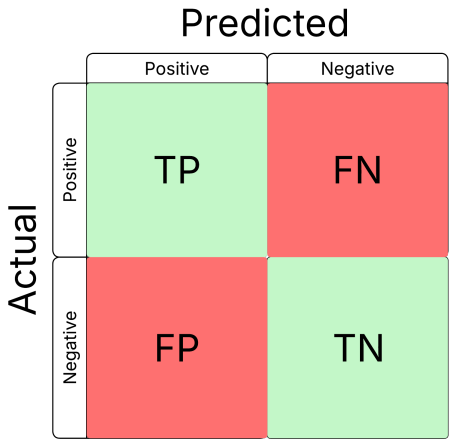


Figure 2.5: Confusion matrix

3
Method

This chapter presents the methodology applied in the project. It outlines the tools used and the procedures for data handling, preprocessing, feature extraction, model development, evaluation, and the case study.

3.1 Tools

This section presents the main tools used throughout the project, including the computational infrastructure, software libraries, the EmotiBit physiological sensing platform, and the algorithm used to estimate SpO_2 from the recorded signals.

3.1.1 Computing clusters

Given the computational complexity and data volume involved in training the models, access to high-performance computing (HPC) clusters was instrumental to ensure efficient model training and evaluation. During the training phase, an HPC cluster named Minerva [40], facilitated by the department of Computer Science and Engineering at Chalmers University of Technology, was utilized.

On Minerva [40], computations were performed through a batch scheduling system called SLURM. Training tasks were defined as batch jobs using shell scripts that specified the required computational resources, including runtime, CPU cores, GPUs, and memory. Subsequently, the jobs were submitted to the scheduler, where they were placed in a queue and executed once the requested resources became available. This approach ensured efficient utilization of computational resources and enabled concurrent execution of multiple tasks.

3.1.2 Python libraries

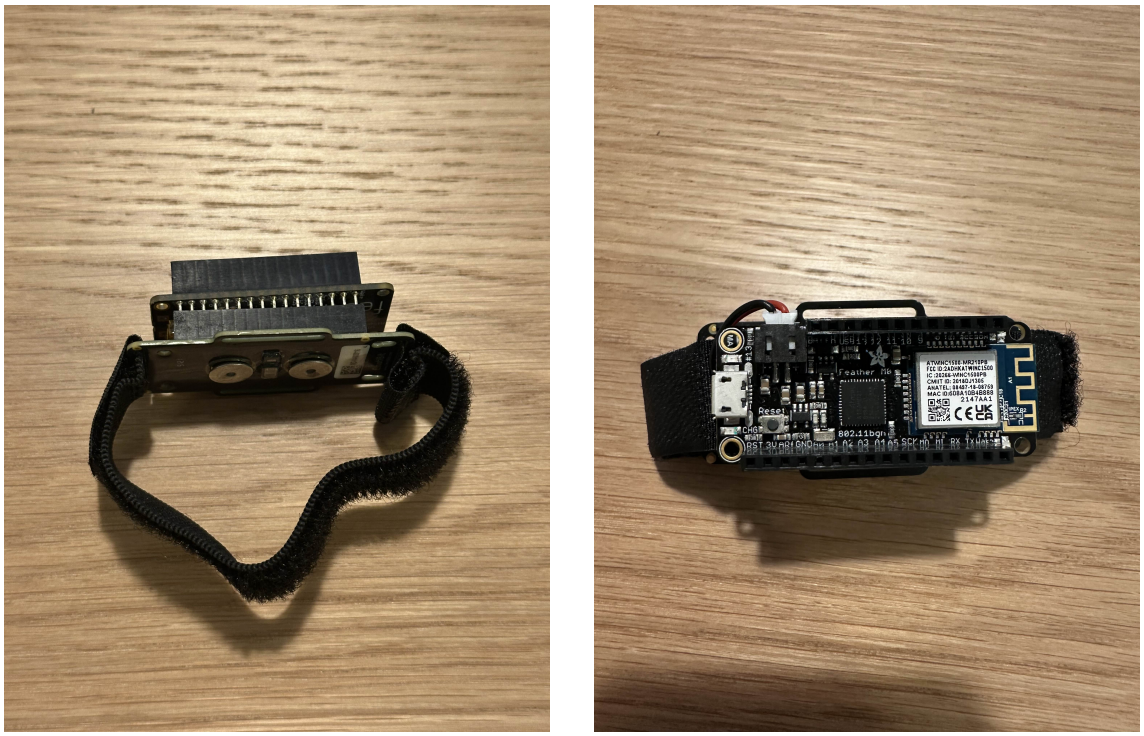
Several Python libraries were used throughout the project for data handling, signal processing, model development, and evaluation. To read the dataset files, two separate libraries were utilized. *SciPy* was used to load MATLAB files containing patient data. *SciPy* is an open-source Python library for scientific computing that provides mathematical functions and algorithms built on top of *NumPy*. Waveform database, which is an open-source library developed by PhysioNet, was used to read the annotated recordings.

The machine learning models were implemented using the Python library *scikit-learn* (*sklearn*). Sklearn is an open-source machine learning library that provides implementations of algorithms such as random forests, support vector machines, and k-nearest neighbors, together with tools for model evaluation, cross-validation, and preprocessing. The complete set of libraries and their use case for the project is summarized in Table A.1.

3.1.3 EmotiBit

Analyzing peripheral physiological signals provides insight into both mental and physiological states [41]. However, current commercial devices for peripheral physiological sensing typically present highly processed data, limiting their suitability for research applications, whereas research-grade

alternatives provide higher-quality data at a greater financial cost. To address these limitations, an open-source physiological sensing platform, EmotiBit [42], was developed by Connected Future Labs [43], providing users with full access to the recorded data [41]. EmotiBit is capable of wirelessly streaming and locally recording data from a multitude of sensors, making it multi-modal. The device is easy to wear and can record photoplethysmogram (PPG) signals, from which heart rate, heart rate variability, respiration, oxygen saturation, and additional physiological metrics can be derived. In particular, SpO_2 can be calculated using an algorithm described in subsection 3.1.4. The EmotiBit and its recorded signals are shown in Figure 3.1 and Figure 3.2 respectively.



(a) Side view **(b)** Front view

Figure 3.1: EmotiBit bracelet used in this project.

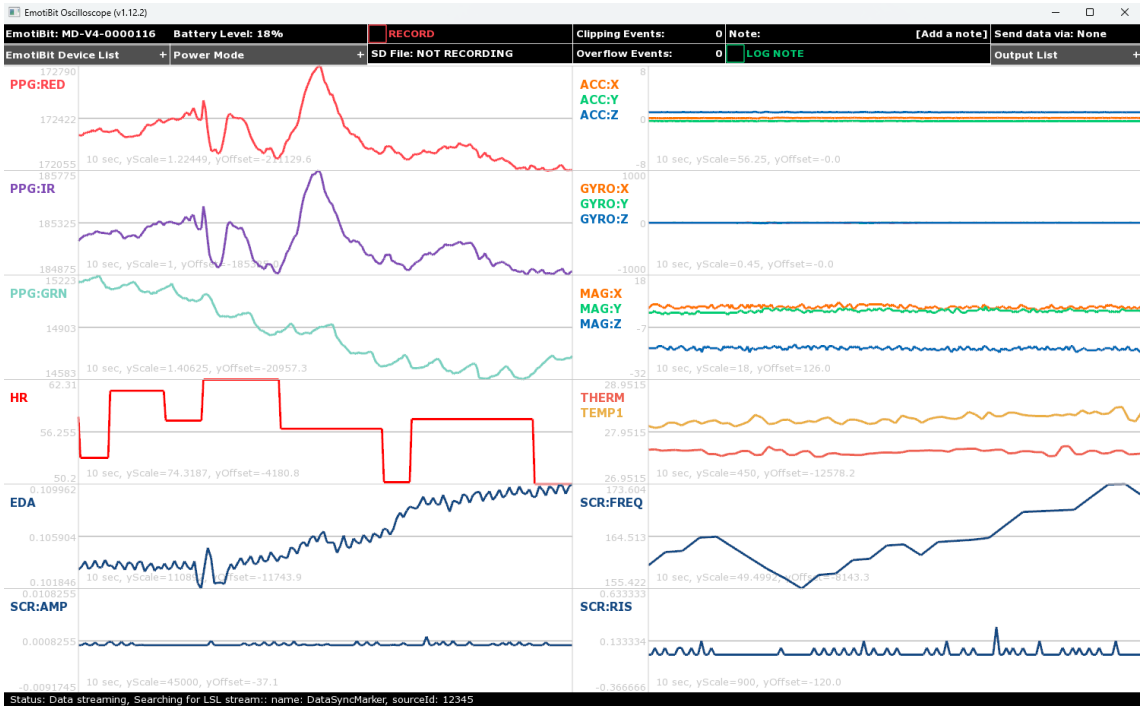


Figure 3.2: Physiological signals measured with an EmotiBit displayed in the included software. The interface shows multiple real-time physiological signals, including photoplethysmography (PPG), heart rate (HR), electrodermal activity (EDA) and temperature.

3.1.4 EmotiBit SpO_2 algorithm

Since the EmotiBit device did not directly measure blood oxygen saturation, it had to be derived from the photoplethysmography (PPG) signals captured by the EmotiBit sensor. These signals captured changes in blood volume using red and infrared (IR) light, which were absorbed differently by oxygenated and deoxygenated hemoglobin.

The PPG signals were then converted to SpO_2 using an algorithm provided by EmotiBit developers [44]. The EmotiBit SpO_2 algorithm estimated blood oxygen saturation by utilizing the different absorption characteristics of oxygenated and deoxygenated hemoglobin. The signals were first preprocessed to reduce noise. The AC component of each PPG signal was estimated using the root mean square of the filtered signal, while the DC components were computed as the mean of the original signal. A ratio-of-ratios was then calculated as [44]:

$$R = \frac{\left(\frac{AC_{\text{red}}}{DC_{\text{red}}}\right)}{\left(\frac{AC_{\text{IR}}}{DC_{\text{IR}}}\right)}.$$

This ratio was mapped to SpO_2 using a quadratic calibration function, and then the result was clipped to a range of 0–100%. See Figure 3.3 for a recording of SpO_2 using PPG signals during sleep. As shown in the figure, the signal contains artifacts, such as abrupt dips, that require preprocessing. A detailed description of the preprocessing steps is provided in section 3.4.

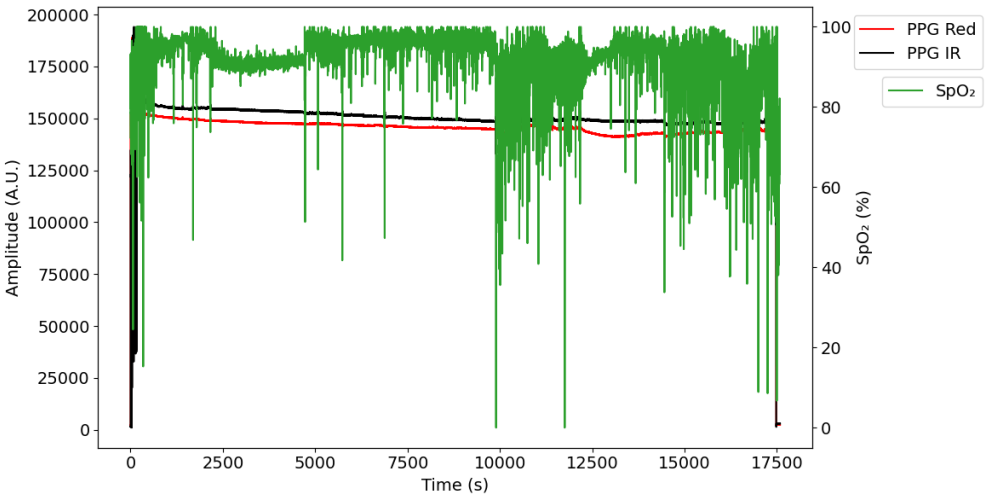


Figure 3.3: Estimated oxygen saturation (SpO_2) calculated from red and infrared photoplethysmography (PPG) signals recorded with the EmotiBit sensor. The estimated SpO_2 signal contains visible artifacts, including abrupt drops to values near 0%, highlighting the need for preprocessing.

3.2 Pipeline overview

This section provides an overview of the processing pipeline for both the classical ML models and the DL model used in this project. The pipeline is composed of the following sequential stages: Data collection, preprocessing, feature extraction, machine learning models, and evaluation. Although the DL model automatically performs feature extraction, this stage is still represented in the flowchart for structural consistency. A visual representation is shown in Figure 3.4.

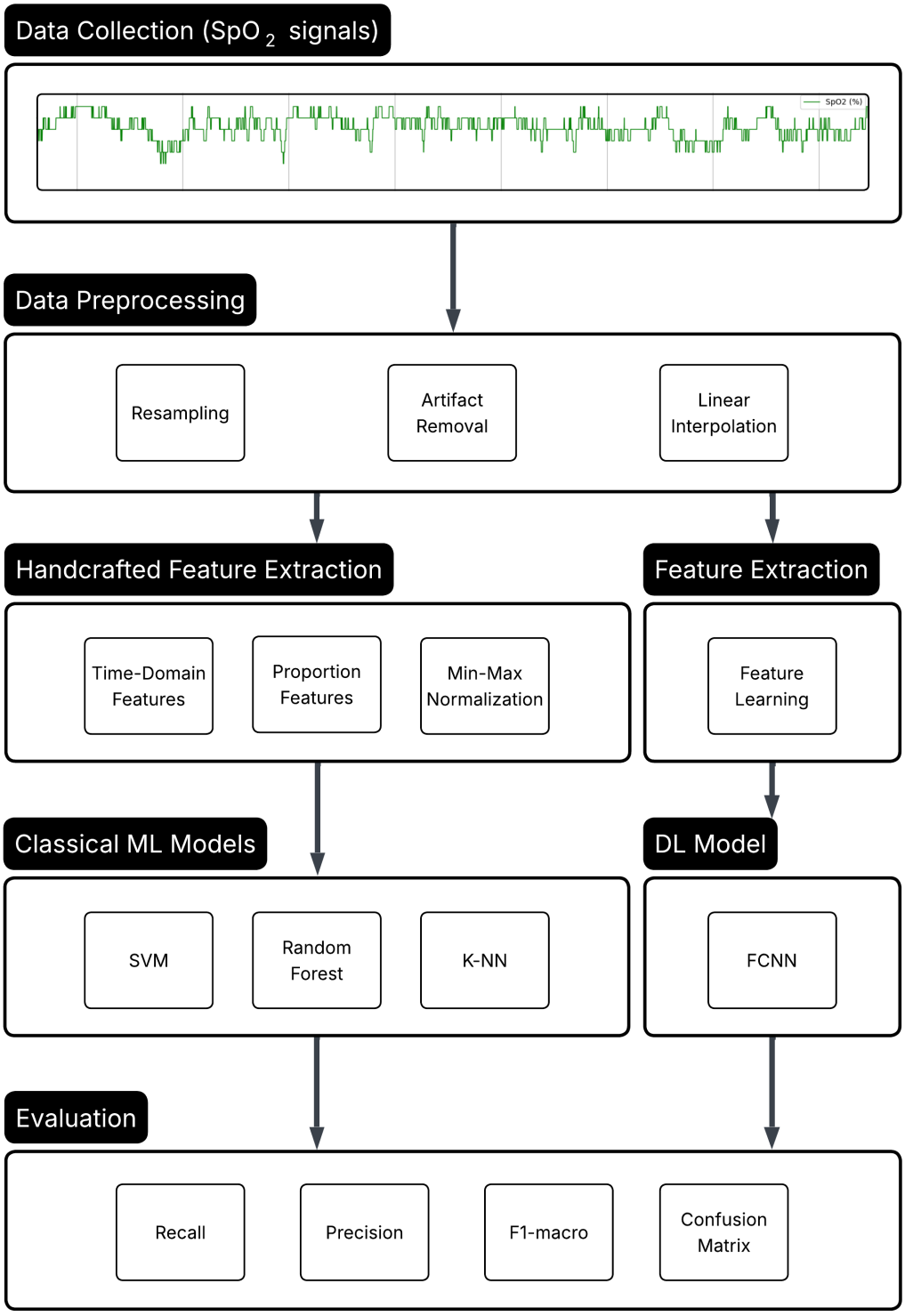


Figure 3.4: An overview of the processing pipeline for both classical ML and DL models used for the detection of sleep apnea using SpO_2. The process includes data collection, preprocessing, feature extraction, model development, and evaluation. Although the DL model performs feature extraction automatically, this stage is still represented in the flowchart for structural consistency.

3.3 Data handling

As discussed in section 2.2 Dataset, the dataset used contained 994 complete PSG recordings. These entries were randomly split into three sets, and remained the same for the training of every ML model. The distribution between the three sets was 80% training, 10% test and 10% validation sets. For the classical ML, the recordings set aside for the validation were merged into the training set. This is because classical machine learning models are not trained in an epoch-based manner and therefore do not inherently require a separate validation set during training.

Initially it was assumed that oxygen saturation was measured using SaO_2, due to this being written in the challenge description. However, during the project an inquiry was sent to the responsible parties and it was concluded that SpO_2 was the actual measurement used. As discussed in section 2.1, the difference between the two is minimal but important nonetheless.

The patient data was stored in MATLAB files as discrete values. A header file containing meta data was present with two values for converting these values to SpO_2 percentage, a baseline and a gain. The conversion was performed using Equation 3.1. The step length between consecutive discrete SpO_2 values was approximately one percentage unit.

$$SpO_2 = \frac{RAW\_VALUE - BASELINE}{GAIN} \tag{3.1}$$

Reduction in file size was accomplished by extracting only the SpO_2 signals from the data files. As no other physiological signals were included in the analysis, the remaining data could safely be discarded. This step reduced the file size by a factor of five. The patient data and annotations were combined into a single Parquet file, a column-oriented binary file format designed for efficient data storage and retrieval in big data analytics.

The selection of the dataset was made primarily based on available physiological signals, with consideration also made for the size of the dataset. Other datasets that included relevant signals were either based on specific parts of the population, or were deemed too narrow in their sample size. Most studies on machine learning for detection of sleep apnea use datasets comprising of 30–1000 samples, with each sample representing a sleep recording from a single subject [10].

3.4 Preprocessing

All physiological signals, with the exception of SpO_2, were originally sampled at a frequency of 200 Hz. However, the SpO_2 signal had been upsampled prior to publication, resulting in duplicated values intended to match the higher sampling frequencies of the other signals. The original sampling rate of the SpO_2 signal was not explicitly provided by the publisher. However, by analyzing the timestamps at which recorded values changed, it was inferred to be 1 Hz. Consequently, the SpO_2 signal was resampled to 1 Hz. This resampling was performed to facilitate easier feature extraction and to reduce storage requirements, effectively reducing the file size by a factor of almost 200.

The raw SpO_2 signal contained several types of artifacts that required preprocessing. In particular, certain recordings included physiologically impossible values, such as oxygen saturation levels below 1%, see Figure 3.5. These anomalies are likely attributable to sensor malfunctions or temporary disconnections of the pulse oximeter. To address such artifacts, two techniques were utilized.

The first of these was the removal of datapoints where the value fell outside of a given range. Other studies focused on SpO_2 have used different ranges, and in 2025 a comparative study of current practices was conducted [45]. The study found that calibration of devices measuring SpO_2 typically is done within the range of 70% to 100%, and that values below this might be inaccurate. Due to this, some studies using SpO_2 opted to use thresholds of 65% or 70%. The comparative study also found that many other studies used a lower threshold of 50% to take equipment errors

into account. Conclusively, the study found that many different techniques are utilized, with some even opting to leave the signal completely unaltered. Analysis of the dataset used in this thesis showed that it contained no reliable recordings with SpO_2 values below 70%. Consequently, the analysis was restricted to the range of 70% to 100%. This interval is also used by other studies not covered by the comparative study [46]. All values outside this interval were therefore replaced with undefined (NaN) values.

The second technique was to calculate the difference between two consecutive measurements and to remove values where the change exceeded 4%. This was done to remove unrealistic and abrupt fluctuations over short time intervals. The selected threshold of 4% is consistent with commonly used preprocessing approaches for SpO_2 signals [47].

Following the artifact removal, the resulting signal contained missing values that needed to be addressed before models could be trained. These gaps were subsequently filled using linear interpolation, where missing values were estimated by constructing a straight line between the nearest valid observations. A visual representation of an unprocessed SpO_2 signal from the training set is shown in Figure 3.5, and the corresponding preprocessed signal is shown in Figure 3.6. A preprocessed SpO_2 signal from the training set with annotations for apnea and hypopnea is shown in Figure 3.7.

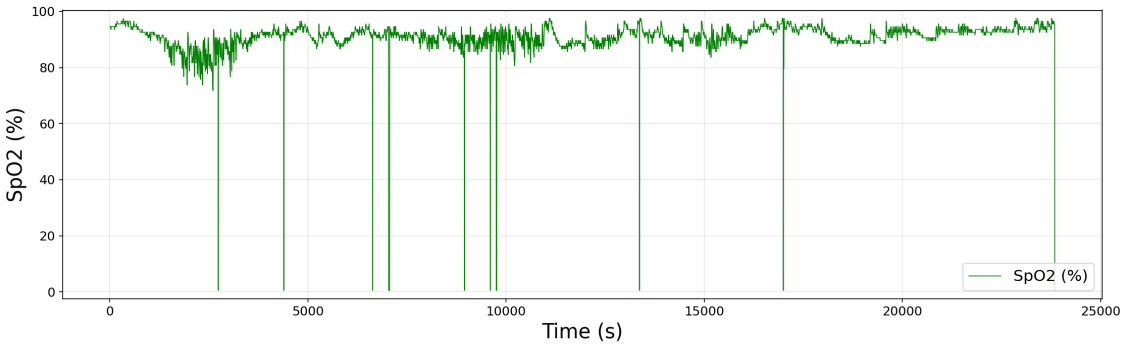


Figure 3.5: Raw (Unprocessed) SpO_2 signal.

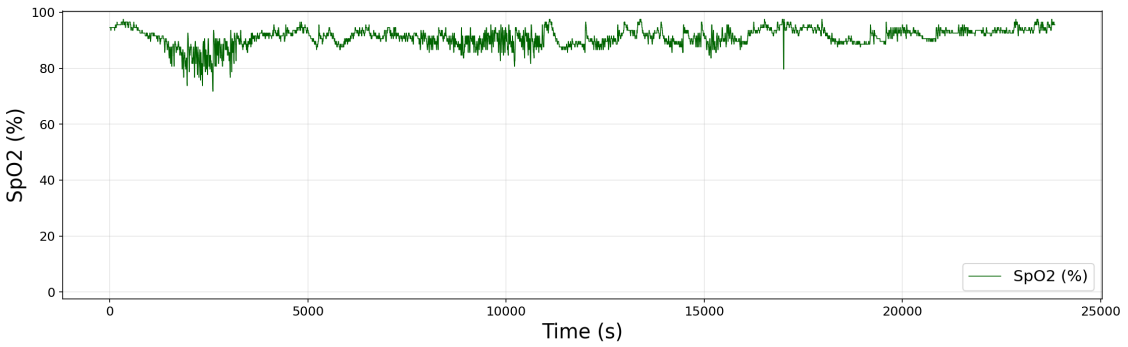


Figure 3.6: Preprocessed SpO_2 signal.

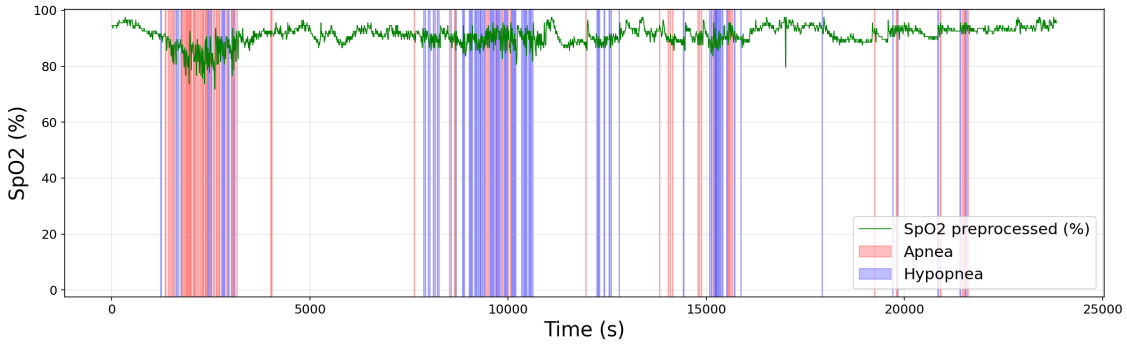


Figure 3.7: Preprocessed SpO_2 signal with apnea (red) and hypopnea (blue) annotations.

3.5 Feature extraction

As previously discussed in section 2.4, the choice of features is critical for effectiveness of classical machine learning models. Since most entries in the dataset contain full nights of sleep, features had to be extracted from smaller windows to be meaningful. Two sliding windows were applied to the data: a short window spanning 10 seconds and an extended window spanning 30 seconds. This allowed features to be constructed from multiple consecutive data points rather than individual observations. Both windows advanced in 5-second increments, producing a 50% overlap between successive short windows and effectively resampling the time axis from a 1-second to a 5-second resolution. The apnea event labels in the training data were similarly resampled to match the new window-based dimensions. A short window was assigned an apnea label if the majority of its data points were annotated as apnea events.

The feature extraction approach in this project was inspired by a study published as part of the PhysioNet Computing in Cardiology Challenge 2018 [48], in which nine time-domain features were derived from patient data; five calculated from the short window and four from the extended window. The features extracted from the short window are the mean, standard deviation, the minimum value, the skewness and kurtosis. These were chosen to capture different statistical properties of the signal, including its central tendency, spread, and distributional shape. From the extended window, four features were derived based on oxygen saturation thresholds adopted from the study mentioned earlier [48]. Each feature represents the proportion of data points within the window that fall into a specific saturation range: above 96%, between 90-96%, between 80-90% and below 80%. Using proportions over the longer window helps capture how much time the patient spends at low oxygen levels, which is more informative than looking at a single point in time. The complete set of extracted features is summarized in Table 3.1.

After feature extraction, normalization using min-max, as shown in Equation 2.1, was applied to bring all features onto a comparable scale. The mean and minimum values were normalized within a fixed range of 70—100%, where 0 corresponds to 70% and 1 corresponds to 100%. This fixed range was chosen to preserve clinically relevant characteristics of the signal rather than normalizing relative to each patient's individual range. All other features were normalized between each patient's minimum and maximum observed values.

Table 3.1: Extracted features from the SpO_2 signal used for classical machine learning.

Feature	Window	Description
Mean	10 s	Average SpO_2 value within the window.
Standard deviation		Measures variation in SpO_2 values within the window.
Minimum		Lowest SpO_2 value observed in the window.
Skewness		Measures asymmetry of the signal distribution around the mean.
Kurtosis		Measures tail heaviness and presence of extreme values.
SpO_2 >96%	30 s	Proportion of observations above 96% oxygen saturation.
SpO_2 90–96%		Proportion of observations between 90% and 96% oxygen saturation.
SpO_2 80–90%		Proportion of observations between 80% and 90% oxygen saturation.
SpO_2 <80%		Proportion of observations below 80% oxygen saturation.

3.6 Model development

All models in this project were implemented in python as subclasses of a common abstract base class. This class defined the necessary abstract methods `train`, `predict`, `save`, and `load`, while `evaluate` was provided as a concrete method since its implementation did not differ between model types. Each classifier was then implemented as a separate subclass, which made it easier to train, evaluate, and compare different models without having to adjust the surrounding code structure for every model type.

The models were trained using the training dataset and later evaluated using separate test data. For the fully connected neural network, to monitor performance, the validation set was used after each training cycle. The classical ML models were trained on the training and validation data combined. Hyperparameters that could affect model performance were kept configurable in each model class, allowing them to be adjusted and compared during evaluation. This was essential because different hyperparameters could influence model performance and generalization ability.

The dataset was considered imbalanced, with a strong predominance of non-apnea over apnea events, and this was accounted for during model development. For models that supported it, class imbalance was addressed by applying either balanced class weights or sample weights during training. This helped mitigate bias toward the majority class and improved the models' ability to learn patterns from the minority class.

After training, each model generated predictions on unseen data using their `predict` function. The trained models could also be saved and loaded again later, which made it possible to reuse the trained classifiers without repeating the training process.

3.6.1 k-nearest neighbors

For the k-NN classifier, the choice of neighbor-search algorithm in the feature space was determined by the properties of the dataset. Brute force is preferred when the training dataset is small, when the chosen distance metric is not compatible with tree-based indexing structures, or when the number of features exceeds 15 [49]. However, when none of these conditions apply, a more optimized method is used. The usual algorithm used is KD Tree (k-dimensional tree), which organizes the training data into a tree structure, reducing lookup complexity from linear to logarithmic. Another

option is Ball Tree, which is employed when the distance metric is not supported by the KD Tree algorithm but is compatible with Ball Tree indexing. Ball Tree structures the training data into a binary search tree, encapsulating subsets of the data points into hyperspheres, allowing large portions of the data to be pruned early during search [49]. Given nine features and a relatively large training dataset, scikit-learn's `KNeighborsClassifier` automatically selected the KD Tree algorithm, as the number of features was below 15 and the Euclidean metric was compatible with KD Tree indexing.

Feature normalization was particularly important for k-NN when using Euclidean metric, as features with larger magnitudes would otherwise dominate the distance calculations, leading to uneven feature contributions [49].

The primary hyperparameter to configure was `k`, which controls the number of neighbors considered when determining the class label during inference (i.e., when the trained model generates predictions for unseen data) [49]. A larger `k` incorporates more neighbors, reducing the influence of individual data points and outliers, resulting in a smoother decision boundary (the boundary in feature space separating the predicted classes) at the cost of increased inference time. A smaller `k` produces a more locally sensitive model, where only the nearest data points influence the prediction, making it more prone to outliers and noise. Rather than standard majority voting, a threshold-based rule is employed to account for the class imbalance. Specifically, a sample is classified as positive if the proportion of positive neighbors exceeds the base rate of positive samples in the training set.

3.6.2 Support vector machines

The non-linear implementation of SVMs from sklearn uses a quadratic programming solver with complexity that scales between $O(n^2)$ and $O(n^3)$ depending on the dataset [50], consequently making non-linear SVMs impractical for larger datasets. The training data used comprised approximately 4 000 000 windows in total, effectively ruling out the use of a non-linear SVM. As such, the linear implementation of SVM from sklearn, `LinearSVC`, was used [51].

The main configurable hyperparameter is called C, which controls the strength of regularization. A low value of C gives stronger regularization, while a high value of C gives weaker regularization and penalizes training errors more heavily. Therefore, C affects the balance between underfitting and overfitting.

To account for the imbalanced dataset, balanced sample weights were computed from the training labels using `compute_sample_weight("balanced", y_tr)`, where `y_tr` is the class labels for the training set. These weights were passed to the fitting function, giving the minority class a larger influence during training.

3.6.3 Random forests

For the random forest model, `RandomForestClassifier` from `sklearn.ensemble` was utilized [52]. The number of decision trees trained on the data is controlled by the parameter `n_estimators`. A larger value increases model stability by reducing the effect of individual tree variation, at the cost of increased training time. The `max_depth` parameter controls the maximum depth of each tree. A deeper tree can capture more complex patterns but risks overfitting, while a shallower tree may underfit the data.

To account for class imbalance, `compute_sample_weight("balanced", y_tr)`, where `y_tr` is the class labels for the training set, was used to assign higher weights to the minority class during training, penalizing misclassifications of positive samples more heavily.

3.6.4 Fully connected neural network

As previously discussed in section 2.5, DL models such as FCNN do not require handcrafted features. Therefore, the feature extraction step was bypassed. However, in contrast to the classical ML models developed, a more specific and manual implementation was required, as there are no predefined templates such as the ones for k-NN, SVM and random forest. For the development of the FCNN, the *PyTorch* library's `torch.nn` module was utilized [53].

A neural network in PyTorch is defined as a Python class that inherits from the base class `nn.Module` [53]. The architecture is specified in the constructor for the class by composing different *layers* from the library. The layers are then wrapped in a sequential container which applies each layer in order during a forward pass, invoked through the `forward` method.

The FCNN implemented in this project took a raw SpO_2 window of fixed length as a flat input vector and passed it through a configurable number of fully connected hidden layers (`nn.Linear`). Each layer was followed by batch normalization (`nn.BatchNorm1d`), which applied Z-score normalization on the output, a ReLU activation (`nn.ReLU`), and optional dropout for regularization (`nn.Dropout`). The final layer produced a single scalar output, representing the probability that the examined window contained an apnea event.

Training was handled outside the architecture class. Input windows were first normalized using the mean and standard deviation computed from the training set. They were then wrapped in PyTorch `Tensor` objects and fed to a `DataLoader`, which divided the data into mini-batches of 1024 windows, and shuffled their order [54]. Class imbalance was addressed by computing a *positive class weight* proportional to the class frequency ratio, which was then passed to the `BCEWithLogitsLoss` loss function. The network parameters were updated each batch via backpropagation using the Adam optimizer. After each training cycle, performance was evaluated on the validation set to monitor generalization.

3.7 Evaluation

To compare model performance, each model was evaluated using the classification metrics defined in section 2.7: recall, precision, F1-score, and confusion matrices. Since feature extraction transformed the time axis into 10-second windows with a 5-second overlap, all metrics reflected how accurately each model classified individual windows as either containing an apnea event or no event.

In practice, these models were expected to produce information about the analyzed patient, indicating whether the patient had sleep apnea or not. This was done by inspecting the output binary vector from inference, where ones represented apnea events in a given 10-second window and zeros represented normal breathing. The default way to determine sleep apnea severity is through AHI, which is explained in chapter 1. Since the recording duration is known, the number of apnea events could be derived from the binary vector. The window vector was converted back to a time vector, where each element represented one second, and a stream of consecutive ones constituted a single event. However, these streams could not be arbitrarily long, as no patient could realistically stop breathing for several minutes and survive. Since the models were not constrained by this, they sometimes labeled intervals of more than 10 minutes as a single apnea event. To address this, labeled intervals exceeding 30 seconds were broken up with a minimum gap of 10 seconds to produce more physiologically realistic predictions.

To ensure fair comparison between the different types of models, a hyperparameter grid search, or sweep, was conducted for each model. A hyperparameter sweep is a method for finding parameters when training a ML model in specific cases, performed by training one model for each combination of parameter options and then comparing the results to find the best configurations for the models. For the random forest and SVM models, the impact of using normalized features were also examined as a part of the grid search by training models on both normalized and non-normalized features. These sweeps were performed on the Minerva computing cluster due to their relatively

high resource requirements and the ability of Minerva to run several training sessions in parallel.

As a final point of evaluation, the best-performing model of each type was examined for signs of underfitting or overfitting. Each model was evaluated independently on the training and test sets, and the absolute difference $|d|$ between corresponding metrics was computed. A threshold of $|d| < 0.10$ was deemed sufficient to conclude the absence of significant over- or underfitting.

3.8 Case study

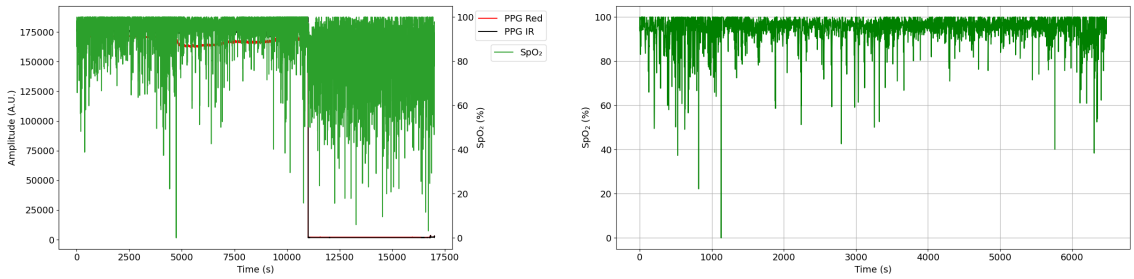
A case study was conducted to evaluate the feasibility of applying the best-performing ML model developed in this study to new data collected in a home environment. Data were collected from two self-reported healthy participants.

3.8.1 Data collection

Physiological data were collected using the EmotiBit wearable biosensor to record SpO_2 via PPG during sleep. The device was secured with a smaller strap and worn on the participant's right index finger. Data were recorded during sleep in a natural, unsupervised home setting. The participants wore the device continuously throughout the night. SpO_2 signals were recorded and stored for offline analysis. Within a couple of minutes after the recording started, the participants went to sleep, and after awakening the recording was stopped.

3.8.2 Data preprocessing

The recorded SpO_2 signal was preprocessed and features were extracted using the same procedure as for the dataset (see section 3.4). However, additional preprocessing was required to account for the characteristics of the home-recorded data. The raw recordings obtained from the EmotiBit device contained segments outside the actual sleep period, including time before sleep and after awakening. These segments exhibited increased motion artifacts and non-sleep-related physiological patterns, which could negatively affect the performance of the machine learning models. To address this, a manual trimming procedure was applied to isolate the portion of the recording corresponding to sleep. First, the initial one hour of each recording was removed to exclude the pre-sleep period. In addition, the last 30 minutes of the recording were removed in order to decrease the likelihood of including post-awakening data. For recordings where the sensor appeared to become loose or was removed before the end of the recording, the endpoint was instead determined through visual inspection of the SpO_2 time series and the corresponding PPG signals it was derived from. Figure 3.8 illustrates an example of this procedure, where the sensor became loose during sleep. Once unreliable segments had been excluded, the trimmed SpO_2 signals were subsequently provided as input to the previously trained sleep apnea detection model.



(a) Full EmotiBit SpO_2 recording. **(b)** Trimmed EmotiBit SpO_2 recording.

Figure 3.8: Comparison between a full and trimmed SpO_2 recording. (a) Full SpO_2 recording, showing a clear dip in the PPG signals when the EmotiBit sensor became loose during sleep. (b) Trimmed SpO_2 signal after exclusion of the first 60 minutes and the segment during which the EmotiBit was no longer attached.

3.8.3 Interpretation of model output

The last part of the case study was to interpret the model output on the preprocessed EmotiBit recordings. As no clinical ground truth in the form of expert annotations was available for the participants of the case study, the model outputs were not used for quantitative performance evaluation. Instead, the AHI values for the different recordings were calculated from the model's predictions and examined, since the healthy participants were not expected to have an AHI above four apnea events per hour, see Table 1.1. To support the interpretation of the case study results, the unique SpO_2 values were examined for both the training dataset and the EmotiBit recordings. For each group of recordings, the mean number of unique SpO_2 values was calculated, together with the minimum and maximum number of unique values observed across recordings. This comparison was used to characterize the variability and resolution of the SpO_2 signal in the two data sources to explain any unexpected model behavior, such as too large AHI values.

4
Results and discussion

This chapter presents the results and compares the performance of the different models. It also discusses key observations regarding dataset characteristics, feature representation, and the strengths and weaknesses of each approach in the context of sleep apnea detection.

4.1 Hyperparameter tuning results

Following are the results of the hyperparameter grid searches performed for each model type, presented through tables and some observations.

4.1.1 k-nearest neighbors

The k-NN hyperparameter grid search, summarized in Table 4.1, evaluated k values ranging from 3 to 2001. Performance was irregular at low values of k, as increasing k did not produce a clear pattern of improvement or decline, which could be attributed to the threshold rule described in subsection 3.6.1. The minimum proportion of positive neighbors required to trigger a positive prediction varied unevenly with k, making the decision boundary sensitive to the exact composition of small neighborhoods. Beyond $k = 51$, F1-macro increased steadily, peaking at $k = 501$ with a score of 0.6113. Further increases in k yielded marginal or negative returns while substantially increasing inference time, as each prediction required computing distances to a larger fraction of the training set. The configuration $k = 501$ was therefore selected as the representative k-NN model.

Table 4.1: k-NN hyperparameter sweep results across different values of k, reporting recall, precision and F1-macro.

k	Recall	Precision	F1-macro
3	0.6857	0.5940	0.6070
5	0.7025	0.5857	0.5839
7	0.7063	0.5808	0.5655
11	0.7010	0.5755	0.5490
15	0.7242	0.5901	0.5827
21	0.7201	0.5841	0.5649
51	0.7406	0.5950	0.5860
101	0.7512	0.6007	0.5961
201	0.7581	0.6051	0.6044
501	**0.7640**	**0.6089**	**0.6113**
901	0.7632	0.6084	0.6105
2001	0.7626	0.6078	0.6091

4.1.2 Support vector machine

The SVM hyperparameter sweep, summarized in Table 4.2, revealed two patterns. First, the model was notably insensitive to the regularization parameter C across all four tested values. The performance varied by less than 0.001 between models with the same option of using normalized features or not, suggesting that the linear kernel had converged to the same decision boundary regardless of the regularization strength. Second, using normalized features had a consistent negative effect. Models not using normalized features outperformed those that did across all metrics, with F1-macro improving from 0.5855 to 0.6174, a relative increase of 5.4%. As such, the SVM model did not benefit from the use of normalized features in this case. Given the negligible sensitivity to C, the configuration $C = 0.01$ without normalization was selected as the representative SVM, achieving the highest F1-macro of 0.6174.

Table 4.2: SVM hyperparameter sweep results across normalized and non-normalized feature settings, reporting recall, precision and F1-macro for different values of C.

Normalized	C	Recall	Precision	F1-macro
Yes	0.01	0.7097	0.5878	0.5855
	0.1	0.7096	0.5877	0.5853
	1.0	0.7096	0.5877	0.5853
	10.0	0.7096	0.5877	0.5853
No	0.01	**0.7289**	**0.6054**	**0.6174**
	0.1	0.7288	0.6054	0.6173
	1.0	0.7288	0.6054	0.6173
	10.0	0.7288	0.6054	0.6173

4.1.3 Random forest

The random forest hyperparameter sweep, summarized in Table 4.3, revealed three patterns. First, the number of estimators n had a negligible effect on performance. F1-macro improved by at most 0.0005 across $n \in \{50, 100, 200\}$ for any fixed depth and normalization setting, suggesting that the ensemble had effectively converged already at $n = 50$. Second, max depth had a clear and consistent positive effect: increasing depth from 10 to 15 improved F1-macro, and further increasing to 20 yielded additional gains, indicating that deeper trees captured more discriminative structure in the data. Third, the effect of normalization was inconsistent and interacted with max depth. At lower depths (10 and 15), configurations without normalization outperformed their normalized counterparts, mirroring the pattern observed for SVM in Table 4.2. However, at depth 20 this relationship reversed. Normalized configurations consistently achieved higher F1-macro than those without normalization. One possible explanation is that at greater depths, trees are more prone to overfitting to the scale of individual features, and normalization acts as a mild regularizer in this regime. Given these observations, the configuration $n = 200$, max depth 20, with normalization was selected as the representative Random forest, achieving the highest F1-macro of 0.6557.

Table 4.3: Random forest hyperparameter sweep results across normalized and non-normalized feature settings. n denotes the number of estimators.

n	Normalized	Max depth	Recall	Precision	F1-macro
50	Yes	10	0.7774	0.6166	0.6239
	Yes	15	0.7793	0.6241	0.6393
	Yes	20	0.7672	0.6320	0.6552
	No	10	0.7814	0.6244	0.6393
	No	15	0.7810	0.6276	0.6455
	No	20	0.7739	0.6283	0.6483
100	Yes	10	0.7775	0.6170	0.6247
	Yes	15	0.7798	0.6241	0.6391
	Yes	20	0.7684	0.6321	0.6553
	No	10	0.7813	0.6243	0.6390
	No	15	0.7810	0.6274	0.6451
	No	20	0.7741	0.6287	0.6489
200	Yes	10	0.7778	0.6172	0.6252
	Yes	15	0.7797	0.6240	0.6390
	Yes	20	0.7686	**0.6324**	**0.6557**
	No	10	**0.7815**	0.6247	0.6398
	No	15	0.7811	0.6275	0.6454
	No	20	0.7744	0.6287	0.6489

4.1.4 Fully connected neural network

The FCNN grid search, summarized in Table 4.4, revealed four patterns. First, a dropout rate of 0.2 consistently outperformed 0.3 across nearly all configurations, suggesting that the lower dropout rate retained sufficient regularization without discarding too much information during training. Second, a max positive weight of 10 yielded higher F1-macro than 15 in almost all cases. A higher positive weight placed greater emphasis on the minority class during training, and the results suggest that $w^+ = 15$ overcorrects for the class imbalance, leading to reduced precision without a sufficient gain in recall. Third, window size had the most pronounced effect of all hyperparameters. Increasing the window from 30 to 60 improved F1-macro substantially, and a further increase to 90 yielded additional gains, indicating that longer temporal context is beneficial for the classification task. Lastly, hidden layer size had a negligible effect on performance. Configurations with 256×128 neurons performed comparably to those with 128×64 across all window sizes and regularization settings. Given these observations, the configuration window size 90, hidden size 128×64, dropout 0.2, and $w^+ = 10$ was selected as the representative FCNN, achieving the highest F1-macro of 0.6939.

Table 4.4: FCNN hyperparameter sweep results. Hidden sizes denote neurons per layer and Max w^+ denote max positive weight.

Window	Hidden sizes	Dropout	Max w^+	Recall	Precision	F1-macro
30	128×64	0.2	10	0.8255	0.6427	0.6641
	128×64	0.2	15	0.8242	0.6460	0.6703
	128×64	0.3	10	0.8250	0.6450	0.6684
	128×64	0.3	15	0.8244	0.6402	0.6695
	256×128	0.2	10	0.8242	0.6476	0.6731
	256×128	0.2	15	0.8246	0.6405	0.6600
	256×128	0.3	10	0.8249	0.6475	0.6727
	256×128	0.3	15	0.8217	0.6392	0.6583
60	128×64	0.2	10	0.8658	0.6466	0.6841
	128×64	0.2	15	0.8657	0.6277	0.6515
	128×64	0.3	10	0.8672	0.6425	0.6776
	128×64	0.3	15	0.8664	0.6297	0.6554
	256×128	0.2	10	0.8661	0.6450	0.6816
	256×128	0.2	15	0.8677	0.6337	0.6626
	256×128	0.3	10	0.8679	0.6433	0.6789
	256×128	0.3	15	0.8663	0.6325	0.6606
90	128×64	0.2	10	0.8819	**0.6488**	**0.6939**
	128×64	0.2	15	**0.8859**	0.6334	0.6702
	128×64	0.3	10	0.8853	0.6407	0.6820
	128×64	0.3	15	0.8844	0.6347	0.6725
	256×128	0.2	10	0.8843	0.6481	0.6929
	256×128	0.2	15	0.8839	0.6349	0.6730
	256×128	0.3	10	0.8855	0.6423	0.6843
	256×128	0.3	15	0.8856	0.6319	0.6679

4.2 Model performance comparison

The evaluation results demonstrate a clear performance hierarchy among the investigated models. The FCNN consistently achieved the strongest overall results, followed by the random forest models, while SVMs and k-NN exhibited comparatively lower performance.

The scatter plot in Figure 4.1 shows that the FCNN clearly outperforms the classical ML models, with random forest in second and then k-NN and SVM interconnected. The SVM models that were evaluated had highly similar values, and are therefore clustered in only 2 points. k-NN models showed a more linearly increasing performance dependent on the selected hyperparameters.

No model achieves a precision above 0.65, which suggests that a notable share of predicted apnea events are false positives. However, in the context of sleep apnea diagnosis, this trade-off may be acceptable, as high recall is generally preferable; missing a true apnea event is clinically more consequential than flagging a false one. The consistently higher recall across all models indicates that they are better tuned to detecting apnea events than avoiding false alarms, which aligns with the priorities of a medical screening tool.

Figure 4.2 further supports the ranking when performance is evaluated using F1-macro. The FCNN achieved the highest score, indicating the most balanced trade-off between recall and precision among the investigated models. The drop in performance for SVM and k-NN is also clearly visible here. An interesting observation is the fact that SVM achieved better F1-macro compared to k-NN, even though Figure 4.1 seems to indicate that k-NN performs better. This behavior stems from the k-NN models increase in performance, measured through recall, is not matched by the

same increase in precision, implying a larger imbalance between correct and incorrect predictions.

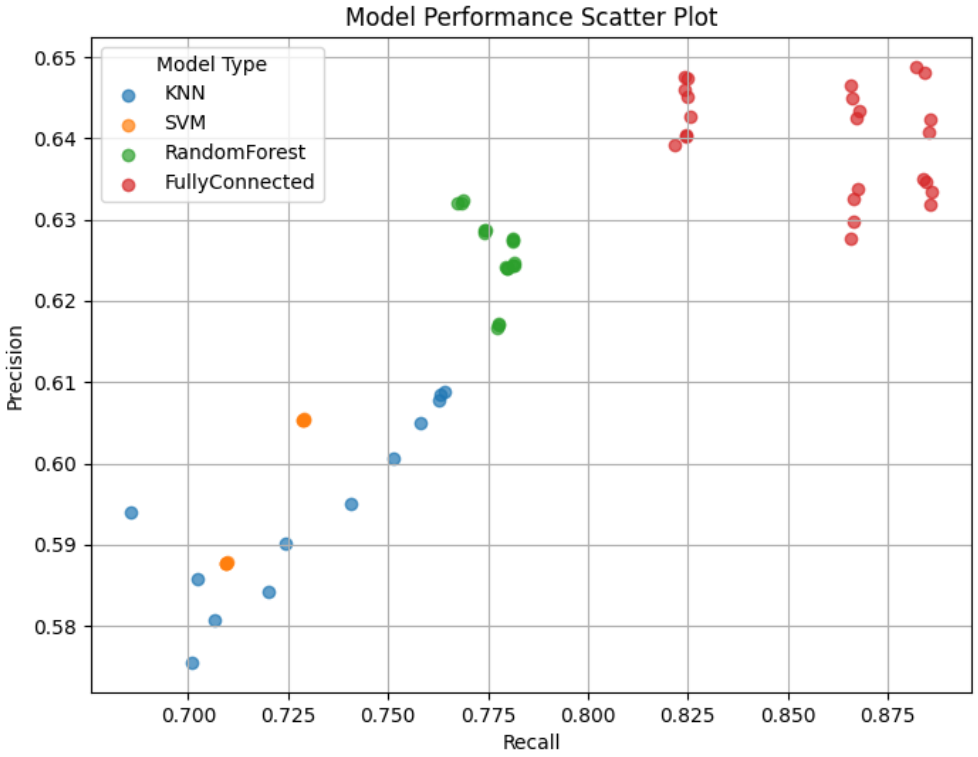


Figure 4.1: Scatterplot of precision vs recall for different implementations of the four model types (k-NN, SVM, random forest and fully connected neural network)

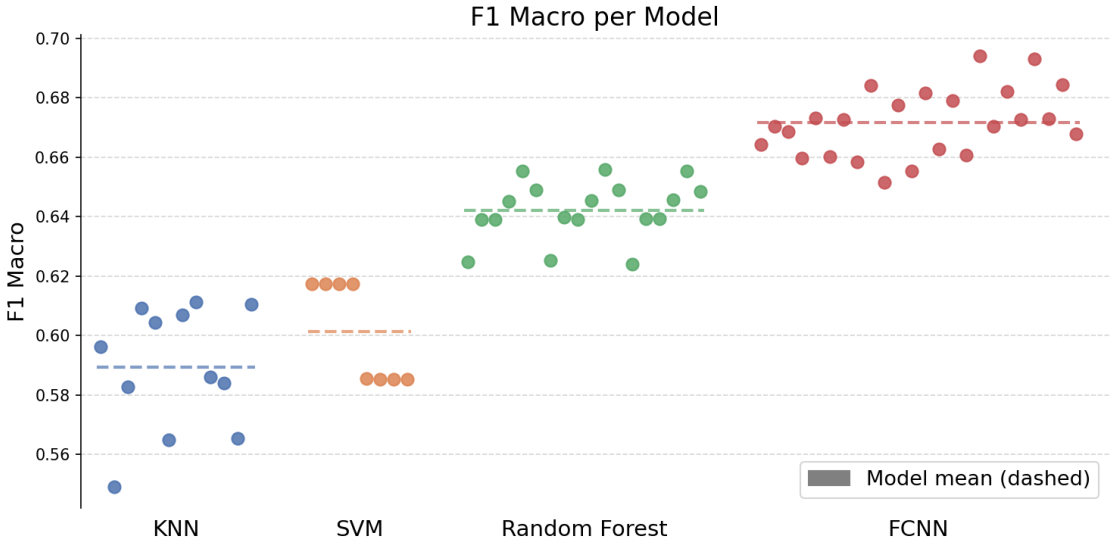


Figure 4.2: F1-macro scores for all model configurations, grouped by model type. Dashed lines indicate the mean score per model.

By utilizing confusion matrices, the individual strengths and weaknesses of each model can be more clearly observed. As seen in Figure 4.3, the FCNN performs best in all 4 categories, whereas the three classical models all differed slightly in which areas their strength and weakness lie. The random forest model, which achieved a higher F1-macro than the k-NN, is more prone to false negatives relative to the k-NN model. The SVM achieved the lowest rate for detecting true positives, which may deem it medically unreliable due to the risk of missing too many apnea cases. Furthermore, most models display higher accuracy on the negative class than the positive class, which is expected given the class imbalance in the dataset. This bias toward the majority class is most pronounced in SVM and random forest. Notably, k-NN achieves higher accuracy on the positive class than the negative, while FCNN achieves the most balanced performance across both classes.

(a) k-NN

(b) SVM

(c) Random forest

(d) Fully connected neural network

Figure 4.3: Confusion matrices, normalized over actual values (row), for the best-performing k-NN, SVM, random forest, and FCNN models, selected by F1-macro.

The generalization ability of the best performing models was analyzed by comparing their training and test performance using the train–test gap. The gap was calculated as the difference between the training score and test score for each metric. A large positive gap indicates overfitting, as it suggests that the model performs better on the training data than on unseen test data. In contrast, a small gap suggests that the model generalizes well.

Overall, the results in Table 4.5 show no strong indication of overfitting for any of the evaluated models, except for k-NN which shows a significant decrease in performance compared to the other three models. This shows that the SVM, random forest, and FCNN models generalize well to unseen data and did not display a strong indication of overfitting. The SVM and FCNN models

showed slightly negative gaps, meaning that their test scores were marginally higher than their training scores. The random forest model showed positive train–test gaps for all metrics, meaning that it performed better on the training data than on the test data. Although the model showed a slightly stronger tendency toward overfitting compared to SVM and FCNN, the train–test gaps were still small, suggesting the model did not suffer from severe overfitting.

In contrast, the k-NN model showed substantially larger train–test gaps across all metrics. The largest gap was observed for F1-macro, followed by recall and precision. This indicates that k-NN performed considerably better on the training data than on the unseen test data, especially considering that k-NN received the highest train scores for all metrics except recall. This large difference suggests that the model had overfitted the training data and was thus less able to generalize to new data compared to the other models.

Table 4.5: Training and test performance of the evaluated models. The gap is computed as training performance minus test performance.

Model	Metric	Train	Test	Train–Test Gap
k-NN	F1-macro	0.8828	0.6143	+0.2685
	Recall	0.8270	0.5825	+0.2445
	Precision	0.9696	0.7817	+0.1879
SVM	F1-macro	0.6044	0.6174	-0.0130
	Recall	0.7154	0.7289	-0.0134
	Precision	0.5950	0.6054	-0.0104
Random forest	F1-macro	0.6797	0.6557	+0.0240
	Recall	0.8143	0.7686	+0.0458
	Precision	0.6495	0.6324	+0.0171
FCNN	F1-macro	0.6774	0.6939	-0.0165
	Recall	0.8749	0.8819	-0.0071
	Precision	0.6341	0.6488	-0.0147

4.3 Impact of dataset characteristics

An important factor influencing the results is the class distribution of the dataset. As previously discussed in section 2.2, and visible in Table 2.1, the reason for visit for patients was either CPAP treatment or diagnosis of sleep apnea. For patients seeking CPAP, they would already have been diagnosed with apnea, and the visit would most likely be to evaluate the effectiveness of the treatment or tune the parameters of their personal breathing masks. Because this project aimed to detect sleep apnea, this means that the dataset used is inherently biased toward the positive class. This observation is also supported by the mean AHI score listed in the table. In the training set used for this project, the mean score for AHI is 19, which would be classified as moderate sleep apnea. This does not reflect real-world screening scenarios, where the majority of individuals would be healthy [1]. Consequently, the models may have developed a tendency to overpredict apnea events, contributing to the relatively low precision observed. This imbalance also affects the generalization ability of the models. While they perform well in identifying apnea events within a clinical dataset, their performance may degrade when applied to a general population.

4.4 Feature representation

The selected set of nine features appears to provide a sufficient representation of the SpO_2 signal for the classification task. The combination of statistical features and proportion-based features allows the models to capture both transient desaturation events and sustained oxygen level patterns.

Although only nine handcrafted features were extracted, the relationship between feature representation and model performance can also be interpreted in relation to the "curse of dimensionality" [55], which describes how increasing the number of features causes the feature space to grow rapidly, making data increasingly sparse and reducing the effectiveness of distance-based learning. This is particularly relevant for k-NN, which relies directly on distances between observations and may therefore be negatively affected by sparsity, helping explain its weaker performance. Overall, the moderate dimensionality in this study suggests that adding more handcrafted features would only be beneficial if they contribute meaningful additional information.

4.5 Explainable AI

ML models that achieve high classification accuracy are not necessarily clinically trustworthy. In the context of sleep apnea detection, where inference could directly impact diagnosis and treatment decisions, the explainability of a model's decision is significant. A qualified person using the model needs to be able to understand not only what decision was made, but also on what basis it was made and which input signals contributed most to that decision. This is particularly important in medical applications where trust and accountability are essential requirements for clinical practice.

The degree of explainability varies across the different models considered in this study. Random forest provides a relatively transparent structure, as feature importance measures can be extracted to indicate which input variables contributed most to the overall decision-making process. The k-NN algorithm offers a different form of interpretability, where predictions can be explained through the notion of similarity to specific training instances. In this case, a given classification can be justified by examining the closest data points in the feature space, effectively linking the decision to concrete patient examples rather than abstract model parameters.

Support vector machines occupy an intermediate position in terms of interpretability. When a linear kernel is used, the model can be interpreted through its learned feature weights, allowing some insight into how individual input variables influence the decision boundary.

FCNN models, as a class of deep learning models, are inherently difficult to interpret due to their non-linear architecture. As discussed in section 2.5, this lack of transparency poses a challenge in medical contexts, where decisions must be both reliable and explainable. While the classical models all have different ways of examining their results discussed above, DL models generally lack this capability. The field of Explainable Artificial Intelligence (XAI) has emerged with the aim to improve the interpretability of complex machine learning models. Two widely used XAI methods are SHAP and LIME [56], both of which could have been applied in this project. However, these approaches were deemed out of scope of this project, as clinically explainable results were not part of the primary objective. Consequently, the FCNN results remain difficult to interpret, and allocating time toward implementing explainability methods could potentially have produced more clinically meaningful results.

4.6 Case study results

As seen in Table 4.6, the estimated AHI was high for all recordings. AHI was ranging from 67.4 to 87.9 events per hour. Recordings 2–4 for participant number 1 produced very similar AHI values, approximately 88 events per hour, despite having different recording durations. Recording 1 for participant 1 resulted in a somewhat lower, but still very elevated, AHI of 67.4 events per hour. Recordings 1–3 for participant number 2 also produced very similar AHI values, with an average of 80 events per hour.

Table 4.6: Apnea events and sleep statistics for the different recordings.

Participant	Recording	Duration (h)	Apnea events	AHI (events/h)
1	1	3.38	228	67.4
	2	0.67	59	87.8
	3	1.80	158	87.8
	4	4.86	427	87.9
2	1	7.38	588	79.6
	2	7.11	592	83.2
	3	2.56	202	79.0

These results indicate that the model detected a large number of apnea events in the EmotiBit data. However, the participants were presumed to be healthy and had no prior diagnosis of sleep apnea. Therefore, the AHI values should be interpreted with caution. In a clinical setting, AHI values above 30 events per hour are considered severe sleep apnea [5], and the mean AHI for the dataset the models were trained on was 19, see Table 2.1. Thus, the values obtained in this case study would suggest an unexpectedly severe condition if interpreted directly. Since this is unlikely for healthy individuals, the results instead suggest that the model has overestimated the number of apnea events when applied to the EmotiBit recordings.

A likely explanation is that the model was applied to data collected with a different measurement system than the one used to obtain the training data. The model was trained and evaluated on labeled sleep apnea data from one type of recording setup, while the case study relied on signals recorded with an Emotibit wearable device in an uncontrolled home environment. This difference is important because the model learned device-specific signal characteristics, including amplitude ranges, noise patterns, sampling behavior, and artifacts. The EmotiBit recordings may differ from the training data in terms of sensor type, sensor placement, signal quality, and sampling characteristics, which could have caused a mismatch between the patterns from the training data and the EmotiBit recordings. As a result, patterns in the EmotiBit recordings may have been incorrectly classified as apnea events.

As part of the comparison between the training dataset and the EmotiBit case study recordings, the number of unique SpO_2 values per recording was calculated after restricting the signal to the range 70–100%.

Table 4.7: Summary statistics for the number of unique SpO_2 values in the range 70–100% per recording for the training dataset and EmotiBit recordings.

Data source	**Mean unique values**	**Min**	**Max**
Training dataset	18	4	30
EmotiBit recordings	19799	17144	23574

Table 4.7 shows a clear difference in the number of unique SpO_2 values between the training dataset and the EmotiBit recordings. After restricting the SpO_2 values to the range 70–100%, the training dataset contained a mean of 18 unique values per recording, while the EmotiBit recordings contained a mean of 19799 unique values per recording. This indicates that the EmotiBit recordings had a finer numerical resolution than the training data. The training dataset also showed a wide range in the number of unique values, from 4 to 30, compared to 17144 to 23574 for the EmotiBit recordings. Therefore, the range of unique values was larger relative to the mean in the training dataset than in the EmotiBit recordings, indicating greater variation in numerical resolution within the training dataset.

The difference in numerical resolution may explain the poor results observed in the case study. Since the model was trained on SpO_2 signals with substantially fewer unique values, it may have learned patterns related to the structure of the training data, rather than only physiological pat-

terns associated with apnea events. When applied to the higher-resolution EmotiBit recordings, normal signal variation may therefore have been interpreted as apnea-related patterns, resulting in elevated AHI values for healthy participants. To bridge the gap in numerical resolution, one possible solution would be to divide the EmotiBit SpO_2 signal into 30 bins, based on the maximum number of unique values from the dataset. This would represent the interval 70–100% and reduce the numerical resolution of the EmotiBit recordings, making their distribution more comparable to the discrete structure observed in the training dataset.

Since no clinical reference measurement was available for the case study recordings, the predicted events could not be verified as true apnea events. Therefore, the AHI values should not be interpreted as a clinical diagnosis, but rather as an indication of how the model behaved on EmotiBit data. Although, it is worth noting that recordings 2–4 for participant 1 and recordings 1-3 for participant 2 produce very similar values for each participant. This suggests that the model behaved consistently when applied to several EmotiBit recordings from two different participants.

Overall, the case study demonstrates that although the model performed well on the dataset used for training and testing, its predictions may not generalize reliably to EmotiBit recordings without further validation. The consistently high AHI values suggest that additional preprocessing and validation against clinically labeled EmotiBit data would be necessary before the model could be used for practical sleep apnea screening.

5

Conclusion and future work

This chapter presents the conclusions of the project and outlines directions for future work.

5.1 Model viability

As previously shown in section 4.2, the FCNN model outperforms the classical models by some margin. This indicates that a deep model's ability to learn features by itself from raw data and its non-linear nature, is well suited for capturing the complex patterns associated with apnea events. However, as mentioned in section 4.5, a deep model's decision making is much more difficult to understand and interpret, a clear downside in the context of medical diagnosis.

The best FCNN model accurately identifies up to 88% of the true apnea and hypopnea events. Additionally, as shown in Figure 5.1, the model does not produce an excessive number of false positive predictions which is a desirable characteristic in preliminary medical diagnosis.

When provided with a record containing many apnea events, as seen in Figure 5.2, the FCNN model can relatively accurately mark the areas of the recording where apnea events are happening. However, all developed models display a comparatively low precision score, clearly visible through their tendency to mark large areas as sleep apnea events rather than individual windows. This behavior indicates that the models lack the ability to precisely detect apnea events, yet demonstrate a capability to characterize whether a patient has recently undergone an apnea event.

The other model types display a similar result as the FCNN model used when producing these two graphs. The low precision may be an effect of the choice of signal and features, and a more complex representation of the data might provide the models with more information which could improve the accuracy of the predictions.

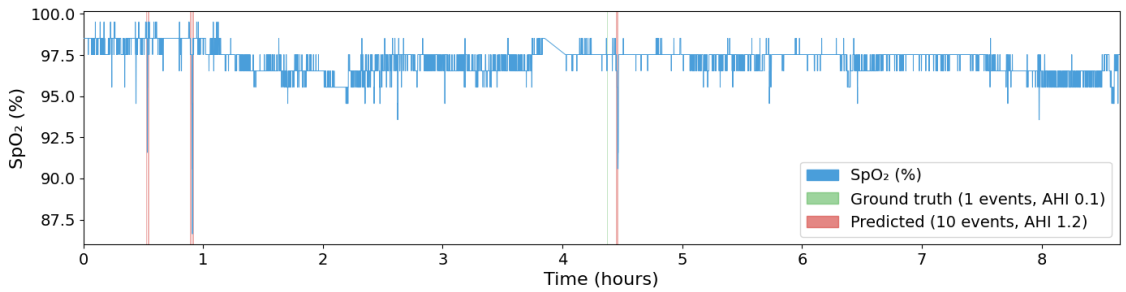


Figure 5.1: An FCNN model's predictions for one record (red), layered over the true annotations (green)

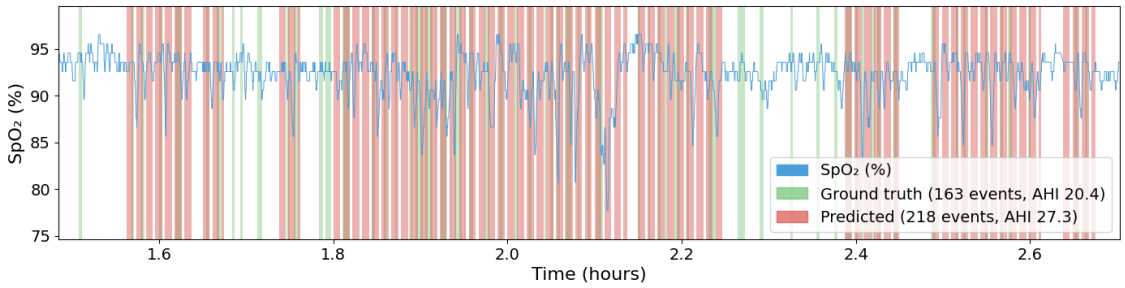


Figure 5.2: Same model's predictions for another record

5.2 Case study and EmotiBit

Applying ML models trained on clinical data on newly recorded data using an EmotiBit remains an interesting field of application, but requires further adaptations than what was conducted during this project. Due to several factors discussed in section 4.6, the model's performance on the data collected in the case study was unreliable and produced implausible results.

5.3 Final conclusion

This study demonstrates that machine learning models can be applied to detect sleep apnea using SpO_2 signals alone. Among the evaluated models, the fully connected neural network achieved the best overall performance, indicating that nonlinear methods are particularly well-suited for capturing the complex patterns associated with apnea events.

5.4 Future work

For future work, a more extensive case study should be conducted to further evaluate the proposed machine learning approaches for sleep apnea detection under a wider range of real-world conditions. This could include a larger and more diverse patient population and longer monitoring periods in order to assess the models' applicability. Alternatively, future case studies could use a wearable device that measures SpO_2 more similarly to sensors used in PSG, therefore reducing the mismatch between recordings from the case study and the recordings from the dataset. An important part of this could also be to include XAI in the scope of the project, since a larger quantity of patients and real-world applicability places larger requirements on explainable results.

Two additional areas of possible future interest are the impact of feature extraction on model performance, as well as training models to differentiate between sleep apnea and hypopnea events. The selection of features has the potential to greatly influence performance of classical ML models, and exploring which features provide the best results for sleep apnea detection could lead to furthered potential of using ML in clinical applications. Distinguishing between hypopnea and sleep apnea is a relevant area of future work. The two conditions behave differently, and training ML models to detect both simultaneously could lead to their performance decreasing compared to models developed for each condition separately.

References

[1] G. Iannella et al., "The global burden of obstructive sleep apnea," *Diagnostics (Basel)*, vol. 15, no. 9, p. 1088, 2025. DOI: 10.3390/diagnostics15091088.

[2] V. Kapur, K. P. Strohl, S. Redline, C. Iber, G. O'Connor, and J. Nieto, "Underdiagnosis of sleep apnea syndrome in u.s. communities," *Sleep and Breathing*, vol. 6, no. 2, pp. 49–54, 2002. DOI: 10.1007/s11325-002-0049-5.

[3] M. Knauert, S. Naik, M. B. Gillespie, and M. Kryger, "Clinical consequences and economic costs of untreated obstructive sleep apnea syndrome," *World Journal of Otorhinolaryngology - Head and Neck Surgery*, vol. 1, no. 1, pp. 17–27, 2015. DOI: https://doi.org/10.1016/j.wjorl.2015.08.001.

[4] S. D. Yalim, N. Bayram, I. Ozdemir, C. Cingi, and N. B. Muluk, "The differences between the subtypes of obstructive sleep apnea among different provinces in turkey: A multicenter study," *European Archives of Oto-Rhino-Laryngology*, vol. 282, no. 5, pp. 2687–2696, 2025. DOI: 10.1007/s00405-025-09271-6.

[5] V. Ho, C. M. Crainiceanu, N. M. Punjabi, S. Redline, and D. J. Gottlieb, "Calibration model for apnea-hypopnea indices: Impact of alternative criteria for hypopneas," *Sleep*, vol. 38, no. 12, pp. 1887–1892, Dec. 2015, ISSN: 0161-8105. DOI: 10.5665/sleep.5234.

[6] C. R. Laratta, N. T. Ayas, M. Povitz, and S. R. Pendharkar, "Diagnosis and treatment of obstructive sleep apnea in adults," *Canadian Medical Association Journal*, vol. 189, no. 48, E1481–E1488, 2017. DOI: 10.1503/cmaj.170296.

[7] P. Zappalà, M. Lentini, S. Ronsivalle, S. Lavalle, L. La Via, and A. Maniaci, "The global socioeconomic burden of obstructive sleep apnea: A comprehensive review," *Healthcare*, vol. 13, 2025, ISSN: 2227-9032. DOI: 10.3390/healthcare13172115.

[8] M. P. Sarala, C. Charitha, C. K. Kala, A. Yaswanthi, and A. Haq, "Sleep disorder prediction using advanced machine learning techniques," *IJSAT-International Journal on Science and Technology*, vol. 16, no. 1, 2025. DOI: https://doi.org/10.71097/IJSAT.v16.i1.2182.

[9] Y. Zhang et al., "Deep learning for obstructive sleep apnea detection and severity assessment: A multimodal signals fusion multiscale transformer model," *Nature and Science of Sleep*, vol. 17, p. 1, 2025, ISSN: 11791608. DOI: 10.2147/NSS.S492806.

[10] M. L. D. Araujo et al., "Status and opportunities of machine learning applications in obstructive sleep apnea: A narrative review," *Computational and Structural Biotechnology Journal*, vol. 28, pp. 167–174, Jan. 2025, ISSN: 2001-0370. DOI: 10.1016/J.CSBJ.2025.04.033.

[11] W. Gu, L. Leung, K. C. Kwok, I.-C. Wu, R. J. Folz, and A. A. Chiang, "Belun ring platform: A novel home sleep apnea testing system for assessment of obstructive sleep apnea," *Journal of Clinical Sleep Medicine*, vol. 16, no. 9, pp. 1611–1617, 2020. DOI: 10.5664/jcsm.8592.

[12] T. Paul et al., "Ecg and spo2 signal-based real-time sleep apnea detection using feed-forward artificial neural network," *AMIA Joint Summits on Translational Science Proceedings*, pp. 379–385, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9285163/.

[13] N. T. H. Trang, T. T. D. Linh, D. Q. Vu, B. T. H. Loan, N. N. Vinh, and T. N. Dang, "Artificial intelligence-based approaches for sleep-related breathing events identification using eeg and ecg signals," *Sleep and Breathing*, vol. 29, no. 5, p. 276, 2025. DOI: 10.1007/s11325-025-03442-9.

[14] J. Zhu, A. Zhou, Q. Gong, Y. Zhou, J. Huang, and Z. Chen, "Detection of sleep apnea from electrocardiogram and pulse oximetry signals using random forest," *Applied Sciences*, vol. 12, no. 9, 2022, ISSN: 2076-3417. DOI: 10.3390/app12094218.

[15] U. N. D. of Economic and S. Affairs, *Goal 3: Ensure healthy lives and promote well-being for all at all ages*, Accessed: May 13, 2026, 2026. [Online]. Available: https://sdgs.un.org/goals/goal3%7D.

[16] C. Pascale, G. Pinna, and N. Artom, "Obstructive sleep apnea and arterial hypertension," *Italian Journal of Medicine*, vol. 11, no. 1, pp. 8–14, Apr. 2016. DOI: 10.4081/itjm.2016.692.

[17] F. J. Nieto, P. E. Peppard, T. Young, L. Finn, K. M. Hla, and R. Farré, "Sleep-disordered breathing and cancer mortality: Results from the wisconsin sleep cohort study," *American Journal of Respiratory and Critical Care Medicine*, vol. 186, no. 2, pp. 190–194, Jul. 2012. DOI: 10.1164/rccm.201201-0130OC.

[18] H. Rahimi-Eichi, J. T. Baker, A. M. Fjell, and R. L. Buckner, "Age- and sex-related differences in sleep patterns and their relations to self-reported sleep and mood," *Sleep Advances*, vol. 6, no. 4, zpaf079, 2025. DOI: 10.1093/sleepadvances/zpaf079.

[19] M. Khadeja and M. Elhossieny, "Accuracy of pulse oximetry in comparison with arterial blood gas," *Research and Opinion in Anesthesia and Intensive Care*, vol. 7, p. 51, 1 2020, ISSN: 2356-9115. DOI: 10.4103/ROAIC.ROAIC_91_18.

[20] M. M. Ghassemi et al., "You snooze, you win: The physionet/computing in cardiology challenge 2018," *Computing in Cardiology*, vol. 2018-September, Sep. 2018, ISSN: 2325887X. DOI: 10.22489/CINC.2018.049.

[21] A. L. Goldberger et al., "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *circulation*, vol. 101, no. 23, pp. 215–220, 2000. DOI: doi.org/10.1161/01.CIR.101.23.e215.

[22] N. S. Freedman, *Primary Care Sleep Medicine: A Practical Guide*. Humana Press, 2007, pp. 131–146, ISBN: 978-1-59745-421-6. DOI: 10.1007/978-1-59745-421-6_12.

[23] M. K. Islam, A. Rastegarnia, and S. Sanei, "Signal artifacts and techniques for artifacts and noise removal," in *Signal Processing Techniques for Computational Health Informatics*, ser. Intelligent Systems Reference Library, Springer, Cham, 2021, pp. 23–79. DOI: 10.1007/978-3-030-54932-9.

[24] A. M. Alqudah and Z. Moussavi, "Bridging signal intelligence and clinical insight: A comprehensive review of feature engineering, model interpretability, and machine learning in biomedical signal analysis," *Applied Sciences*, vol. 15, no. 22, 2025, ISSN: 2076-3417. DOI: 10.3390/app152212036.

[25] A. Demircioğlu, "The effect of feature normalization methods in radiomics," *Insights into Imaging*, vol. 15, no. 1, 2024. DOI: 10.1186/s13244-023-01575-7.

[26] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011, ISBN: 0123814790.

[27] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*, 3rd ed. San Diego, CA: Academic Press, 2004.

[28] A. Burkov, *The Hundred-Page Machine Learning Book*. Quebec City, Canada: Andriy Burkov, 2019, ISBN: 9781999579500. [Online]. Available: http://ema.cri-info.cm/wp-content/uploads/2019/07/2019BurkovTheHundred-pageMachineLearning.pdf.

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

[30] S. Notley and M. Magdon-Ismail, "Examining the use of neural networks for feature extraction: A comparative analysis using deep learning, support vector machines, and k-nearest neighbor classifiers," *arXiv preprint arXiv:1805.02294*, 2018. DOI: https://doi.org/10.48550/arXiv.1805.02294.

[31] O. Faust, Y. Hagiwara, T. J. Hong, O. S. Lih, and U. R. Acharya, "Deep learning for healthcare applications based on physiological signals: A review," *Computer Methods and Programs in Biomedicine*, vol. 161, pp. 1–13, 2018, ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2018.04.005.

[32] S. Hong, Y. Zhou, J. Shang, C. Xiao, and J. Sun, "Opportunities and challenges of deep learning methods for electrocardiogram data: A systematic review," *Computers in Biology*

and Medicine, vol. 122, p. 103 801, 2020, ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2020.103801.

[33] S. Somani et al., "Deep learning and the electrocardiogram: Review of the current state-of-the-art," *EP Europace*, vol. 23, pp. 1179–1191, 8 Aug. 2021, ISSN: 1099-5129. DOI: 10.1093/EUROPACE/EUAA377.

[34] S. S. Skiena, *The Data Science Design Manual* (Texts in Computer Science), 1st ed. Cham: Springer, 2017, ISBN: 978-3-319-55444-0. DOI: 10.1007/978-3-319-55444-0.

[35] Z. Zhang, "Introduction to machine learning: K-nearest neighbors," *Annals of Translational Medicine*, vol. 4, no. 11, p. 218, 2016. DOI: 10.21037/atm.2016.03.37.

[36] G. Louppe, "Understanding random forests: From theory to practice," PhD dissertation, Ph.D. dissertation, University of Liège, Jul. 2014. arXiv: 1407.7502 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1407.7502.

[37] "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[38] S. A. Hicks et al., "On evaluation metrics for medical applications of artificial intelligence," *Scientific Reports*, vol. 12, Art. no. 5979, 2022. DOI: 10.1038/s41598-022-09954-8.

[39] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: An overview," *arXiv preprint arXiv:2008.05756*, 2020. DOI: 10.48550/arXiv.2008.05756.

[40] M. Karppa. "Minerva." GitLab repository, accessed 2026-04-16. [Online]. Available: https://git.chalmers.se/karppa/minerva.

[41] S. M. Montgomery, N. Nair, P. Chen, and S. Dikker, "Introducing emotibit, an open-source multi-modal sensor for measuring research-grade physiological signals," *Science Talks*, vol. 6, p. 100 181, 2023, ISSN: 2772-5693. DOI: https://doi.org/10.1016/j.sctalk.2023.100181.

[42] EmotiBit, *Emotibit shop - wearable biometric sensing for any project.* [Online]. Available: https://www.emotibit.com/.

[43] Connected Future Labs, *Connected future labs*, https://www.connectedfuturelabs.com/, Accessed: 2026-06-02, 2026.

[44] EmotiBit, *Emotibit brainflow spo2 algorithm*, https://github.com/EmotiBit/EmotiBit_Brainflow_SpO2_Algorithm, Accessed: 2026-04-15, 2025.

[45] N. H. Hoang and Z. Liang, "Ai-driven sleep apnea screening with overnight blood oxygen saturation: Current practices and future directions," *Frontiers in Digital Health*, vol. 7, 2025. DOI: 10.3389/fdgth.2025.1510166.

[46] C. Singtothong and T. Siriborvornratanakul, "Deep-learning based sleep apnea detection using sleep sound, spo2, and pulse rate," *International Journal of Information Technology*, vol. 16, pp. 4869–4874, 2024. DOI: 10.1007/s41870-024-01906-x.

[47] M. Jayawardhana and P. de Chazal, "Enhanced detection of sleep apnoea using heart-rate, respiration effort and oxygen saturation derived from a photoplethysmography sensor," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2017, pp. 121–124. DOI: 10.1109/EMBC.2017.8036777.

[48] S. Parvaneh, J. Rubin, A. Samadani, and G. Katuwal, "Automatic detection of arousals during sleep using multiple physiological signals," *Computing in Cardiology*, 2018. DOI: 10.22489/CinC.2018.152.

[49] scikit-learn developers, *Neighbors*, https://scikit-learn.org/stable/modules/neighbors.html, Accessed: May 5, 2026.

[50] scikit-learn developers, *Support vector machines*, https://https://scikit-learn.org/stable/modules/svm.html, Accessed: May. 18, 2026.

[51] scikit-learn developers, *sklearn.svm.LinearSVC*, https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html, Accessed: Apr. 30, 2026.

[52] scikit-learn developers, *sklearn.ensemble.RandomForestClassifier*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, Accessed: Apr. 30, 2026.

[53] pyTorch developers, *torch.nn*, https://docs.pytorch.org/docs/2.12/nn.html, Accessed: May. 18, 2026.

[54] pyTorch developers, *torch.tensor*, https://docs.pytorch.org/docs/2.12/tensors.html, Accessed: May. 18, 2026.

[55] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). New York, NY: Springer, 2006, ISBN: 978-0-387-31073-2. [Online]. Available: https://link.springer.com/book/9780387310732.

[56] A. M. Salih et al., "A perspective on explainable artificial intelligence methods: Shap and lime," *Advanced Intelligent Systems*, vol. 7, no. 1, 2024. DOI: 10.1002/aisy.202400304.

A

Python libraries

Table A.1: Python libraries used in the project.

Library	Version	Usage
Scientific computing		
NumPy	2.4.3	Numerical array operations; core data structure for signal processing and model input
pandas	3.0.1	Tabular data handling; loading, transforming, and storing signal data as DataFrames
SciPy	1.17.1	Reading MATLAB `.mat` files; statistical functions (skewness, kurtosis) for feature extraction
Data management		
wfdb	4.3.1	Reading WFDB-format physiological records from the PhysioNet database
PyArrow	23.0.1	Parquet file I/O for efficient storage and retrieval of processed datasets
Machine learning		
scikit-learn	1.8.0	Classical ML models (SVM, KNN, RF), evaluation metrics, and train/test splitting
joblib	1.5.3	Serialization and deserialization of trained scikit-learn model objects
Deep learning		
PyTorch	2.11.0	Neural network models (fully-connected); training and inference
tqdm	4.67.3	Progress bars during neural network training loops
Feature extraction		
AntroPy	0.2.1	Entropy and complexity measures (e.g. permutation entropy, sample entropy) for biosignal features
Visualization		
Matplotlib	3.10.8	Plotting of raw and processed signals, apnoea event annotations, and model results
seaborn	0.13.2	Statistical data visualization (distribution plots, heatmaps)
CLI and interface		
questionary	2.1.1	Interactive terminal prompts for the command-line interface
Rich	14.3.3	Formatted console output, styled tables, and status messages in the CLI
Notebook support		
Jupyter	1.0.0+	Interactive notebook environment for exploratory analysis and prototyping
ipykernel	6.25.0+	Python kernel enabling Jupyter notebooks to execute project code

B

Generative AI

During the writing of this report, generative AI tools were used to improve the language of already written text. The tools helped with grammar, clarity, and wording. All ideas and structure were developed by the authors. Furthermore, AI was utilized to assist with LaTeX syntax.

C

Source Code

The complete source code for this project, including all preprocessing pipelines, model implementations, training scripts, CLI, and evaluation utilities, is publicly available on GitHub. The repository contains the full codebase developed throughout this thesis work and can be accessed through the following link:

github.com/herr-ek/37A-Kandidat-Sleepy-AI