The University of Sheffield

Multi-sensory Fusion and Low Latency Sleep Apnea Detection

Saba Firdaus Ansaria

Supervisor: Prof. Lyudmila Mihaylova

A dissertation submitted in partial fulfilment of the requirements for the degree of MSc in Robotics

in the

Department of Automatic Control and Systems Engineering

September 25, 2024

Declaration

All sentences or passages quoted in this document from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.

Name: ______________________________

Signature: ______________________________

Date: ______________________________

Acknowledgement

Alhamdulillah! I am grateful to the Almighty for everything. I am highly obliged to the Department of ACSE for allowing me to select the topic for the Project Dissertation. I am grateful to my supervisor, Prof. Lyudmila Mihaylova, for her consistent support, guidance, supervision, and comments that helped me navigate the project journey. She has been patient throughout this research and encouraged me in the right direction. I am thankful to Ms Dongjin Yang, who assisted me in explaining the data and for our friendly chats after the meetings. I would also like to thank the Sheffield Children's NHS Foundation Trust and the group of doctors who evaluated the patients and collected the data. I am thankful to my classmates who supported me and worked with my anxieties and whining. I am grateful to my father, mother, father-in-law and mother-in-law for their support and continuous encouragement. I am grateful to my sister-in-law and brother for their immense love and care. My most enormous thanks are to my husband, whose patience encouraged me in every step of the journey.

Executive Summary

Sleep Apnea is a condition that disturbs normal breathing as one is asleep. Various health factors can cause sleep apnea, triggering several health issues such as heart diseases, high blood pressure, insomnia, etc. Sleep apnea detection is important because, according to an NHS study, undiagnosed people will cost NHS twice the diagnosed ones. Most of the previous research mainly focused on adult sleep apnea data corpora. This research work focuses on children's sleep apnea data. The children's sleep study data collected from Sheffield Children's NHS Foundation Trust hospital is very small compared to the contemporary corpora for sleep apnea detection tasks which is a challenge. This research paved a step-by-step analysis process, demonstrated the challenges at each step, and gave efficient solutions supported by empirical results. Firstly, the research has efficiently used the unlabelled children's sleep study data and labelled it with an unsupervised clustering algorithm. Then the features are visualised to understand the data-feature distribution. By using this empirical understanding of the data, different supervised statistical learning models have been used to learn the relationships among the multisensory input. Each step of these explorations is visualised and analysed. Finally, a low-resource efficient end-to-end multisensory fusion network has been presented with a multichannel-convolutional neural network. The multichannel-convolutional neural network achieved 98.1% accuracy on the test set, and the tradeoffs between performance and model size have been discussed. Finally, the research prescribes the best possible solutions for the children's sleep apnea data. The whole methodology and analysis approach will help further enrich children's sleep apnea research.

Abstract

Sleep Apnoea is a condition that disturbs normal breathing as one is asleep. Various health factors can cause sleep apnea, triggering several health issues such as heart diseases, high blood pressure, insomnia, etc. Diagnosis of sleep apnea is a substantial medical concern because, according to an NHS study, undiagnosed people will cost NHS twice the diagnosed ones. Polysomnography(PSG) is the gold standard diagnostic test to detect sleep apnea. Previous research has used the PSG test and processed the single or multisensory data to detect sleep apnea events. However, most of the research focused on adult sleep apnea data. Child sleep apnea studies are relatively few because PSG tests require the subject to go to the hospital or sleep study centre and stay there overnight. This is a complicated scenario for kids due to obvious reasons. It is also hard for children to have a regular sleep cycle in the hospital or sleep study centre. For this research, children's sleep study data is acquired from Sheffield Children's NHS Foundation Trust hospital and analysed and processed with statistical learning models. The main challenge is that the dataset, with only eight participants, is tiny compared to the apnea datasets used in previous research. Therefore, the choice of the statistical model needs to be made very carefully. Also, the prediction latency needs to be very low, as it can be used in low-resource scenarios. For this work, the raw corpus from Sheffield Children's NHS Foundation Trust hospital is parsed, processed, filtered and the dataset is prepared. Support vector machine, logistic regression and neural network models are used to learn the data distribution and apnea events. Their performance and latency are compared to suggest an optimal machine learning solution for this dataset. A multichannel convolutional neural network is used to fuse the five sensory inputs with weighted learnable parameters, which got 98.1% accuracy on the test set. Furthermore, the dataset is visualised with the model's decision boundaries to analyse the multisensory features and their relationships. These findings will be helpful in low-resource sleep apnea detection research and, most importantly, children's sleep apnea research. Finally, this work discussed the tradeoffs between various statistical learning algorithms and proposed a future research path.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sleep apnea (SA) is a sleep disorder that affects breathing while a person is asleep and is a condition predominant in all age groups. However, males are twice as likely to be affected by sleep apnea than females [1]. People suffering from sleep apnea may go through brief periods of shallow breathing or no breathing while sleeping. There are four major sleep apnea-related disorders, central sleep apnea, obstructive sleep apnea, sleep-related hypoxemia, and hypoventilation [2]. Brain's inability to send the proper signal to the lungs to control breathing patterns causes central sleep apnea. It is different from obstructive sleep apnea (OSA), which is caused by the upper respiratory tract's inability to pass air while sleeping. OSA occurs due to sleeping patterns, obesity or respiratory diseases. Sleep apnea sometimes causes hypoxemia which is low oxygen saturation in blood or a sudden drop in oxygen saturation blood. Hypoventilation is slow breathing which does not meet the oxygen need of the body, and carbon dioxide level increases in the body. OSA is a more commonly occurring sleep disorder than central sleep apnea. Sleep apnea directly does not cause death often. However, it causes other comorbidities like heart disease and high blood pressure, which are detrimental to a person's well-being. Although sleep apnea is treatable in most cases, 80% of the patients are undiagnosed [3]. According to an NHS study, about three million people have undiagnosed severe or moderate sleep apnea. The cost of NHS for an undiagnosed sleep apnea patient is twice that of the diagnosed patient. With that estimate, the NHS will use as much as ninety-six million pounds for the treatment of undiagnosed sleep apnea patients [4].

Generally, polysomnography (PSG), a multisensory method, is the most popular method for diagnosing sleep apnea. PSG employs sensors such as Electrooculogram (EOG), Electromyography (EMG), and Electroencephalogram (EEG) for observing the health parameters during a sleep study. PSG is time-consuming because the patient needs to go to the hospital or sleep centre to stay overnight, and generally, there are long patient queues in terms of waiting time. Therefore, arguably this method displaces the participants from their normal sleeping zones, which may disrupt their normal sleep patterns. Also, PSG is an arduous and costly process [5]. So, there is an emerging need for a cheap affordable, non-intrusive solution for diagnosing OSA that does not require the patient to go elsewhere for a sleep study. [6] proposed a framework for online SA detection with wearable sensors which used filtered ECG-based feature delineation to detect abnormality in cardiac rhythms. [7] used a framework consisting of Internet of Things (IoT) enabled wearable sensors with pruned 1D convolutional neural network (CNN) to detect sleep apnea. These works majorly used ECG signals for their framework. Rather than external filtering shown in [6], [7] hypothesised 1D CNNs will act as denoising filters inherently. Support vector machine models are also used as back-end classifiers [8, 9]. The previous research is further discussed in Chapter 3. Sensory inputs like ECG and SPO2 are mainly explored in the literature. Most of the sleep apnea detection research was performed with ECG signals. In this research, I explored other sensory inputs like Pulse, CO2, Snoring, and RIP airflow along with SPO2 for sleep apnea detection.

Different sleep studies have been done since the early 2000s; they were pivotal for carrying out sleep apnea research. One of the major and initial corpora for sleep apnea research was CAP data [10], where 66 men and 42 women participated. Apnea-ECG database [11] also provided important push to the sleep apnea research. A large number of people who participated in the 2018 database offered by PhysioNet/PhysioBank. The dataset contains 1,983 subjects, and they were monitored at an MGH sleep laboratory for the diagnosis of sleep disorders. The experiments were conducted in a multisensory framework, and seven signals were included, including EEG, EOG, EMG, respiration signal from the abdomen and chest, airflow and oxygen saturation (SPO2). However, all these datasets mainly contained adult participants, and ages ranged from 26-70 years. Children's sleep data is not well explored in sleep apnea research. In this research, I explore the children's sleep study multisensory data from Sheffield NHS Children's Trust for sleep apnea and abnormality detection in a sleep cycle. The dataset and data preparation is further discussed in Chapter 4.

1.1 Research Objectives

The above discussion clears some of the major accomplishments and trends in sleep apnea research that shape my project's research objectives. Children's data for sleep apnea research is very much unexplored. In this research, we have acquired children's sleep study data from Sheffield NHS Children's foundation trust hospital, and the goal is to use multisensory information other than ECG for sleep apnea research. The first research objective is to filter and prepare the raw data for this research (discussed in Chapter 4).

The first thing noticed with this data is that the data is unlabeled, and the sleep apnea events and the clear/detailed medical analysis of the data are not available. This problem can be alleviated using unsupervised algorithms to pseudo-label the data. The second objective is to cluster the data in an unsupervised way and pseudo-label the data. Also, it is crucial to visualise the clustered data for further analysis (discussed in Chapter 5).

It has been noticed that the raw data contains only eight participants, and each sleep study averages 10.5 hours. For sleep apnea research, it is a relatively small dataset compared to the standard adult sleep apnea datasets [10, 11]. The third research objective is to explore low-resource and machine learning solutions for the children's data for sleep apnea detection research. The solutions need to have a low-latency prediction time with a low computational resource. The goal for sleep apnea detection is to have cheap computational solutions that can run on low-powered devices and wearables (discussed in Chapter 7, 8).

The main objectives are summarise below

1. To prepare Sheffield NHS Children's foundation trust sleep apnea data for detecting sleep apnea events with statistical models.

2. The corpus is small, and the number of apnea and non-apnea events are unbalanced. So the goal is to explore low-resource solutions with a balanced dataset preparation.

3. To perform an in-depth literature review and present the current findings compared with the previous research work in this domain.

4. To use a multisensory fusion network by evaluating different techniques using the deep neural network models.

5. To evaluate and visualise the model learning at every step for interpreting different stages of representation learning.

1.2 Thesis structure

The structure of the thesis is as follows: Chapter 1 presents the introduction to this research domain and discusses the research motivations & objectives. Chapter 2 discusses the health factors and impacts with sleep apnea. Chapter 3 presents the background research with different machine learning models and corpora. Chapter 4 outlines and explains the dataset and the data preparation pipeline. Chapter 5 explores unsupervised learning on the children's sleep apnea data and the primary sensory information. Chapter 6 presents the unsupervised representation of the data for further analysis. Chapter 7 and 8 explores supervised machine learning models with their latency and multisensory features for sleep apnea event prediction. Finally, chapter 9 concludes this research and proposes future development based on this research.

Chapter 2

Sleep apnea: Health Impacts

Sleep Disorder Breathing(SDB) is the general term primarily used for identifying unusual sleep patterns observed in humans during breathing. It remains grossly under-recognised in most cases, resulting in adverse medical and psychosocial consequences. Specific sleep therapies for neurological disorders relate to a high morbidity rate, and health services' cost has significantly increased ([12]).

2.1 Symptoms

Sleep Apnoea is a condition that disturbs normal breathing as one is asleep. The main reason behind this is often an obstruction that blocks the back of the throat so that air cannot reach the lungs properly. This condition lasts for about 10 seconds or more, and the pause in breathing automatically forces one to wake up and start breathing again. This occurs several times during the night, making it hard for the patients to get enough oxygen for breathing and often prevents obtaining good quality sleep ([13]).

Sleep apnoea is mostly caused due to obesity which results in fat deposition around the upper airway. Patients with Sleep apnoea often experience gasping, snorting or choking noises, snoring loudly, and waking up frequently at night. During the daytime, they feel fatigued, experience excessive daytime sleepiness, often wake up with a dry mouth and headache, and it is difficult to concentrate on work (which leads to road accidents in the case of professional drivers). Memory loss, insomnia and sudden mood swings affect cognitive and behavioural skills. Patients may suffer from depression if it becomes uncontrollable. If anyone suffers from the above symptoms, it is highly recommended to discuss with their doctor immediately, which can help them to reduce their symptoms and mitigate future health problems ([13]).

2.2 Diagnostic Framework & Sensory Indicators

Polysomnography(PSG) is the diagnostic test that can be performed in a sleep clinic. PSG test is considered high standard as it requires a specific sleep clinic supervised by trained nurses ([14]). Specific bio-instrument technologies calculate various physiological and biomedical terms or parameters. During the test, the airflow in the upper level of the throat, electroencephalogram(EEG), electrocardiogram(ECG), electromyogram(EMG), electrooculogram(EOG) and Oxygen saturation(SpO2) is monitored while the patient is sleeping. It is time-consuming and expensive, so ordinary people cannot afford the diagnosis ([15, 12]).

EEG is a process of recording the electrical activity in the brain, during which the scalp is cleaned first and attached to the electrodes. ECG is the process of recording the electrical signals in the heart. The skin gets connected to sensors that detect the signals produced during each heartbeat.

EMG records the electrical activity in response to the muscle's nerve stimulation. Electrodes such as tiny needles are inserted through the skin into the muscles. EOG records the non-uniform eye movements using electrodes attached to the skin near the eyes due to the difference in the voltage between the cornea and the retina. SpO2 is recorded using a pulse oximeter that measures the oxygen percentage in the patient's blood. Moreover, the respiratory effort or snoring generates thoracic and abdominal signals along with acoustic signals that help detect sleep apnoea. The parameters calculated by the above instruments are used to draw the inference by specific measurements to deduce that the patient is suffering from sleep apnoea ([12, 15, 16]), including:

1. Oxygen Desaturation Index(ODI) is calculated from SpO2, which signifies the number of times there is a reduction in oxygen level in the blood for more than 10 seconds divided by the number of sleep hours.

2. Apnea-hypopnoea index(AHI) is the number of times the patient suffers from either apnea or hypopnoea while sleeping during one night divided by the total number of sleep hours. So, it can be defined as the AHI score that is the occurrence of hypopnoea or apnea per sleep hour. We can determine the condition of the patient using the AHI score:

   $AHI < 5$ —**normal** ,

   $5 \leq AHI < 15$ -**mild**,

   $15 \leq AHI < 30$ -**moderate**,

   and $AHI \geq 30$ —**severe**.

3. Respiratory Disturbance Index (RDI) factor is the occurrence of respiratory distress during one's sleep. The index helps determine the degree of apnea calculated during a sleep study. The formula of RDI is:

   $RDI = \frac{(Number of apnoeas + Number of hypopnea + Number of RERAs)}{total number of hours of sleep}$

   Here RERA is the number of sleep arousals due to a rise in respiratory effort.

2.3 Taxonomy, Health Impacts & Management

SDB encircles a wide range of sleep-related breathing disorders that covers Obstructive sleep apnoea(OSA)(Apnoea and Hypoapnoea), Central Apnoea and Mixed Apnoea.

2.3.1 Obstructive Sleep Apnoea(OSA)

One of the most general types of SDB is Obstructive Sleep Apnoea(OSA), found in children and adults. OSA is caused due to excessive relaxation of the muscles in the back of the throat, hindering the typical air passage. The structures such as the roof of the mouth, the soft palate, the tiny triangular tissue dangling from the soft palate known as the uvula, the tongue and the tonsils are supported by these muscles. With the relaxation of the muscle, the passage becomes narrow or shut as one breathes, disturbing the typical breathing pattern for 10 seconds or more. Consequently, the blood-oxygen level drops, leading to the blood's high carbon dioxide content. Furthermore, the brain senses this unusual reduced breathing and the patient arouses suddenly to renew the airflow. Usually, the patient could not remember this awakening due to its short length.

This pattern gets repeated, ranging from 5 times to 30 or even more each hour all night ([17, 16]). The patient awakes with choking, gasping or snorting and repairs itself within one to two deep breaths and returns to sleep. This condition, in turn, damages the patient's ability to have a deep, sound sleep throughout the night, hence resulting in sleepiness during the daytime. Most patients do not even realise that they have interrupted sleep patterns throughout the night. It results in either complete cessation - Apnoea or partial collapse of the airway- Hypoapnoea that limits the breathing rate. The total cessation of airflow in the oronasal passage lasts at least 10 seconds. Hypopnea is an event of shallow breathing that results in an abnormal drop in the respiratory rate. We can also define hypopnea with the help of the following characteristic exhibited by the patient :

1. There is an airflow drop in the oronasal cavity $\geq 30\%$ from the standard line related to a fall in the oxyhaemoglobin saturation that is $\geq 4\%$.
2. There is an airflow drop in the oronasal cavity $\geq 50\%$ from the standard line related to a fall in the oxyhaemoglobin saturation $\geq 3\%$.
3. Electroencephalographic arousals along with the above conditions. Moreover, the (Apnoea-hypopnoea index)AHI plays a crucial role in determining the intensity of OSA, which is fatal to the human body.

Though OSA can develop in any person, certain groups are more vulnerable, including obesity, age, sex, lifestyle choices, and pre-existing diseases. Obese people with fat deposition around the neck region, People with a medical history of Asthma, hypertension, diabetes, hypothyroidism and Polycystic ovary syndrome can suffer from OSA. With the increase in age, the risk of OSA increases as the quality of deep sleep reduces. Some people might inherit the narrow air passage or develop swelling in their tonsils, blocking the airway. Patients suffering from Chronic nasal congestion are twice vulnerable to OSA due to their narrow airways. People who smoke or drink alcohol are more likely to develop OSA.

Weight lose management programs may treat OSA, and exercise is recommended for obese people. Though it does not entirely eradicate OSA, it aids in maintaining blood pressure, enhances the quality of life and reduces daytime sleepiness. A therapy of Continuous positive airway pressure(CPAP) is highly recommended, which involves wearing a face mask while sleeping that maintains the airflow in the upper airway passage near the throat. Sometimes changing the sleep position also helps people.

Learning to sleep on the sides might help patients, compared to the supine position (sleeping on the back), which makes the condition even worse.

2.3.2 Central Sleep Apnoea(CSA)

Central Sleep Apnoea(CSA) is a sleep disorder which occurs when you experience a cessation during breathing because your brain stops sending signals to your muscles to take in air. It usually lasts for 10 to 30 seconds in fragments or cycles, further reducing the oxygen level in the blood ([18]). Chemoreceptors in a healthy person send impulses to the brain, which sends reflex signals to the larynx for opening and movement in the rib cage muscles and diaphragm. The muscles inflate the chest cavity and the air loads to the lungs. In Central sleep apnoea, this neurological feedback mechanism fails to maintain homeostasis. It differs from OSA, during which the muscle near the oronasal cavity relaxes while sleeping, resulting in complete or partial airflow blockage (Apnoea or hypopnoea). It is less frequent than OSA, accounting for 0.9% compared to OSA. CSA usually occurs due to prolonged illness or any medical condition that predominantly strikes the lower brainstem or the nerves connected to the muscles ([18]).

Congenital central hypoventilation syndrome (CCHS), Obesity hypoventilation syndrome(OHS), cardiac disease, Cheyne-Stokes breathing(CSB), high altitude-induced periodic breathing, idiopathic Central Sleep Apnoea(ICSA), Narcotic-induced central sleep apnoea, Neurological disorders such as Parkinson's disease, Alzheimer's disease, Hypothyroid disease, Kidney failure are the various exemplifications of CSA. With the rise in global obesity, a prevalence is observed in the OHS patient. Though ICSA is less common, it comprises less than 5% of patients assigned to sleep clinics. CSA has seen in a healthy person who experiences recurring breathing problems at high altitudes. Cheyne-Stokes breathing is prevalent in heart failure or stroke suffered patients. Each cycle comprises accelerated, slow and paused breathing and starts over again—these cycles repeat for about 30 seconds to 2 minutes. Patients on certain opioid medications also experience disturbed breathing patterns. CSA occurrence is more frequent in older adults, mainly those above 65. The deterioration in health affects their sleep patterns, making them more vulnerable to CSA. CCHS, or Ondine's curse, is a gene-linked disease affecting one in 200,000 children worldwide. CSA is observed in young babies, causing a breathing space of about 20 seconds. Patients suffering from OSA using continuous positive airway pressure(CPAP) erupt into central sleep apnoea, known as treatment-emergent central sleep apnoea. One of the symptoms observed explicitly during CSA is the absence of abdominal-thoracic movement for about 10 seconds or even longer while the patient sleeps and breaks in breathing. After awakening, patients cannot voluntarily exert extra effort to control the midriff and muscles near the thorax.

After the diagnosis, the condition of patients suffering from CSA improved by minimising the medication doses or withdrawal in due course. An insertable device is known as the remede System used as a pacemaker approved by the FDA for CSA patients. It stimulates the phrenic nerve to send signals to the diaphragm, records the respiratory signals and aids in normal breathing. The patients do not need to wear the mask at night, and the device automatically activates without sound. Bilevel positive airway pressure(BiPAP) is another machine that helps maintain two-way air pressure while inhaling and exhaling. During inhalation, it increases air pressure; during exhalation, it decreases air pressure.

2.3.3 Mixed Apnea

Mixed Apnoea or Complex Sleep Apnoea is the medical condition when the patient suffers from both obstructive sleep apnoea and central sleep apnoea ([19]). Some patients suffering from mixed apnea tend to have OSA in the first instance and gradually start developing pauses while breathing, ranging from 20 to 30 per hour every night. In

OSA, a patient gets relieved using a CPAP(continuous positive airway pressure), but this is not the case with complex apnea patients. Though OSA seems to disappear, they still experience breathing problems, which results in the appearance of CSA, and patients start experiencing repetitive pauses while breathing, consequently sleeping in fragments. In a study involving 223 patients for one month, the Mayo Clinic Sleep Disorders Centre found that 15 per cent of overall apneic patients constituted Complex apnea patients, 84 per cent with OSA, and 0.4 per cent had CSA. Males suffer more from Complex Apnoea than females due to their unstable respiratory control ([19]).

Furthermore, patients with a medical history of sleep and cardiovascular issues tend to be affected by Complex Apnoea. Patients with Complex apnea complain less of frequent waking up after their initial sleep than CSA patients, who are 32% to 79%. The diagnosis of complex Apnoea is based on the sleep patterns diagnosed at the medical centres followed by inadequate counter feedback of CPAP. Therapy involving BiPAP and ASV(Adaptive servo ventilators machines) is a treatment for mixed apnea over CPAP, the most effective treatment is still in research.

Chapter 3

Background research

This chapter details the recent advancements and machine learning research in sleep apnea research. Various health factors can cause sleep apnea, and sleep apnea events can be detected using different health parameters. Previous research explored most of the sensory information (related to PSG) for sleep apnea event detection. Over the years, the interest in sleep apnea research has grown significantly ([12]). Two main aspects of sleep apnea research are sensor types for data collection & feature engineering and machine learning classifiers. These aspects are discussed in the following sections.

3.1 Sensors & Features

One of the significant indicators of apnea is blood oxygen level. The classical approach for oxygen saturation analysis uses the oxygen desaturation index (ODI), where the cumulative saturation value under a certain threshold denotes an abnormality in the sleep cycle or an apnea event. Various approaches have been proposed to measure this threshold level. A pointwise relative variability measure was proposed, where the change in the SPO2 levels in three different points indicated an occurrence of an apnea event over a certain period of time (10-90 seconds) ([20]). Central tendency measure (CTM), which also represents quantitative variability, was used for oxygen desaturation measurement ([21]). [21] further compared non-linear methods to measure this variability. Spectral density calculation, entropy, and Lempel-Ziv complexity (LZC) are also used with SPO2 for initial feature computation and further used with different classifiers ([22, 23]). Furthermore, dimension reduction with principal component analysis was proposed for SPO2 feature engineering.

The other standard sensory input used for sleep apnea detection is ECG waveforms. Wavelet transformation is widely used to decompose ECG waveforms, and the waveform coefficients are used to form feature components. The number of components/coefficients used are differed by the type of classifiers and initial research hypothesis in various research. Therefore, more or less, the only variation regarding the wavelet approach is in the number of components and the wavelet decomposition algorithms (Debauches 4 wavelet, Tunable-Q factor wavelet transform etc.) ([24, 25, 26]). PCA has been used for dimension reduction in wavelet feature processing in [25]. Heart rate variability (HRV) and the interbeat (RR) measurements have been proposed for analysing ECG intervals among the QRS points ([27]). These measurements can either be taken from the raw data or the derived feature coefficients. Extracting time-frequency distribution and spectral clustering has been a popular approach for these purposes ([28, 29]). Linear frequency Cepstral coefficients were widely used in this context. Spatiotemporal entropy represents the uncertainty of the signal over a given window. Spectral entropy density and spatial bands are also used for HSV analysis ([30]). [31] mapped the RR intervals into a disease state space with a probability density. The state transition of the RR interval signal is measured by the parameter shifts in the Gamma distribution using a multistate cumulative sum method. Furthermore, using the exponential likelihood ratio test backward elimination, the false points in the state space transition are detected and eliminated. The severity of the disease is predicted using a severity index based on the state change points. A similar approach of index-based cross-correlation with a mixture of the abovementioned approaches is followed by several research ([32, 33, 34]).

Oronasal airflow is also used as a direct indicator of breathing disorders during the sleep cycle. It is used for OSA detection. [35] used an oronasal airflow signal and filtered it with a low pass Butterworth filter to remove the signal artefacts and normalised them for each subject. The normalised signal was temporally segmented, and three significant aspects of the signal were analysed, including the total area covered by the respiration signal to measure airflow volume, variance in the temporal regions for analysing in-flow and out-flow rate and the upper 90th percentile. For spatiotemporal analysis, Hilbert-Huang transform and respiratory rate variability (RRV) were used with threshold cutoff ([36, 37]). [38] introduced a different approach for oronasal airflow measurement. They measured respiratory effort signal (RI) and the interval between breaths by measuring the impedance of a wire coil which was strapped around the patient's rib cage. Therefore, they got the RI peak heights value and peak-to-peak time and the event of long pauses and flat-lining.

The breathing sounds, such as snoring, were also used to detect the events of OSA. The acoustic signals were converted to feature vectors using fast Fourier transforms, Mel Frequency Cepstral Coefficient (MFCC) for formant analysis ([39, 40, 41]). Various features, such as duration of the respiratory event, average ventilation, non-respiratory event rates, and nasal formants, were derived from these analyses. Apnea and no-apnea-related snoring were differentiated to detect the apnea events. Voice activity detection was also proposed for use in this regard ([42]). Tracheal sound analysis for OSA was proposed and analysed by [43, 44]. The acoustic and suprasternal pressure sensors were combined to detect snoring and intrathoracic pressure variations in OSA events because OSA may affect the resonance produced by the upper airway. However, it is crucial to mention here that the respiratory signals sometimes need more preprocessing to remove heartbeat sounds from the respiratory signals.

Several research based on the combination of different sensory information has been conducted, and the most prevalent combination is ECG and SPO2. These are the most indicative individual indicators for OSA, as well as they perform exceptionally well when combined ([45, 46, 47, 48]).

These features are used with different statistical models depending on the feature type and initial research hypotheses. Previous works with these models are discussed in the next section.

3.2 Machine Learning Models

3.2.1 Classical Machine Learning Models

Among the classical machine learning techniques, Support Vector Machines(SVM) were mostly used with the single sensory and multi-sensory information ([25, 31, 5, 35]). SVM was used with different kernels, such as linear, polynomial, and RBF. The choice of the kernel is dependent on the features. If the features are additive and linear in nature, such as SpO2, linear kernels were used for the Support Vector Classifier(SVC). If the features are complex such as MFCC, Cepstral coefficient, wavelets, RBF or polynomial kernels were used ([5, 49]). The advantage of using SVC is its simplicity and low latency time. SVM learn decision boundaries that also help to interpret the samples with model learnability.

Linear Discriminant Analysis(LDA) and Logistic Regression (LR) are also proposed with the combination of single sensory and multisensory features to detect OSA events ([31, 49, 20, 50, 51]). These models are simple yet powerful if there is less variability in the feature vectors. Due to the linear projection, LDA cannot learn efficiently with uncorrelated feature components. Both LDA and LR have similar limitations. Both of them are weak models for modelling non-linear data, and if the number of samples is less than the number of feature dimensions, these models do not learn the data distribution efficiently.

Sequential models were also used for treating the sensory information as Spatio-temporal multiplicative data. In this regard, Markovian Chains such as Hidden Markov Models were mainly used ([52, 53]). The assumption of frame independence in HMMs was particularly useful for multi-timeframe time series analysis, and it works better than previous models for high variability Spatio-temporal data. These state-space models are also interpretable to analyse the probability of state transition based on the input data. The decision trees regarding the HMMs are also helpful in fine-tuning the models. These models are mainly used with the respiratory acoustics signal and ECG signals ([53]). However, HMM-based models are more complex to train and have relatively high latency compared to SVMs and Logistic Regression.

3.2.2 Neural Network Based Models

With the advent of neural networks, research works with neural network-based classifiers have been presented. Generally, in OSA detection task, neural networks can de-correlated the data and learn the covariate shift better than previously mentioned models. Due to the simplicity of the sensory information for sleep apnea, shallow neural networks have been used in OSA detection. [23] used a small three-layered NN with oxygen saturation index as input, and the task was two-class classification (OSA positive, OSA negative). A similar experimental regime was used in [54, 55]. The OSA classification problem has often been treated as a binary classification problem. Rather than detecting apnea, hypopnea, and severe apnea, the task was simplified to detect just the presence of an apnea-related event. Like the classical models, sequence modelling has been done in OSA detection research, especially with recurrent neural networks (RNNs). RNNs such as long-short term memory networks (LSTM) have been proposed in the literature with multisensory time series sleep study information. Surprisingly, with a very small number of samples, these shallow RNNs were effective ([56, 57, 58]). Convolutional neural networks (CNN) have been shown to be effective for sleep apnea research and OSA detection as well. CNN can be perceived as stochastic filters, and they remove the dependency on rigorous feature engineering in the preprocessing step. CNN can learn feature representations directly from raw data ([59, 60]). Later research proposed hybrid CNN-LSTM models to utilise the benefits from both convolutional and recurrent structures ([61, 62, 63]). In general, the neural network-based models improve over the classical models like SVM and LR. Neural network-based models attain a significant advantage for non-linear decision boundary learning over classical machine learning models like SVM, LR, and GMMs. However, NN models come with a significant increase in latency both in the training process and prediction. Hence the use of a specific model depends on the problem statement, the available number of samples, data complexity and available computational resource. Modern wearable technologies like smartwatches and fitness bands run on low-power chips. This scenario demands low-resource solutions both computationally and data-centric. Therefore a parity among complexity-performance needs to be achieved.

Chapter 4

Data Preparation and Methodology

Children's sleep study data has been obtained from Sheffield Children's NHS Foundation Trust hospital. The data is filtered and pre-processed for this research. This chapter describes the process and the research methodology used in this research.

Patient ID	**Sensors**	**Total Sleep data (hours)**	**Max Apnea Duration (seconds)**	**Remarks**
PI001	SPO2, Pulse, RIP combine, CO2, Snore	11.38	14	brief awakenings throughout the study, desaturation as low as 85% AHI = 5.8 59 apnea events.
PI002	SPO2, Pulse, RIP combine, CO2, Snore	10.45	10	brief awakenings throughout the study, desaturation as low as 93% AHI = 12.8 111 apnea events.
PI004	SPO2, Pulse, RIP combine, CO2, Snore	8.19	19	desaturation as low as 84% AHI = 42.9 65 apnea events.
PI005	SPO2, Pulse, RIP combine, CO2, Snore	9.07	26	desaturation as low as 80% AHI = 17.6 44 apnea events.
PI006	SPO2, Pulse, RIP combine, CO2, Snore	7.54	12	AHI = 0.4
PI007	SPO2, Pulse, RIP combine, CO2, Snore	10.35	12	desaturation as low as 89% AHI = 1.8 12 apnea events.
PI009	SPO2, Pulse, RIP combine, CO2, Snore	10.05	7	AHI = 0.4
PI010	SPO2, Pulse, RIP combine, CO2, Snore	10.05	14	AHI = 1.4, 12 apnea events

Table 4.1: *Overall Children's data description*

4.1 Data description

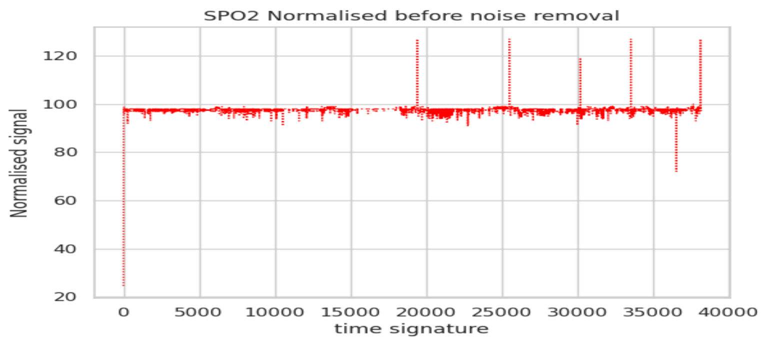
Sensor Name	**Description**
SpO2	oxygen saturation levels
CO2	measures carbon dioxide levels
Snore	Snore channel that picks up the vibration of snoring
Pulse	Pulse rate of the subject.
RIP combined	Sum of two respiratory signal channels RIP abdomen: respiratory inductance band that measure abdominal breathing RIP thorax: respiratory inductance band that measure chest breathing

Table 4.2: *Overall Children's sensory data description*

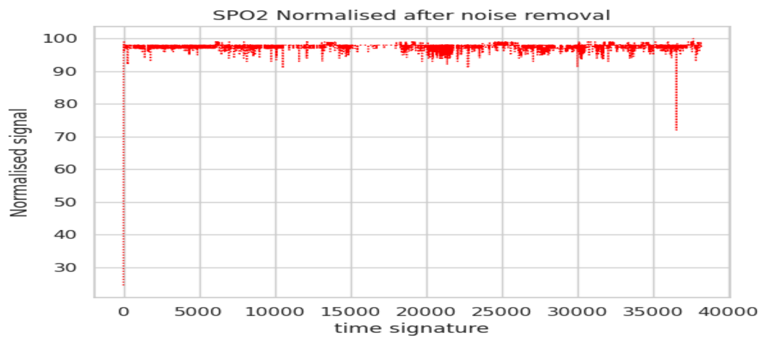
This research mainly focus on children's sleep apnea with the data collected from Sheffield Children's NHS Foundation Trust hospital. Eight participants' data are included in the children's sleep study corpora. Approximately a total of 76.8 hours of sleep study data was collected. So, on average, 9.6 hours of data are available per subject. However, apnea events are very few all over the sleep study. The maximum apnea duration per subject varied from 7-26 seconds. The desaturation index and AHI among the subjects varied significantly. The data description with the doctor's brief evaluation remark is shown in Table 4.1. Five sensory information is considered for this work. The sensors are described in Table 4.2. These sensor readings are saved in a text file format with various sampling rates. The sampling rate among the various sensors varied from 4-250 per second, depending on the sensor. These text files are parsed according to the sampling rate, and initial features are prepared.

4.2 Noise Filtering

The polysomnography sleep study (PSG) contains analogue sensors. The sensors sometimes attain noisy readings. These readings are removed from the time-series data. Time synchronisation is one of the main challenges for noise filtering in multisensory



(a) *Unnormalised SPO2 before noise removal*



(b) *Unnormalised SPO2 after noise removal*

Figure 4.1: *SPO2 without normalisation visualisation before/after visualisation*

information. Different time frames are removed from the sensory inputs. However, these sensors need to be time synchronised to create multisensory features. In this research, SPO2 is considered the primary sensor. The SPO2 sensor readings vary from 0 to 100. Garbage values (negative, high values) are removed from the SPO2 time series from the sleep study. These noisy timeframes are removed from the other sensory information as well, and all the sensors are time synchronised. In a similar way, all the sensors are filtered to remove noise. An example of SPO2 sensor data visualisation is presented in Figure 4.1a and 4.1b. Figure 4.1a shows the sensory readings with noise and after noise removal Figure 4.1b shows significant reduction in oxygen saturation levels. The timeframe on Y-axis is approximately 40000 seconds, and the X-axis shows the sensory readings.

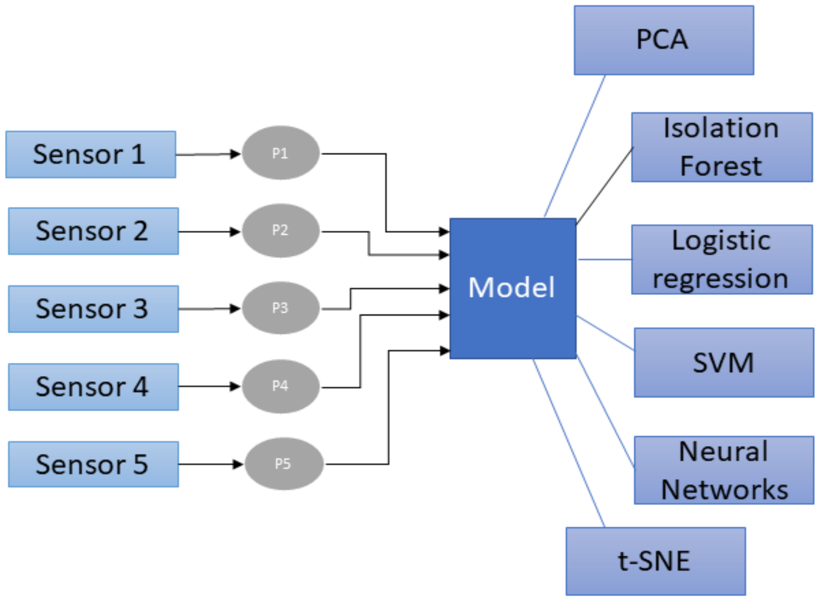


Figure 4.2: *General research framework overview*

4.3 Research Methodology

The first thing to notice about the dataset is that the dataset is unlabelled. Furthermore, the dataset has only eight participants. So, the sample size is also small. Therefore the research methodology should explore the solutions that can perform well with low resource data. The overall models and multisensor pipeline are summarised in Figure 4.2.

1. **Multisensor features:** The multisensory features are time synchronised and normalised with feature range (0,1).

2. **Flexible for weighted feature streams:** The weights are P1, P2, P3, P4, and P5 for the sensor streams for having the flexibility of weighted feature streams. The feature context is set as 10 seconds. Five-second backwards and five-second forward features with the target frame. Totalling 11 seconds context.

3. **Pseudo-label data after unsupervised:** The dataset is unlabelled. So, an unsupervised Isolation forest is employed to cluster the abnormalities or outliers in the primary sensor (SPO2). The outlier ratio is tuned using the doctor's evaluation of each patient regarding the number of apnea events per sleep study.

4. **Clustering:** The outlier timeframes are marked as apnea events. Each time frame is added with 10 second backward/forward context and synchronised with the other sensory information in those timeframes to create 55 length feature vector. A similar approach is made to get the normal sleep event timeframes. The samples for apnea events and non-apnea events are equal. These features are visualised to examine the clusters using PCA and t-SNE.

5. **Train the labelled data with supervised models:** The samples are trained with supervised models such as SVM, LR, and Neural Networks. The prediction latency and computational resources are compared with their performance.

6. **Visualisation of decision boundaries, clusters and latency:** Finally, the decision boundaries are visualised to interpret the decision boundaries according to the feature-length, type and dimension.

Chapter 5

Unsupervised Abnormality Detection for Sleep Apnea

As discussed in the previous section, the sleep apnea data is filtered, and SPO2 is chosen as the primary sensor. The dataset is unlabelled, and the multisensory data has a very limited manual evaluation of the sleep states (doctor's evaluation). However, it is observed in the doctor's report that very few apnea events occurred out of the 76.8 hours of sleep data. Therefore very few anomaly events are there in the data. In this scenario, Isolation forest ([64, 65]) has been used to detect those few abnormal regions in the time-series SPO2 data as anomaly data points. SPO2 sensor readings are the simplest, and they can directly correlate to sleep apnea events due to immediate oxygen desaturation. Therefore, SPO2 is chosen as the primary sensor. Anomaly detection is the method of searching for the data points that follows an alienated path. The data point deviates from the regular norms and follows specific offbeat patterns, defined as outliers. Anomaly detection has an extensive application in diverse fields due to its insightful nature. For example, a tumour in the breast can be diagnosed by identifying an abnormality in the Magnetic Resonance Imaging (MRI) scan. A

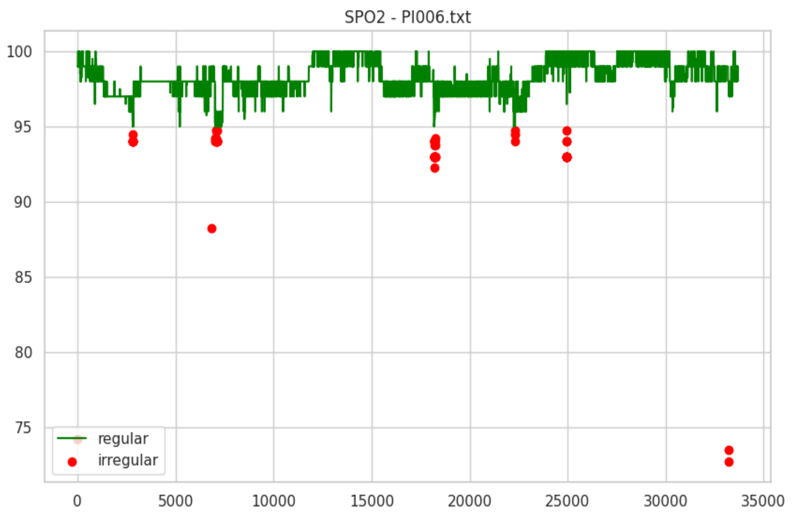


Figure 5.1: *Optimal Isolation forest SPO2 abnormality cluster (PI006)*

fraudulent transaction is identified using abnormalities in the credit card transaction or a sudden rise or fall of stocks. Detecting abnormalities in the above fields requires algorithms that intelligently recognise the pattern.

Commonly, the model based on the idea of anomaly detection involves building the concept that is normal and then specifying the points that do not obey any normality. This native method has certain drawbacks, such as selecting anomalies that are typical values or pointing out too few anomalies. The disadvantages are overcome with the concept of Isolation Forest ([64]).

5.1 Isolation Forest

Isolation Forest is a different technique for isolating abnormal data points instead of constructing an average model. Isolation Forest(IF) uses the powerful subsampling technique that enhances the algorithm's speed and low memory intake. Isolation Forest is performed with the detection of those points that are fewer in number and located away from the normal instances, which makes it more accessible. Hence, a tree structure is constructed to isolate these data points. The points that encounter proximity to the tree's root are the anomalies, and the points located towards the end of the tree are normal data points. The property of isolation forms the fundamental quality to detect the abnormality, and this tree is called the Isolation Tree or iTree ([64]). Figure 5.1 shows the anomaly regions detected with IF in patient PI006's SPO2 data over a 10.45-hour sleep study.

Isolation Forests(IF) are more likely to a Random forests based on the concept of Decision trees. As the data points are not labelled, it is defined as an unsupervised model that constructs an ensemble of iTrees. The properties of the Isolation Forest that are different from the usual procedure are stated below ([64]):

1. The isolation property of iTrees makes them feasible to use the sub-sampling method. The swamping and masking effect reduces with a small sample and enables the building of enhanced iTrees.

2. There is no need to calculate density or distance to detect the abnormality. Hence, it mitigates the problem of computational cost.

3. Isolation Forest achieves the linear time complexity with low memory intake and low constant compared to other methods that perform best only using high memory.

4. Isolation Forest quickly deals with data of high dimensions and irrelevant features.

In Isolation Forest, a point is segregated from the rest of the data points by random partitioning. The random partitioning is performed recursively until every single point is isolated. This act approaches the distinguishable anomalies, resulting in a shorter path to the abnormalities in the iTree. Therefore, the collection of shorter paths to specific data points creates a forest of random trees susceptible to anomalies. During the recursive partition, the selection of features is randomly, and then a random splitting is performed, dividing the maximum and minimum feature values.

In iTrees, the number of partitions done to separate a point is equal to the distance between the root node and the last node. The calculation of expected path length is done by finding the mean path length of all the iTRee along with different sets of partitions generated randomly.

Let N be the node of the Isolation Tree. N may be an external node without any child, or it can be an internal node with one test and two offspring nodes (N_l, N_r). The test comprises a feature 'a' and 'b' being the split value (where $b > a$) segregates the data points into (N_l, N_r).

Let us consider a sample $T = (t_1, t_2, ...t_m)$ consisting m data points from a d-dimensional distribution. The isolation tree is constructed, and the sample T is partitioned recursively by randomly selecting the feature 'a' and splitting it with the value 'b'. The procedure continues till the $|T|$ is equal to 1 and all the data points in T achieve equal values.

Now assume all the data points are entirely different. Until iTree achieves total growth, the data points can be isolated as an external node. Let m be the number of external nodes, $m-1$ be the number of internal nodes, and $2m-1$ be the total number of nodes with constrained memory requirement that develops linearly in m.

Anomaly Detection mainly generates a grading system that focuses on the hierarchy of anomalies. Hence, data points can get sorted according to their anomaly level, which describes the path length or precisely anomaly scores. The path length $(h(t))$ of a point t can be defined as the course distance in an iTree covered by a point t starting from the root node till it ends at the external node. An Anomaly Score can be computed by the algorithm based on the study of the equivalence between the structure of the iTree and Binary Search Tree(BST), where the termination point in iTree correlates to the unsuccessful search in BST. Consequently, the approximate average value of the path length for the external node in iTree equals an unsuccessful BST search.

$$c(m) = 2H(m-1)–(2(m-1)/m) \tag{5.1}$$

Here H(j) is the harmonic number, and it can be estimated by $ln(i) + 0.5772156649$ (Euler's Constant). $c(m)$ is the average of $h(t)$ given m that normalises $h(t)$. Therefore, the anomaly score s of a data point t can be expressed as

$$s(t, m) = 2^{-E(h(t))/c(m)} \tag{5.2}$$

$$when\ E(h(t)) \to c(m)\ then\ s \to \frac{1}{2} \tag{5.3}$$

$$when\ E(h(t)) \to 0\ then\ s \to 1 \tag{5.4}$$

$$when\ E(h(t)) \to (m-1)\ then\ s \to 0 \tag{5.5}$$

From the anomaly score, we can make the evaluation

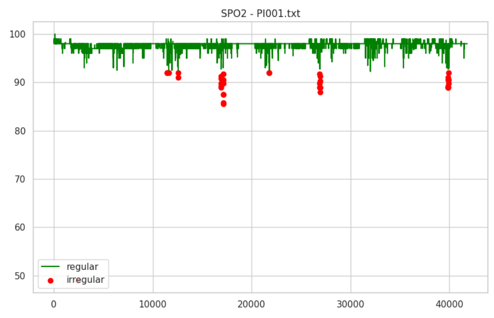
1. If the data point returns s very close to 1, then they are definite anomalies.

2. If the data point returns s, which is much smaller than 0.5, then the points are pretty normal.

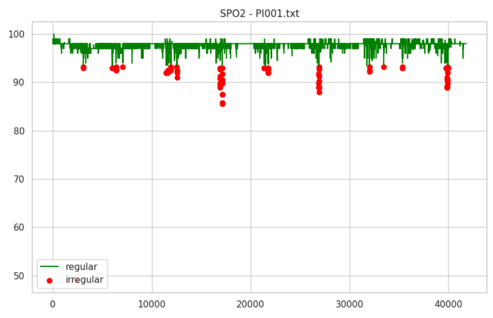
3. If all data points return $s \approx 0.5$, then the whole sample does not have any unique anomaly.

5.2 Experiments and Results

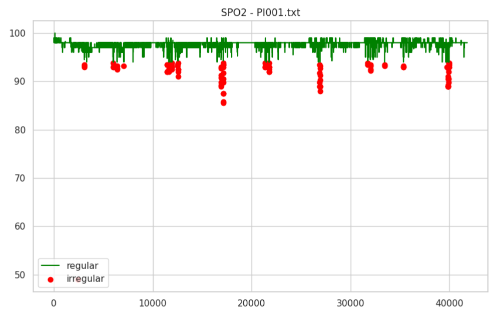
As IF detects the outliers, it is crucial to have prior knowledge of the outlier proportion. In the NHS children's sleep apnea corpus, every sleep study has certain sleep apnea events mentioned by the investigating doctors. The exact apnea event times are not mentioned, but the number of apnea events is mentioned. This prior knowledge is used to set the initial outlier proportion to the isolation forest algorithm. The outlier proportion or anomaly fraction is tested from 0.001 to 0.05, and some of the results are visualised in Figure 5.2. The data is normalised, and scikit-learn library ([66]) has been used to implement IF algorithm. The anomaly timeframes have been labelled as probable apnea events.



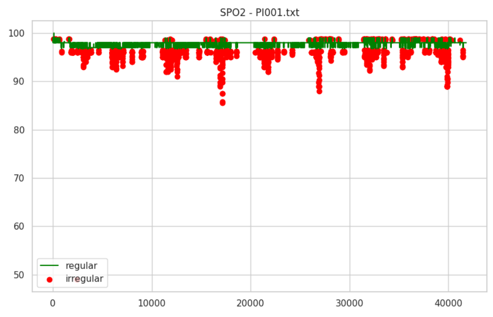
(a) *Anomaly fraction 0.001*



(b) *Anomaly fraction 0.002*



(c) *Anomaly fraction 0.003*



(d) *Anomaly fraction 0.05*

Figure 5.2: *Isolation forest SPO2 abnormality cluster comparison.*

5.3 Results & Discussion

The green points are the normal readings, and the red points are the abnormal or apnea events. The SPO2 threshold for the anomaly is relative to the patient's normal SPO2 levels. Certain degradation of oxygen levels over a 5-10 second period has been treated here as an abnormality. The results clearly show that with the increasing anomaly fraction, the Isolation forest classifier starts marking the non-anomaly regions as an anomaly or abnormal points. That is why the *anomaly fraction* has been set manually to keep a similar number of anomaly events with the investigating doctor's evaluation. For example, the fraction is 0.005 in PI001, but it is 0.002 in PI002. Between Figure 5.2a and Figure 5.2c & 5.2c, there is a huge difference regarding the anomaly regions. The difference is directly related to the threshold and decision node selection discussed in Section 5.1. The abnormal regions become redundant when the anomaly fraction reaches 0.05. From this point (*anomaly fraction=0.05*), IF output clusters became redundant.

Unsupervised anomaly detection is a very crucial step in this research. Because the apnea timeframes are labelled using this step. Each of the patient's sleep data has been clustered using different *anomaly fraction*, and the optimal outcomes have been chosen that resonate with the investigating doctor's brief evaluation. These timeframes are pseudo-labelled and synchronised with the other sensors, and feature vectors are formed, discussed in the next chapter.

Chapter 6

Unsupervised Clustering of Multisensory feature

The abnormal or apnea events are marked and pseudo-labelled in the previous chapter with SPO2 sensory information. The abnormal time frames for probable apnea events are marked, and the same time frames are extracted from the other sensory information. This chapter discusses multisensory feature processing, and the samples are visualised for interpreting the features. Firstly, the feature-creating process is discussed, and then the cluster visualisation process is briefly discussed. Finally, the experimental scenarios and the results are explained to pave the path for the following chapters.

6.1 Multisensory Feature

As mentioned in Chapter 2, a sleep apnea event triggers when an oxygen-deprived state persists for an average of 5-10 seconds. The SPO2 information and CO2, Pulse, RIP sum, and Snore may vary in the apnea time duration. As SPO2 signal information is simple and clearly indicates an apnea event, the apnea time frames are detected with isolation forest and marked as apnea. A similar number of non-apnea times are sampled from the rest of the sleep study, and they are labelled as normal samples. For each event, 5 seconds of backward and 5 seconds of forwarding context are added along with the apnea time frame, totalling 11 seconds of context. Similar time frame data points are extracted from the other four sensors, and they are concatenated to create the feature vector. Hence the length of the feature vector per sample is 55. The sample distribution over the participants in the sleep study is shown in Figure 6.1.

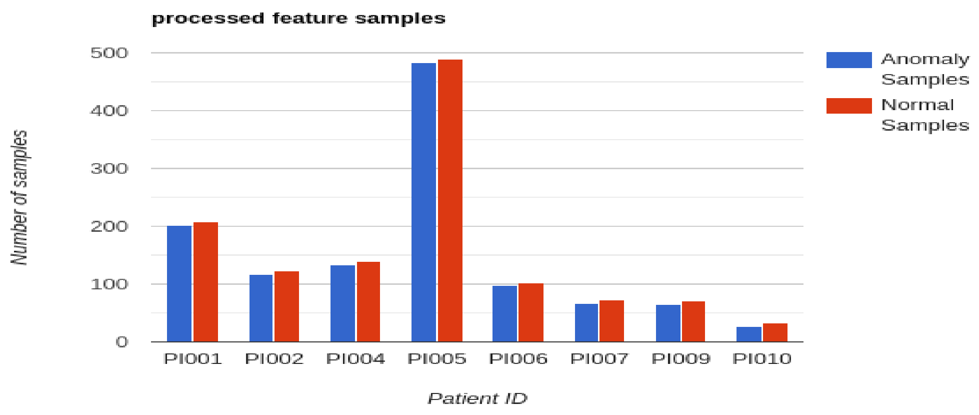


Figure 6.1: *Number of normal-anomaly sample distribution over the participants*

The normal and apnea samples are balanced equally for better model training and representation. The total number of samples are 2476. The apnea samples are 1239, and the non-apnea (normal) samples are 1237. The total sample set is divided into 85% training and 15% test set. For further research, this sample set is sampled for 5-fold cross-validation. As the SPO2 features are directly correlated because the partitions are based on these features, the other four sensory features are less correlated to the labels (apnea/normal). The feature projects are further analysed by visualising the principal components of the features.

6.2 Principal Component Analysis

Principal Component Analysis(PCA) is the method of calculating the principal components used for dimensionality reduction of large data sets while conserving the information that helps maintain the variation in data as much as possible ([67]). It is a strategy where the input variables are combined in a particular manner to eliminate the less significant variables while preserving the fundamental parts of most variables. PCA acts as a complimentary transformation where the new variables formed are less correlated and relatively independent, benefiting the linear model. The new variables included are called Principal components of the data generated by the linear combination of the original variables. So, the number of principal components depends on the variables in the data set.

PCA extracts most of the information from the original variables and compresses it into the first principal component, thereby distributing the remaining information in the second principal component. This process goes on till all the information is extracted from these components. Furthermore, PCA dumps the components containing a low percentage of information. PCA comprises five basic steps involving the method of Normalisation of the data set and computing the Covariance values, Eigen vectors and Eigen values ([68]).

1. Normalisation helps standardise the original variables so that the variables contribute equally to the data analysis. If the data is learned profoundly, diving into the details, one can observe the difference between the original variables that respond to a significant value. Hence, the distinction between the variables will disrupt the small range of variables. The normalisation equation is

$$v = (value - mean)/standarddeviation \tag{6.1}$$

2. The calculation of the Covariance matrix helps to relatively comprehend the varying nature of the variables of the original data from the mean. Moreover, to observe the existence of any relationship between the variables. Principal Component Analysis is applied when the correlation between the variables is high and regulates the redundancy that may exist in the information. This is one of the reasons to determine the covariance matrix. The diagonal of the covariance matrix contains the variance of the variables. It is known that the covariance is commutative ($Cov(p, q) = Cov(q, p)$); it can be observed that the covariance matrix is symmetric based on the diagonal of the matrix. Here, the signs of the covariance are to be considered that decide whether the two variables are correlated (that is, positive sign) or inversely correlated(that is, negative sign).

3. The eigenvectors and eigenvalues of the covariance matrix are calculated to find possible line projections and data variability to characterise the data.

4. A best-fitted line is obtained by rotating about the origin(mean). The rotation continues till it gets the largest value of the sum of the squares of the distance between the mean and the projection data set points(that is, projection of the points on the line) among them. After obtaining that line(principal component), we find the contribution of the part of variables.

5. Finally, the feature vector formed with the eigenvectors reformed the native axes of the data set to the principal components. This transformation can be represented as

$$\textit{Resulted data set} = \textit{Feature Vector}^T \times \textit{Normalized Native Data Set}^T. \tag{6.2}$$

6.3 t-distributed Stochastic Neighbor Embedding

t-distributed stochastic neighbour embedding (t-SNE) is an unsupervised statistical technique used for high-dimensional data visualisation, providing a position to each instance in a two or three-dimensional space. It trains each of the instances in high-dimensional space to a two or three-dimensional instance in a specific manner that the instances located nearby are similar to each other and have high probability value, and the instances located at distant points are dissimilar and have low probability.

The t-SNE algorithm is divided into the following steps ([69])

1. During the first step, the stochastic neighbour embedding changes the Euclidean distance between the instances in the high-dimensional space to the conditional probabilities signifying their homogeneity. The degree of homogeneity of the instance l_j to l_i is represented by the conditional probability $X_{j/i}$ such the instance l_i chooses l_j as its neighbour. In the Gaussian distribution centred at x_i, the instances are selected as neighbours of l_i relative to their probability density. The conditional probability values vary for the nearby instances being high relative to the distant instance being infinitesimally low for appropriate variance(sigma) values of the Gaussian. The conditional probability $X_{j/i}$ can be mathematically expressed as:

$$X_{j/i} = \exp(-\|l_i - l_j\|^2)/2\sigma_i^2)/\sum_{k\neq i}\exp(-\|l_i - l_k\|^2/2\sigma_i^2) \tag{6.3}$$

   where $X_{i/i} = 0$

2. During the second step, in the low dimensional space, an almost identical probability distribution is defined $q(j/i)$ for instances $m(i)$ and $m(j)$, where the variance of the Gaussian is set to $1/srqt(2)$. The Conditional probability repre-

sents the homogeneity of the instance $m(i)$ to $m(j)$, and it can be mathematically represented as

$$Y_{j/i} = \exp(-\|m_i - m_j\|^2)/2\sigma_i^2)/\sum_{k\neq i}\exp(-\|m_i - m_k\|^2/2\sigma_i^2) \tag{6.4}$$

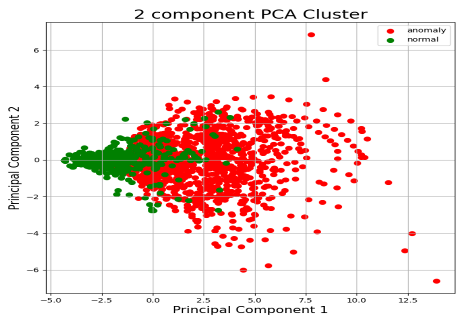
where $Y_{i/i} = 0$

3. SNE basically focuses on searching for a low-dimensional demonstration that reduces the heterogeneity between $X_{j/i}$ and $Y_{j/i}$, and Kullback-Leibler divergence represents the measured value. SNE decreases the summation value of KL divergence (Kullback-Leibler divergence) over all the instances implementing the gradient descent method. The perplexity is defined as an even estimate of the productive number of proximate. The range of the perplexity varies between 5 and 50, which signifies the effects of SNE.

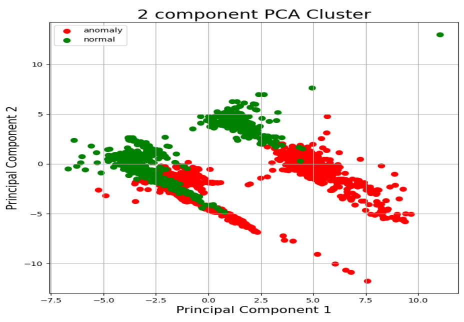
t-SNE is a non-linear dimensionality reduction algorithm because it portrays high-dimensional instances to a low dimension instance. t-SNE is used for clustering data. In the SNE, the data visualisation is feasibly good, but it is quite hard to optimise the cost function because of the issue defined as a 'crowding problem'. So, the crowding problem is solved with the t-SNE because it uses Student-t distribution instead of Gaussian.

6.4 Experiments

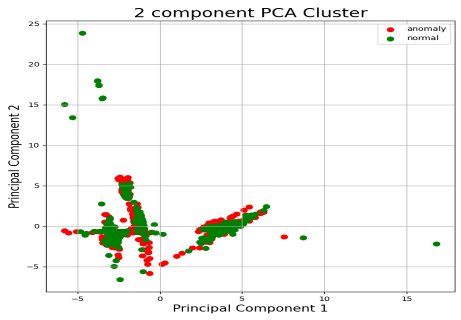
The features formed in Section 6.1, are used in the experimental scenarios. PCA and t-SNE are used to visualise the samples. Two major motivations for these visualisations are



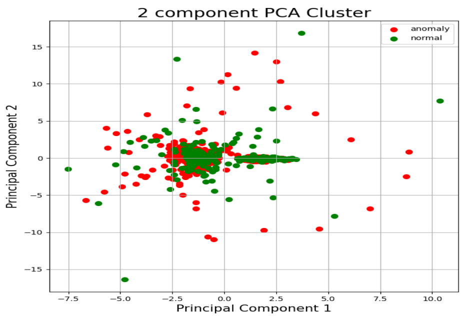
(a) *PCA with SPO2 only feature (11 dim)*



(b) *PCA with SPO2,RIP sum, CO2, Pulse, Snore (55 dim)*



(c) *PCA with RIP sum, CO2, Pulse, Snore (44 dim)*



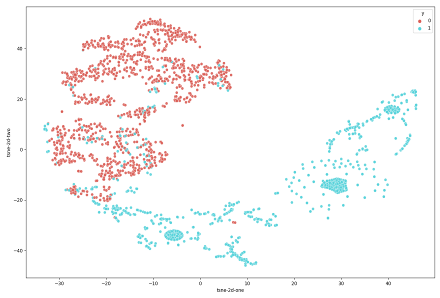
(d) *PCA with Snore (11 dim)*

Figure 6.2: *PCA visualisation with sensor feature combinations*

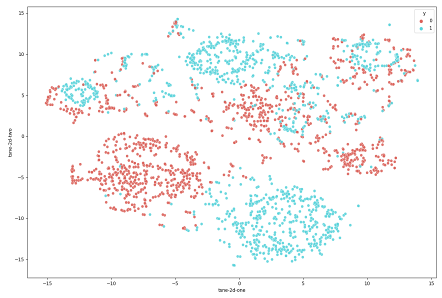
1. To understand the feature correlation with the pseudo-labels and to test the quality of unsupervised anomaly learning in Chapter 5.

2. To understand the correlation between different sensory combinations.

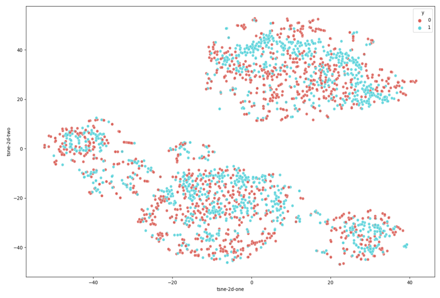
In the first scenario, PCA and t-SNE have been done with single sensory information (SPO2) and multisensory information (SPO2, CO2, Pulse, RIPs, Snore). The length of the feature vector is 11 and 55. The motivation of this experiment is to see if adding a non-correlated sensory variable along with SPO2 is degrading the feature representation. For the second scenario, except for SPO2, the rest of the sensory information in the feature vector has been used. Before preparing the feature vectors, the assumption was that all the other sensors would show irregular readings compared to the non-apnea time frame at a given sleep apnea time frame. The assumption would be correct if the PCA and t-SNE clusters have clear non-overlapping boundaries with the other four sensory features. The plots are presented in Figure



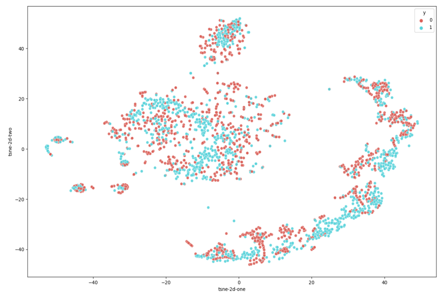
(a) *t-SNE with SPO2 only feature (11 dim)*



(b) *t-SNE with SPO2,RIP sum, CO2, Pulse, Snore (55 dim)*



(c) *t-SNE with RIP sum, CO2, Pulse, Snore (44 dim)*



(d) *t-SNE with Snore (11 dim)*

Figure 6.3: *t-SNE visualisation with sensor feature combinations*

6.5 Results & Discussion

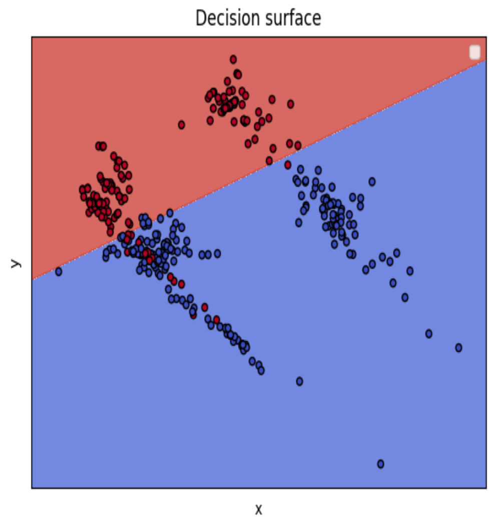
The resulted plots are presented and compared in Figure 6.2 and 6.3. The red regions denote the apnea samples, and the green regions denote the normal samples in Figure 6.2. The red regions denote the apnea samples, and the blue regions denote the normal samples in Figure 6.3. As the SPO2 data was previously clustered with isolation forest and labelled based on that, the PCA representation with 11 dimensions SPO2 feature vector has the best representation. When the other four sensors are added with SPO2, it adds variability, and it can be clearly seen between Figure 6.2a and 6.2b. However, the sample data points between the clusters are overlapping, but a clear cluster distinction can still be observed. But after removing SPO2 from the features, a drastic overlap can be observed in Figure 6.2c and 6.2d. Similar behaviour is observed in Figure 6.3. The t-SNE plots are optimised after experimenting with multiple perplexities and iteration numbers. The datapoint clusters are better in the t-SNE plots rather than the PCA plots because t-SNE uses the PCA components to distil further the relationships based on mutual information and joint probability distribution. The t-SNE plots deteriorate less than the PCA figures.

The results may imply that the four sensors are not correlated with SPO2. The other possibility might be that these above-mentioned clustering algorithms can not learn decision boundaries with the non-linear relations between SPO2 and CO2, Pulse, RIPs, Snore. Furthermore, Figure 6.2c, 6.2d, 6.3c, 6.3d show that the four sensors are not linearly dependent with the event classes; apnea (anomaly) and normal. To alleviate this problem, supervised models are trained with these pseudo-labelled samples discussed in Chapter 7 and 8.

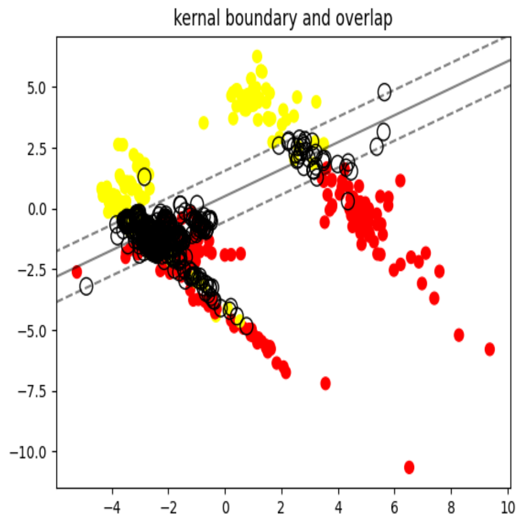
Chapter 7

Supervised Sleep apnea detection

The pseudo-labelled data from Section 6.1 has been used in this chapter for examining the data with supervised learning. The previous chapters clearly showed that after adding the other four sensory information with SPO2, the feature representation did not have a clear boundary with unsupervised t-SNE. The principal components with PCA also showed uncertainty (Figure 6.3, 6.2). In this chapter, a support vector machine-based classifier is used with the Sheffield Children's NHS Foundation Trust sleep apnea dataset to explore the pseudo-labels further and supervised model-based decision boundary learning. SVMs ([70]) have been used widely as one of the major classifiers for sleep apnea detection and analysis. The previous research has been discussed in Section 3.2.1. The next sections briefly discuss SVM kernel boundaries and model learning paradigm, then pave the research path, and finally discuss the results and visualisation.



(a) *SVM decision surface with all feature*



(b) *SVM decision surface with all feature*

Figure 7.1: *SVM classifier boundary and solution space for all features*

7.1 Support Vector Machine

In recent years, Support Vector Machine (SVM)'s discriminative power has grabbed the attention of various machine learning communities. SVM performs significantly well, even if the input data set is small. During the learning phase, SVM acquires a subset of the support vectors, which forms only a small portion of the entire data set. This small set of support vectors represents the classification tasks ([67, 71]). Classification evaluates a set of data points and constructs a model splitting the data points into the required number of classes. SVM algorithm performs in a k-dimensional space where k represents the number of features ([70]). The data points are plotted in the k-dimensional space with specific coordinate values. After that, the data classification is performed by searching a hyper-plane separating the two classes.

Let us assume some data points fit in any of the two classes, so the objective is to select the class to which a new data point belongs. In Support Vector Machine, data points appear as the k-dimensional vector comprising k numbers. We need to find a hyperplane with a (k-1) dimension that separates the data points; such a hyperplane is called a linear classifier. Various hyperplanes can be selected that separate the data points. Still, the one quality that represents the best hyperplane is the one with maximum separation, defined as the margin dividing two classes. So, a hyperplane with a maximum distance from its nearby data points on each side is considered the best; if such a kind of hyperplane is present, then it is defined as a maximum-margin hyperplane and the maximum-margin classifier, also known as a linear classifier.

Let us assume a sample of training dataset consisting of k points represented as follows: $(p_1, q_1), (p_2, q_2), \ldots.., (p_k, q_k)$. where q_i value is either 1 or -1 that denotes the class in which the data point p_i will fit in m – dimension space. The task is to find a "maximum marginal hyperplane that divides the data points p_i for $q_i = 1$ for one group and $q_i = -1$ for the other group. Then the task evaluation is done with the distance between the closest point, and the hyperplane is maximum from any one of the groups. The set of points p_i satisfying the hyperplane expressed as

$$x^T p - b = 0, \tag{7.1}$$

where x represents as a normal vector to the hyperplane and $b/\|x\|$ is the parameter that evaluates the hyperplane offsets from the origin along the normal vector x. For linearly separable training data, two parallel hyperplanes are chosen that divide the data of the two classes so that the distance between them is maximum. A hyperplane with a maximum-margin hyperplane is selected in the middle of them. The hyperplane, along with a normalized set of data, can be expressed as:-

case 1:- points above this boundary and on this with label 1

$$x^T p - b = 1 \tag{7.2}$$

case 2:-points below this boundary and on this with label -1

$$x^T p - b = -1 \tag{7.3}$$

To maximize the distance between the planes where the distance is $2/\left\|x\right\|$, the value of $\|x\|$ is to be minimized. To calculate the distance, use the distance formula from the point to the plane equation. A constraint is added to restrain the data points from falling into the margin, that is, for each i the constraint is either

$$x_i^T p - b \geq 1, if q_i = 1 \tag{7.4}$$

$$or,\ x_i^T p - b \leq -1, if q_i = -1 \tag{7.5}$$

The data points must lie on the right side of the margin according to the constraints state. So, the optimization can be stated as follows :

$$Minimize\ \ \|x\| \tag{7.6}$$

$$subject\ to \tag{7.7}$$

$$q_i(x_i^T p - b) \geq 1, for i = 1, 2, 3, \ldots, m \tag{7.8}$$

To find the solution for the classifier $p \mapsto sgn(x^T p - b)$, (where $sgn(.)$ is the sign function). The value of x and b are used. Hence, the data points p_i that lie close to the max-margin hyperplane are significant for the main geometric illustration and the data points p_i are known as support vectors.

The non-linearly separable data points, the SVM can be expanded with the use of hinge loss function

$$max(0, 1 - q_i(x_i^T p - b)) \tag{7.9}$$

The i-th target value is q_i (where the value is either one or -1), and the ith output value is $x_i^T p - b$ The value of the function is zero, the data point p_i lies on the right side of the margin. In contrast, if it lies on the wrong side, the value of the function becomes proportional to the margin's distance from the data point.

For the parameter $\lambda > 0$, Minimizing the expression below can lead to the goal of optimization in SVM,

$$\lambda \|x\|^2 + [\frac{1}{m} \sum_{i=1}^{m} max(0, 1 - q_i(x^T p_i - b))] \tag{7.10}$$

The λ helps adjust the increasing value of the margin size to ensure the correct side of the margin for the data point. If the value of λ is considered small, the expression behaves similar to linearly separable data points.

7.2 Experiments

This section mainly interprets the SVM classifier learning, performance, and decision boundaries with our multisensory dataset. Furthermore, Logistic regression has also been used for comparison ([72]). Logistic regression is a parametric linear model with low latency, and it performs fairly for learning linear relations. The features prepared in Section 6.1 are used for supervised learning. Support vector machine (Section 7.1) and Logistic regression (LR) models are used as low-latency low resource classifier. 11 dimension SPO2 feature and 55 dimension $(SPO2 + CO2 + Pulse + RIPs + Snore)$

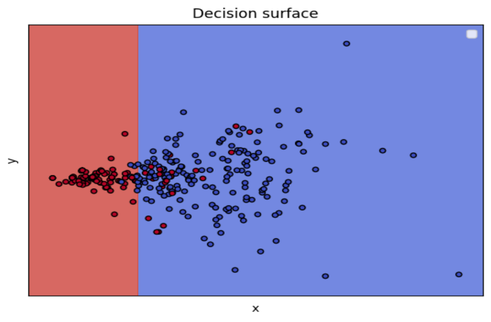
Sensory Features	**Model**	**Latency (ms)**	**Accuracy (%)**
SPO2 (11)	SVM	1.6	**94.8**
SPO2, CO2,Pulse, RIP,Snore (55)	SVM	2.3	94.2
SPO2 (11)	LR	0.15	**94.5**
SPO2, CO2,Pulse, RIP,Snore (55)	LR	0.16	94.2
CO2,Pulse, RIP,Snore (44)	SVM	16.74	61.0
Snore (11)	SVM	10.48	48.5

Table 7.1: *SVM, LR comparison with features, latency and performance*

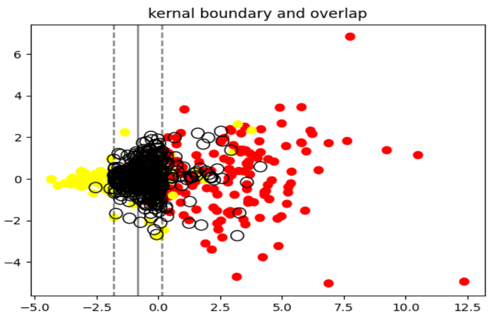
features were used with both SVM and LR models.

The raw data was partitioned using SPO2 as the primary sensor. Therefore, it is natural that SPO2 features would be directly correlated to the task. However, we do not know the correlation of the other four sensors' information with the task. Therefore, 44-dimension features (Sum RIP, CO2, Pulse, Snore) and 11-dimension snore features have also been trained and tested with SVM and LR.

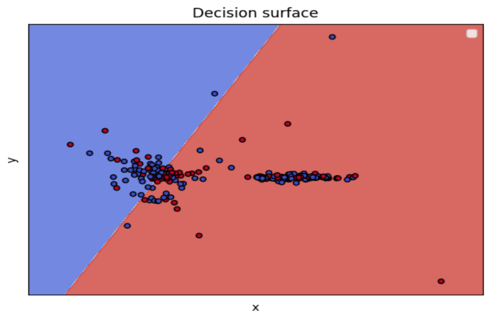
The models are trained with *Scikit-learn* library ([66]). The latency for each of the four scenarios are compared. The performace-latency table is shown in Table 7.1. The decision boundaries are shown in Figure 7.1a and 7.1b. Furthermore, SPO2 and Snore sensory features are compared for their correlation with the target labels. The decision boundaries are compared in Figure 7.2a and 7.2c. The support vector classifier coefficients are visualised for the decision boundaries in Figure 7.2b, 7.2d.



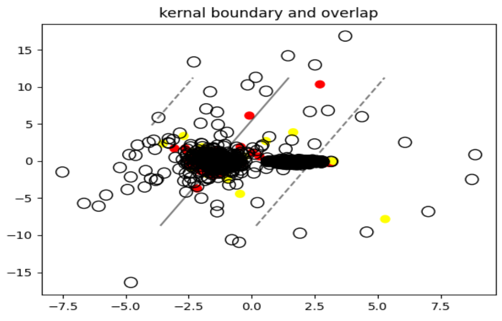
(a) *SVM decision surface with 'SPO2' feature*



(b) *SVM decision boundary and coefficients with 'SPO2' feature*



(c) *SVM decision surface with 'Snore' feature*



(d) *SVM decision boundary and coefficients with 'Snore' feature*

Figure 7.2: *SVM classifier boundary and solution space comparison*

7.3 Results & Discussion

According to Table 7.1 the number of feature dimensions increases the latency time in SVM because it creates a hyperplane depending on the feature dimension. In the logistic regression model, the increase in latency based on feature dimension is significantly less than SVM. The best results in both SVM and LR are achieved with the 11-dimension SPO2 feature. The performance slightly deteriorated when the whole sensor set was concatenated (55 dim). This clearly shows the presence of high variability non-linear characteristics of CO2, RIP sum, Snore, and Pulse. SVM and LR are linear models which are not as effective as neural networks for learning non-linear data. The best result in SVMs is achieved with the 11-dimension SPO2 feature and 55-dimension five sensory feature at 94.8% and 94.2%. The LR models also performed similarly in the same scenario. This signifies that SPO2 is highly correlated with the sleep apnea detection task. However, the performance deteriorates by about 30% when the SPO2 feature vector is removed. The accuracy with CO2, RIP sum, Snore, and Pulse is 61.0%. The accuracy with 11-dimension Snore data is 48.5%. Given that the sleep apnea detection task here is a binary classification task, the 11-dimension Snore result (48.5%) shows that the model is predicting randomly and has not converged yet. Therefore, SVM classifiers can not learn the task-specific data distribution from the four sensory features as it does with the five sensory features.

Similar trends are observed with the decision boundaries. The decision boundaries are almost similar with SPO2 single sensor data and multisensor five sensor data. However, the number of misclassification is slightly more in the multisensor data. This clearly indicates that the four sensory information increase uncertainty, which linear models like SVM and LR can not alleviate. Figure 7.2d clearly shows the SVM classifier coefficients are all over the boundaries and can not create clear boundaries among the data class. The datapoint projection and the data boundaries clearly show that the SVM classifier can not learn from 'Snore' data feature points. The latency times are also affected when we discard the SPO2 feature due to the high convergence time taken by the SVM classifier.

One of the reasons for the SVM and LR to perform poorly with CO2, RIP sum, Snore, and Pulse data may be related to the use of raw features. These four sensory information have not been simplified with a feature transformation like FFT and wavelets. As a result, the SVM and LR classifiers are unable to learn the non-linear distribution.

Chapter 8

Neural network and Multisensor fusion for Sleep Apnea detection

Previously in Chapter 7, it was evident that SVM, LR models could not learn the decision boundaries based on the four sensory features alone. Those sensory information has been given end-to-end with minimal pre-processing. Chapter 3, Section 3.1 discusses that various types of feature extraction such as FFT, wavelet transform, cumulative sum have been used to preprocess and simplify the sensory information. However, this research aims to minimise extra computation as much as possible. Without feature compression and correlation SVM, LR models have shown inefficiency when used with Snore, sum RIP, CO2, and Pulse data. This chapter explores neural network-based solutions for our low-resource data. This chapter has mainly focused on three models: feed-forward neural network based shallow model, shallow CNN-feedforward model, and multichannel CNN-feedforward model. All the models used here are shallow compared to the deep neural architectures ([61, 59, 7]). The motivation here is to explore end-to-end sleep apnea feature learning with shallow neural architectures. The next sections briefly discuss the model architectures and the working principles. Previous research works regarding sleep apnea and neural networks have been discussed in Section 3.2.2.

8.1 Feedforward Neural Network

The feedforward paradigm in multi-layer perceptrons or feedforward neural networks is a bioinspired homeostasis process that adjusts the internal states of the model in continuous space. The initial research on feedforward neural networks started with the theoretical connectionist frameworks [73]. [74] hypothesised that an upsurge in synaptic efficacy results from the recurrence of reverberatory actions between adjacent neurons. [75], [76] among others, further expanded the connectionist paradigm in neurons. From a computation point of view, a feedforward neural network is a function approximator. Therefore, a classifier, $y = f^{*}(m)$ maps the input m to a categorical distribution n.

The multi-layer FNN showed the problem of the absence of a learning algorithm, which was solved using the backpropagation algorithm. The multi-layer FNN consists of an input layer, one or more hidden layers (lying between the input and the output layer) and an output layer. The dimension of input data determines the number of input layers, the type of neural network problem determines the number of output layers and the complexity of problems estimates the number of hidden layers. A higher number of hidden layers grows the computational time due to higher numbers of parameters which further raises the chances of an overfitting problem. This problem leads to memorising the data rather than generalising it. Regularisation techniques can partially solve these issues ([77]).

There are various hidden unit activation functions in the multi-layer FNN apart from Binary step activation function (that is used for single layer neural networks) such as Logistic or Sigmoid [78], Hyperbolic [79] , Rectified linear unit (ReLU) [80] defined as

$$Logistic\ or\ Sigmoid\ \sigma(r) = 1/1 + e^{-r}, range\ (0, 1) \tag{8.1}$$

$$Hyperbolic\ tangent\ \tan h(r) = (e^r - e^{-r})/(e^r + e^{-r}), range\ (-1, 1) \tag{8.2}$$

$$\begin{aligned} Rectified\ Linear\ Unit\ f(z) &= 0,\ \ if\ z \leq 0 \\ &= z,\ \ if\ z > 0 \\ &= max(0, z),\ \ range\ [0, \propto) \end{aligned} \tag{8.3}$$

Logistic or Sigmoid activation function can be applied for the output layer and hidden layers in the multi-layer neural network. The input and output layers can build a non-linear relationship using the sigmoid activation function, but there is a vanishing gradient problem ([81, 79]). The derivative values of the sigmoid activation function are small and away from the origin that rapidly reaches zero. During backpropagation in multi-layer neural networks, the chain rule is applied where the derivative values are multiplied, and multiplying small values produces a very small number. So, the updated values of the weight become minimum and hence a slower learning algorithm. To avoid this problem, the ReLU activation function is used in deep neural networks with many hidden layers which do not suffer from the vanishing gradient problem. Hyperbolic tangent(tanh) is also used as the activation function similar to sigmoid, but its derivatives are larger, resulting in a lower rate of vanishing gradient. ReLU has been used extensively in all the layers except the last ones in this research. Softmax activation is used in the last layers of the models shown in this section.

The cost function of the supervised learning models is dependent on the output activation function, which in turn is highly dependent on the choice of a machine learning problem that is binary classification, multiclass classification or regression ([67, 82]). The output unit activation function for binary classification is considered in a Bernoulli distribution defined as a Sigmoid output unit activation function. Here the expected values ($\hat{q}$) are predicted as a probability (0,1), and it is expressed as

$$\hat{q} = P(Q = 1|p^{R-1}; w^R) = f(z^R) = 1/(1 + e^{-w^{(R)T}p^{R-1}}) \tag{8.4}$$

For N class, multi-class classification, the output unit is considered in a multinoulli distribution and the expected value is expressed as :

$$\hat{q}_j = P(Q = j|p^{R-1}; w_j^R) = f(z_j^R) = exp(w_j^{(R)T}p^{R-1}) / \sum_{k=1}^{N} exp(w_k^{(R)T}p^{R-1}) \tag{8.5}$$

where j=1,2,.....,N. The linear regression output unit activation function in the Gaussian distribution for the original value q is defined as:

$$\hat{q}_j = f(z_j^R) = w_j^{(R)T}p^{R-1}, \tag{8.6}$$

where R denotes the final layer in the neural network. Finally, the Deep Feedforward Neural Networks can be represented as combination of layers of simple linear functions,

$$s_j^t = \sum_{i=1}^{N_t} w_{ji}^t r_i^{t-1} + b_j^t \tag{8.7}$$

$$= w_j^{(t)T}s^{t-1}, \tag{8.8}$$

for $j = 1, 2, ..., J_t$.

where s_j^t is the activation at layer t, r_i^{t-1} is the input from the layer $t-1$ to layer t, w_{ji}^t is the weight parameter at the layer t, b_j^t is the bias or error parameter,

$w_j^t = (b_j^t, w_{j1}^t, w_{j2}^t, ..., w_{jN_t}^t)$, $r^{t-1} = (1, r_1^{t-1}, r_2^{t-1}, ..., r_{N_t}^{t-1})$. To obtain the output r_j^t, a non-linear activation function $h(.)$ is applied on the activation s_j^t :

$$r_j^t = h(s_j^t \tag{8.9}$$

The final layer can be obtained by passing the activation through the output unit activation function f(.) :

$$\hat{q}_j = f(s_j^Z) \tag{8.10}$$

where the total number of layers in the neural network is denoted as Z. Finally, the Z layers, deep neural network is expressed as

$$\hat{q} = f(W^Z h(....h(W^{(2)} h(W^{(1)} r)))) \tag{8.11}$$

where $W^t = (w_1^t, w_2^t, ..., w_{J_t}^t)^T$, and $r = r^0$

8.2 Convolutional Neural Network

The Convolutional neural network (CNN) architecture has gridlike overlapping kernels for parameter sharing among the whole layer [83]. Convolutional neural architecture is a biologically inspired neural network which resembles loosely to the information processing in the visual cortex ([84]). [85] proposed the idea of a weight-sharing information processing model mimicking a cat's visual cortex. In mathematical terms, the word 'convolution' comes from the mathematical convolution operation. The convolution function operates on two functions to yield a resulting function computing the subsequent overlap of function shifts on each other. If $m(x)$ and $n(x)$ are two functions over a continuous variable y, the convolution over an infinite interval would be

$$f(\mathbf{x}) * g(\mathbf{x}) = \int_{-\infty}^{+\infty} f(\gamma) \times g(\mathbf{x} - \gamma) d\gamma, \tag{8.12}$$

Here γ is the continuous time step, $*$ is the convolution operator. In signal processing, $n[y]$ is the impulse function over $m[y]$. In image processing, the interval is finite. The $n[y]$ is the local receptive field that shares the same set of weights on different regions of the input data/embeddings $m[y]$. The layers extract abstract features and combine the set of outputs to form feature maps. Kernels of size $[p \times q \times T]$ ([height $\times$ width $\times$ depth], and $t = 1, 2, \cdots, T$) are used, the t^{th} convolutional feature map can be denoted as:

$$conv_t = R\left(\sum_k n_k * y_k\right), \tag{8.13}$$

where n_k is the k^{th} kernel and y_k ($k = 1, 2, \cdots, K$) is the k^{th} input feature map of size $[A \times B]$ and $R(\cdot)$ is the activation function for introducing nonlinearity in order to convert the output of convolution into nonlinear complex problem space ([86, 87]).

Typically, CNNs have several convolution layers accompanied by spatial/temporal sub-sampling layers([83])(an example shown in Figure 8.1). Normalisation is performed batch-wise (batch normalisation) and channel-wise (instance normalisation) for better parametric learning by normalising the distribution in the embeddings. These shared regions, sparsed interactions, and equivariant representations ([82]) allow the CNNs to work with variable-length data for better representation. The sparsity permits faster learning and convergence as well. As a result, the convolution process parameter sharing has better efficiency than the standard fully-connected neural layers.

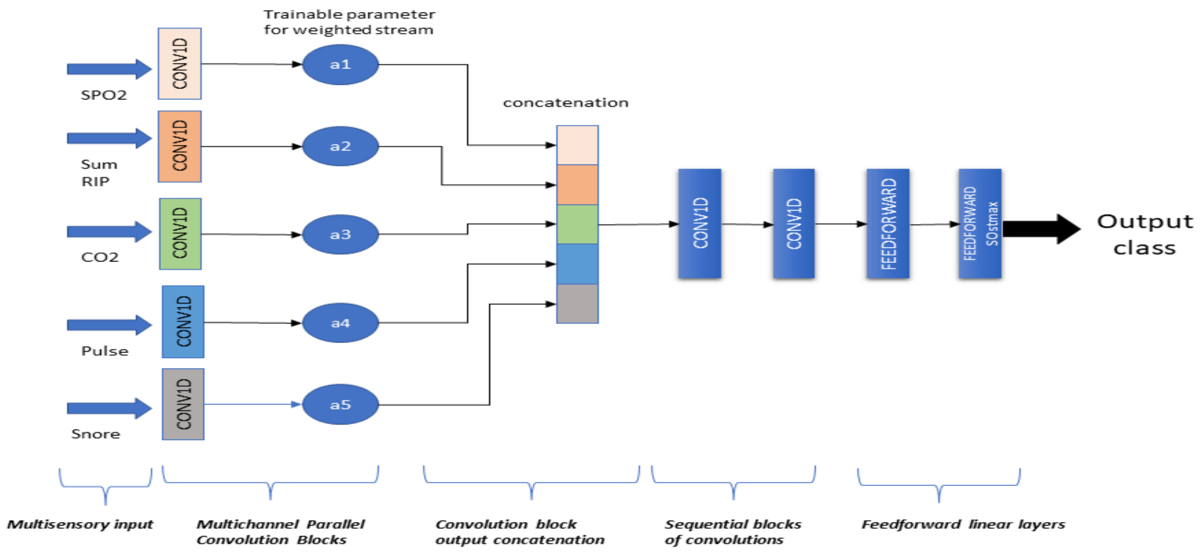

Trainable parameter for weighted stream
SPO2
Sum RIP
CO2
Pulse
Snore
CONV1D
a1
a2
a3
a4
a5
concatenation
CONV1D
CONV1D
FEEDFORWARD
FEEDFORWARD SOstmax
Output class
Multisensory input
Multichannel Parallel Convolution Blocks
Convolution block output concatenation
Sequential blocks of convolutions
Feedforward linear layers

Figure 8.1: *Multichannel Shallow CNN Model with feedforward inear layer*

8.3 Multichannel Convolutional Neural Network

In traditional CNN blocks, the CNN layers are stacked together. So, in theory, the input embedding gets a similar transformation at individual layers because they all share the same kernel weight matrix for learning the representation. In multichannel neural networks, different parallel CNN layers attend separate parts of the input embedding/information. Thus each of the separate parts of the input goes through a different transformation function, and they are connected together. The assumption here is that these separate regions go through different perspectives (kernels) of projection which helps create a better representation of the overall information/input-embedding. In principle, this works as a mixture of experts' model or simply a mixture model. Figure 8.1 shows the multichannel model architecture used in this research.

If $f(\cdot)$ is denoted here as convolution and f_1, f_2, f_3, f_4, f_5 are the parallel convolution layers the multichannel layers are computed as follows

$$embedding_x = concat(f_1(x1)\alpha_1 + f_2(x2)\alpha_2 + f_3(x3)\alpha_3 + f_4(x4)\alpha_4 + f_5(x5)\alpha_5) \quad (8.14)$$

Here $x1$ is the *sensor 1* input, $x2$ is the *sensor 2* input and so on. The final output is $embedding_x$. $\alpha_1, \alpha_2, ... \alpha_5$ are trainable parameters learned through backpropagation with the rest of the model. All the parameters and weights are initialised randomly. These learnable parameters give the network a weighted feature concatenation approach as well as learning the importance of the sensor streams while training. Furthermore, these weights give the flexibility to control the convolutional streams/channels manually if needed by disabling the gradient learning in those parameters.

8.4 Experiments

Similar experimental scenarios are followed as mentioned in Section 7.2. The 11-dimension feature SPO2, 55-dimension feature (SPO2, Sum RIP, CO2, Pulse, Snore), 44-dimension feature (Sum RIP, CO2, Pulse, Snore), and 11-dimension snore features have been used with different neural model architectures. Three different neural network models have been used mainly, i.e., the shallow feedforward model, the shallow convolutional-feedforward model and the multichannel convolutional-feedforward model. The models are trained on the Pytorch framework ([88]). The number of target classes for this task is two. Therefore, the binary cross-entropy cost function is used. For optimisation, stochastic gradient descent has been used ([89]). The learning rate has been set as 0.01. In this scenario, we have a small training sample set compared to the standard neural network training set sample size.

Furthermore, the multichannel CNN architecture in Section 8.3 and Figure 8.1 shows a mixture model based on weighted trainable parameter. The weighted repre-

Features (length)	**Model**	**Layer Description**	**Num of parameters**	**Latency (ms)**	**Accuracy (%)**
SPO2, CO2,Pulse, RIP,Snore (55)	Multichannel CNN + feedforward	5x parallel conv1d(1,64,5),1x Conv1d(64,128,5) 1x Conv1d(128,64,5), 1x Conv1d(64,64,5),1x Linear(64,128), 1x Linear(64,64), 1x Linear(64,1)	121222	9.46	98.1
SPO2, CO2,Pulse, RIP,Snore (55)	CNN + feedforward	1x Conv1d(64,128,5) 1x Conv1d(128,64,5), 1x Conv1d(64,64,5),1x Linear(64,128), 1x Linear(64,64), 1x Linear(64,1)	115713	5.68	97.0
SPO2, CO2,Pulse, RIP,Snore (55)	feedforward	1x Linear(55,128), 1x Linear(128,128), 1x Linear(128,1)	23809	2.03	96.7
SPO2 (11)	CNN + feedforward	1x Conv1d(64,64,5),1x Conv1d(64,64,3), 1x Linear(64,128), 1x Linear(128,64), 1x Linear(64,1)	45761	3.7	95.1
SPO2 (11)	feedforward	1x Linear(11,128), 1x Linear(128,128), 1x Linear(128,1)	23809	2.03	94.8
CO2,Pulse, RIP,Snore (44)	feedforward	1x Linear(44,128), 1x Linear(128,128), 1x Linear(128,1)	22401	2.11	72.0
CO2,Pulse, RIP,Snore (44)	Multichannel CNN + feedforward	4x parallel conv1d(1,64,3),1x Conv1d(64,128,3) 1x Conv1d(128,64,3), 1x Conv1d(64,64,3),1x Linear(320,128), 1x Linear(128,64), 1x Linear(64,1)	112390	10.7	81.8
Snore (11)	feedforward	1x Linear(11,128), 1x Linear(128,128), 1x Linear(128,1)	18177	2.31	67.1

Table 8.1: *Comparison of results and latency among shallow NN models*

Features (length)	**Model**	**Learnable Weighting Parameters**				
		a1	**a2**	**a3**	**a4**	**a5**
SPO2, RIP, CO2, Pulse, Snore (55)	Multichannel CNN + feedforward	0.7394	0.2854	0.5518	0.7390	0.5167
RIP, CO2, Pulse, Snore (44)	Multichannel CNN + feedforward	--	0.7100	0.6609	0.3181	0.8531

Table 8.2: *Multichannel CNN channel weights and multisensor channels*

sentations are concatenated and passed as input to the next layers. These trainable parameters show the feature importance in each sensor stream, loosely indicating the correlation of that sensory information with the given apnea classification task. The weights are shown in Table 8.2. Each of the weights are assigned to specific sensor channels and these weights learns task-specific feature ranking.

8.5 Results & Discussion

The results are shown in Table 8.1. The latency and performance are compared throughout the models over different input features. The multichannel CNN model performs best, but it has the highest prediction latency as well. For a more fair comparison, the total number of trainable parameters in each of these models is also given. It can be clearly seen that number of parameters has a huge impact on the improvement in the performance of the CNN models. However, the shallow feedforward models have performed exceptionally well compared to their parameter size. This has happened mostly because the features are relatively simple for a neural network model to learn the covariate shift. The shallow feedforward model with 23809 parameters is almost 4.5 times smaller than the Multichannel-CNNN + feedforward model (121222), but with 55 feature dimensions, the relative accuracy difference is 1.4%.

However, the effectiveness of the multichannel-CNN + feedforward model can be observed while using five sensory information (SPO2, CO2, Pulse, RIP, Snore (55)) vs four sensory information (CO2, Pulse, RIP, Snore (44)). The feedforward, CNN, SVM, and LR models are significantly less efficient in this task than the Multichannel-CNNN + feedforward model. Compare to the SVM & LR results in Table 7.1 the Multichannel-CNNN + feedforward model achieved 20.8% improvement while using four sensor data (CO2, Pulse, RIP,Snore (44)) and 19.6% improvement single sensory 'Snore' data.

While comparing the five sensor input data (SPO2, CO2, Pulse, RIP,Snore (55)), the Multichannel-CNNN + feedforward model has 3.3% improvement (between Table 7.1 and Table 8.1). The dataset was initially partitioned using SPO2 anomaly information. So it is natural that all the classifiers performed consistantly while having the SPO2 information.

Table 8.1 shows that even with shallower layer depths, the CNN and feedforward neural network learned task-specific representations in the end-to-end regime. The four sensory information were fed with only normalisation but without any feature transformation. The SVM and LR models failed to model these raw features and performed poorly. However, the convolutional layers in the parallel convolutional blocks acted as feature transformation functions, and they learned task-specific feature representations.

The weight distribution in Table 8.2 clearly shows the sensory channel weights. The weight for the 'SPO2' channel is a1, for 'RIP' channel a2, for 'CO2' channel a3, for 'Pulse' channel a4, and for 'Snore' channel a5. While using five sensors, a1 and a4 channels got the highest weights which loosely implies that 'SPO2' and 'Pulse' were highly correlated for this apnea detection task and also 'SPO2' and 'Pulse' are positively correlated with each other. After discarding the 'SPO2' sensor, when we use the four sensory information, a2 and a5 got the highest weights, which signifies the higher correlation between 'RIP' and 'Snore'. Both 'RIP' and 'Snore' signify the respiratory tract state. Therefore, the neural network model intuitively learns the task-specific biological correlation from the data.

After examining the results, the following inferences can be drawn.

1. Shallow neural networks can be efficiently used with the children's sleep apnea data. If the dataset is balanced, the NN models learn comparatively better than SVM and LR models with low-resource data.

2. Convolutional neural networks perform task-specific feature transformation with the raw sensory information. Here removing the need for doing feature engineering separately. The latency is comparatively high than SVMs, but still, it is 9.46 milliseconds. Therefore, these can easily be deployed in real time.

3. The highest number of parameters is 120k, which can easily be implemented and

executed in low-power wearables and chips.

4. The multichannel-CNN + feedforward model acts as a mixture model that generates the highest performance. This is achieved because the parallel convolutional blocks learned varied task-specific representations, which added more generalisation to the NN training.

Chapter 9

Conclusions

In this research, a children's sleep apnea data has been explored for detecting apnea events in low-latency and low-resource statistical learning models. The dataset has been prepared with the raw sensory information from five sensors and analysed with both unsupervised and supervised methods. The decision boundaries are visualised for more in-depth analysis.

It has been observed that with single sensory data, SPO2 SVMs performed better compared to multisensory data. However, the SVM classifier's performance degraded immensely when other single sensory features were used, like Snore or Pulse. This can be linked to the sparseness of SVM's solution classifier. It samples a subset of training examples for the support vectors for learning boundary. Therefore, feature selection is important for SVM. This phenomena is further visualised with the Figure 7.1a and 7.1b.

However, for NNs, the scenario is quite different. They perform quite similarly with single sensory and multisensory data (if not better with multisensory data). The reason is that NNs process the whole feature vector over all the training samples. Thus, learning the covariate shift in the data and better generalisation. Also, neural networks are data hungry, so the NN's learned better-generalised representation with the increasing feature vector size.

The multichannel CNN network performs better with increased latency. The fundamental reason is that the parallel CNN blocks learn different representations from 5 different sensors, and then they are fused together with learnable weighted parameters. This acts like a mixture model, which causes better learning and prediction. The significance of the multichannel-CNN model can be observed while using five sensory information (SPO2, CO2, Pulse, RIP, Snore (55)) vs four sensory information (CO2, Pulse, RIP, Snore (44)). The feedforward, CNN, SVM, and LR models are significantly less efficient in this task than the Multichannel-CNN model. Due to the small size of the dataset, very big and deep neural networks with large parameter size has not been used here.

Although it can be argued that the number of parameters is significantly high in the NN models, the effect of the number of parameters in similar architecture can be analysed in future work. The children's OSA dataset used here is small, and the parameter number vs dataset size tradeoff analysis may give helpful direction for optimising the OSA detection systems.

Another possible future direction could be to explore the temporal feature selection for reducing uncertainty in these small low, resource models. One of the significant weaknesses of this work is that the supervised training is mainly dependent on unsupervised abnormality detection based on the SPO2 feature. Although SPO2 is a major indicator for an apnea event, the dependency for abnormality clustering based on a single sensor is a weakness. Future work may consider manually transcribing the apnea events or consider multisensory anomaly detection for the pseudo-labelling task. Pseudo-labelling from a pretrained apnea model can also be a pragmatic future path.

This research has presented an in-depth analysis and step-by-step problem-solving for a children's sleep study corpus. The research framework is intuitive and generic and can be used to solve similar problems in low-resource corpora.

Bibliography

[1] Terry Young, Mari Palta, Jerome Dempsey, James Skatrud, Steven Weber, and Safwan Badr. The occurrence of sleep-disordered breathing among middle-aged adults. *New England Journal of Medicine*, 328(17):1230–1235, 1993.

[2] Michael J. Sateia. International classification of sleep disorders-third edition: highlights and modifications. *Chest*, 146 5:1387–1394, 2014.

[3] Abed Nassir and Ofer Barnea. Wireless body-area network for detection of sleep disorders. *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*, pages 1–5, 2012.

[4] Sleep Apnoea Trust. Obstructive sleep apnoea research, June 2022.

[5] Ahsan Khandoker, Jayavardhana Gubbi, and Marimuthu Palaniswami. Automated scoring of obstructive sleep apnea and hypopnea events using short-term electrocardiogram recordings. *Information Technology in Biomedicine, IEEE Transactions on*, 13:1057 – 1067, 12 2009.

[6] Grégoire Surrel, Amir Aminifar, Francisco Rincón, Srinivasan Murali, and David Atienza. Online obstructive sleep apnea detection on medical wearable sensors. *IEEE Transactions on Biomedical Circuits and Systems*, 12(4):762–773, 2018.

[7] Arlene John, Barry Cardiff, and Deepu John. A 1d-cnn based deep learning

technique for sleep apnea detection in iot sensors. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[8] Jing Zhou, Xiao-ming Wu, and Wei-jie Zeng. Automatic detection of sleep apnea based on eeg detrended fluctuation analysis and support vector machine. *Journal of clinical monitoring and computing*, 29(6):767–772, 2015.

[9] Hemant Sharma and K.K. Sharma. An algorithm for sleep apnea detection from single-lead ecg using hermite basis functions. *Computers in Biology and Medicine*, 77:116–124, 2016.

[10] Mario Giovanni Terzano, Liborio Parrino, Adriano Sherieri, Ronald Chervin, Sudhansu Chokroverty, Christian Guilleminault, Max Hirshkowitz, Mark Mahowald, Harvey Moldofsky, Agostino Rosa, Robert Thomas, and Arthur Walters. Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (cap) in human sleep. *Sleep Medicine*, 2(6):537–553, 2001.

[11] Thomas Penzel, George B Moody, Roger G Mark, Ary L Goldberger, and J Hermann Peter. The apnea-ecg database. In *Computers in Cardiology 2000. Vol. 27 (Cat. 00CH37163)*, pages 255–258. IEEE, 2000.

[12] Anita Ramachandran and Anupama Karuppiah. A survey on recent advances in machine learning based sleep apnea detection systems. In *Healthcare*, volume 9, page 914. MDPI, 2021.

[13] Mayo Clinic. Sleep apnea, Jul 2020.

[14] Fabio Mendonca, Sheikh Shanawaz Mostafa, Antonio G Ravelo-Garcia, Fernando Morgado-Dias, and Thomas Penzel. A review of obstructive sleep apnea detection approaches. *IEEE journal of biomedical and health informatics*, 23(2):825–837, 2018.

[15] Nancy R Foldvary-Schaefer and Tina E Waters. Sleep-disordered breathing. *CONTINUUM: Lifelong Learning in Neurology*, 23(4):1093–1116, 2017.

[16] Tarek Gharibeh and Reena Mehra. Obstructive sleep apnea syndrome: Natural history, diagnosis, and emerging treatment options. *Nature and Science of Sleep*, 2:233–255, 09 2010.

[17] Lucia Spicuzza, Daniela Caruso, and Giuseppe Di Maria. Obstructive sleep apnoea syndrome and its management. *Therapeutic advances in chronic disease*, 6(5):273–285, 2015.

[18] Danny J. Eckert, Amy S. Jordan, Pankaj Merchia, and Atul Malhotra. Central sleep apnea. *Chest*, 131(2):595–607, 2007.

[19] Mayo Clinic. Mayo clinic discovers new type of sleep apnea, Sep 2006.

[20] Daniel Alvarez, Ana Cerezo-Hernández, Andrea Crespo, Gonzalo Gutiérrez-Tobal, Fernando Vaquerizo-Villar, Verónica Barroso-García, Fernando Moreno, C. Arroyo, Tomas Albi, Roberto Hornero, and Felix del Campo. A machine learning-based test for adult sleep apnoea screening at home using oximetry and airflow. *Scientific Reports*, 10, 03 2020.

[21] Daniel Alvarez, Roberto Hornero, Daniel Abásolo, Felix del Campo, and Carlos Zamarron. Nonlinear characteristics of blood oxygen saturation from nocturnal oximetry for obstructive sleep apnoea detection. *Physiological measurement*, 27:399–412, 05 2006.

[22] Roberto Hornero, Daniel Álvarez, Daniel E. Abásolo, Félix del Campo, and Carlos Zamarrón. Utility of approximate entropy from overnight pulse oximetry data in the diagnosis of the obstructive sleep apnea syndrome. *IEEE Transactions on Biomedical Engineering*, 54:107–113, 2007.

[23] Laiali Almazaydeh, Miad Faezipour, and Khaled Elleithy. A neural network system for detection of obstructive sleep apnea through spo2 signal features. *International Journal of Advanced Computer Science and Applications*, 3:7–11, 05 2012.

[24] Ahsan H. Khandoker, Jayavardhana Gubbi, and Marimuthu Palaniswami. Automated scoring of obstructive sleep apnea and hypopnea events using short-term electrocardiogram recordings. *IEEE Transactions on Information Technology in Biomedicine*, 13(6):1057–1067, 2009.

[25] Vega Pradana Rachim, Gang Li, and Wan-Young Chung. Sleep apnea classification using ecg-signal wavelet-pca features. *Bio-medical materials and engineering*, 24:2875–82, 09 2014.

[26] Ahnaf Rashik Hassan. Computer-aided obstructive sleep apnea detection using normal inverse gaussian parameters and adaptive boosting. *Biomedical Signal Processing and Control*, 29, 05 2016.

[27] A. F. Quiceno-Manrique, J. B. Alonso-Hernandez, C.M. Travieso-Gonzalez, M. A. Ferrer-Ballester, and G. Castellanos-Dominguez. Detection of obstructive sleep apnea in ecg recordings using time-frequency distributions and dynamic features. In *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5559–5562, 2009.

[28] J. D. Martínez-Vargas, L. M. Sepúlveda-Cano, and G. Castellanos-Dominguez. On determining available stochastic features by spectral splitting in obstructive sleep apnea detection. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 6079–6082, 2011.

[29] Jayavardhana Gubbi, Ahsan Khandoker, and Marimuthu Palaniswami. Classifica-

tion of sleep apnea types using wavelet packet analysis of short-term ecg signals. *Journal of clinical monitoring and computing*, 26:1–11, 12 2011.

[30] Gonzalo C. Gutiérrez-Tobal, Daniel Álvarez, Javier Gómez-Pilar, Félix del Campo, and Roberto Hornero. Assessment of time and frequency domain entropies to detect sleep apnoea in heart rate variability recordings from men and women. *Entropy*, 17:123–141, 2015.

[31] Lili Chen and Xi Zhang. State-based general gamma cusum for modeling heart rate variability using electrocardiography signals. *IEEE Transactions on Automation Science and Engineering*, 14(2):1160–1171, 2017.

[32] Clamarion Maier, H Wenz, and H Dickhaus. Robust detection of sleep apnea from holter ecgs joint assessment of modulations in qrs amplitude and respiratory myogram interference. *Methods of information in medicine*, 53:303–307, 07 2014.

[33] Antonio G. García, Juan L. Navarro-Mesa, Ubay Casanova-Blancas, Sofía González, Pedro Quintana, Iván Guerra-Moreno, Jose Canino, and Eduardo Hernandez-Pérez. Application of the permutation entropy over the heart rate variability for the improvement of electrocardiogram-based sleep breathing pause detection. *Entropy*, 17:914–927, 03 2015.

[34] Sofía González, Juan L. Navarro-Mesa, Gabriel Juliá-Serdá, Jan Kraemer, Niels Wessel, and Antonio G. García. Heart rate variability feature selection in the presence of sleep apnea: An expert system for the characterization and detection of the disorder. *Computers in Biology and Medicine*, 91, 10 2017.

[35] Bijoylaxmi Koley and Debangshu Dey. Automated detection of apnea and hypopnea events. In *2012 Third International Conference on Emerging Applications of Information Technology*, pages 85–88, 2012.

[36] P Caseiro, Rui Fonseca-Pinto, and A Andrade. Screening of obstructive sleep apnea using hilbert–huang decomposition of oronasal airway pressure recordings. *Medical engineering & physics*, 32(6):561–568, 2010.

[37] Gonzalo Gutiérrez-Tobal, Roberto Hornero, Daniel Alvarez, J. Victor Marcos, and Felix del Campo. Linear and nonlinear analysis of airflow recordings to help in sleep apnoea-hypopnoea syndrome diagnosis. *Physiological measurement*, 33:1261–75, 07 2012.

[38] Anirudh Thommandram, J. Mikael Eklund, and Carolyn McGregor. Detection of apnoea from respiratory time series data using clinically recognizable features and knn classification. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5013–5016, 2013.

[39] T. Rosenwein, E. Dafna, A. Tarasiuk, and Y. Zigel. Breath-by-breath detection of apneic events for osa severity estimation using non-contact audio recordings. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 7688–7691, 2015.

[40] Thanawan Praydas, Booncharoen Wongkittisuksa, and Sawit Tanthanuch. Obstructive sleep apnea severity multiclass classification using analysis of snoring sounds. In *Proc. 2nd World Congr. Elect. Eng. Comput. Syst. Sci.*, 2016.

[41] Rubén Pozo, José Luis Blanco Murillo, Luis Gómez, Eduardo Gonzalo, José Ramírez, and Doroteo Toledano. Assessment of severe apnoea through voice analysis, automatic speech, and speaker recognition techniques. *EURASIP J. Adv. Sig. Proc.*, 2009, 12 2009.

[42] Laiali Almazaydeh, Khaled M. Elleithy, Miad Faezipour, and Ahmad Abushakra. Apnea detection based on respiratory signal classification. In *EUSPN/ICTH*, 2013.

[43] Thomas Penzel and AbdelKebir Sabil. The use of tracheal sounds for the diagnosis of sleep apnoea. *Breathe*, 13(2):e37–e45, 2017.

[44] Christoph Kalkbrenner, Manuel Eichenlaub, Stefan Rüdiger, Cornelia Kropf-Sanchen, Wolfgang Rottbauer, and Rainer Brucher. Apnea and heart rate detection from tracheal body sounds for the diagnosis of sleep-related breathing disorders. *Medical Biological Engineering Computing*, 56, 08 2017.

[45] Carlos Zamarron, Francisco Gude, Francisco-Javier González-Barcala, Jose Rodriguez, and Pablo Romero. Utility of oxygen saturation and heart rate spectral analysis obtained from pulse oximetric recordings in the diagnosis of sleep apnea syndrome. *Chest*, 123:1567–76, 06 2003.

[46] Daniel Alvarez, Roberto Hornero, J Victor Marcos, Felix Del Campo, and Miguel Lopez. Spectral analysis of electroencephalogram and oximetric signals in obstructive sleep apnea diagnosis. In *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 400–403. IEEE, 2009.

[47] Baile Xie and Hlaing Minn. Real-time sleep apnea detection by classifier combination. *IEEE Transactions on information technology in biomedicine*, 16(3):469–477, 2012.

[48] Antonio G Ravelo-García, Jan F Kraemer, Juan L Navarro-Mesa, Eduardo Hernández-Pérez, Javier Navarro-Esteva, Gabriel Juliá-Serdá, Thomas Penzel, and Niels Wessel. Oxygen saturation and rr intervals feature selection for sleep apnea detection. *Entropy*, 17(5):2932–2957, 2015.

[49] Ahsan H Khandoker, Chandan K Karmakar, and Marimuthu Palaniswami. Automated recognition of patients with obstructive sleep apnoea using wavelet-based

features of electrocardiogram recordings. *Computers in Biology and Medicine*, 39(1):88–96, 2009.

[50] J Víctor Marcos, Roberto Hornero, Daniel Álvarez, Félix Del Campo, and Mateo Aboy. Automated detection of obstructive sleep apnoea syndrome from oxygen saturation recordings using linear discriminant analysis. *Medical & biological engineering & computing*, 48(9):895–902, 2010.

[51] Ainara Garde, Parastoo Dehkordi, Walter Karlen, David Wensley, Mark Ansermino, and Guy Dumont. Development of a screening tool for sleep disordered breathing in children using the phone oximeter tm. *PloS one*, 9, 11 2014.

[52] Carlos M Travieso, Jesús B Alonso, Marcos del Pozo-Baños, Jaime R Ticay-Rivas, and Karmele Lopez-de Ipiña. Automatic apnea identification by transformation of the cepstral domain. *Cognitive Computation*, 5(4):558–565, 2013.

[53] Changyue Song, Kaibo Liu, Xi Zhang, Lili Chen, and Xiaochen Xian. An obstructive sleep apnea detection approach using a discriminative hidden markov model from ecg signals. *IEEE Transactions on Biomedical Engineering*, 63(7):1532–1542, 2016.

[54] J. Victor Marcos, Roberto Hornero, Daniel Alvarez, Felix del Campo, Carlos Zamarron, and Miguel Lopez-Coronado. Utility of multilayer perceptron neural network classifiers in the diagnosis of the obstructive sleep apnoea syndrome from nocturnal oximetry. *Computer methods and programs in biomedicine*, 92:79–89, 11 2008.

[55] D. Álvarez, G. C. Gutiérrez-Tobal, F. Vaquerizo-Villar, V. Barroso-García, A. Crespo, C. A. Arroyo, F. Del Campo, and R. Hornero. Automated analysis of unattended portable oximetry by means of bayesian neural networks to assist

in the diagnosis of sleep apnea. In *2016 Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE/PAHCE)*, pages 1–4, 2016.

[56] Rahul Krishnan Pathinarupothi, Ekanath Srihari Rangan, EA Gopalakrishnan, R Vinaykumar, KP Soman, et al. Single sensor techniques for sleep apnea diagnosis using deep learning. In *2017 IEEE international conference on healthcare informatics (ICHI)*, pages 524–529. IEEE, 2017.

[57] Jakub Drzazga and Bogusław Cyganek. An lstm network for apnea and hypopnea episodes detection in respiratory signals. *Sensors*, 21(17):5858, 2021.

[58] Oliver Faust, Ragab Barika, Alex Shenfield, Edward J Ciaccio, and U Rajendra Acharya. Accurate detection of sleep apnea with long short-term memory network based on rr interval signals. *Knowledge-Based Systems*, 212:106591, 2021.

[59] Arlene John, Barry Cardiff, and Deepu John. A 1d-cnn based deep learning technique for sleep apnea detection in iot sensors. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[60] Yunxiang Bai, Luqiao Zhang, Dechao Wan, Yu Xie, and Hanghang Deng. Detection of sleep apnea syndrome by CNN based on ECG. *Journal of Physics: Conference Series*, 1757(1):012043, jan 2021.

[61] Junming Zhang, Zhen Tang, Jinfeng Gao, Li Lin, Zhiliang Liu, Haitao Wu, Fang Liu, and Ruxian Yao. Automatic detection of obstructive sleep apnea events using a deep cnn-lstm model. *Computational Intelligence and Neuroscience*, 2021, 2021.

[62] Hyun Bin Kwon, Dongyeon Son, Dongseok Lee, Heenam Yoon, Mi Hyun Lee, Yu Jin Lee, Sang Ho Choi, and Kwang Suk Park. Hybrid cnn-lstm network for

real-time apnea-hypopnea event detection based on ir-uwb radar. *IEEE Access*, 10:17556–17564, 2022.

[63] Xiaolong Liang, Xing Qiao, and Yongtao Li. Obstructive sleep apnea detection using combination of cnn and lstm techniques. In *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pages 1733–1736, 2019.

[64] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[65] Zhiguo Ding and Minrui Fei. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. *IFAC Proceedings Volumes*, 46(20):12–17, 2013. 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013.

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

[67] Christopher M Bishop. *Pattern recognition and machine learning*, volume 4. Springer.

[68] Lindsay I Smith. A tutorial on principal components analysis. 2002.

[69] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[70] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[71] Wikipedia. Support-vector machine — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Support-vector%20machineoldid=1107938510, 2022. [Online; accessed 12-September-2022].

[72] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[73] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[74] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory.* Psychology Press, 1949.

[75] Fahlman and Hinton. Connectionist architectures for artificial intelligence. *Computer*, 20(1):100–109, 1987.

[76] Geoffrey E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1):185–234, 1989.

[77] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[78] Sridhar Narayan. The generalized sigmoid activation function: Competitive supervised learning. *Information Sciences*, 99(1):69–82, 1997.

[79] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[80] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[81] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[82] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[83] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[84] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.

[85] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[86] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010.

[87] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolution Network. *ICML Deep Learning Workshop*, pages 1–5, 2015.

[88] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and

Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[89] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Appendices

Appendix

```
  ### initial anomaly detection and visualisation

  import os

import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import matplotlib.pyplot as plt
from datetime import datetime
from datetime import timedelta
from pandas.plotting import register_matplotlib_converters
from mpl_toolkits.mplot3d import Axes3D

from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
register_matplotlib_converters()
from time import time
import seaborn as sns
sns.set(style="whitegrid")
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.covariance import EllipticEnvelope
from sklearn import preprocessing


def signal_visualisation(samples, time, title):
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
    scaler = scaler.fit(np.array(samples).reshape(-1, 1))
    normalised_samples = scaler.transform(np.array(samples).reshape(-1, 1))
    normalised_samples = normalised_samples[:-1000]
    plt.plot(normalised_samples, ls='dotted', c='red', lw=1)
    # plt.plot(samples, ls='dotted', c='red', lw=1)
    plt.title(title)
    plt.ylabel("Normalised signal")
    plt.xlabel("time signature")
    plt.show()


def sample_extraction(filepath):
    from scipy import stats

    samples = []
    time = []
    sample_rate = 1
    with open(filepath,"r") as f:
        for _ in range(2):
            next(f)
        for line in f:
```

```
        sample_rate = int(line.strip().split(":")[1].strip())
        break
    for _ in range(4):
        next(f)
    counter = 0
    mean = 0
    sensor = filepath.split("/")[-1]
    print("sample rate in {} is {}".format(sensor,sample_rate))
    time_prev = 0
    for line in f:
        timestamp = line.strip().split(";")[0].split(",")[0]
        if counter == sample_rate:
            mean = float(mean/sample_rate)
            counter = 0
            time.append(time_prev)
            samples.append(mean)
            mean = 0
        # print(line.strip().split(";")[1].strip())
        mean = mean + float(line.strip().split(";")[1].strip())
        counter = counter + 1
        time_prev = timestamp

print(len(samples),len(time))
pivot = 0
for i in range(len(time)):
    if "06:56:59" in time[i]:
        break
    else:
        pivot+=1
samples = samples[:pivot]
time = time[:pivot]
```

```
    return samples,time


def Isolation_forest(sample,time,sensor,plot=True):
    from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
    from sklearn.covariance import EllipticEnvelope
    from sklearn.ensemble import IsolationForest

    original_sample = sample
    outliers_fraction = float(.003)
    plt.rc('figure', figsize=(12, 6))
    plt.rc('font', size=15)
    original_sample = np.array(original_sample)
    sample = np.array(sample)
    scaler = StandardScaler()
    # np_scaled = scaler.fit_transform(sample.reshape(-1, 1))
    model = IsolationForest(contamination=outliers_fraction)
    model.fit(sample.reshape(-1, 1))
    anomaly = model.predict(sample.reshape(-1,1))
    # model.fit(np_scaled)
    # anomaly = model.predict(np_scaled)
    # for i,j in zip(sample,time):
    #     print(i,j)
    print(anomaly.shape)
    fig, ax = plt.subplots(figsize=(10, 6))
    anomaly_points = []
    anomaly_times = []
    normal_points = []
    nomral_times = []
    for i in range(anomaly.shape[0]):
        if anomaly[i]==-1:
            # print(i,anomaly[i],sample[i],original_sample[i])
            anomaly_points.append(original_sample[i])
            anomaly_times.append(i)
        else:
            normal_points.append(original_sample[i])
            nomral_times.append(i)

    if plot:
        ax.plot(nomral_times, normal_points, color='green', label='regular')
        # ax.plot(anomaly_times, anomaly_points, color='red', label='anomaly')
        # ax.scatter(nomral_times,normal_points,  color='green', label='regular')

        ax.scatter(anomaly_times, anomaly_points, color='red', label='irregular')
        plt.legend(loc="lower left")
        title = sensor.split("/")[-1]
        plt.title(title)
        plt.show()

    return anomaly



def multi_sensor_visualisation(rootdir,sensornames):
    filelist = os.listdir(rootdir)
    for file in filelist:
        counter = 0
        for i in sensornames:
            if i.split(" ")[0] in file.split(" ")[0]:
                filepath = os.path.join(rootdir,file)
                print(i)
                sample, time = sample_extraction(filepath)
                print(i,len(sample),len(time))
                Isolation_forest(sample,time,i)
                counter+=1


def filter_noise(sample, time):
    new_sample = []
    new_time = []
    removed_time = []
    for i in range(len(sample)):
        if sample[i] <= 100 and sample[i] > 0:
            new_sample.append(sample[i])
            new_time.append(time[i])
        else:
            removed_time.append(time[i])
    return new_sample,new_time, removed_time


def remove_indexes(sample, time, removed_time):
    new_sample = []
    new_time = []
    for i in range(len(time)):
        if time[i] not in removed_time:
            new_sample.append(sample[i])
            new_time.append(time[i])
    return new_sample,new_time


def multi_sensor_fusion(rootdir, primary_sensorname, auxilary_sensorname, save=False):
    import itertools
    import random
    import pickle


    filelist = os.listdir(rootdir)
    for file in filelist:
        if primary_sensorname in file:
            primary_sensor = os.path.join(rootdir,file)
            break
    primary_sensor_sample,primary_sensor_time = sample_extraction(primary_sensor)
    # signal_visualisation(primary_sensor_sample, primary_sensor_time, "SPO2 Normalised before noise
    # exit(-1)
    primary_sensor_sample, primary_sensor_time, removed_time = filter_noise(primary_sensor_sample,pri
    # print(removed_time)
    # signal_visualisation(primary_sensor_sample, primary_sensor_time,
    "SPO2 Normalised after noise removal")
    # exit(-1)
    primary_anomaly = Isolation_forest(primary_sensor_sample,primary_sensor_time,
                                       primary_sensor,plot=True)
    # exit(-1)
    fused_features_anomaly = {}
    fused_features_normal = {}
    primary_feature_samples_num = len(primary_anomaly)
    context_size = 5

    anomaly_time = []
    normal_time = []
    for i in range(len(primary_anomaly)):
        if primary_anomaly[i] == -1:
            if (i-context_size-1 > 0) and (i+context_size+1 < primary_feature_samples_num):
                back_context = primary_sensor_sample[i-context_size:i]
                front_context = primary_sensor_sample[i+1:i+context_size+1]
                # print(len(back_context),len(front_context))
                # print(primary_sensor_sample[i-5:i+5])
                # print(back_context,primary_sensor_sample[i],front_context)
                # print(primary_sensor_time[i])
                feature = list(itertools.chain(back_context,[primary_sensor_sample[i]],front_context)
                fused_features_anomaly[primary_sensor_time[i]] = feature
                anomaly_time.append(primary_sensor_time[i])

    # print(anomaly_time)
    # print(fused_features_anomaly)
    anomaly_features_number = len(fused_features_anomaly)
    normal_samples_number = anomaly_features_number + 5
    normal_samples_index = random.sample(range(10,len(primary_anomaly)-context_size-1),normal_samples

    counter = 0
    for i in normal_samples_index:
        if primary_anomaly[i]==1:
            back_context = primary_sensor_sample[i - context_size:i]
            front_context = primary_sensor_sample[i + 1:i + context_size + 1]
            # print(len(back_context),len(front_context))
            # print(primary_sensor_sample[i-5:i+5])
            # print(back_context,primary_sensor_sample[i],front_context)
            # print(primary_sensor_time[i])
            feature = list(itertools.chain(back_context,
            [primary_sensor_sample[i]], front_context))
            fused_features_normal[primary_sensor_time[i]] = feature
            normal_time.append(primary_sensor_time[i])
            counter+=1
            if counter > normal_samples_number:
                break

    # print(fused_features_normal)
```

```
filelist = os.listdir(rootdir)
for file in filelist:
    counter = 0
    for i in auxilary_sensorname:
        if i.split(" ")[0] in file.split(" ")[0]:
            print(file)
            filepath = os.path.join(rootdir,file)
            # print(i)
            sample, time = sample_extraction(filepath)
            sample, time = remove_indexes(sample, time, removed_time)
            print(i,len(sample),len(time))
            # Isolation_forest(sample,time,i)
            for i in range(len(primary_anomaly)):
                if primary_anomaly[i] == -1:
                    if (i - context_size - 1 > 0) and (i + context_size + 1 < primary_feature_sam
                        back_context = sample[i - context_size:i]
                        front_context = sample[i + 1:i + context_size + 1]
                        # print(len(back_context),len(front_context))
                        # print(sample[i-5:i+5])
                        # print(back_context,sample[i],front_context)
                        # print(time[i])
                        feature = list(itertools.chain(back_context, [sample[i]], front_context))
                        if time[i] in fused_features_anomaly:
                            prev_features = fused_features_anomaly[time[i]]
                            new_feature_list = list(itertools.chain(prev_features, feature))
                            fused_features_anomaly[time[i]] = new_feature_list
                        else:
                            print(time[i]," is not present")
                            exit(-1)
            # print(fused_features_anomaly)
            counter = 0
            for i in normal_samples_index:
                if primary_anomaly[i] == 1:
                    back_context = sample[i - context_size:i]
                    front_context = sample[i + 1:i + context_size + 1]
                    # print(len(back_context),len(front_context))
                    # print(primary_sensor_sample[i-5:i+5])
                    # print(back_context,primary_sensor_sample[i],front_context)
                    # print(primary_sensor_time[i])
                    feature = list(itertools.chain(back_context,
                    [sample[i]], front_context))
                    if time[i] in fused_features_normal:
                        prev_features = fused_features_normal[time[i]]
                        new_feature_list = list(itertools.chain(prev_features, feature))
                        fused_features_normal[time[i]] = new_feature_list
                    else:
                        print(time[i], " is not present")
                        exit(-1)
                    counter += 1
                    if counter > normal_samples_number:
                        break
            # print(fused_features_normal)
            counter+=1
print(len(fused_features_anomaly))
print(len(fused_features_normal))
anomaly_path = "save/"+rootdir.split("/")[-3]+"_fused_features_anomaly.pkl"
normal_path = "save/"+rootdir.split("/")[-3]+"_fused_features_normal.pkl"
if save:
    with open(anomaly_path, ’wb’) as f:
        pickle.dump(fused_features_anomaly, f)
        with open(normal_path, 'wb') as f:
            pickle.dump(fused_features_normal, f)



if __name__ == '__main__':
    # filepath = "/home/saba/Documents/sba/data/ChildData/PI001/Raw data/SPO2 - PI001.txt"
    # sample, time = sample_extraction(filepath)
    # # print(sample)
    # Isolation_forest(sample,time)
    rootdir = "/home/saba/Documents/sba/data/ChildData/PI006/Raw data/"
    auxilary_sensorname = ["CO2","Pulse","Sum RIPS","Snore"]
    primary_sensorname = "SPO2"
    multi_sensor_fusion(rootdir, primary_sensorname, auxilary_sensorname, save=False)


#### create feature files

import numpy as np
import pandas as pd
import os
import pickle

pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

if __name__ == '__main__':
    feature_files_dir = "/home/saba/PycharmProjects/sba/save"
    files = os.listdir(feature_files_dir)
    df = pd.DataFrame(columns=["id", "time", "feature", "label"])
    counter = 0
    for file in files:
        print(file)
        uid = file.split(".")[0].split("_")
        # print(uid[0],uid[3])
        with open(os.path.join(feature_files_dir,file),"rb") as f:
            dictionary = pickle.load(f)
            # print(len(dictionary))
            for key, value in dictionary.items():
                # print(key,value)
                value = [round(i,2) for i in value]
                value = np.asarray(value)
                unique_id = uid[0]+"_"+uid[3]+"_"+key
                new_row = {'id':unique_id, "time":key, "feature":value, "label":uid[3]}
                df.loc[counter] = new_row
                counter+=1
    print(df)
    save_ = feature_files_dir + "/features_combined"
    df.to_pickle(save_)
    df = df.sample(frac=1).reset_index(drop=True)
    print(df)
    save_ = feature_files_dir + "/features_shuffle"
    df.to_pickle(save_)



#### visualisation of PCA cluster and PCA analysis



import numpy as np
from sklearn.preprocessing import StandardScaler
```

```
import pickle
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline



def load_dic(path):
    df = pd.read_pickle(path)
    return df



if __name__ == '__main__':
    dic1 = "/home/saba/PycharmProjects/sba/save/features_shuffle"
    df1 = load_dic(dic1)

    x_ = df1.loc[:, ['feature']].values
    y = df1.loc[:, ['label']].values

    x = []
    for i in range(x_.shape[0]):
        x.append(x_[i][0])

    # exit(-1)
    x = StandardScaler().fit_transform(x)

    # print(x.shape)
    #
    # df1.to_csv(dic1+"save.csv", sep='\t', encoding='utf-8')
    # exit(-1)
    x = x[:,:11]
    pca = PCA(n_components=2)

    principal_components = pca.fit_transform(x)

    principalDataframe = pd.DataFrame(data=principal_components,
    columns=['PC1','PC2'])
    finalDf = pd.concat([principalDataframe, df1[['label']]], axis=1)

    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(1, 1, 1)
    ax.set_xlabel('Principal Component 1', fontsize=15)
    ax.set_ylabel('Principal Component 2', fontsize=15)
    ax.set_title('2 component PCA Cluster', fontsize=20)
    targets = ['anomaly', 'normal']
    colors = ['r', 'g']
    for target, color in zip(targets, colors):
        indicesToKeep = finalDf['label'] == target
        ax.scatter(finalDf.loc[indicesToKeep, 'PC1']
                   , finalDf.loc[indicesToKeep, 'PC2']
                   , c=color
                   , s=50)
    ax.legend(targets)
    ax.grid()
    fig.show()


######## t-SNE visualisation and clustering

from __future__ import print_function
import time
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

import numpy as np
from sklearn.preprocessing import StandardScaler
import pickle
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt




def load_dic(path):
    df = pd.read_pickle(path)
    return df



if __name__ == '__main__':
    dic1 = "/home/saba/PycharmProjects/sba/save/features_shuffle"
    df1 = load_dic(dic1)

    x_ = df1.loc[:, ['feature']].values
    y_ = df1.loc[:, ['label']].values

    y = []
    for i in range(y_.shape[0]):
        if y_[i] =="normal":
```

```
        y.append(1)
    elif y_[i] == "anomaly":
        y.append(0)
    else:
        exit(-1)
# exit(-1)
# print(y)

# exit(-1)

x = []
for i in range(x_.shape[0]):
    x.append(x_[i][0])

# exit(-1)
x = StandardScaler().fit_transform(x)
# x = x[:,11:]

pca = PCA(n_components=3)
pca_result = pca.fit_transform(x)

time_start = time.time()
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300) #40
tsne_results = tsne.fit_transform(x)
print('t-SNE done! Time elapsed: {} seconds'.format(time.time() - time_start))

# exit(-1)
df_subset = {}

df_subset['pca-one'] = pca_result[:, 0]
df_subset['pca-two'] = pca_result[:, 1]
```

```
df_subset['pca-three'] = pca_result[:, 2]

df_subset['tsne-2d-one'] = tsne_results[:, 0]
df_subset['tsne-2d-two'] = tsne_results[:, 1]

df_subset['y'] = y
plt.figure(figsize=(16, 10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="y",
    palette=sns.color_palette("hls", 2),
    data=df_subset,
    legend="full",
    alpha=0.9
)

plt.figure(figsize=(16, 7))
ax1 = plt.subplot(1, 2, 1)
sns.scatterplot(
    x="pca-one", y="pca-two",
    hue="y",
    palette=sns.color_palette("hls", 2),
    data=df_subset,
    legend="full",
    alpha=0.3,
    ax=ax1
)
ax2 = plt.subplot(1, 2, 2)
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="y",
        palette=sns.color_palette("hls", 2),
        data=df_subset,
        legend="full",
        alpha=0.3,
        ax=ax2
    )

    plt.show()

###### SVM and LR classification and latency



import numpy as np
from sklearn.preprocessing import StandardScaler
import pickle
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import time
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

import numpy as np
from sklearn.preprocessing import StandardScaler
import pickle
```

```
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Sklearn modules & classes
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
from sklearn import metrics


def load_dic(path):
    df = pd.read_pickle(path)
    return df


if __name__ == '__main__':
    dic1 = "/home/saba/PycharmProjects/sba/save/features_shuffle"
    df1 = load_dic(dic1)
    x_ = df1.loc[:, ['feature']].values
    y_ = df1.loc[:, ['label']].values

    y = []
    for i in range(y_.shape[0]):
        if y_[i] == "normal":
            y.append(1)
        elif y_[i] == "anomaly":
            y.append(0)
        else:
            exit(-1)

    x = []
    for i in range(x_.shape[0]):
        x.append(x_[i][0])

    # exit(-1)
    x = StandardScaler().fit_transform(x)
    x = x[:,:11]
    length = len(x)
    print(length)
    train_num = int(length*0.85)

    x_train = x[:train_num]
    y_train = y[:train_num]

    x_test = x[train_num:]
    y_test = y[train_num:]
    print(len(x_train),len(x_test),len(x))
    print(len(x_train[0]),len(x_test[0]),len(x[0]))

    # Instantiate the Support Vector Classifier (SVC)
    svc = SVC(C=1.0, random_state=1, kernel='linear')

    # Fit the model
    svc.fit(x_train, y_train)

    time_start = time.time()
```

```
# Make the predictions
y_predict = svc.predict(x_test)
print('SVM done! Time elapsed: {} seconds'.format(time.time() - time_start))
# Measure the performance
print("Accuracy score %.3f" % metrics.accuracy_score(y_test, y_predict))
print(metrics.classification_report(y_test, y_predict))

print("Logistic Regression")

# Initialize and fit the Model
model = LogisticRegression()
model.fit(x_train, y_train)

time_start = time.time()
# Make prediction on the test set
pred = model.predict(x_test)

print('Logistic regression done! Time elapsed:
{} seconds'.format(time.time() - time_start))

# calculating precision and reall
precision = metrics.precision_score(y_test, pred)
recall = metrics.recall_score(y_test, pred)
print("Accuracy score %.3f" % metrics.accuracy_score(y_test, pred))
print('Precision: ', precision)
print('Recall: ', recall)
print(metrics.classification_report(y_test, pred))
# Plotting Precision-Recall Curve
# disp = metrics.plot_precision_recall_curve(model, x_test, y_test)
# disp.plot()
# plt.show()
```

```
#####  SVM boundary visualisation

import numpy as np
from sklearn.preprocessing import StandardScaler
import pickle
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import time
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

import numpy as np
from sklearn.preprocessing import StandardScaler
import pickle
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Sklearn modules & classes
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
from sklearn import metrics


def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


def load_dic(path):
    df = pd.read_pickle(path)
    return df


def visualise_style1(x,y,svc):
    fig, ax = plt.subplots()
    # title for the plots
    title = ('Decision surface') #of linear SVC
    # Set-up grid for plotting.
    X0, X1 = x[:, 0], x[:, 1]
    xx, yy = make_meshgrid(X0, X1)

    plot_contours(ax, svc, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_ylabel('y')
    ax.set_xlabel('x')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    ax.legend()
    plt.show()

def visualise_style2(x,y,svc):
    ax = plt.gca()
    plt.scatter(x[:, 0], x[:, 1], c=y, s=50, cmap='autumn')
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = svc.decision_function(xy).reshape(XX.shape)

    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])
    ax.scatter(svc.support_vectors_[:, 0], svc.support_vectors_[:, 1], s=100,
               linewidth=1, facecolors='none', edgecolors='k')
    ax.set_title("kernal boundary and overlap") #SVC
    plt.show()


if __name__ == '__main__':
    dic1 = "/home/saba/PycharmProjects/sba/save/features_shuffle"
    df1 = load_dic(dic1)
    x_ = df1.loc[:, ['feature']].values
    y_ = df1.loc[:, ['label']].values

    y = []
    for i in range(y_.shape[0]):
        if y_[i] == "normal":
            y.append(1)
        elif y_[i] == "anomaly":
            y.append(0)
        else:
            exit(-1)

    x = []
    for i in range(x_.shape[0]):
        x.append(x_[i][0])

    # exit(-1)
    # x = x[:,11:22]

    x = StandardScaler().fit_transform(x)

    print(x.shape)
```

```
x = x[:,11:]
# exit(-1)
pca = PCA(n_components=2)
pca_result = pca.fit_transform(x)

x = pca_result

print(x.shape)

# exit(-1)

length = len(x)
print(length)
train_num = int(length*0.85)



x_train = x[:train_num]
y_train = y[:train_num]

x_test = x[train_num:]
y_test = y[train_num:]
print(len(x_train),len(x_test),len(x))
print(len(x_train[0]),len(x_test[0]),len(x[0]))

# Instantiate the Support Vector Classifier (SVC)
svc = SVC(C=1.0, random_state=1, kernel='linear')

# Fit the model
svc.fit(x_train, y_train)

# Make the predictions
```

```
    y_predict = svc.predict(x_test)

    # Measure the performance
    print("Accuracy score %.3f" % metrics.accuracy_score(y_test, y_predict))
    print(metrics.classification_report(y_test, y_predict))
    visualise_style1(x_test, y_test, svc)
    visualise_style2(x_test,y_test,svc)

####### neural network models and training loop and prediction


# first neural network with keras tutorial
import time

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
import torch
import numpy as np
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch import nn
import matplotlib.pyplot as plt
from torch import optim
import itertools
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

```
# load the dataset


def load_dic(path):
    df = pd.read_pickle(path)
    return df


def dataprep(datapath):
    df1 = load_dic(datapath)
    x_ = df1.loc[:, ['feature']].values
    y_ = df1.loc[:, ['label']].values

    y = []
    for i in range(y_.shape[0]):
        if y_[i] == "normal":
            y.append(1)
        elif y_[i] == "anomaly":
            y.append(0)
        else:
            exit(-1)

    x = []
    for i in range(x_.shape[0]):
        x.append(x_[i][0])

    # exit(-1)
    x = StandardScaler().fit_transform(x)
    # x = x[:,:11]
    # split into input (X) and output (y) variables

    length = len(x)
    print(length)

    train_num = int(length * 0.85)


    x_train = x[:train_num]

    y_train = y[:train_num]


    x_test = x[train_num:]

    y_test = y[train_num:]


    print(len(x_train), len(x_test), len(x))

    print(len(x_train[0]), len(x_test[0]), len(x[0]))


    return x_train,y_train,x_test,y_test


# Convert data to torch tensors

class Data(Dataset):

    def __init__(self, X, y):

        self.X = torch.from_numpy(X.astype(np.float32))

        self.y = torch.from_numpy(y.astype(np.float32))

        self.len = self.X.shape[0]


    def __getitem__(self, index):

        return self.X[index], self.y[index]


    def __len__(self):

        return self.len




class NeuralNetwork_simple(nn.Module):

    def __init__(self, input_dim, hidden_dim, output_dim):

        super(NeuralNetwork_simple, self).__init__()
        self.layer_1 = nn.Linear(input_dim, hidden_dim)
        nn.init.kaiming_uniform_(self.layer_1.weight, nonlinearity="relu")
        self.layer_2 = nn.Linear(hidden_dim, hidden_dim)
        nn.init.kaiming_uniform_(self.layer_2.weight, nonlinearity="relu")
        self.layer_3 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = torch.relu(self.layer_1(x))
        x = torch.relu(self.layer_2(x))
        x = torch.sigmoid(self.layer_3(x))

        return x

class CNN_simple(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(1, 64, 3)
        self.pool = nn.MaxPool1d(2, 2)
        self.conv2 = nn.Conv1d(64, 128, 3)
        self.conv3 = nn.Conv1d(128, 64, 3)
        self.fc1 = nn.Linear(64 * 5 , 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        # print(x.shape)
        x = x.unsqueeze(1)
        # print(x.shape)
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        print(x.shape)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x


class CNN_simple2(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(1, 64, 8)
        self.pool = nn.MaxPool1d(2, 2)
        self.conv2 = nn.Conv1d(64, 128, 5)
        self.conv3 = nn.Conv1d(128, 64, 5)
        self.fc1 = nn.Linear(64 * 3 , 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        # print(x.shape)
        x = x.unsqueeze(1)
        # print(x.shape)
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        # print(x.shape)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x


class CNN_simple_11dim(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(1, 64, 5)
        self.pool = nn.MaxPool1d(2, 1)
        self.conv2 = nn.Conv1d(64, 64, 3)
        self.fc1 = nn.Linear(64 * 3 , 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        # print(x.shape)
        x = x.unsqueeze(1)
        # print(x.shape)
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # x = self.pool(F.relu(self.conv3(x)))
        # print(x.shape)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x

class CNN_simple3(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(1, 64, 5)
        self.conv2 = nn.Conv1d(1, 64, 5)
        self.conv3 = nn.Conv1d(1, 64, 5)
        self.conv4 = nn.Conv1d(1, 64, 5)
        self.conv5 = nn.Conv1d(1, 64, 5)
        self.x1 = nn.Parameter(torch.rand(1))
        self.x2 = nn.Parameter(torch.rand(1))
        self.x3 = nn.Parameter(torch.rand(1))
        self.x4 = nn.Parameter(torch.rand(1))
        self.x5 = nn.Parameter(torch.rand(1))

        self.convx1 = nn.Conv1d(64, 128, 5)
        self.convx2 = nn.Conv1d(128, 64, 5)
        self.convx3 = nn.Conv1d(64, 64, 5)
        self.pool = nn.MaxPool1d(2, 2)

        self.fc1 = nn.Linear(64 * 1 , 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        # print(x.shape)
        x = x.unsqueeze(1)
        x1 = x[:,:,:11]
        x2 = x[:, :, 11:22]
        x3 = x[:, :, 22:33]
        x4 = x[:, :, 33:44]
        x5 = x[:, :, 44:55]
        # print(x.shape)
        # x1 = torch.mul(F.relu(self.conv1(x1)),self.x1)
        x1 = torch.mul(F.relu(self.conv1(x1)), self.x1)
        x2 = torch.mul(F.relu(self.conv2(x2)),self.x2)
        x3 = torch.mul(F.relu(self.conv3(x3)),self.x3)
        x4 = torch.mul(F.relu(self.conv4(x4)),self.x4)
        x5 = torch.mul(F.relu(self.conv5(x5)),self.x5)


        x = torch.cat((x1,x2,x3,x4,x5),dim=2)
        print(self.x1,self.x2,self.x3,self.x4,self.x5)
        # print(x.shape)
        x = self.pool(F.relu(self.convx1(x)))
        x = self.pool(F.relu(self.convx2(x)))
        x = F.relu(self.convx3(x))
        # print(x.shape)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x


class CNN_simple_sensors(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(1, 64, 3)
        self.conv2 = nn.Conv1d(1, 64, 3)
        self.conv3 = nn.Conv1d(1, 64, 3)
        self.conv4 = nn.Conv1d(1, 64, 3)
        self.conv5 = nn.Conv1d(1, 64, 3)
        self.x1 = nn.Parameter(torch.rand(1))
        self.x2 = nn.Parameter(torch.rand(1))
        self.x3 = nn.Parameter(torch.rand(1))
        self.x4 = nn.Parameter(torch.rand(1))
        self.x5 = nn.Parameter(torch.rand(1))
        self.convx1 = nn.Conv1d(64, 128, 3)
        self.convx2 = nn.Conv1d(128, 64, 3)
        self.convx3 = nn.Conv1d(64, 64, 3)
        self.pool = nn.MaxPool1d(2, 2)

        self.fc1 = nn.Linear(64 * 5 , 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        # print(x.shape)
        x = x.unsqueeze(1)
        # x1 = x[:,:,:11]
        x2 = x[:, :, 11:22]
        x3 = x[:, :, 22:33]
        x4 = x[:, :, 33:44]
        x5 = x[:, :, 44:55]
        # print(x.shape)
        # x1 = torch.mul(F.relu(self.conv1(x1)),self.x1)
        # x1 = torch.mul(F.relu(self.conv1(x1)), self.x1)
        x2 = torch.mul(F.relu(self.conv2(x2)),self.x2)
        x3 = torch.mul(F.relu(self.conv3(x3)),self.x3)
        x4 = torch.mul(F.relu(self.conv4(x4)),self.x4)
        x5 = torch.mul(F.relu(self.conv5(x5)),self.x5)


        x = torch.cat((x2,x3,x4,x5),dim=2)
        print(self.x2,self.x3,self.x4,self.x5)
        # print(x.shape)
        x = self.pool(F.relu(self.convx1(x)))
        x = self.pool(F.relu(self.convx2(x)))

        x = F.relu(self.convx3(x))

        # print(x.shape)

        x = torch.flatten(x, 1) # flatten all dimensions except batch

        x = F.relu(self.fc1(x))

        x = F.relu(self.fc2(x))

        x = torch.sigmoid(self.fc3(x))

        return x



class NeuralNetwork_four_sensor(nn.Module):

    def __init__(self, input_dim, hidden_dim, output_dim):

        super(NeuralNetwork_four_sensor, self).__init__()

        self.layer_1 = nn.Linear(input_dim, hidden_dim)

        nn.init.kaiming_uniform_(self.layer_1.weight, nonlinearity="relu")

        self.layer_2 = nn.Linear(hidden_dim, hidden_dim)

        nn.init.kaiming_uniform_(self.layer_2.weight, nonlinearity="relu")

        self.layer_3 = nn.Linear(hidden_dim, output_dim)


    def forward(self, x):

        # x = x.unsqueeze(1)

        # x1 = x[:,:,:11]

        print(x.shape)

        x = x[:,44:55]

        x = torch.relu(self.layer_1(x))

        x = torch.relu(self.layer_2(x))

        x = torch.sigmoid(self.layer_3(x))


        return x
```

```
if __name__ == '__main__':
    dic1 = "/home/saba/PycharmProjects/sba/save/features_shuffle"
    X_train,y_train,X_test,y_test = dataprep(dic1)
    X_train = np.asarray(X_train)
    y_train = np.asarray(y_train)

    X_test = np.asarray(X_test)
    y_test = np.asarray(y_test)

    batch_size = 100
    # Instantiate training and test data
    train_data = Data(X_train, y_train)
    train_dataloader = DataLoader(dataset=train_data,
    batch_size=batch_size, shuffle=True)

    test_data = Data(X_test, y_test)
    test_dataloader = DataLoader(dataset=test_data,
    batch_size=batch_size, shuffle=True)

    input_dim = 11
    hidden_dim = 128
    output_dim = 1

    # model = NeuralNetwork_simple(input_dim, hidden_dim, output_dim)
    # model = NeuralNetwork_four_sensor(input_dim, hidden_dim, output_dim)

    # print(model)
    model = CNN_simple3()
    # model = CNN_simple_sensors()
    print(model)
```

```
learning_rate = 0.001


loss_fn = nn.BCELoss()
print("params:",sum(p.numel() for p in model.parameters() if p.requires_grad))
# optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)


optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
num_epochs = 100
loss_values = []


for epoch in range(num_epochs):
    for X, y in train_dataloader:
        # zero the parameter gradients
        optimizer.zero_grad()


        # forward + backward + optimize
        pred = model(X)
        # print(pred.shape)
        # print(y.unsqueeze(-1))
        loss = loss_fn(pred, y.unsqueeze(-1))
        loss_values.append(loss.item())
        loss.backward()
        optimizer.step()

print("Training Complete")


"""
Training Complete
"""


# step = np.linspace(0, 100, 10500)
```

```
#
# fig, ax = plt.subplots(figsize=(8, 5))
# plt.plot(step, np.array(loss_values))
# plt.title("Step-wise Loss")
# plt.xlabel("Epochs")
# plt.ylabel("Loss")
# plt.show()

"""
We’re not training so we don’t need to calculate the gradients for our outputs
"""
y_pred = []
y_hyp = []
total = 0
correct = 0
time_start = time.time()

with torch.no_grad():
    for X, y in test_dataloader:
        outputs = model(X)
        predicted = np.where(outputs < 0.5, 0, 1)
        predicted = list(itertools.chain(*predicted))
        y_pred.append(predicted)
        y_hyp.append(y)
        total += y.size(0)
        correct += (predicted == y.numpy()).sum().item()

print(’NN done! Time elapsed: {} seconds’.format(time.time() - time_start))
print("Accuracy of the network on the {}
test instances is:{}".format(total,(100 * correct // total)))
```

```
    y_pred = list(itertools.chain(*y_pred))
    y_hyp = list(itertools.chain(*y_hyp))

    print(classification_report(y_hyp, y_pred))

    cf_matrix = confusion_matrix(y_hyp, y_pred)

    plt.subplots(figsize=(8, 5))

    sns.heatmap(cf_matrix, annot=True, cbar=False, fmt="g")

    plt.show()

#### data parrsing

import os
import pywt
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing



def file_parser(directory_path):
    filelist = os.listdir(directory_path)
    for file in filelist:
        filepath = os.path.join(directory_path,file)
        with open(filepath, "r" ) as f1:
            count = 0
            for _ in range(5):
                next(f1)
            for line in f1:
                count = count+1
            print(file, count)



def sample_extraction(filepath):
    from scipy import stats

    samples = []
    time = []
    sample_rate = 1
    with open(filepath,"r") as f:
        for _ in range(2):
            next(f)
        for line in f:
            sample_rate = int(line.strip().split(":")[1].strip())
            break
        for _ in range(4):
            next(f)
        counter = 0
        mean = 0
        sensor = filepath.split("/")[-1]
        print("sample rate in {} is {}".format(sensor,sample_rate))
        time_prev = 0
        for line in f:
            timestamp = line.strip().split(";")[0].split(",")[0]
            if counter == sample_rate:
                mean = float(mean/sample_rate)
                counter = 0
                time.append(time_prev)
                samples.append(mean)
                mean = 0
            # print(line.strip().split(";")[1].strip())
            mean = mean + float(line.strip().split(";")[1].strip())
            counter = counter + 1
            time_prev = timestamp

    print(len(samples),len(time))
    pivot = 0
    for i in range(len(time)):
        if "06:56:59" in time[i]:
            break
        else:
            pivot+=1
    samples = samples[:pivot]
    time = time[:pivot]
    return samples,time

def wavelet_coefficient_generation(samples,time,mode, title):
    # normalised_samples = preprocessing.normalize(np.array(samples).reshape(-1,1))
    scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
    scaler = scaler.fit(np.array(samples).reshape(-1,1))
    normalised_samples = scaler.transform(np.array(samples).reshape(-1,1))
    # print(normalised_samples.shape)
    cA, cD = pywt.dwt(normalised_samples,'db2')
    coefficients = (cA,cD)
    for i, ci in enumerate(coefficients):
        plt.imshow(ci.reshape(1, -1),
        extent=[0, normalised_samples.shape[0], i + 0.5, i + 1.5],
        cmap='inferno', aspect='auto',
                    interpolation='nearest')
    plt.plot(normalised_samples, ls='dotted', c='red', lw=1)
    plt.ylim(0.5, len(coefficients) + 0.5)
    # set the y-lim to include the six horizontal images
    # optionally relabel the y-axis (the given labeling is 1,2,3,...)
    plt.yticks(range(1, len(coefficients) + 1), ['cA', 'cD'])
    plt.title(title)
    # plt.savefig(path)
    plt.show()

def signal_visualisation(samples, time, title):
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
    scaler = scaler.fit(np.array(samples).reshape(-1, 1))
    normalised_samples = scaler.transform(np.array(samples).reshape(-1, 1))
    normalised_samples = normalised_samples[200:-1000]
    plt.plot(normalised_samples, ls='dotted', c='red', lw=1)
    plt.title(title)
    plt.ylabel("Normalised signal")
    plt.xlabel("time signature")
    plt.show()




if __name__ == '__main__':
    directory_path = "/home/saba/Documents/sba/data/ChildData/PI001/Analysed data"
    file_parser(directory_path)
    # filepath = "/home/saba/Documents/sba/data/ChildData/PI001/Raw data/SPO2 - PI001.txt"
    # sample, time = sample_extraction(filepath)
    # for i,j in zip(sample,time):
    #     print(i,j)
    # print((sample))
    # print((sample))
    # signal_visualisation(sample,time,"SpO2 normalised")
    # wavelet_coefficient_generation(sample,time,"db1","SpO2 normalised")


    # filepath = "/home/saba/Documents/sba/data/ChildData/PI001/Raw data/Sum RIPs - PI001.txt"
    # sample, time = sample_extraction(filepath)
    # signal_visualisation(sample, time, "SUM RIPs normalised")
    # wavelet_coefficient_generation(sample, time, "db1", "SUM RIPs normalised")


    filepath = "/home/saba/Documents/sba/data/ChildData/PI001/Raw data/CO2 - PI001.txt"
    sample, time = sample_extraction(filepath)
    # print(len(sample))
    # print(len(sample))
    #
    signal_visualisation(sample, time, "Co2 normalised")
    # wavelet_coefficient_generation(sample, time, "db1", "Co2 normalised")
```

Project Management

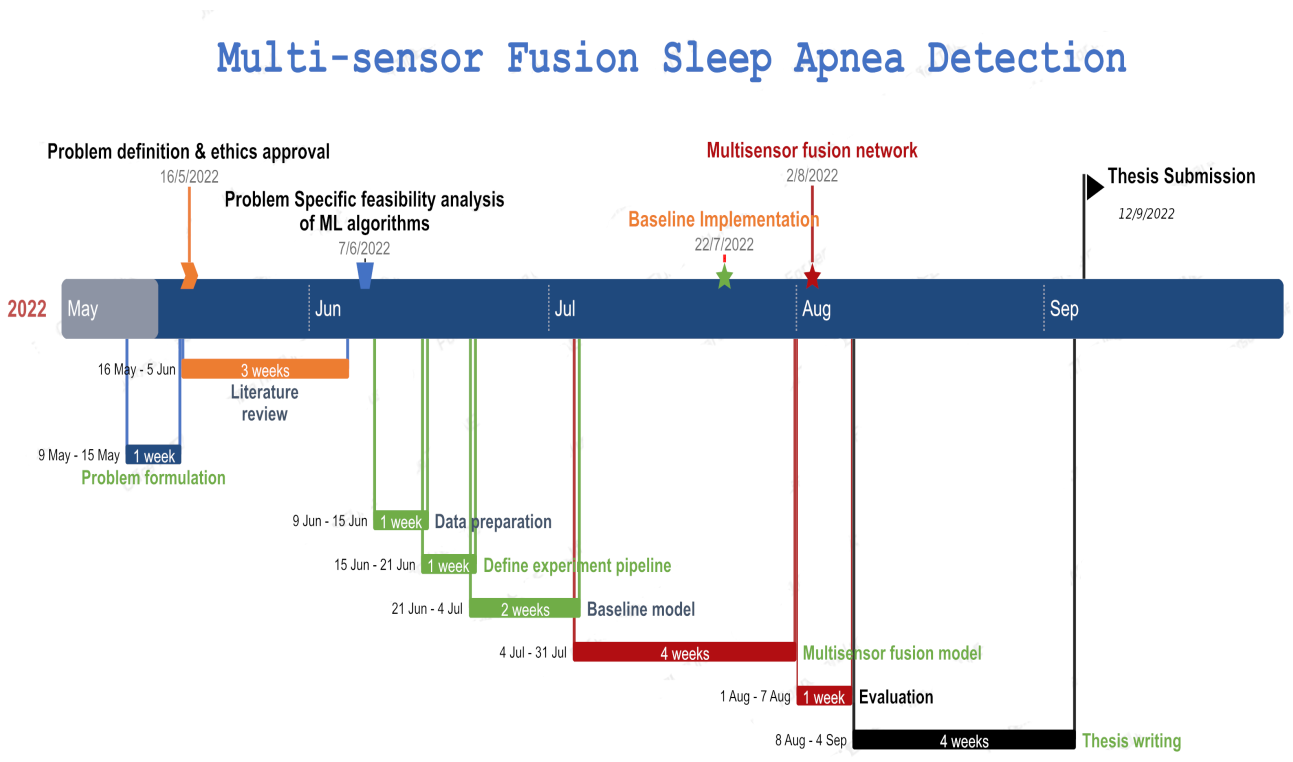


Figure 1: *Project timeline*

The project is finished with the following steps

- Initially, I was given the problem statement by my supervisor Prof. Lyudmila Mihaylova. She provided me with the children's sleep study data and walked me through the terms and technicalities.

- After I understood the problem statement, I did an in-depth literature review regarding the health factors and parameters concerning sleep apnea.
- My supervisor guided me to read the machine learning paper, and I did an in-depth review of the previous ML papers regarding sleep apnea detection. I found out that SVM and neural networks have been used popularly and effectively.
- I started reading and exploring low resource and shallow neural network models because the dataset is small.
- I faced a challenge because the data I received was not labelled. I came to know about unsupervised anomaly detection in time series data from a student of my supervisor.
- I started the implementation process, and firstly I parsed the data and synchronised all the sensors with similar time steps. Then I programmed the unsupervised Isolation forest using scikitlearn library and found the anomaly regions.
- , Therefore, I labelled the data using the anomaly timeframes and prepared a multisensor feature vector. I used the feature vector with various models mentioned in the dissertation. I compared their performance, latency and intermediate decision boundary representation.
- At the same time, in parallel, I started writing the literature review chapters and data processing chapters. After finishing all the experiments, I started writing the technical chapters.
- Finally, I reviewed the chapters multiple times and put appropriate references wherever necessary.