UNIVERSITY OF THESSALY

Development of a Portable System for Collecting and Processing Bio-signals and Sounds to Support the Diagnosis of Sleep Apnea

by

Raisa Angelidou

A thesis submitted in partial fulfillment for the Bachelor of Computer Science & Biomedical Informatics

in the
Department of Computer Science & Biomedical Informatics

March 2015

"Sleep is that golden chain that ties health and our bodies together."

- Thomas Dekker

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Περίληψη

Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική

Ραΐσα Αγγελίδου

Στην παρούσα εργασία παρουσιάζεται μια μέθοδος για την παρακολούθηση του ύπνου στο σπίτι, με στόχο την υποστήριξη της διάγνωσης της υπνικής άπνοιας. Δεδομένου ότι τα προβλήματα ύπνου επηρεάζουν το 24% των ανδρών και το 9% των γυναικών, σε όλο τον κόσμο, είναι σημαντικό το γεγονός ότι προτεινόμενο σύστημα είναι πρακτικό και χαμηλού κόστους. Σε αντίθεση με τις περισσότερες τεχνικές παρακολούθησης του ύπνου, η προτεινόμενη είναι κατάλληλη ακόμη και για μακροχρόνια παρακολούθηση. Αναπτύχθηκε μια αποτελεσματική εφαρμογή με σκοπό την ανίχνευση του ροχαλητού και συμβάντων υπνικής άπνοιας.

Η επικράτηση του ροχαλητού είναι αρκετά υψηλή ανάμεσα στον πληθυσμό και συχνά σύμπτωμα του συνδρόμου της αποφρακτικής άπνοιας στον ύπνο. Επομένως, κατασκευάστηκε ένα σύστημα καταγραφής ήχου, χωρίς επιτήρηση, χρησιμοποιώντας τον υπολογιστή Raspberry Pi, ένα απλό μικρόφωνο και σύνδεση στο διαδίκτυο. Ακόμη, αναπτύχθηκε μια εφαρμογή με τη χρήση της γλώσσας Java, έτσι ώστε να αντληθεί γνώση σχετικά με τα ηχητικά δείγματα και να δοθεί μια εκτίμηση σχετικά με το εάν ο χρήστης μπορεί να πάσχει από το σύνδρομο της αποφρακτικής άπνοιας ή όχι.

Ως κύριο χαρακτηριστικά για την ταξινόμηση των ηχητικών δειγμάτων σε κατηγορίες, χρησιμοποιήθηκαν οι συντελεστές της συχνότητας Mel. Στη συνέχεια,διεξήχθησαν αρκετά πειράματα για να αξιολογηθούν τα αποτελέσματα του μοντέλου που δημιουργήθηκε. Αυτή η εργασία προτείνει μια διακριτική μέθοδος παρακολούθησης του ύπνου, όπου κυρίαρχες μετρούμενες παράμετροι είναι οι αναπνοές και το ροχαλητό.

UNIVERSITY OF THESSALY

Abstract

Department of Computer Science & Biomedical Informatics

by Raisa Angelidou

This Thesis presents a method for sleep monitoring at home, aiming to support the diagnosis of sleep apnea. Since sleep problems affect 24% of men and 9% of women, worldwide, it is essential that the proposed system is convenient and low-cost. Unlike most sleep measurement techniques, the suggested one is also suited for long-term monitoring. In this study, an efficient application was developed in order to detect snoring and apneic events.
The prevalence of snoring is quite high among population and also an often symptom of the obstructive sleep apnea syndrome (OSAS). Thus an unattended recording system was built using the Raspberry Pi computer, a simple microphone and Internet connection. A Java application was also developed, so as to extract knowledge concerning the sound samples and get an estimate on whether the user may be suffering from OSAS or not.
For audio feature extraction the jAudio JAVA API was utilized, taking advantage of cross-platform portability and design benefits. Then, using the WEKA API the extracted properties were fed to machine learning algorithms. The performance of selected classifiers was evaluated and J48 decision tree was chosen (accuracy and sensitivity at 94,1%).
Mel-frequency cepstral coefficients were used as the prime feature in order to classify the audio samples into categories. Following, several experiments were conducted to evaluate the output of the created model. This thesis suggests an unobtrusive sleep tracking method, where the dominant measured parameter is breathing and snoring sounds.

Acknowledgements

Working on the sleep apnea project for the past 2 years has been a unique experience. However, many people aided in the completion of this thesis.
I am grateful to my supervisor, Dr. Ilias Maglogiannis, for providing knowledge, guidance and definitely patience. I, also, want to thank my family and friends for their support throughout my undergraduate years at the University of Thessaly.
Lastly, I want to thank Marios for being there along the way.

Contents

Greek Abstract **ii**

Abstract **iii**

Acknowledgements **iv**

List of Figures **vii**

List of Tables **viii**

Abbreviations **ix**

1 Introduction **1**
- 1.1 Sleep Apnea Syndrome . . . 1
- 1.2 Physiologic Sleep . . . 2
- 1.3 Sleep Disordered Breathing . . . 3
- 1.4 Snoring . . . 3
- 1.5 Snore Detection . . . 4
- 1.6 Audio Feature Extraction . . . 5
- 1.7 Thesis Outline . . . 6

2 Background **7**
- 2.1 Sleep Apnea Overview . . . 7
  - 2.1.1 Obstructive Sleep Apnea . . . 8
  - 2.1.2 Central Sleep Apnea . . . 10
  - 2.1.3 Mixed Sleep Apnea . . . 11
- 2.2 Review of Diagnostic Methods-Context of Interest . . . 12
  - 2.2.1 Polysomnography . . . 12
  - 2.2.2 Home Monitoring . . . 13
- 2.3 Data Mining-Machine Learning . . . 15
- 2.4 The Raspberry Pi . . . 16
- 2.5 JAudio . . . 17
- 2.6 The WEKA Project . . . 19
  - 2.6.1 Weka Data Set . . . 20
  - 2.6.2 Classification . . . 21
  - 2.6.3 MFCC . . . 22
  - 2.6.4 J48 Decision Tree-The C4.5 Algorithm . . . 23

3 Material and Methods **25**
3.1 System Architecture . . . 25
3.1.1 Recording System . . . 25
3.1.2 Application . . . 28

4 Results **30**
4.1 Audio Recordings Observations . . . 30
4.2 Feature Extraction Application . . . 31
4.2.1 Model Evaluation . . . 31
4.2.2 Experimental Data and Test Results . . . 33

5 Conclusion-Future Work **38**

Bibliography **40**

Code-Java **42**

Raspberry Pi-Bash Script **52**

List of Figures

1.1 Association between Clinical Findings 4
1.2 Typical Classification System 6

2.1 Obstructive Sleep Apnea Anatomy 9
2.2 Central and Obstructive Apneic Events (Baldwin cm, 2002, pp. 37:633-654) . 11
2.3 PSG 12
2.4 The Data Mining Process 15
2.5 The Raspberry Pi Computer 16
2.6 JAudio GUI Homescreen 18
2.7 Sampling Rate Selection 19
2.8 WEKA Startup Screen 20
2.9 WEKA Explorer 21
2.10 MFCC Pipeline 23

3.1 System Architecture Diagram 25
3.2 The Raspberry Pi Recording System 26
3.3 Script Audio Recording Log 27
3.4 Raspberry Pi Script 27
3.5 Labeled ARFF File 28
3.6 Feature Extraction Flow Chart 29

4.1 Classifiers: Performance Graphical Representation 32
4.2 Java Application-GUI Wireframe 34
4.3 Java Application-GUI Implementation 34
4.4 File Selection in Java Application 35
4.5 Java Application-GUI Output 35

List of Tables

1.1 OSA Severity Classes 3

4.1 Memory occupied for Recordings 30
4.2 Results of each Classifier 31
4.3 Classifiers: Errors and Accuracy 32
4.4 Application Test Results for Window Size:64ms 36
4.5 Application Test Results for Window Size:128ms 36
4.6 Application Test Results for Window Size:256ms 37

Abbreviations

AHI	**A**pnea **H**ypopnea **I**ndex
ARFF	**A**ttribute-**R**elation **F**ile**F**ormat
OSAS	**O**bstructive-**S**leep **A**pnea **S**yndrome
PSG	**P**oly**s**omno**g**raphy

Chapter 1

Introduction

1.1 Sleep Apnea Syndrome

Before Thomas Edison's invention of the light bulb, people slept an average of 10 hours a night; today the average is 6.9 hours of sleep on weeknights and 7.5 hours per night on weekends, although it still is doubtless that sleep is an essential part of the overall physical and mental welfare. Adequate sleep is vital to daytime alertness and outstanding activity. Sleep has a crucial role in the daily functioning of the nervous system and allows one to behave both biologically and mentally. Studies demonstrate that sleep is an imperative prerequisite for the regular performance of the body's immune system and ability to fight disease and sickness. The significance of a good night's sleep cannot be underrated. Yet, a recent study found that more people cut short on the amount of sleep or suffer from other sleep problems. Sleep difficulties appear in 75 percent of the population, while 2 in every 5 individuals complain about sleep deprivation. Only 30 percent of adults are getting the recommended eight hours of sleep each night. It is but no wonder that there is excessive research regarding sleep worldwide. Surveys conducted by the National Sleep Foundation (USA) report that 60 percent of adults have sleep problems several nights a week. Likewise, more than 40 percent of adults experience daytime hypersomnolence severe enough to interfere with their daily activities at least a few days each month.
An estimated 50-70 million adults in the United States have chronic sleep and wakefulness disorders. Sleep difficulties, some of which are avertible, are associated with chronic conditions, mental disorders, health-risk behaviors, limitations of daily functioning, injury, and mortality. According to a report from Centers for Disease Control and Prevention (CDC) summarizing the distribution of several sleep difficulties, it is determined that 37.9 percent of the respondents fall asleep unintentionally at least once in a month, while 48.0 percent snoring. Moreover, 4.7 percent doze or fall asleep while driving, similarly once in 30 days, consequently almost 20 percent of all serious car crash injuries in the general population are associated with driver sleepiness, independent of alcohol effects.

Humans spend about one-third of their lives asleep. There are two types of sleep, non-rapid eye movement (NREM) sleep and rapid eye movement (REM) sleep. According to sleep stages defined by Rechtschaffen and Kales, 1968, NREM sleep is divided into four stages, representing a continuum of relative depth. Each has unique characteristics including variations in brain wave patterns, eye movements, and muscle tone. Circadian rhythms, the daily rhythms in physiology and behavior, regulate the sleep-wake cycle.
Sleep stages have been identified by R&K through their detectable characteristics using electroencephalogram (EEG), electrooculogram (EOG) and electromyogram (EMG). Although it is a slow and expensive procedure, the sleep stage interpretations provided by R&K are the most widely applied in clinical and research sleep studies. Polysomnography is presently the standard technique to assess sleep parameters and also the gold standard for a definitive diagnosis of a certain and common sleep disorder, Obstructive Sleep Apnea. Polysomnography provides reliable data to enable sleep staging and the scoring of arousals, and to discriminate obstructive from central apneas. Polysomnography requires the following channels in order to evaluate sleep-related breathing disorders: EEG, EOG, chin EMG, airflow, arterial oxygen saturation, respiratory effort and ECG or heart rate. Considering PSG is an expensive and time consuming procedure, since patients need to be admitted at a hospital sleep laboratory for at least one night, other recording strategies and modalities have been developed.
Portable monitoring (PM) or respiratory polygraphy (RP) has recently been accepted as an alternative to overnight polysomnography (PSG) (Thurnheer 2007) for the evaluation of suspected obstructive sleep apnea. It has been reported in a number of papers that the PM is appropriate for the recording of sleep apnea 1.Portable monitoring has several advantages: increased accessibility, better patient acceptance, convenience of home recording, low cost and applicability to telemedicine. Although studies comparing portable devices to standard PSG are few, portable devices can be useful in diagnosing OSA in patients with a high pretest probability of OSA based on clinical features.
OSA cannot be diagnosed without the recording of breathing patterns during sleep. Thus, the principal aim of this thesis is to develop a portable but efficient monitoring device in order to support the diagnosis of obstructive sleep apnea exploiting snore sounds, a low-cost computer, the Raspberry Pi, and data mining techniques.

1.2 Physiologic Sleep

A few of the world's most remarkable philosophers and scientists have attempted to describe sleep behavior. The invention of electroencephalography (Hans Berger, 1929) is closely connected with modern sleep research primarily by allowing scientific categorization of sleep and its various stages. Normal human sleep can be divided into rapid eye movement (REM) sleep characterized by desynchronized EEG signals, muscle atony, and dreaming, and non-rapid eye movement (NREM) sleep characterized by synchronous EEG patterns. People begin the sleep cycle with a period of NREM sleep and after progressing through deeper NREM stages (stages II, III, and IV), the first episode of REM sleep is reached. Subsequently, cycles of NREM and REM occur, each lasting approximately 90 minutes. NREM constitutes the majority of sleep time – approximately 75% – while REM covers the remaining quarter.

1.3 Sleep Disordered Breathing

Sleep disordered breathing (SDB) comprises a vital connection between sleep and heart diseases. Its commonness in the general population is eminently high, affecting 24 and 9 percent of middle-aged men and women, respectively. Characteristic of SDB is the cessation or decrease in respiratory airflow during sleep, causing a fall in peripheral oxygen concentration of at least 4 percent. The number of these repetitive pauses in breathing divided by the number of hours of sleep defines the apnea-hypopnea index (AHI). A diagnosis of sleep disordered breathing is given to patients who are found to have AHI ≥5. Based on the mechanism of airflow cessation, SDB can be classified into two categories: obstructive sleep apnea (OSA), and central sleep apnea (CSA). Based on the AHI values, the severity of OSA is classified as follows (Table 1.1)

mild	5–15 events per hour of sleep
moderate	15–30 events per hour of sleep
severe	$>$ 30 events per hour of sleep

Table 1.1: Classes of Obstructive Sleep Apnea based on severity

1.4 Snoring

Snoring, apneas and hypopneas are indications of raised upper airway resistance usually due to anatomical factors that tighten upper airway. Snoring patients represent a range of clinical and sleep study findings from non-sleepy, non-apneic snoring to severe obstructive sleep apnea syndrome (OSA) (Figure 1.1). Upper airway resistance syndrome (UARS) occupies an in-between position in this spectrum.

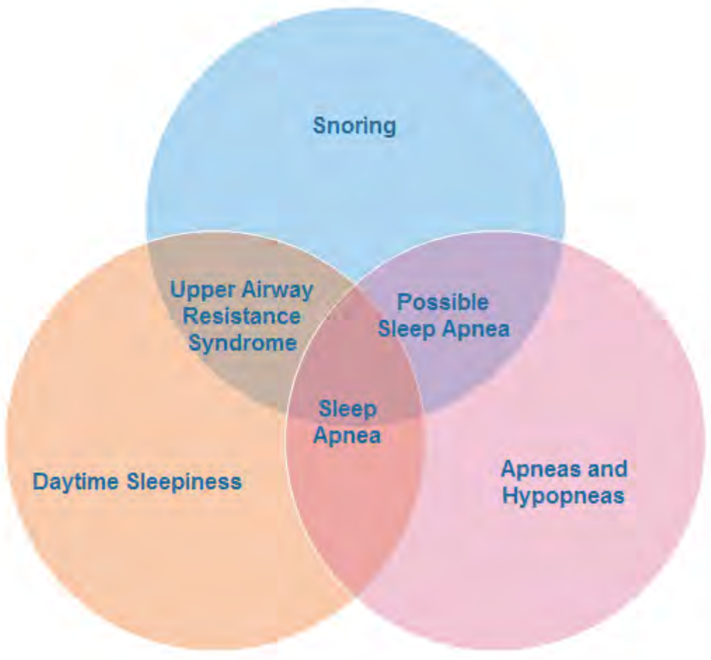


FIGURE 1.1: Schematic Illustration of the Association of Apneas and Hypopneas, Daytime Sleepiness and Snoring to Sleep Apnea

The prevalence of snoring is high and can significantly impact on quality of life. An impartial diagnosis of snoring usually requires a sleep study. The conditions causing sleep-disordered breathing, particularly OSA, have received much thought lately. Patients with OSA experience from total or partial airway obstruction caused by pharyngeal collapse during sleep, which leads to heavy snoring or choking, constant awakenings, interrupted sleep and hypersomnolence during the day. Not cured OSA is an important health problem that results in poor quality of life, increased danger for other diseases – cardiovascular diseases in particular. Additionally, OSA patients are at enhanced risk for higher mortality and traffic accidents. Since about 4% of men population and 2% of women have OSA, diagnosis and treatment options for OSA are crucial for health management.

1.5 Snore Detection

Automatic analysis of snoring sounds can be strongly associated with the following task: simple snoring versus OSA classification (a simple snorer can be defined as an individual having snoring habit without having any kind of sleep apnea). Detection of snoring episodes in a full-night recording of sleep sounds is a fundamental step in all these tasks. Until recently, related studies were based on manual segmentation of snoring episodes; there is a very limited amount of work on automatic detection. In the past, the energy and the zero crossing rate has been used as the feature to identify snoring episodes. Energy and zero-crossing rate are commonly used for audio signal activity detection; however, they are not known as having strong discriminative capabilities in classification. Duckitt et al (2006) adopted speech processing techniques for snore detection. Mel-frequency-cepstral coefficients (MFCCs) were used as the features in a hidden Markov model (HMM) based classification framework. The characteristics of a speech waveform in a transition segment between two phonemes of the sequence may deviate considerably more than those observed over the core segments of the neighboring phonemes. Phonemes are commonly modelled by three state HMMs in order to represent initial, core and final segments distinctively. However, sleep sounds are of dominant discrete nature. Furthermore, snoring sounds remain quite stationary over their intervals of existence. These observations suggest the possibility of using computationally less intensive classification approaches. MFCCs are very widely used in speech signal characterization much more than they are used with other types of audio signals. Sleep sound recordings contain not only sounds produced by humans but also sounds from other sources. Therefore, sound feature definition and classification methods in automatic snoring episode detection still appear as a ground of exploration.

1.6 Audio Feature Extraction

The main purpose of this thesis is to accurately determine useful content out of certain audio samples of one's overnight breathing and/or snoring. All sound classification systems, as such, start with the identification of audio signals' components. Thus, the first step of an aforementioned automatic system is to extract features. Many efficient methods have been developed in the speech/ speaker recognition field. It is basically a two-step procedure. First, feature extraction must be carefully performed and then classification, using models. A typical classification system is simply illustrated below.
A time-domain signal, like audio recordings, contains too much irrelevant data to use directly for classification. Thus, feature extraction is inevitable, while using the right features is crucial for a reliable result. Finally, since the prime target of a classification method is to properly allocate previously unobserved instances, therefore it is important to use an appropriate feature in the design of the classifier.

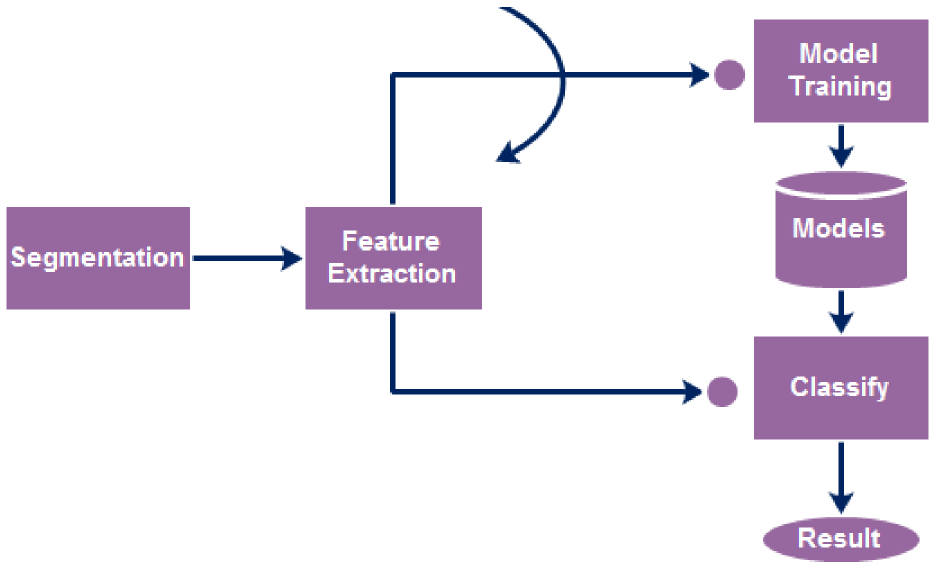


Figure 1.2: Typical Classification System

1.7 Thesis Outline

The motivation of this study was to develop an effective system to detect snoring episodes that can be sufficiently fast to process certain short-length recordings overnight in a reasonable amount of time but also with the fewest possible dependencies, such as cost to construct, platform to be run on, user knowledge and experience and lastly need for skilled technician. The suggested system is roughly a two-step process. Firstly, sound episodes are recorded and then these episodes are classified into certain categories of snoring or breathing characterized by Mel-frequency cepstral coefficients (MFCCs), widely used in literature for the analysis of snore sounds and sleep apnea screening. Our method overcomes the problems mentioned above and which are often found in the literature, since it makes use of Raspberry Pi, a low cost credit card-sized computer. The main benefit of the system, however, is the use of Java and, some open source libraries, making it portable and also fast.
This thesis is divided into 4 chapters. Chapter 1 describes all the background and theory needed to comprehend this approach. Chapter 2 presents the methodology used in the architectural design of this snore detection system. Chapter 3 discusses the results of some experiments performed. Chapter 4 summarizes the main conclusions of this thesis and also offers an outlook for future work.

Chapter 2

Background

2.1 Sleep Apnea Overview

Sleep apnea is very common, as common as type 2 diabetes. Yet still because of the lack of awareness by the public and healthcare professionals, the vast majority of sleep apnea patients remain undiagnosed and therefore untreated, despite the fact that this serious disorder can have serious consequences. Originating from the Greek word “apnea”, meaning without breath, sleep apnea is explained as uncontrolled interruptions of breathing that occurs while a patient is asleep. The duration of each pause may vary from 10 to 60 seconds. Patients may suffer from hundreds of these episodes throughout their sleep. As long as a breathing pause lasts, there is insufficient or no airflow to the lungs and also the blood oxygen levels are reduced.

There are three categories of this sleep disorder: obstructive, central, and mixed. Obstructive sleep apnea is caused by a blockage of the airway, usually when the soft tissue in the rear of the throat collapses and closes during sleep. In central sleep apnea, the airway is not blocked but the brain fails to signal the muscles to breathe. Mixed sleep apnea, as the name implies, is a combination of the two. Of the three, obstructive sleep apnea, frequently called OSA for short, is the most ordinary. With each apnea event, the brain rouses the sleeper, usually only partially, to signal breathing to resume. As a result, the patient’s sleep is extremely fragmented and of poor quality. Despite the difference in their principal cause, in all three categories, people with untreated sleep apnea stop breathing repeatedly during their sleep. This signals the patient’s brain to wake him/her up and the brain signals the relevant muscles to acquire increased airflow. In most cases the sleeper is unaware of these breath pauses because they don’t trigger a full awakening. Obstructed breathing sometimes halts with a gasp. This problematic way of breathing disrupts the patient’s normal sleep pattern and leads to both interim and lasting impacts on health. A family member or bed partner may be the first to recognize indications of sleep apnea.

Undiagnosed, sleep apnea can have serious and life-shortening consequences causing high

blood pressure and other cardiovascular diseases, stroke, diabetes, depression, memory problems, weight gain, impotence, and headaches. Moreover, untreated sleep apnea may be responsible for job impairment and motor vehicle crashes.
Sleep apnea is seen more regularly among men than women. The most dominant symptom is intensely loud snoring, sometimes remarkably loud that bed partners find it intolerable. Although not everyone who snores has sleep apnea it is the major symptom. Others signs and symptoms of sleep apnea include:

- Morning headaches
- Memory or learning difficulties and inability to concentrate
- Feeling irritable, depressed, or having mood swings or personality changes
- Waking up frequently to urinate
- Dry mouth or sore throat when waking up

Risk factors include being male, overweight, and over the age of 40, but sleep apnea can strike anyone at any age, even children. In children, sleep apnea can cause hyperactivity, poor school performance, and angry or hostile behavior. Children who have sleep apnea also may breathe through their mouth instead of their nose during the day. Fortunately, sleep apnea can be diagnosed and treated. Sleep studies in a sleep laboratory or home monitoring devices can show decisively the presence of sleep apnea and its severity. Several treatment options exist, and definitely research into additional options advances.

2.1.1 Obstructive Sleep Apnea

Obstructive Sleep Apnea (OSA) syndrome is characterized by the repetition of full or partial closure of the upper airway during sleep. Full occlusion of the UA is described as obstructive apnea while partial cessation is defined as hypopnea. Intense snoring and excessive daytime sleepiness are quite commonly noticed among individuals likely to suffer from OSA.
An obstructive apnea is a pause of ten or more seconds in respiration associated with continuing ventilatory effort. Obstructive hypopneas are decreases in ventilation, though not complete cessation of it, with an associated decrease in dissolved oxygen (oxygen saturation) or awakening from sleep. A diagnosis of OSA syndrome is accepted when a patient has an apnea-hypopnea index greater than 5 and has indications of excessive daytime sleepiness. Besides apneic events there a few other risk factors that may indicate the presence of OSA syndrome:

- Disruptive snoring
- Witnessed apnea or gasping

- Obesity and/or enlarged neck size
- Hypersomnolence (not common in children or in heart failure)
- Other: male gender, crowded-appearing pharyngeal airway, increased blood pressure, morning headache, sexual dysfunction, behavioral changes (especially in children)

Pharyngeal collapse in patients with OSA generally takes place at the rear of the tongue, uvula, and soft palate or some combo of the above structures. Since this part of the upper airway is not supported by bone or cartilage, it may collapse, because muscle tone provides rigidity and this tone can change over sleep (Figure 2.1). The main irregularity in patients with OSA is an anatomically small pharyngeal airway occurring from obesity, bone and soft tissue structures, or, in children, tonsils and adenoids. While awake, this leads to increased airflow resistance and greater intrapharyngeal negative pressure during breathing. However, during an apneic and hypopneic event there is actually this much of ventilatory effort that ultimately leads to arousal from sleep to terminate the apneic event. Thus, an upper airway that requires involuntary muscle activation to maintain patency during wakefulness may be vulnerable to failure during sleep.

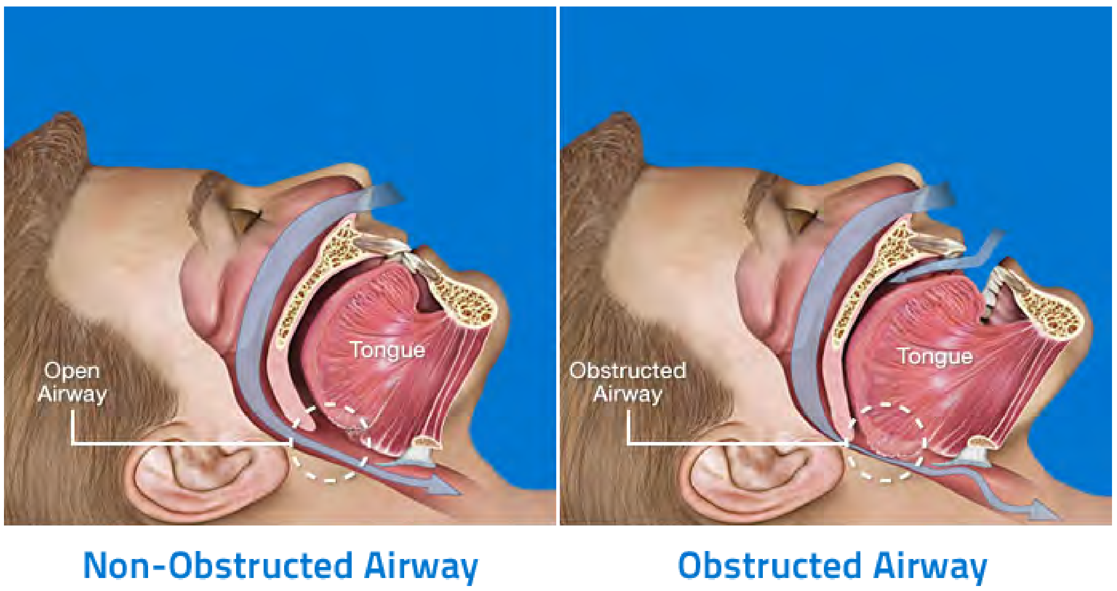


FIGURE 2.1: Obstructive Sleep Apnea Anatomy

Obstructive sleep apnea can be mild to severe. Airway might just tighten or get completely blocked. When the airway collapses a little it becomes blocked only partially. This makes it difficult for air to pass into the lungs during inhalation. Any air that remains through the obstruction may cause snoring. At times the airway may become entirely blocked resulting to no airflow. With little or no airflow to the lungs, there is a reduction of blood oxygen levels. This signals the brain to disrupt the sleep procedure which help in opening the airway. Occasionally obstructed breathing ends with a gasp. The normal breathing cycle continues

until the next episode of obstruction. Snoring is constantly connected with this disorder and is also one of the first nocturnal signs of OSA. Once one begins to sleep, the upper airway goes through functional and structural changes, prompting to spatially and transiently distributed locations conducive to snore sound generation.
OSA is a highly prevalent disease leading to a range of downstream complications such as increased risk of ischemic heart disease, stroke, type II diabetes, neurocognitive dysfunction, and increased vulnerability to accidents. If diagnosed soon, its' devastating consequences can be prevented. This risky sleep disorder has a great prevalence among men and women, but unfortunately more than 80% of OSA cases remain not diagnosed. With polysomnography (PSG) being the current reference standard used for OSA diagnosis, a low cost and unattended screening technique is urgently required.

2.1.2 Central Sleep Apnea

In central (non-obstructive) apnea an absence of chest and abdominal wall movement is reported along with a lack of airflow and unnoticeable upper airway blockage (Figure 2.2). The absence of chest and abdominal movement is the key to discriminate central apnea from the obstructive type. CSA is similarly, characterized by repetitive pauses of oxygen uptake during sleep, but unlike OSA the pauses are followed by a ventilatory drive deficiency. A central apnea is a ≥10-second pause in ventilation with no associated respiratory effort. CSA syndrome is generally present when a patient has more than 5 central apneas per hour of sleep and the related symptoms of frequent awakenings and/or daytime sleepiness. Since central apneas may also appear in an individual with obstructive apneas there is a requirement of 50 to 80 percent of the apneic events to be central and not obstructive, in deciding whether CSA is the syndrome from which the patient suffers. CSA is strongly associated with Cheyne-Stokes respiration (CSR) that generally occurs in patients with heart failure, neurovascular disorders and dementia.

- Congestive heart failure
- Paroxysmal nocturnal dyspnea
- Witnessed apnea
- Fatigue/hypersomnolence
- Other: male gender, older age, mitral regurgitation, atrial fibrillation, CSR while awake, periodic breathing during exercise, hyperventilation with hypocapnia

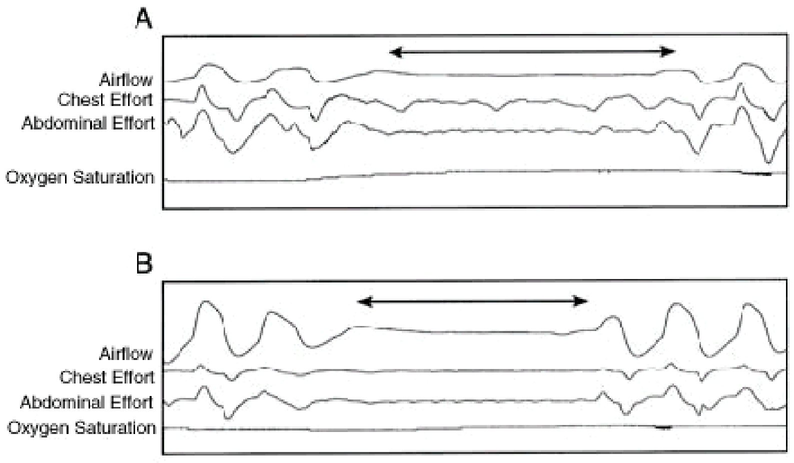


FIGURE 2.2: Central and Obstructive Apneic Events (Baldwin cm, 2002, pp. 37:633-654)

The diagnosis of the central sleep apnea syndrome is not easily made by doctors and likewise OSA requires a full-night polysomnography to determine the frequency and pattern of central apnea. Full-night polysomnography remains the standard, considering unattended monitoring systems, or merely oximetry, in the diagnosis of CSA are not widely accepted as in OSA detecting methods. Patients with heart failure will show a periodic breathing pattern even during wakefulness and exercise. Therefore, polysomnography is often helpful to ignore simultaneous obstructive apnea and to guide treatment for the nocturnal periodic breathing.

2.1.3 Mixed Sleep Apnea

Complex or mixed sleep apnea (CompSAS) is the term used to describe the form of sleep disordered breathing in which repeated central apneas (>5/hour) persist or emerge when obstructive events are eliminated with positive airway pressure (PAP) and for which there is no clear etiology. By the currently accepted definition, the central events must comprise more than half of the remnant sleep disordered breathing events or lead to a periodic breathing pattern which on positive airway pressure therapy becomes predominant and disruptive and the central apnea index (CAI) is more than 5 events per hour. It is actively being debated whether to include those with central apneas associated with narcotics or Cheyne-Stokes breathing due to systolic heart failure, but it is clear that there is a need for a diagnostic category for patients with treatment emergent central apneas—especially for those in whom the central apnea does not resolve with chronic CPAP treatment. Complex sleep apnea has also been often termed CPAP-related periodic breathing, complex disturbed breathing during sleep, and CPAP-related CSA. The prevalence ranges from 0.56% to 18% with no clear predictive characteristics as compared to obstructive sleep apnea. Prognosis is similar to obstructive sleep apnea, however the prevalence of this syndrome is somewhat variable in part due to the diversity of the population being studied and is significantly affected by factors such as narcotic use, BMI, and also heart failure.

2.2 Review of Diagnostic Methods-Context of Interest

2.2.1 Polysomnography

One of the most common tests in order to diagnose obstructive sleep apnea syndrome (OSAS) is laboratory-based polysomnography (PSG). It can determine the severity of the disease and validate numerous other sleep disorders.
Polysomnography is conducted by recording multiple physiologic parameters related to sleep and wakefulness. This picture below shows an example of an apnea event while a patient is sleeping.
Through PSG the number of respiratory events (ie, obstructive, central, or complex) and the resultant low level of oxygen in blood and awakenings related to the respiratory events can be monitored and calculated.
To determine both if OSAS is present and the degree of the disorder it is often enough to have a single-night PSG. Nevertheless, those of patients that are highly possible to suffer from the disease, but have a low AHI, may need several nights of monitoring. Moreover, other factors, such as laboratory equipment or scoring techniques may also be of significance. This technique is ultimately used to evaluate sleep and also wakefulness disorders that influence one's sleep.

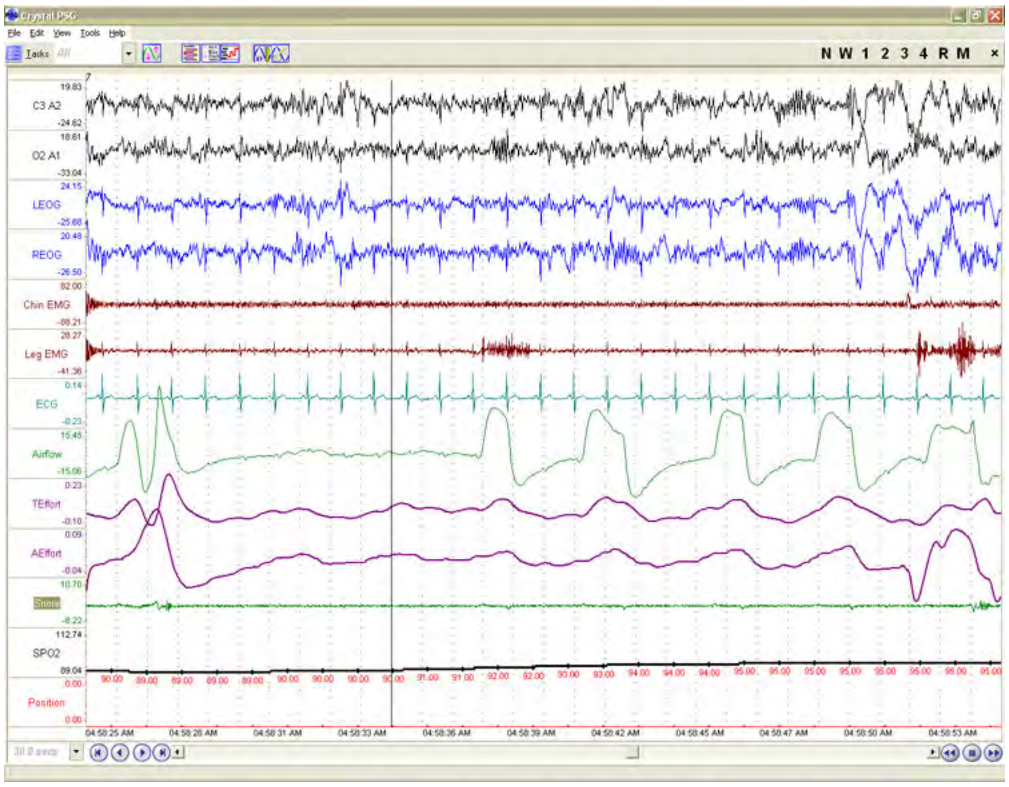


Figure 2.3: PSG - http://www.clevemed.com/terms/sleep_apnea.shtml

Parameters Monitored
Three different tests take place, in particular electroencephalography (EEG), electrooculography (EOG), and electromyography (EMG).
Firstly, PSG uses one EEG channel to monitor sleep stages, while at times additional EEG channels can be used, particularly in patients with epilepsy. To monitor both horizontal and vertical eye movements two EOG channels are needed. By using electrodes inherent voltage is picked up, which is necessary to evaluate eye movements for the documentation of the beginning of REM sleep and the presence of slow-rolling eye-movements. Lastly, an EMG channel records atonia or lack of it during the previous stage and also measures any excessive grinding of the teeth. EMG can also help in determining effort during respiratory events.
In order to monitor airflow, one oral thermistor channel evaluates the absence of air or not. Thermistor is the preferred method for the evaluation of apneas whereas a nasal pressure transducer channel measures airflow restriction more sensitively, and is the preferred method for the evaluation of hypopneas.
Other parameters monitored in a sleep study include electrocardiography, pulse oximetry, respiratory effort and even sound recordings to measure snoring. Some optional parameters of a sleep study include body temperature and pressure and pH at various esophageal levels.

2.2.2 Home Monitoring

The laboratory and overnight PSG has been the gold standard for estimating sleep physiology related to sleep apnea. The PSG has been appropriate for the diagnosis of obstructive sleep apnea (OSA), but cal also identify numerous other sleep disorders, like narcolepsy or restless leg syndrome, since the diagnostic criteria for certain sleep disorders are altogether clinical.
Although PSG has an obvious purpose in clinical sleep medicine, portable devices that can evaluate sleep patterns at home have emerged because of their low-cost and convenience. Such home monitoring equipment aims in assisting patients with sleep disordered breathing. Not only home-based sleep measurements can help in sleep apnea diagnosis, but can also benefit patients with insomnia.
These devices have no medical approval, but since they are inexpensive they are utilized from customers for self-help use. Commercially available sleep monitors based on different types of data collected are growing progressively in the wellness market, however the clinical market have an involvement for improved metrics of sleep activity. Long-term home monitoring avoids several restrictions of the laboratory PSG, like the inconvenient sleeping environment and the single monitoring. Since sleep is a changing process it is crucial to be able to measure multiple nights of sleep for medical, research, and wellness reasons. The potential to provide a more pragmatic platform to capture many of sleep data is actually offered by home monitoring. Long-term data is promising to turn out helpful for the discovery of underlying patterns of sleep inconsistency or the comparison of sleep with activities like

exercise, naps, food, caffeine, alcohol, and stress. The outcome of all the above activities, which vary from day to day, can lead to complex effects and interactions, thus large data sets are needed to denote sleep correlations.
Measuring sleep towards these different components can only be accomplished through the use of long-term data and not just a single-night PSG, with the goal of personalized evaluations. The goal of personal well-being is served by portable monitoring related to contexts not relevant to the area of sleep medicine.
Categories of available devices developed explicitly for sleep tracking and quality monitoring are shortly reviewed below. This area will definitely continue to expand swiftly as new devices are going to be introduced while technology advances. It would be an omission not to acknowledge that non energizing sleep can be connected to numerous medical and psychiatric ailments, therefore medical consultation is advised in any case.

- Sleep-monitoring based on brain activity signals includes devices that record electroencephalogram (EEG), electromyogram (EMG) and electrooculogram (EOG) signals. Data streams are used by a neural network and classifications of wake, light NREM, deep NREM, and REM sleep are rendered in 30-second time periods. Deep NREM sleep corresponds to slow wave sleep or stage N3. The term “deep” is often linked to this stage because of the high-amplitude and low-frequency EEG signal pattern and because awakening from this stage of sleep is very difficult.

- Sleep-monitoring based on autonomic signals is about systems that consist both of embedded sensors(with or without wired adhesive electrodes) that measure respiratory and cardiac physiology and ports that are available for optional EMG and EOG electrodes. Data signals stored locally in the device include electrocardiogram (ECG), actigraphy, and the individual’s body position. The trunk actigraphy signal is used to assess total sleep time, sleep efficiency and the number of awakenings that happened during the sleep time.

- Sleep-monitoring based on movement consists of wearable monitors. These are small devices which one can wear on the wrist, clip to clothing, or carry in his pocket. Features of this monitoring include a pedometer and altimeter, a calorie counting device, movement detection by actigraphy and a clock. The investigation of movement returns standard sleep related metrics such as a discrimination between sleep and wake states, total time of sleep, sleep latency, and an “arousal index” based on events of body motion during presumed sleep time. Such applications report graphs of total sleep time and a distinction between light sleep, deep sleep, and wake.

- • Bed-based sleep monitors include pressure-sensing pads that record heart rate, respiration rate, snoring, and body movement. In addition, sensors that measure respiration, heart rate, snoring, coughing, and movement have been developed. A Bayesian classifier algorithm can be utilized to combine properties of respiration with movement signals in order to distinguish sleep versus wake with modest accuracy compared to concurrent PSG evaluation These types of monitors connect the data streams with machine learning algorithms to optimize correspondence with human PSG scoring.

2.3 Data Mining-Machine Learning

A rapidly developing research area of Computer Science is that of Machine Learning and consequently Data Mining. This advancement is attributed to the surge in areas related to data analysis research, resulting Machine Learning to be an acknowledged research area, particularly connected with the discovery of models, patterns, and other regularities occurring in series of data. Machine learning approaches can be divided into Symbolic and Statistical approaches. Of the above, this thesis is making use of the former namely the J48 decision tree. In general, data mining (sometimes called knowledge discovery) is the analysis of data from different views in order to extract useful information. Data mining software allows users to evaluate data from many different perspectives, classify it and outline the relationships identified.

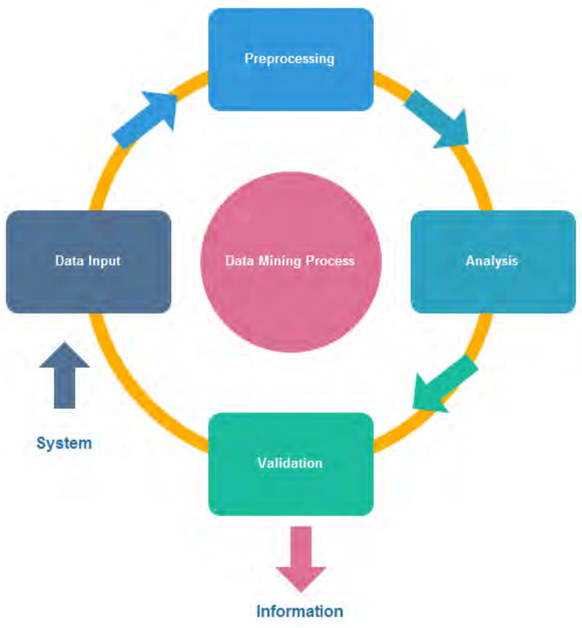


Figure 2.4: The Data Mining Process

The data mining process(Figure 2.4) is commonly defined with the following 4 stages:

1. Data Selection: A closer look at the data can determine whether some of them need to be removed for reducing the complexity and time resources needed for the system. This stage can also identify data quality issues.

2. Preprocessing: This phase covers all the tasks involved in creating the case table needed to build the model.(i.e attribute selection and data transformation).

3. Analysis: The phase of selecting and applying numerous modeling techniques and improving the parameters for optimum results.

4. Validation: This stage of the data mining process, evaluates how well the model satisfies the originally stated objective. This final step of knowledge discovery verifies that the patterns produced by the data mining algorithms occur in the expended data set.

2.4 The Raspberry Pi

The Raspberry Pi(Figure 2.5), primarily introduced in 2006, is a credit card sized computer, running a dedicated version of Linux, the Raspbian, developed by the Raspberry Pi Foundation, situated in the UK. With all of its components mounted on a single board, it is capable of offering many of the features regularly expected from a computer in the 21st century. However, the Pi carries out all the above with only an ARM processor and a very low price. Costs are kept so low since the Pi was based on a concept that teachers and academics would be able to inspire children. However, is now common to use the Pi for scientific purposes and prototyping projects, due to the plethora of advantages it offers.

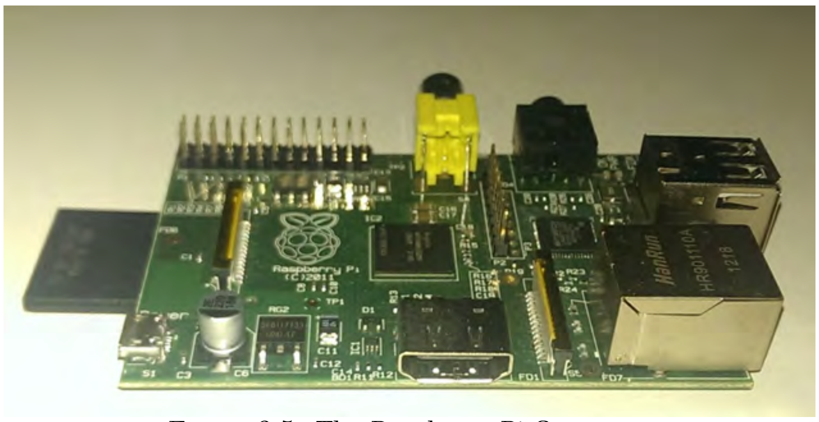


Figure 2.5: The Raspberry Pi Computer

Raspberry Pi can be considered as a microcontroller development board, such as Arduino, or even as a laptop replacement. In fact it is no coincidence that the latest version, the Raspberry Pi 2, introduced in January 2015, can now support a special Internet of Things version of the Windows OS.
At, first, the Raspberry Pi could be found in two versions, model A and model B. Although model A was cheaper than B, there are some significant differences between the two that prohibited the use of model A in this study, primarily to the lack of an Ethernet Port.
Below are described all the parts of the Model B Board, used in the development of this thesis:

The Processor

The core of the device is Broadcom BCM2835 System on a Chip (SoC), featuring single core 700 MHz ARM11 and 512MB of RAM.

The Secure Digital (SD) Card slot

Since there's no hard drive on the Pi, everything is stored on an SD Card.

The USB port

On the Model B there are two USB 2.0 ports, but only one on the Model A. A powered external hub can be used if needed.

Ethernet port

The model B has a standard RJ45 Ethernet port

HDMI connector

The HDMI port provides digital video and audio output, supporting 14 different video resolutions.

Status LEDs

The Pi has five indicator LEDs that provide visual feedback

Analog Audio output

A standard 3.5mm mini analog audio jack, intended to drive high impedance loads, but unfortunately, the quality of the analog output is much less than the HDMI audio output.

Composite video out

A standard RCA-type jack that provides extremely low-resolution composite NTSC or PAL video signals.

Power input

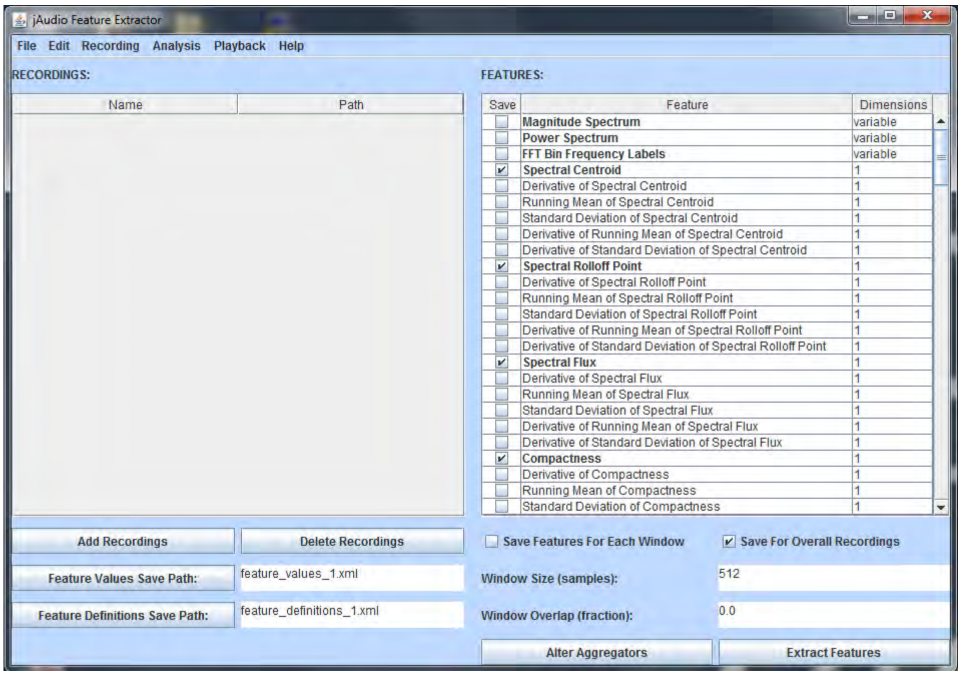
A microUSB connector is used to supply power.

The Raspberry Pi runs Linux for an operating system. Linux is technically just the kernel, and an operating system is much more than that; the Raspberry Pi used for this thesis study utilizes the Raspbian Wheezy as an operating system, which is a Debian based free OS.

2.5 JAudio

The jAudio library is a Digital Signal Processing project developed to provide a program for audio feature extraction developed using Java. Audio feature extraction involves extracting properties, such as power spectrum and statistical summaries and numerous other attributes. These properties can then be fed to machine learning toolkits (in our case WEKA) to automatically extract properties from unknown audio samples. Although Jaudio can be used

through a GUI (Figure 2.6) or a command-line interface, in this Thesis it was embedded in the developed Java code.
jAudio was written in Java in order to take advantage of Java's capabilities such as cross-platform portability and design benefits. It contains a custom low-level audio layer which supplements Java's limited core audio support and allows dealing with jAudio features directly with structures of sample values rather than concerning about buffering and format conversion issues. The fact that jAudio provides a Java API made it ideal for this Thesis' feature extraction system.



FEATURES:

Save	Feature	Dimensions
☐	**Magnitude Spectrum**	variable
☐	**Power Spectrum**	variable
☐	**FFT Bin Frequency Labels**	variable
✓	**Spectral Centroid**	1
☐	Derivative of Spectral Centroid	1
☐	Running Mean of Spectral Centroid	1
☐	Standard Deviation of Spectral Centroid	1
☐	Derivative of Running Mean of Spectral Centroid	1
☐	Derivative of Standard Deviation of Spectral Centroid	1
✓	**Spectral Rolloff Point**	1
☐	Derivative of Spectral Rolloff Point	1
☐	Running Mean of Spectral Rolloff Point	1
☐	Standard Deviation of Spectral Rolloff Point	1
☐	Derivative of Running Mean of Spectral Rolloff Point	1
☐	Derivative of Standard Deviation of Spectral Rolloff Point	1
✓	**Spectral Flux**	1
☐	Derivative of Spectral Flux	1
☐	Running Mean of Spectral Flux	1
☐	Standard Deviation of Spectral Flux	1
☐	Derivative of Running Mean of Spectral Flux	1
☐	Derivative of Standard Deviation of Spectral Flux	1
✓	**Compactness**	1
☐	Derivative of Compactness	1
☐	Running Mean of Compactness	1
☐	Standard Deviation of Compactness	1

☐ Save Features For Each Window ✓ Save For Overall Recordings

Feature Values Save Path: feature_values_1.xml

Feature Definitions Save Path: feature_definitions_1.xml

Window Size (samples): 512

Window Overlap (fraction): 0.0

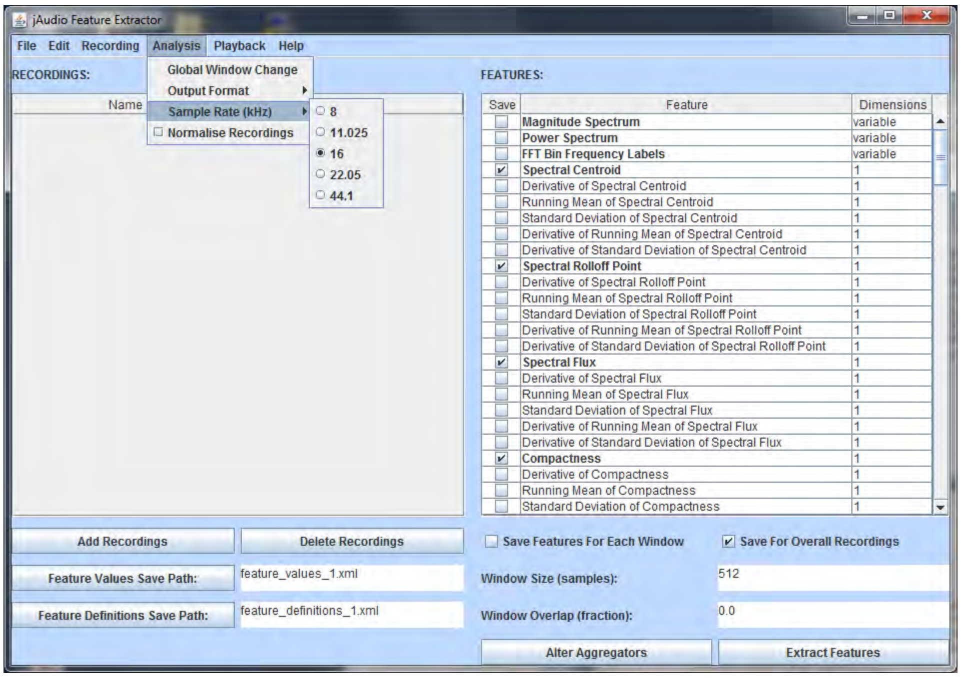
FIGURE 2.6: JAudio GUI Homescreen

jAudio uses two original file formats that represent extracted features in a specific format. Both of these files are written in XML format, making them easy to read and also to debug. Of these files, the first one contains feature definitions in order to explain the details of each feature and the other one contains the feature values. As a result, classification systems do not require any connection with the feature extraction system in order to access the feature definitions. Moreover, it is possible to reuse the same definitions with multiple data sets.
Unlike the conventional ways of storing feature values, the file formats mentioned keep two particular relationships. That is the logical relationship between the components of multidimensional features and the one between an overall audio sample and its window. Using this format, certain features can not only be stored for overall recordings, but as separate features for each window as well. jAudio supports the ARFF format used by the popular WEKA analysis toolkit making it easier to integrate it with the Weka API. By treating

all multidimensional features as collections of individual features, export to ARFF files can be done.

JAudio API

By using the jAudio API, several processing methods can be performed in order to extract certain audio information. Of these, the most suitable in the case of this application, are the batch files, used to set up a sequence of continuous experiments to run. The batch files are written in XML format, containing all the information needed to automatically load files, process them and output the feature values into the selected format. The information in included in such files may also set values like sampling rate, window size, normalization, output file format. In order to make use of the convenience offered by batch files, jAudio API comes with classes that assist in their parsing and getting all the relevant information for the feature extraction procedure.



Sample Rate (kHz): 8 [✓ 16] 11.025 22.05 44.1

FEATURES:

Save	Feature	Dimensions
	Magnitude Spectrum	variable
	Power Spectrum	variable
	FFT Bin Frequency Labels	variable
✓	**Spectral Centroid**	1
	Derivative of Spectral Centroid	1
	Running Mean of Spectral Centroid	1
	Standard Deviation of Spectral Centroid	1
	Derivative of Running Mean of Spectral Centroid	1
	Derivative of Standard Deviation of Spectral Centroid	1
✓	**Spectral Rolloff Point**	1
	Derivative of Spectral Rolloff Point	1
	Running Mean of Spectral Rolloff Point	1
	Standard Deviation of Spectral Rolloff Point	1
	Derivative of Running Mean of Spectral Rolloff Point	1
	Derivative of Standard Deviation of Spectral Rolloff Point	1
✓	**Spectral Flux**	1
	Derivative of Spectral Flux	1
	Running Mean of Spectral Flux	1
	Standard Deviation of Spectral Flux	1
	Derivative of Running Mean of Spectral Flux	1
	Derivative of Standard Deviation of Spectral Flux	1
✓	**Compactness**	1
	Derivative of Compactness	1
	Running Mean of Compactness	1
	Standard Deviation of Compactness	1

☐ Save Features For Each Window ☑ Save For Overall Recordings

Feature Values Save Path: feature_values_1.xml

Feature Definitions Save Path: feature_definitions_1.xml

Window Size (samples): 512

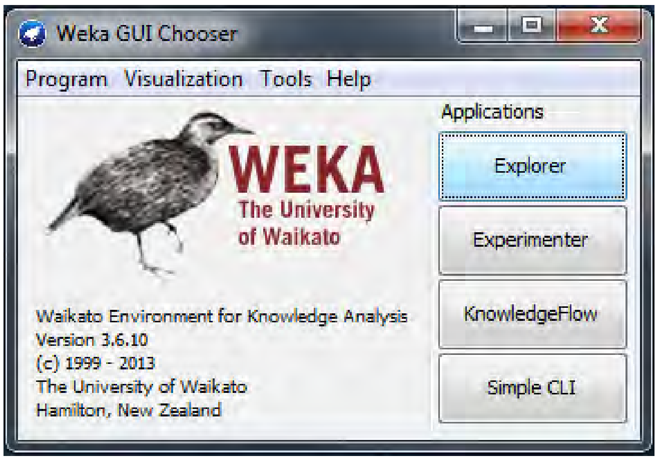
Window Overlap (fraction): 0.0

Figure 2.7: Sampling Rate Selection

2.6 The WEKA Project

WEKA is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. WEKA is a piece of software that can be used for big data mining as well. It is the product of the University of Waikato (New Zealand) and was first implemented in its modern form in 1997.

It uses the GNU General Public License (GPL). The software is written in the Java™ language and contains a GUI for interacting with data files and producing visual results (think tables and curves), the algorithms can be applied directly to a dataset (Figure 2.8). It also has a general API and can be embedded like any other library. In this thesis WEKA has been called from the developed JAVA code.



Waikato Environment for Knowledge Analysis
Version 3.6.10
(c) 1999 - 2013
The University of Waikato
Hamilton, New Zealand

FIGURE 2.8: WEKA Startup Screen

Starting up with WEKA, the GUI chooser pops up letting the user select from four ways to work with WEKA and data. For the purposes of the proposed system the Explorer option is more than adequate.

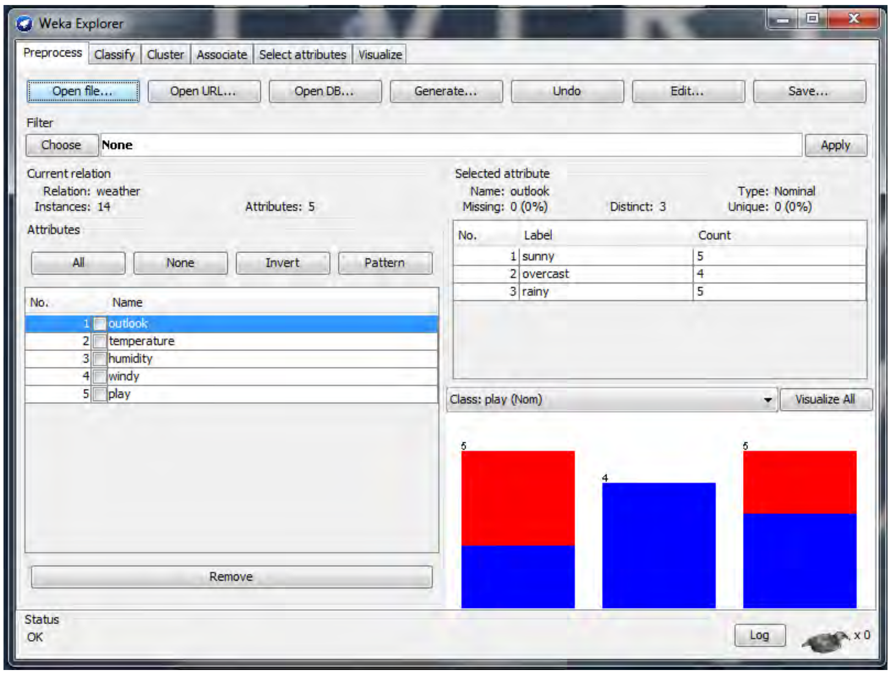
2.6.1 Weka Data Set

The number one method for loading data into WEKA is the Attribute-Relation File Format (ARFF). An ARFF file is an ASCII text file that describes a list of instances sharing a set of attributes, developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the WEKA machine learning software. ARFF files have two distinct sections. The first section is the Header information, which is followed the Data information. The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. The ARFF Data section of the file contains the data declaration line and the actual instance lines. Each instance is represented on a single line, where every new line denotes a new instance.

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}
```

```
@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

The Preprocess tab (Figure 2.9) offers information about the data set, statistical information about each column, such as maximum, minimum, average and standard deviation, but visualization of the data as well.



Filter
None

Current relation
Relation: weather
Instances: 14
Attributes: 5

Attributes

No.	Name
1	outlook
2	temperature
3	humidity
4	windy
5	play

Selected attribute
Name: outlook
Type: Nominal
Missing: 0 (0%)
Distinct: 3
Unique: 0 (0%)

No.	Label	Count
1	sunny	5
2	overcast	4
3	rainy	5

Class: play (Nom)

Status
OK

FIGURE 2.9: WEKA Explorer

2.6.2 Classification

Classification is a data mining algorithm that creates a step-by-step guide for how to determine the output of a new data instance. This is achieved using training and testing data set. Training data is loaded into a machine learning algorithm, in this case Naïve Bayes, so as to produce a classifier. The produced classifier can be tested using some independent test data

in order to be evaluated. In any case, it is important to use a different dataset for a reliable evaluation.
The concept of using a ”training set” to produce the model is common and simple. A dataset with known output values is used to build the model. Then, every time a new, unclassified, data point is loaded, with an unknown output value, the unknown values are predicted and the expected output file is produced. The test data ensure that the model will accurately predict future unknown values. The output from such a model looks like:

```
Time taken to build model: 0 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances          13               92.8571 \%
Incorrectly Classified Instances         1                7.1429 \%
Kappa statistic                          0.8372
Mean absolute error                      0.2798
Root mean squared error                  0.3315
Relative absolute error                 60.2576 \%
Root relative squared error             69.1352 \%
Total Number of Instances               14

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0.2        0.9        1         0.947      0.911    yes
                 0.8       0          1          0.8       0.889      0.911    no
Weighted Avg.    0.929     0.129      0.936      0.929     0.926      0.911

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = yes
 1 4 | b = no
```

Important elements for the evaluation of the above output model are the “correctly” and “incorrectly classified instances” and also the “confusion matrix” showing false positive and false negative classified number of instances. Moreover, accuracy displays whether or not a model is worthy. The final step to validate a classification tree is to run a test set through the model and ensure that the accuracy of the model when evaluating the test set is not different from the training set.

2.6.3 MFCC

The extraction and selection of the best parametric representation of acoustic signals is an important task in the design of any speech recognition system; it significantly affects the recognition performance. A compact representation would be provided by a set of mel-frequency cepstrum coefficients (MFCC), which are the results of a cosine transform of the real logarithm of the short-term energy spectrum expressed on a mel-frequency scale. The MFCCs are proved more efficient. The calculation of the MFCC includes the steps shown in Figure 2.10.

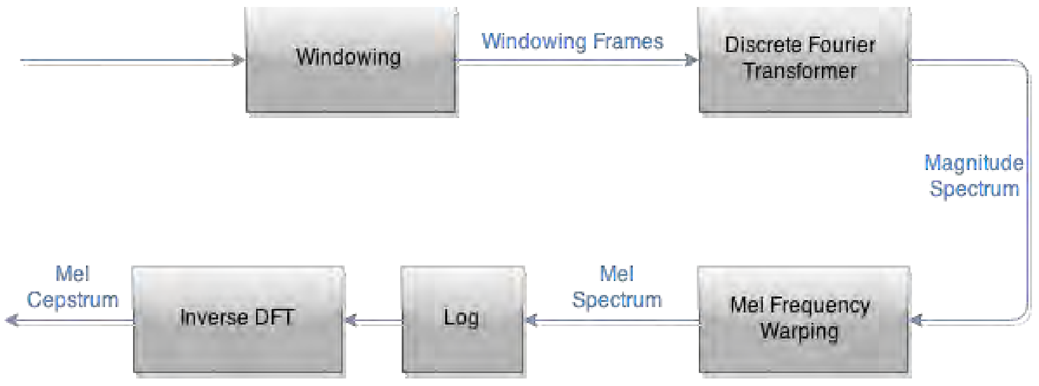


FIGURE 2.10: MFCC Pipeline

MFCC is based on the human peripheral auditory system. The human perception of the frequency con-tents of sounds for speech signals does not follow a linear scale. Thus for each tone with an actual fre-quency t measured in Hz, a subjective pitch is measured on a scale called the 'Mel Scale' .The mel frequency scale is a linear frequency spacing below 1000 Hz and logarithmic spacing above 1kHz.As a reference point, the pitch of a 1 kHz tone, 40 dB above the perceptual hearing threshold, is defined as 1000 Mels. The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

2.6.4 J48 Decision Tree-The C4.5 Algorithm

C4.5 is an algorithm used to spawn a decision tree developed by Ross Quinlan. The decision trees generated by C4.5 can be used for classification, thus, C4.5 is often mentioned as a statistical classifier. C4.5 builds decision trees from a set of training data, where the training data is a set of already classified samples. Each sample has the same structure, consisting of a number of attribute/value pairs.
At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sub lists.

This algorithm has a few base cases. All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class. None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class. Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.
J48 is an open source Java implementation of the C4.5 algorithm in the WEKA data mining tool. J48 implements Quinlan's C4.5 algorithm for generating a pruned or unpruned C4.5 decision tree. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by J48 can be used for classification. J48 builds decision trees from a set of labeled training data using the concept of information entropy. It uses the fact that each attribute of the data can be used to make a decision by splitting the data into smaller subsets. J48 examines the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. To make the decision, the attribute with the highest normalized information gain is used. Then the algorithm recurs on the smaller subsets. The splitting procedure stops if all instances in a subset belong to the same class. Then a leaf node is created in the decision tree telling to choose that class. But it can also happen that none of the features give any information gain. In this case J48 creates a decision node higher up in the tree using the expected value of the class.J48 can handle both continuous and discrete attributes, training data with missing attribute values and attributes with differing costs. Further it provides an option for pruning trees after creation.

Chapter 3

Material and Methods

3.1 System Architecture

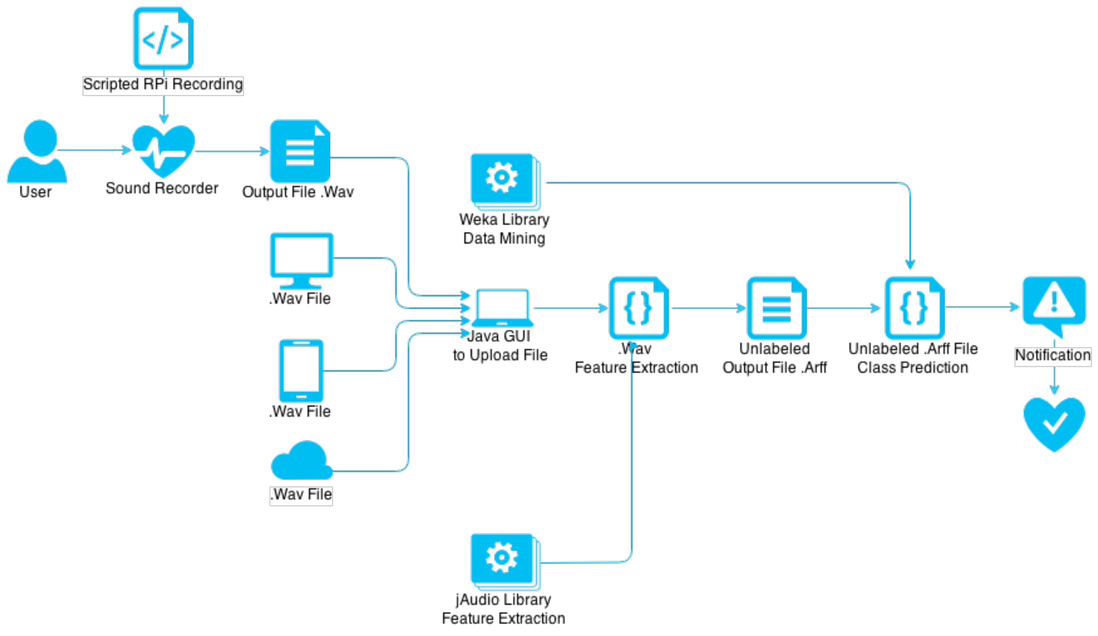


FIGURE 3.1: System Architecture Diagram

3.1.1 Recording System

The architecture used to implement the proposed system can be described in two parts (Figure 3.1). First, an audio recording system was built, using the Raspberry Pi computer. The system is rather simple; it is automatic, unattended and extremely low in hardware requirements (Figure 3.2).
Hardware needed for the recording system:

1. A Raspberry Pi computer, model B

2. Ethernet cable or USB WiFi Adapter

3. USB Sound card, since Raspberry Pi does not have an audio input embedded

4. Simple 3,5 mm jack microphone or a common 3,5 mm jack headset

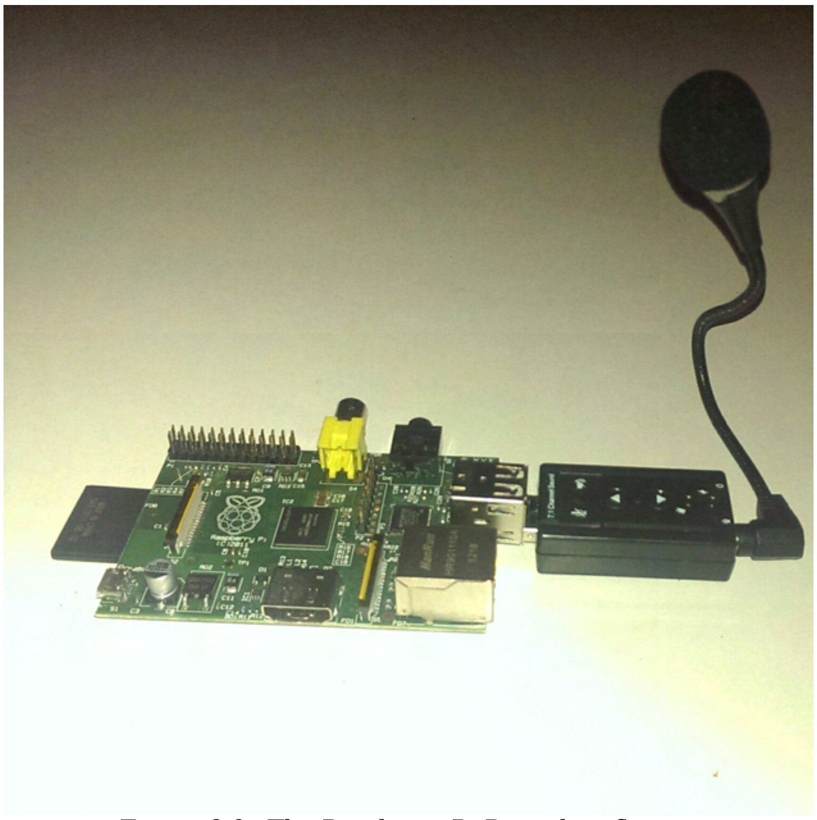
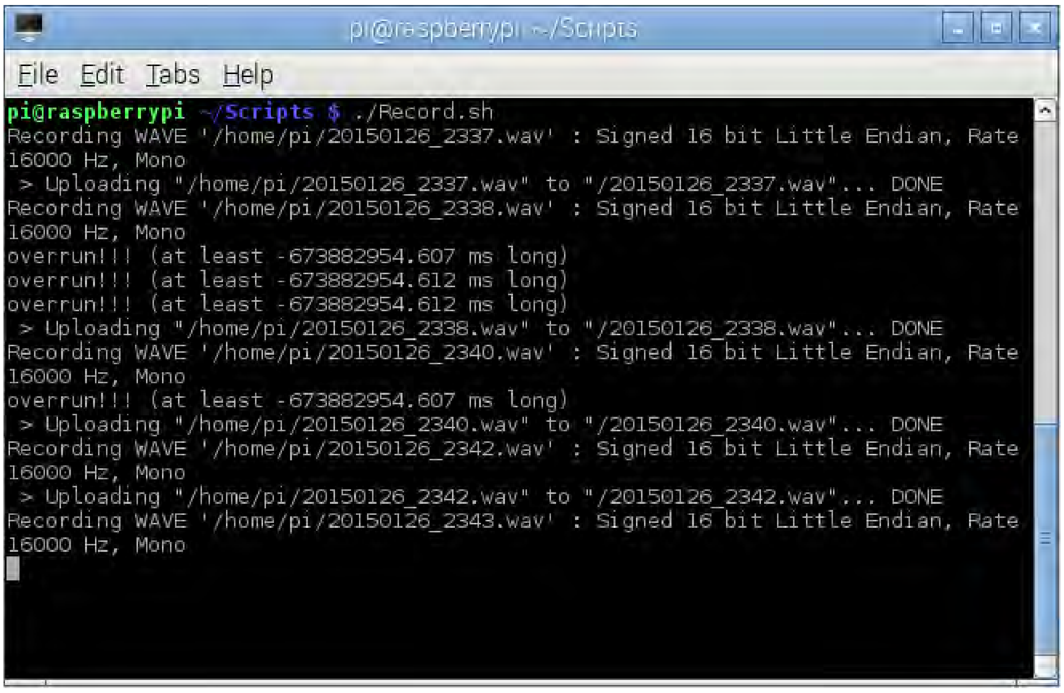


Figure 3.2: The Raspberry Pi Recording System

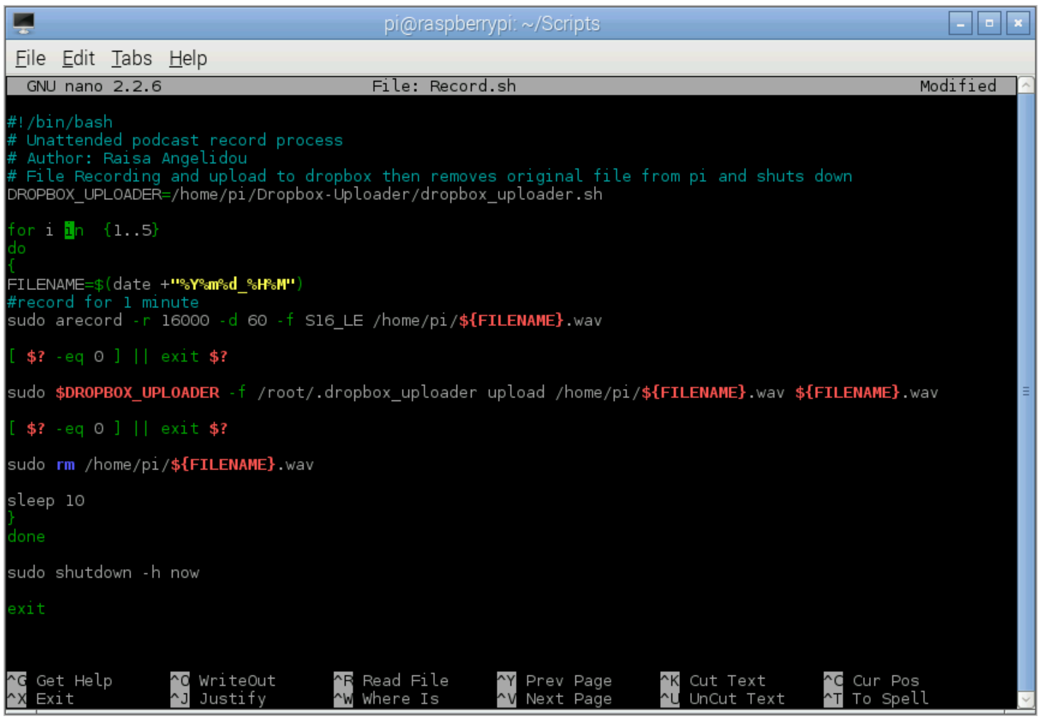
Raspberry Pi has two USB ports on chip; both were used in order to connect the above peripherals. One was used for the WiFi adapter and one for the system's major component, the USB sound card alongside the mic. Then the computer was programmed accordingly, the sound card was set up and the volume options of the external microphone, as first in line. Also, was made sure that the Raspberry Pi computer always boots up with the above settings, ALSA utilities were installed as the recording, processing and playback application of the system. Following, a script was written, recording several segments of the users sleep sounds; that is breathing but snoring sounds as well. In-between these recordings, the previous segment are uploaded to the users Dropbox in a special folder. For that part Dropbox Uploader was also set up (Figure 3.3).



```
pi@raspberrypi ~/Scripts $ ./Record.sh
Recording WAVE '/home/pi/20150126_2337.wav' : Signed 16 bit Little Endian, Rate
16000 Hz, Mono
 > Uploading "/home/pi/20150126_2337.wav" to "/20150126_2337.wav"... DONE
Recording WAVE '/home/pi/20150126_2338.wav' : Signed 16 bit Little Endian, Rate
16000 Hz, Mono
overrun!!! (at least -673882954.607 ms long)
overrun!!! (at least -673882954.612 ms long)
overrun!!! (at least -673882954.612 ms long)
 > Uploading "/home/pi/20150126_2338.wav" to "/20150126_2338.wav"... DONE
Recording WAVE '/home/pi/20150126_2340.wav' : Signed 16 bit Little Endian, Rate
16000 Hz, Mono
overrun!!! (at least -673882954.607 ms long)
 > Uploading "/home/pi/20150126_2340.wav" to "/20150126_2340.wav"... DONE
Recording WAVE '/home/pi/20150126_2342.wav' : Signed 16 bit Little Endian, Rate
16000 Hz, Mono
 > Uploading "/home/pi/20150126_2342.wav" to "/20150126_2342.wav"... DONE
Recording WAVE '/home/pi/20150126_2343.wav' : Signed 16 bit Little Endian, Rate
16000 Hz, Mono
```

Figure 3.3: Script Audio Recording Log

Regarding the audio recording settings, the segments are recorded at a 16 KHz sampling rate, WAVE. Also, the buffer data is being delivered as a 16bit little endian byte array (Figure 3.4). Finally, the original recorded files are removed from the Raspberry Pi.

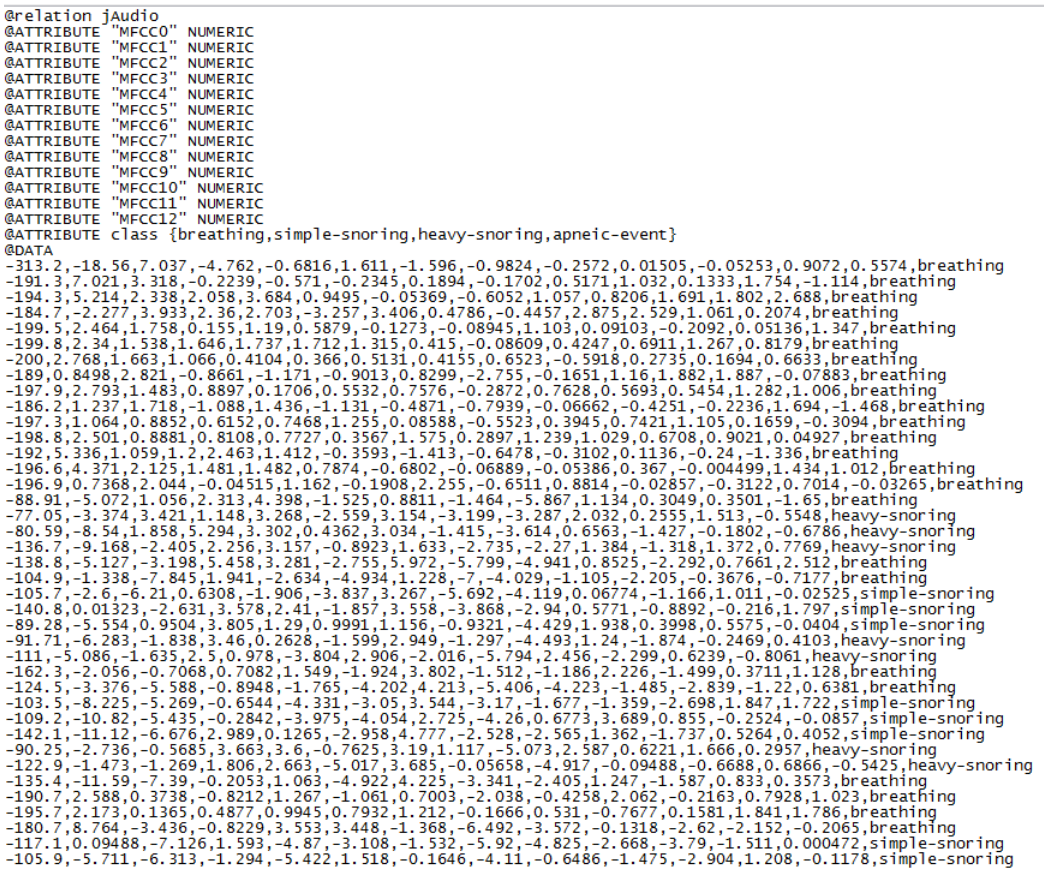


```
  GNU nano 2.2.6                    File: Record.sh                         Modified

#!/bin/bash
# Unattended podcast record process
# Author: Raisa Angelidou
# File Recording and upload to dropbox then removes original file from pi and shuts down
DROPBOX_UPLOADER=/home/pi/Dropbox-Uploader/dropbox_uploader.sh

for i in  {1..5}
do
{
FILENAME=$(date +"%Y%m%d_%H%M")
#record for 1 minute
sudo arecord -r 16000 -d 60 -f S16_LE /home/pi/${FILENAME}.wav

[ $? -eq 0 ] || exit $?

sudo $DROPBOX_UPLOADER -f /root/.dropbox_uploader upload /home/pi/${FILENAME}.wav ${FILENAME}.wav

[ $? -eq 0 ] || exit $?

sudo rm /home/pi/${FILENAME}.wav

sleep 10
}
done

sudo shutdown -h now

exit


^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

Figure 3.4: Raspberry Pi Script

3.1.2 Application

Secondly, a JAVA code was developed using jAudio library for the recorded samples feature extraction, and WEKA library for training and classification. Both jAudio and WEKA are developed in JAVA, offer an open-source API as well as documentation.A step at a time explanation of the code is described as follows:
The jAudio graphical user interface (GUI) was used for the feature extraction of some apneic audio samples. The sole feature used throughout this system is the Mel-frequency cepstral coefficients (MFCCs), widely used in literature for the analysis of snore sounds and sleep apnea screening. Again, the audio file format was chosen to be WAVE, single channel and sampling rate at 16 KHz. The above procedure has an output file in ARFF format which was manually labeled into four classes and then used to train WEKA (Figure 3.5).



```
@relation jAudio
@ATTRIBUTE "MFCC0" NUMERIC
@ATTRIBUTE "MFCC1" NUMERIC
@ATTRIBUTE "MFCC2" NUMERIC
@ATTRIBUTE "MFCC3" NUMERIC
@ATTRIBUTE "MFCC4" NUMERIC
@ATTRIBUTE "MFCC5" NUMERIC
@ATTRIBUTE "MFCC6" NUMERIC
@ATTRIBUTE "MFCC7" NUMERIC
@ATTRIBUTE "MFCC8" NUMERIC
@ATTRIBUTE "MFCC9" NUMERIC
@ATTRIBUTE "MFCC10" NUMERIC
@ATTRIBUTE "MFCC11" NUMERIC
@ATTRIBUTE "MFCC12" NUMERIC
@ATTRIBUTE class {breathing,simple-snoring,heavy-snoring,apneic-event}
@DATA
-313.2,-18.56,7.037,-4.762,-0.6816,1.611,-1.596,-0.9824,-0.2572,0.01505,-0.05253,0.9072,0.5574,breathing
-191.3,7.021,3.318,-0.2239,-0.571,-0.2345,0.1894,-0.1702,0.5171,1.032,0.1333,1.754,-1.114,breathing
-194.3,5.214,2.338,2.058,3.684,0.9495,-0.05369,-0.6052,1.057,0.8206,1.691,1.802,2.688,breathing
-184.7,-2.277,3.933,2.36,2.703,-3.257,3.406,0.4786,-0.4457,2.875,2.529,1.061,0.2074,breathing
-199.5,2.464,1.758,0.155,1.19,0.5879,-0.1273,-0.08945,1.103,0.09103,-0.2092,0.05136,1.347,breathing
-199.8,2.34,1.538,1.646,1.737,1.712,1.315,0.415,-0.08609,0.4247,0.6911,1.267,0.8179,breathing
-200,2.768,1.663,1.066,0.4104,0.366,0.5131,0.4155,0.6523,-0.5918,0.2735,0.1694,0.6633,breathing
-189,0.8498,2.821,-0.8661,-1.171,-0.9013,0.8299,-2.755,-0.1651,1.16,1.882,1.887,-0.07883,breathing
-197.9,2.793,1.483,0.8897,0.1706,0.5532,0.7576,-0.2872,0.7628,0.5693,0.5454,1.282,1.006,breathing
-186.2,1.237,1.718,-1.088,1.436,-1.131,-0.4871,-0.7939,-0.06662,-0.4251,-0.2236,1.694,-1.468,breathing
-197.3,1.064,0.8852,0.6152,0.7468,1.255,0.08588,-0.5523,0.3945,0.7421,1.105,0.1659,-0.3094,breathing
-198.8,2.501,0.8881,0.8108,0.7727,0.3567,1.575,0.2897,1.239,1.029,0.6708,0.9021,0.04927,breathing
-192,5.336,1.059,1.2,2.463,1.412,-0.3593,-1.413,-0.6478,-0.3102,0.1136,-0.24,-1.336,breathing
-196.6,4.371,2.125,1.481,1.482,0.7874,-0.6802,-0.06889,-0.05386,0.367,-0.004499,1.434,1.012,breathing
-196.9,0.7368,2.044,-0.04515,1.162,-0.1908,2.255,-0.6511,0.8814,-0.02857,-0.3122,0.7014,-0.03265,breathing
-88.91,-5.072,1.056,2.313,4.398,-1.525,0.8811,-1.464,-5.867,1.134,0.3049,0.3501,-1.65,breathing
-77.05,-3.374,3.421,1.148,3.268,-2.559,3.154,-3.199,-3.287,2.032,0.2555,1.513,-0.5548,heavy-snoring
-80.59,-8.54,1.858,5.294,3.302,0.4362,3.034,-1.415,-3.614,0.6563,-1.427,-0.1802,-0.6786,heavy-snoring
-136.7,-9.168,-2.405,2.256,3.157,-0.8923,1.633,-2.735,-2.27,1.384,-1.318,1.372,0.7769,heavy-snoring
-138.8,-5.127,-3.198,5.458,3.281,-2.755,5.972,-5.799,-4.941,0.8525,-2.292,0.7661,2.512,breathing
-104.9,-1.338,-7.845,1.941,-2.634,-4.934,1.228,-7,-4.029,-1.105,-2.205,-0.3676,-0.7177,breathing
-105.7,-2.6,-6.21,0.6308,-1.906,-3.837,3.267,-5.692,-4.119,0.06774,-1.166,1.011,-0.02525,simple-snoring
-140.8,0.01323,-2.631,3.578,2.41,-1.857,3.558,-3.868,-2.94,0.5771,-0.8892,-0.216,1.797,simple-snoring
-89.28,-5.554,0.9504,3.805,1.29,0.9991,1.156,-0.9321,-4.429,1.938,0.3998,0.5575,-0.0404,simple-snoring
-91.71,-6.283,-1.838,3.46,0.2628,-1.599,2.949,-1.297,-4.493,1.24,-1.874,-0.2469,0.4103,heavy-snoring
-111,-5.086,-1.635,2.5,0.978,-3.804,2.906,-2.016,-5.794,2.456,-2.299,0.6239,-0.8061,heavy-snoring
-162.3,-2.056,-0.7068,0.7082,1.549,-1.924,3.802,-1.512,-1.186,2.226,-1.499,0.3711,1.128,breathing
-124.5,-3.376,-5.588,-0.8948,-1.765,-4.202,4.213,-5.406,-4.223,-1.485,-2.839,-1.22,0.6381,breathing
-103.5,-8.225,-5.269,-0.6544,-4.331,-3.05,3.544,-3.17,-1.677,-1.359,-2.698,1.847,1.722,simple-snoring
-109.2,-10.82,-5.435,-0.2842,-3.975,-4.054,2.725,-4.26,0.6773,3.689,0.855,-0.2524,-0.0857,simple-snoring
-142.1,-11.12,-6.676,2.989,0.1265,-2.958,4.777,-2.528,-2.565,1.362,-1.737,0.5264,0.4052,simple-snoring
-90.25,-2.736,-0.5685,3.663,3.6,-0.7625,3.19,1.117,-5.073,2.587,0.6221,1.666,0.2957,heavy-snoring
-122.9,-1.473,-1.269,1.806,2.663,-5.017,3.685,-0.05658,-4.917,-0.09488,-0.6688,0.6866,-0.5425,heavy-snoring
-135.4,-11.59,-7.39,-0.2053,1.063,-4.922,4.225,-3.341,-2.405,1.247,-1.587,0.833,0.3573,breathing
-190.7,2.588,0.3738,-0.8212,1.267,-1.061,0.7003,-2.038,-0.4258,2.062,-0.2163,0.7928,1.023,breathing
-195.7,2.173,0.1365,0.4877,0.9945,0.7932,1.212,-0.1666,0.531,-0.7677,0.1581,1.841,1.786,breathing
-180.7,8.764,-3.436,-0.8229,3.553,3.448,-1.368,-6.492,-3.572,-0.1318,-2.62,-2.152,-0.2065,breathing
-117.1,0.09488,-7.126,1.593,-4.87,-3.108,-1.532,-5.92,-4.825,-2.668,-3.79,-1.511,0.000472,simple-snoring
-105.9,-5.711,-6.313,-1.294,-5.422,1.518,-0.1646,-4.11,-0.6486,-1.475,-2.904,1.208,-0.1178,simple-snoring
```

Figure 3.5: Labeled ARFF File

Four classes were chosen in order to label the instances in relation to a categorization of their physical meaning. These attributes are numeric in accordance with WEKA datatypes, they are real numbers and the classes can be described as follows:

- breathing, instances that represent one’s normal breathing sounds

- simple-snoring,instances that suggest sounds of indistinct snoring
- heavy-snoring, instances of rather loud and intense snoring
- apneic-event, instances of breathing halt

Next, the audio files are that were recorded with Raspberry Pi are imported and batch files are used for feature extraction. The output files are, again, unclassified. The algorithm used for the production of the unclassified arff files is illustrated in detail in a flowchart.

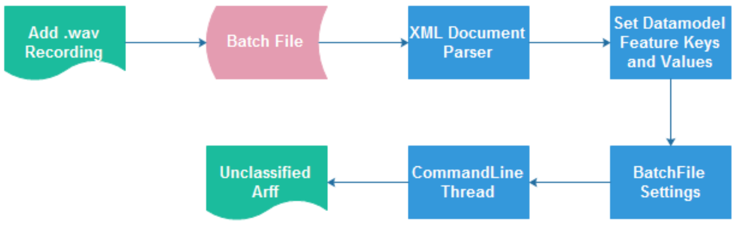


FIGURE 3.6: Feature Extraction Flow Chart

RecordingInfo: A class for holding references to audio recording files. The name of the .wav file is given as a parameter in it's methods.

Batch: Data type used to represent a batch file. Utilizes methods for setting the required recording and-selected window size, as well as Data model for the process.

XMLDocumentParser: A holder class for the XMLDocumentParser method. This method is a general pur-pose method for loading an XML file, testing that the file exists, validating it as a valid XML file, ensuring that it is of the correct type, parsing it and extracting its data into the required form. Informative error exceptions are thrown in a format that can be displayed directly to users.

DataModel: This class is used to set the features we want to utilize from a possible feature extraction. Using this class' methods, we can set feature keys and values.

CommandLineThread: A class which extends the Java's Thread Class properties. It is used to create a thread which will execute the feature extraction, taking as a parameter a batch instance. After this process is completed, an unlabeled arff file is constructed.

Following, WEKA classes are used for classification, and in particular J48 decision tree is used to predict the class of each instance. Finally, the duration of simple and heavy snoring is calculated, but most importantly the program is counting apneic events (>10 second breathing pauses) in relation to sample duration in order to conclude to apnea-hypopnea index and the severity of the patient's condition.

Chapter 4

Results

4.1 Audio Recordings Observations

The process of recording some audio samples and the construction of the corresponding system using Raspberry Pi, for the purposes of this thesis, can be called simple, economic and convenient. The user simply needs to power up the computer that of course, must first be connected to the Internet.

The memory requirements for storing the samples are not a restriction. Samples are ultimately only stored temporarily in the Raspberry Pi itself and then are uploaded to cloud. In our case, dropbox was used, but any other service can be used (owncloud, own server, amazon etc). However, even if the requirements in memory are limiting, it does not particularly affect the system that was constructed, since the way in which the recording of the samples is carried out requires:

Duration	Channels	Sampling Rate	Bits	Memory Occupied
1min	1	16000 Hz	16	**1.83 MB**
5min	1	16000 Hz	16	**9 Mb**

Table 4.1: Memory occupied for Recordings

Consequently, the output files can be characterized as studio quality, with a bit-rate at 256 kbps (bit-rate as in information stored per unit of time).

After the recording of each sample is complete, the wav file is automatically deleted and releases the memory occupied in the SD card of the Raspberry Pi. So, even if it is desirable to record longer audio files or more small samples, it is possible. Moreover, the selected sampling rate creates files with high sound quality, but occupies little memory. Besides, selecting one channel is common in speech applications and subsequently facilitates the feature extraction of the samples.

4.2 Feature Extraction Application

4.2.1 Model Evaluation

There is a substantial amount of research with machine learning algorithm such as Bayes Network, Radial Basis Function, Decision tree and pruning, Single Conjunctive Rule Learner and Nearest Neighbors Algo-rithm. Attaining the correct diagnosis of certain important information is a major problem in medical sci-ence. Machine learning resolves such difficulties with the help of several artificial intelligence algorithms performing as classifiers. To gauge and investigate the performance on the selected classification meth-ods or algorithms namely Bayes Network Classifier, Radial Basis Function, Decision Tree with pruning, Single Conjunctive Rule Learner and Nearest Neighbor, the same experiment procedure as suggested by WEKA was used. In WEKA, all data is considered as instances and features in the data are known as attrib-utes.
The performance results of the classifiers are partitioned into several sub items for easier analysis and evaluation. On the first part, correctly and incorrectly classified instances will be partitioned in numeric and percentage value and subsequently Kappa statistic, mean absolute error and root mean squared error will be in numeric value only. The relative absolute error and root relative squared error in percentage for references and evaluation. The results of the simulation are shown in Tables 4.2 and 4.3 below. Table 4.2 mainly summa-rizes the result based on accuracy for each classifier. Meanwhile, Table 4.3 shows the result based on error. Figure 4.1 is the graphical representation of the algorithms' performance. From the above tables it is clear that J48 was chosen as the best classifier.

	Correctly Classified Instances %	**Kappa Statictic**
CONJUNCTIVE RULE	77.6185	0.494
OneR	86.3466	0.7127
SVM	86.5337	0.7059
BAYES NET	85.6608	0.7077
NAIVE BAYES SIMPLE	84.5387	0.6747
NAIVE BAYES	84.5387	0.6744
J48	94.0773	0.8732

TABLE 4.2: Results of each Classifier

	Mean Absolute Error	**Root Mean Squareð Error**	**Precision**	**Recall**
CONJUNCTIVE RULE	0.1620	0.2852	0.641	0.776
OneR	0.0683	0.2613	0.861	0.863
SVM	0.2633	0.3323	0.856	0.865
BAYES NET	0.0792	0.2441	0.857	0.857
NAIVE BAYES SIMPLE	0.0816	0.2585	0.837	0.845
NAIVE BAYES	0.0815	0.2585	0.837	0.845
J48	0.0509	0.1595	0.941	0.941

Table 4.3: Classifiers: Errors and Accuracy

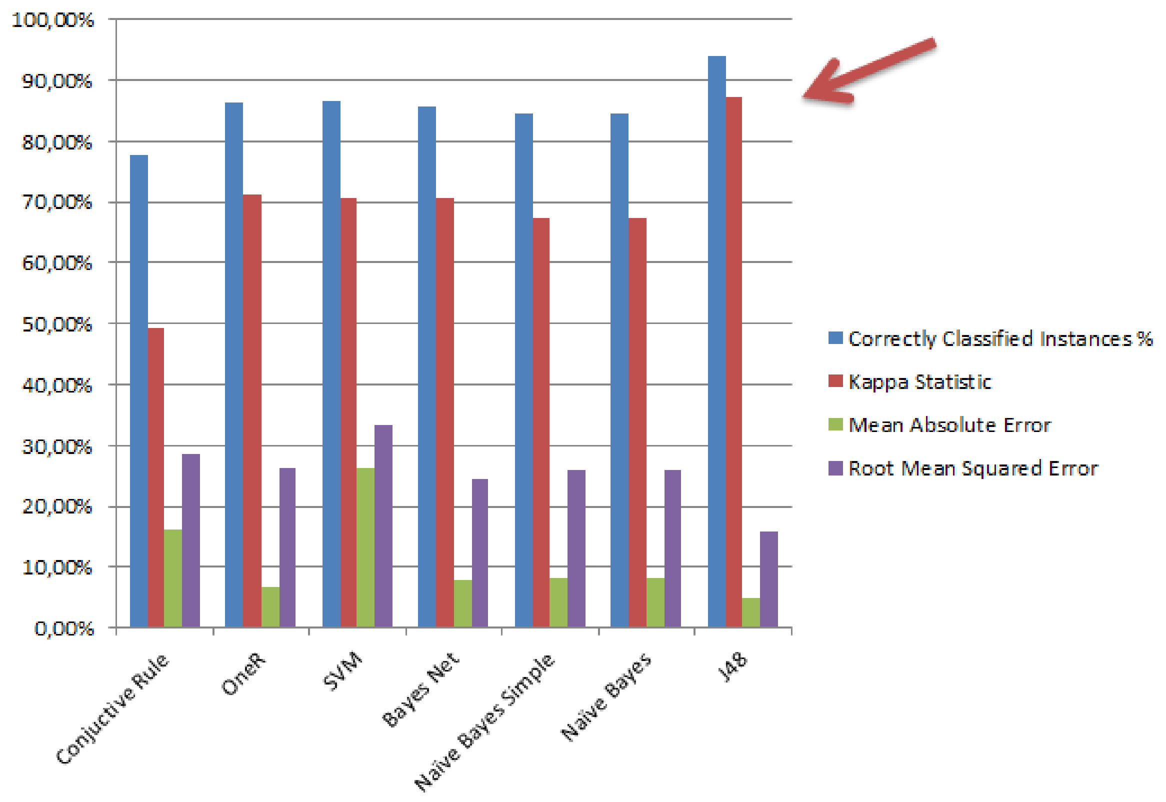


Figure 4.1: Classifiers: Performance Graphical Representation

As previously mentioned, this thesis uses the concept of a ”training set” so as to build a model capable of accurately predicting future unknown values. A dataset with known output values was manually labeled and used to build the model. The output from such that model is described below:

```
Listing 3. Output from WEKA's classification model
```

```
Correctly Classified Instances        1531               95.4489 \%
Incorrectly Classified Instances        73                4.5511 \%
Kappa statistic                          0.9045
Mean absolute error                      0.0401
Root mean squared error                  0.1417
Relative absolute error                 16.4341 \%
Root relative squared error             40.566  \%
Total Number of Instances             1604

=== Detailed Accuracy By Class ===

     TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
     0.824     0.013      0.929     0.824     0.873      0.942    breathing
     0.939     0.006      0.944     0.939     0.942      0.998    simple-snoring
     0.912     0.002      0.954     0.912     0.932      0.998    heavy-snoring
     0.992     0.087      0.961     0.992     0.976      0.969    apneic-event
Weighted Avg.    0.954     0.063      0.954     0.954      0.953      0.968

=== Confusion Matrix ===

    a    b    c    d   <-- classified as
  224    5    2   41 |    a = breathing
    6  153    1    3 |    b = simple-snoring
    3    3   62    0 |    c = heavy-snoring
    8    1    0 1092 |    d = apneic-event
```

The percentage of correctly classified instances is often called accuracy, in this case it is 95.45%, which is quite reliable. ROC Area, or area under the ROC curve, is a preferred measure, and one of the most important values output by Weka. An optimal classifier will have ROC area values approaching 1, in this case 0.968. The error rates are used for numeric prediction rather than classification.
Moreover, true positives rate, as in the instances correctly classified as a given class, in this model equals 0.954 and false positives rate in this case 0.063. Specificity can be calculated as 1-FP rate. Further interpreting the results of the model, precision is the proportion of instances that are truly of a class divided by the total instances classified as that class. Recall, also known as sensitivity in biomedical informatics, equals 0.954 and is the proportion of instances classified as a given class divided by the actual total in that class (equivalent to TP rate).

4.2.2 Experimental Data and Test Results

Ultimately, the user can perform the process described above using the graphical user interface that was developed (Figure 4.3). The user has the following options-buttons to make use of:

1. Upload an audio file in WAVE format
2. Select from 3 different window sizes

3. Run the program
4. Repeat

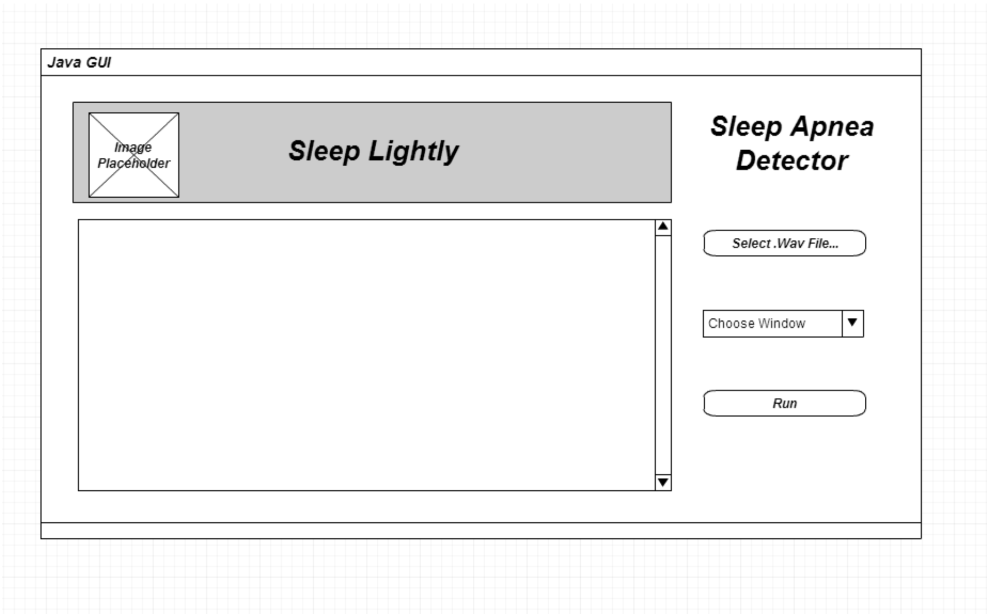
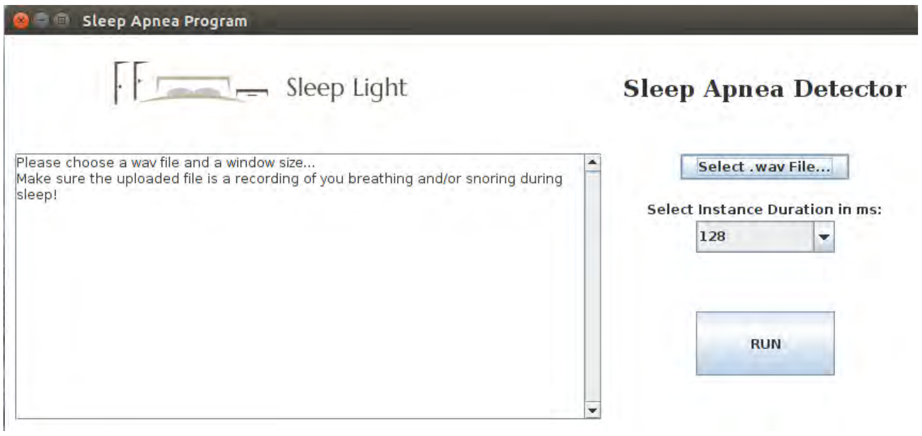


FIGURE 4.2: Java Application-GUI Wireframe



Please choose a wav file and a window size...
Make sure the uploaded file is a recording of you breathing and/or snoring during sleep!

Select Instance Duration in ms: 128

FIGURE 4.3: Java Application-GUI Implementation

As mentioned earlier in this paper, MFCCs were used as the feature for extracting data from unknown clips of sleep sounds. Furthermore, the classifier J48 was used to build the model of unknown values.
The following screenshots present the steps of uploading an unknown .wav file and the final parameters calculated after the feature extraction and class prediction stages.

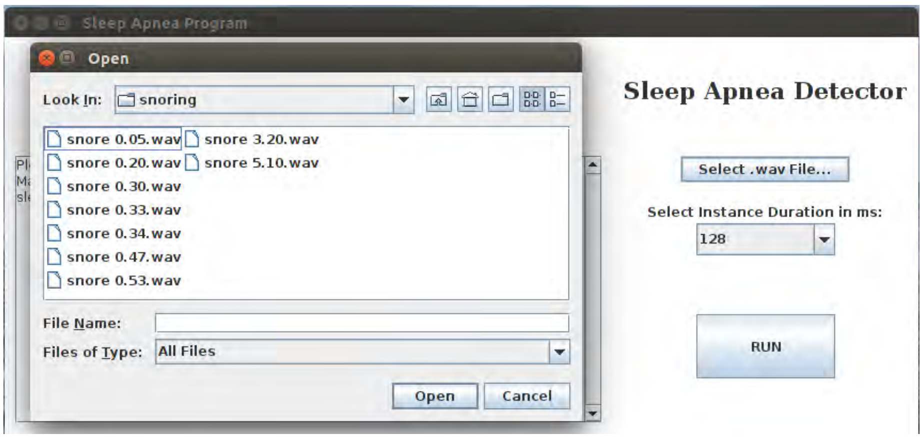
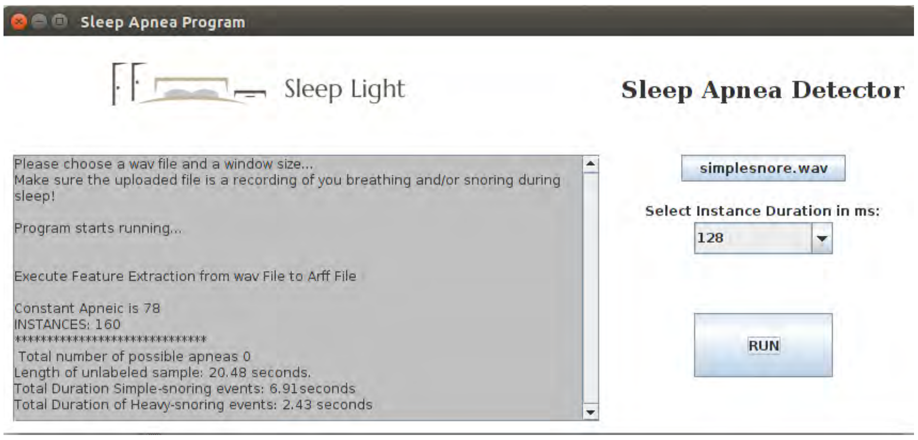


Figure 4.4: File Selection in Java Application



```
Please choose a wav file and a window size...
Make sure the uploaded file is a recording of you breathing and/or snoring during
sleep!

Program starts running...


Execute Feature Extraction from wav File to Arff File

Constant Apneic is 78
INSTANCES: 160
******************************
 Total number of possible apneas 0
Length of unlabeled sample: 20.48 seconds.
Total Duration Simple-snoring events: 6.91seconds
Total Duration of Heavy-snoring events: 2.43 seconds
```

Figure 4.5: Java Application-GUI Output

Finally, the model was tested using 10 audio files.The test was performed for 3 different window sizes, in particular, instances lasting 64ms,128ms and 256ms. The following tables present some indicative characteristics related to the content of these audio files and their relationship with sleep apnea events:

file #	**Apneas**	**Duration**	**simple-snore**	**heavy-snore**
1	0	20.48	6.59	0.64
2	1	30.46	3.97	0,00
3	0	34.05	17.02	3.26
4	1	34.43	12.48	2.56
5	1	48.00	26.24	6.08
6	1	53.82	13.31	4.54
7	2	55.3	12.86	1.22
8	6	138.11	44.22	1.73
9	6	201.86	23.94	49.34
10	0	310.27	190.85	48.32

Table 4.4: Application Test Results for Window Size:64ms

file #	**Apneas**	**Duration**	**simple-snore**	**heavy-snore**
1	0	20.48	6.91	2.43
2	1	30.46	4.61	0
3	0	34.05	13.31	3.58
4	1	34.43	15.87	0.13
5	0	48.00	11.78	12.16
6	0	53.89	9.86	10.5
7	1	55.3	8.89	0.13
8	6	138.11	28.42	10.5
9	6	201.86	12.67	60.54
10	0	310.27	120.83	92.29

Table 4.5: Application Test Results for Window Size:128ms

file #	Apneas	Duration	simple-snore	heavy-snore
1	0	20.48	8.45	0.26
2	1	30.46	5.38	0.26
3	0	34.05	20.22	0.51
4	1	34.43	14.85	0.51
5	1	48.00	31.23	1.79
6	1	54.02	22.02	4.01
7	2	55.3	14.59	0.00
8	5	138.11	44.54	0.00
9	8	201.86	13.06	64.51
10	0	310.27	216.58	27.14

Table 4.6: Application Test Results for Window Size:256ms

The audio files used to test the model and practically evaluate the application were of apneic as well as plain snoring samples. As illustrated in the above tables, files 1-7 and 10 had none or a few apneas (events of complete breath pause lasting 10+ seconds) while files 8 and 9 had 5 or more apneic events regardless of the window size used. These two samples, were actually characterized as apneic beforehand, making the class prediction of the developed model trustworthy.

Chapter 5

Conclusion-Future Work

The aim of this thesis has been to develop a portable self-diagnose sleep apnea detection system. The above goal was achieved by developing a low-cost sound recording system using the Raspberry Pi computer and an 3.5 mm jack microphone, as described in Chapter 3. Following, an application was written in Java to reach a conclusion regarding one's possibility of suffering from obstructive sleep apnea or not.

Using the developed GUI the potential patient could upload their uncategorized audio samples of breathing and/or snoring during sleep and get an estimate of apneic events among the file duration as well as data concerning their snoring. In more detail, the application measured the plethora and duration of apneic events, and also the duration of simple and heavy snoring over the audio sample. The suggested system, could consecutively run numerous audio samples and with 3 different options of window size, as in duration per instance.

The suggested methodology has been evaluated with unknown audio samples as input. A 10-subject validation has been carried out illustrating a rather good accuracy, as presented in Chapter 4.

The expanded purpose of this thesis, however, has been long-term sleep and subsequently sleep apnea measurement. Unlike, short-term sleep measurements, lasting several days or weeks, this system can be used for long-term sleep tracking, since it is portable and does not require the presence of a specialized technician. This thesis suggests an unobtrusive sleep tracking method, where the primary measured parameter is breathing and snoring sounds. Currently, as it has been portrayed in Chapter 2, most sleep self-tracking devices in the market are based on wearable technologies such as wristbands, watches or headbands, yet this technique may have an eminent role in the future, since it causes no disturbance to the user's sleep. The comfort offered by this system is an essential advantage as it increases the possibility of making sleep tracking a long-lasting habit.

In addition, it should be emphasized that this is a self-diagnose system and the results exported cannot safely conclude whether the potential patient suffers from or not. Including an oximeter to measure oxygen saturation, or an Arduino attached heart rate sensor, can add up in making the system even more accurate. Nonetheless, these may be future expansions of this thesis. Finally, since the recorded audio samples are already uploaded to the cloud from the Raspberry Pi, another improvement of the system would be the ability to use the developed application online, too.

In short, research in unattended and also non-intrusive sleep self-tracking is a productive field. There is potential for a considerable public health impact by raising awareness among people in order to help improve their sleep patterns and learn more about sleep-disordered breathing. The prime ambition is to make sleep tracking a user-friendly and prevalent activity aiming In the advancement of one's sleep and consequently well-being. This thesis contributes in reaching that target.

Bibliography

[1] Mack, David C., et al, *A passive and portable system for monitoring heart rate and detecting sleep apnea and arousals: Preliminary validation* Distributed Diagnosis and Home Healthcare, 2006. D2H2. 1st Transdisciplinary Conference on. IEEE, 2006.

[2] American Academy of Sleep Medicine, and Conrad Iber, *The AASM manual for the scoring of sleep and associated events: rules, terminology and technical specifications* American Academy of Sleep Medicine, 2007

[3] Chesson Jr, M. D., et al *Practice parameters for the use of actigraphy in the assessment of sleep and sleep disorders: an update for 2007*, Sleep 30.4 (2007): 519

[4] Force, Adult Obstructive Sleep Apnea Task, and American Academy of Sleep Medicine, *Clinical guideline for the evaluation, management and long-term care of obstructive sleep apnea in adults*, Journal of clinical sleep medicine: JCSM: official publication of the American Academy of Sleep Medicine 5.3 (2009): 263

[5] Kushida, Clete A., et al, *Practice parameters for the indications for polysomnography and related procedures: an update for 2005* , Sleep 28.4 (2005): 499-521.

[6] Quinlan, J. Ross, *C4. 5: programs for machine learning*, Elsevier, 2014

[7] McEnnis, Daniel, Cory McKay, and Ichiro Fujinaga, *Overview of on-demand metadata extraction network (omen)*, Proceedings of the Seventh International Conference on Music Information Retrieval (ISMIR'06). 2006.

[8] 8. Richardson, Matt, and Shawn Wallace, *Getting Started with Raspberry Pi*, O'Reilly Media, Inc., 2012.

[9] *Dropbox Uploader*

[10] Ittichaichareon, Chadawan, Siwat Suksri, and Thaweesak Yingthawornsuk, *Speech recognition using MFCC.*, International Conference on Computer Graphics, Simulation and Modeling (ICGSM'2012) July. 2012.

[11] Skomro, Robert P., et al., *Outcomes of home-based diagnosis and treatment of obstructive sleep apnea*, CHEST Journal 138.2 (2010): 257-263.

[12] Tice, Jeffrey A, *Portable Devices Used for Home Testing in Obstructive Sleep*, California Technology Assessment Forum, 2009.

[13] Nieto, F. Javier, et al, *Association of sleep-disordered breathing, sleep apnea, and hypertension in a large community-based study*, Jama 283.14 (2000): 1829-1836

[14] *The Raspberry Pi* 2014

[15] Hall, Mark, et al. , *The WEKA data mining software: an update*, ACM SIGKDD explorations newsletter 11.1 (2009): 10-18.

[16] Blanco, José Luis, et al. , *Improving automatic detection of obstructive sleep apnea through nonlinear analysis of sustained speech*, Cognitive Computation5.4 (2013): 458-472

[17] Calisti, M., et al, *Automatic detection of snore episodes from full night sound recordings: home and clinical application*, Proceedings of the 3rd advanced voice function assessment international workshop. 2009.

[18] Goldshtein, Evgenia, Ariel Tarasiuk, and Yaniv Zigel , *Automatic detection of obstructive sleep apnea using speech signals.*, Biomedical Engineering, IEEE Transactions on 58.5 (2011): 1373-1382.

[19] Azarbarzin, Ali, and Zahra Moussavi , *Automatic and unsupervised snore sound extraction from respiratory sound signals*, Biomedical Engineering, IEEE Transactions on 58.5 (2011): 1156-1162.

[20] Flemons, W. Ward, et al , *Home diagnosis of sleep apnea: a systematic review of the literature: an evidence review cosponsored by the American Academy of Sleep Medicine, the American College of Chest Physicians, and the American Thoracic Society*, CHEST Journal 124.4 (2003): 1543-1579.

[21] Duckitt, W. D., S. K. Tuomi, and T. R. Niesler., *Automatic detection, segmentation and assessment of snoring from ambient acoustic data.*,Physiological measurement 27.10 (2006): 1047.

[22] Rechtschaffen, A, Kales, A (eds)., *A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects*,U.S. Public Health Service, U.S. Government Printing Office, Washington D.C. 1968

Code-Java

.

```
import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.io.*;
import java.text.DecimalFormat;
import java.util.Date;

import jAudioFeatureExtractor.DataModel;
import jAudioFeatureExtractor.ACE.DataTypes.Batch;
import jAudioFeatureExtractor.DataTypes.RecordingInfo;
import jAudioFeatureExtractor.ACE.XMLParsers.XMLDocumentParser;
import jAudioFeatureExtractor.CommandLineThread;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.trees.J48;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instances;

public class ptixiaki extends JFrame
{
        final int FRAME_WIDTH =600 ;
        final int FRAME_HEIGHT =400;
        final int RUN_WIDTH =130;
        final int RUN_HEIGHT =60;
        final int  TEXT_AREA_WIDTH =550;
        final int  TEXT_AREA_HEIGHT = 250;
        final int DROPDOWN_WIDTH = 130;
        final int DROPDOWN_HEIGHT = 30;

        //Gui Variables
        public JFrame f;        //frame is like a window
        public JPanel panel;    //displays all the components on the frame
        public ImageIcon logo;
        public JLabel logo_label,label_window_size_prompt;
        public JLabel label_welcome_text;
        public JButton openButton,button_run_program;
        public JScrollPane areaScrollPane ;
        public JTextArea text_area;
        public JTextField gui_input_window_size;
        public JComboBox optionList ;
        public BufferedReader buf_reader;
        public JFileChooser fileChooser;
        public File file;


        File[] files ;
        int returnval;
        public String fileName,filePathName;
        public String string_input_window_size;
        public String selectedItem;
         //Create the combo box
        public String[] optionStrings = { "64", "128", "256" };


        //Feature Extraction variables
        //Initialize variables here in order to be visible anywhere
        Batch b = null;
        Evaluation eTest = null;
        Classifier cls=null;
        Instances isTrainingSet=null;
        BufferedReader inputReader=null;
        //Window size set by user
        public int window_size;

        //Constructor
        public ptixiaki()
        {

                fileChooser=new JFileChooser();
                openButton = new JButton( new AbstractAction("Select .wav File...")
                {
                        @Override
                        public void actionPerformed(ActionEvent e)
                        {
                                //SELECT WAV ACTION
                                if(e.getSource()==openButton)
                                {
                                        returnval= fileChooser.showOpenDialog(null);
                                        //if someone has successfully chosen a file
                                        if(returnval==JFileChooser.APPROVE_OPTION)
                                        {
                                                text_area.setBackground(Color.WHITE);
                                                files = new File[1];
                                                file=fileChooser.getSelectedFile();
                                                files[0]=file;

                                                fileName=file.getName();
                                                openButton.setText(fileName);
                                                filePathName=file.getAbsolutePath();
                                                String extension = fileName.substring(fileName
   ↪ .lastIndexOf(".") + 1, fileName.length());
                                                String wavExtension = "wav";
                                                if (!extension.equals(wavExtension))
                                                {
```

```
                                                JOptionPane.showMessageDialog(null,
↪ "Choose a .wav file!!");
                                      }
                              }

                      }
              }
        });

        button_run_program = new JButton( new AbstractAction("RUN")
        {
            @Override
            public void actionPerformed( ActionEvent e ) {

            //RUN BUTTON ACTION
            if(e.getSource()==button_run_program)
            {

                text_area.append("\n\nProgram starts running... \n\n");
                        //Read the selected value from optionList and print it

                //-----------MAIN CODE---------
                //If a wav file has been selected then run program
                if(fileChooser.getSelectedFile()==null)
                        JOptionPane.showMessageDialog(null, "Please select a .wav file
↪  before running the program!");
                else
                {
                        runprogram();
                }
            }
         }
    });

        button_run_program.setPreferredSize(new Dimension(RUN_WIDTH, RUN_HEIGHT));

        //------Create a panel which will be assigned to window----------------
        panel=new JPanel(new GridBagLayout());
        panel.setBackground( Color.WHITE);
        //-----------------------------------------------------------------

        //We create a frame window
        f = new JFrame("Sleep Apnea Program");
        f.setVisible(true);
        f.setSize(FRAME_WIDTH,FRAME_HEIGHT);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.getContentPane().add(panel,BorderLayout.CENTER);
        //------------------------------------------------------------------

        //------Create an ImageIcon and assign it to a Label----------------
        logo= new ImageIcon(getClass().getResource("/resources/sleep.png"));
        logo_label=new JLabel (logo);

        //------Label to show Title: "Sleep Apnea Detector"----------------
        label_welcome_text = new JLabel("Sleep Apnea Detector");
        label_welcome_text.setFont(new Font("Serif", Font.BOLD, 22));
        //-------------------------------------------------------------------

        //------Label to show "Enter window size:"---------------------------
        label_window_size_prompt = new JLabel("Select Instance Duration in ms:");
        //-------------------------------------------------------------------

        //------Text-Area and assignment to a scrollpanel--------------------
        text_area = new JTextArea("Please choose a wav file and a window size...\nMake
↪  sure the uploaded file is a recording of you breathing and/or snoring during sleep!");
        text_area.setLineWrap(true);

        text_area.setWrapStyleWord(true);

        areaScrollPane = new JScrollPane(text_area);
        areaScrollPane.setVerticalScrollBarPolicy(JScrollPane.
↪ VERTICAL_SCROLLBAR_ALWAYS);
        areaScrollPane.setPreferredSize(new Dimension(TEXT_AREA_WIDTH,
↪ TEXT_AREA_HEIGHT));

        //-------------------------------------------------------------------

        //-----Window Size ComboBox(Drop-down)--------------------------------
        optionList = new JComboBox(optionStrings);
        optionList.setSelectedIndex(1);
        optionList.setPreferredSize(new Dimension(DROPDOWN_WIDTH,DROPDOWN_HEIGHT));
        optionList.setFocusable(false);

        //-----------------------------------------------------------------

        //now add panel to frame at certain position!
        f.add(panel);

        //Build the desired layout using GridBagConstraints
        GridBagConstraints c=new GridBagConstraints();
        c.insets=new Insets(10,10,10,10);
        c.gridx=0;
        c.gridy=0;
        panel.add(logo_label,c);
        c.gridx=1;
        c.gridy=0;
        panel.add(label_welcome_text,c);
        c.gridwidth=1;
        c.gridx=1;
        c.gridy=1;
        c.anchor=GridBagConstraints.LINE_END;
        c.anchor=GridBagConstraints.CENTER;
        panel.add(openButton,c);
        c.gridwidth=1;
        c.gridx=0;
        c.gridy=1;
        c.gridheight=4;
        panel.add(areaScrollPane,c);
        c.gridheight=1;
        c.gridx=1;
```

```
            c.gridy=2;
            c.anchor=GridBagConstraints.PAGE_END;
            c.insets=new Insets(10,0,0,0);
            panel.add(label_window_size_prompt,c);
            c.gridx=1;
            c.gridy=3;
            c.insets=new Insets(3,0,0,0);
            c.anchor=GridBagConstraints.CENTER;
            panel.add(optionList,c);
            c.gridx=1;
            c.gridy=4;
            c.anchor=GridBagConstraints.CENTER;
            panel.add(button_run_program,c);
            f.pack();

    }





public void runprogram()
    {

    try
    {
            //Read file for use as a Training Set and create Instances isTrainingSet
            selectedItem = (String) optionList.getSelectedItem();

            if(selectedItem.equals("64"))
        {
                    inputReader=new BufferedReader(new FileReader(getClass().getResource("
↪ /resources/1024.arff").getFile()));
                    window_size=1024;
        }
            else if(selectedItem.equals("128"))
        {
                    inputReader=new BufferedReader(new FileReader(getClass().getResource("
↪ /resources/2048.arff").getFile()));
                    window_size=2048;
        }
            else if(selectedItem.equals("256"))
        {
                    inputReader=new BufferedReader(new FileReader(getClass().getResource("
↪ /resources/4096.arff").getFile()));
                    window_size=4096;
        }
            else
            {
                    System.out.println("SELECT WINDOW SIZE!!!");
                    text_area.append("\nSELECT WINDOW SIZE!!!");
            }


                    isTrainingSet=new Instances(inputReader);
                    isTrainingSet.setClassIndex(isTrainingSet.numAttributes()-1);
```

```
                        //Create Classifiers and evaluate

                        cls=new J48();
                        cls.buildClassifier(isTrainingSet);
                        eTest=new Evaluation(isTrainingSet);

                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        //Set audio file(wav) and batch file(xml)
        RecordingInfo recording= new RecordingInfo(filePathName);

        File batch=new File(getClass().getResource("/resources/mfcc.xml").getFile());

         if (!batch.exists())
         {
                 System.out.println("Batch file does not exist");
             text_area.append("\nBatch file does not exist");
         }
         else
         {
                 //Create variables/objects representing file data
                 Object[] o = new Object[]{};
                 try {
                                 o=(Object[]) XMLDocumentParser.parseXMLDocument(getClass().
↪ getResource("/resources/mfcc.xml").getFile(),"batchFile");

                                 b = (Batch)(o[0]);
                     //Set xml with features to extract

                                 DataModel dm=new DataModel(getClass().getResource("/resources/
↪ features.xml").getFile(),null);
                                 dm.featureKey=new FileOutputStream(new File(b.getDestinationFK
↪ ()));
                                 dm.featureValue=new FileOutputStream(new File(b.
↪ getDestinationFV()));
                                 b.setDataModel(dm);

                                 b.setRecording(new RecordingInfo[]{recording});
                                 b.setWindowSize(window_size);
                     //Create thread which executes the feature extraction
                                 CommandLineThread clt = new CommandLineThread(b);

                                 System.out.print("Execute Feature Extraction from wav File to
↪ Arff File\n");
                                 text_area.append("\nExecute Feature Extraction from wav File
↪ to Arff File\n");
                                 clt.start();

                                 while (clt.isAlive())
                                 {
                                         if (System.in.available() > 0)
```

```
                                    {
                                            clt.cancel();
                                    }
                                    clt.join(1000);

                            }

                            } catch (Exception e)
                            {
                             System.out.println("Error parsing the batch file");
                                     System.out.println(e.getMessage());
                                     text_area.append("\nError parsing the batch file");
                                     text_area.append("\n"+e.getMessage());
                            }

            //Create object to read data from arff file
                    BufferedReader testData_inputReader;
                            try {
                                    //Create Instances isTestingSet from the extracted
↪ arff file
                                    testData_inputReader = new BufferedReader(new
↪ FileReader(b.getDestinationFV()));
                                    Instances isTestingSet=new Instances(
↪ testData_inputReader);


                            //--Create 4 categories in order to classify each Instance--//

                            FastVector fvClassVal=new FastVector(4);
                            fvClassVal.addElement("breathing");
                            fvClassVal.addElement("simple-snoring");
                            fvClassVal.addElement("heavy-snoring");
                            fvClassVal.addElement("apneic-event");
                            Attribute ClassAttribute=new Attribute("class",fvClassVal);


                            isTestingSet.insertAttributeAt(ClassAttribute,isTestingSet.
↪ numAttributes());

                            isTestingSet.setClassIndex(isTestingSet.numAttributes()-1);

                            eTest.evaluateModel(cls, isTrainingSet);
                            //Load Testing set to labeled Instances
                            Instances labeled=new Instances(isTestingSet);

                            //Counters and flags for apneic events
                            int counter_apneic=0, counter_heavy=0,counter_simple=0,apneas
↪ =0;

                            int constant_apneic=0;

                            boolean flag=false;
                            if(selectedItem.equals("64"))
                    {
                                    constant_apneic=156;
                    }
                            else if(selectedItem.equals("128"))
                    {
                                constant_apneic=78;
                    }
                          else if(selectedItem.equals("256"))
                    {
                                constant_apneic=39;
                    }
                          text_area.append("\nConstant Apneic is "+constant_apneic);
                          //----ADD CODE FOR CORRECT VARIABLE WHICH DEPENDS ON
↪ WINDOW_SIZE----

                          //
↪ -------------------------------------------------------------------
                          //Loop to fill labeled Instances and count APNEIC EVENTS
                          DecimalFormat df = new DecimalFormat();
                          df.setMaximumFractionDigits(2);
                          int counter_max=78;
                          int ct=0;
                          text_area.append("\nINSTANCES: "+isTestingSet.numInstances());
                          for(int i=0;i<isTestingSet.numInstances();i++)
                          {
                                double clsLabel=cls.classifyInstance(isTestingSet.
↪ instance(i));
                                labeled.instance(i).setClassValue(clsLabel);
                    //If category of Instance is apneic event-->increase counter_apneic
                    //if there are more than  continuous apneic events, raise flag

                                if(clsLabel==3.0 )
                                {
                                        counter_apneic++;
                                        if(counter_apneic>=constant_apneic)
                                        {
                                                if(flag==false)
                                                        ct=i-constant_apneic;
                                                flag=true;
                                                counter_max++;

                                        }
                                }
                                else
                                {
                                        if(flag==true)
                                                {
                                                        counter_apneic=0;
                                                        flag=false;
                                                    //Print the number of Instance
↪ where the last apneic event happened
                              System.out.println(i-1 +" Possible apneic event!");

                              text_area.append("\nPossible apneic event from : "+df.
↪ format(ct*(b.getWindowSize()/b.getSamplingRate()))+" to: "+df.format((i-1)*(b.
↪ getWindowSize()/b.getSamplingRate())));
                              apneas++;
                              ct=0;
                                                }
                                        }

                                        if(clsLabel==1.0)
                                                counter_simple++;
                                        if(clsLabel==2.0)
                                                counter_heavy++;
                                }

                                double file_duration_in_seconds = isTestingSet.numInstances()
↪ *(b.getWindowSize()/b.getSamplingRate());
                                double instance_duration = (b.getWindowSize()/b.
↪ getSamplingRate());
                                System.out.println("******************************\n ");
                                System.out.println("Total number of possible apneas "+ apneas)
↪ ;
                                        System.out.println("Length of unlabeled sample: " +
↪ isTestingSet.numInstances()*(b.getWindowSize()/b.getSamplingRate())+" seconds.");
                                        text_area.append("\n******************************\n "
↪ +"Total number of possible apneas "+ apneas+"\nLength of unlabeled sample: " + df.
↪ format(isTestingSet.numInstances()*(b.getWindowSize()/b.getSamplingRate()))+" seconds."
↪ );
                                        System.out.println("\nNumber of Simple-snoring events:
↪  "+counter_simple);
                                        System.out.println("\nNumber of Heavy-snoring events:
↪ "+counter_heavy);

                                        //Duration of simple-snoring events
                                        double duration_of_simple_snoring_events=
↪ instance_duration*counter_simple;
                                        double duration_of_heavy_snoring_events=
↪ instance_duration*counter_heavy;

                                        text_area.append("\nTotal Duration Simple-snoring
↪ events: "+df.format(duration_of_simple_snoring_events)+ "seconds");
                                        text_area.append("\nTotal Duration of Heavy-snoring
↪ events: "+df.format(duration_of_heavy_snoring_events)+ " seconds");

                                        double index_of_simple_snoring =
↪ duration_of_simple_snoring_events/file_duration_in_seconds;
                                        double index_of_heavy_snoring =
↪ duration_of_heavy_snoring_events/file_duration_in_seconds;


                                //Write the labeled data to a new arff file
                                        Date date = new Date();
                                BufferedWriter writer=new BufferedWriter(new FileWriter(date.
↪ toString()+".arff"));

                                writer.write(labeled.toString());
                                writer.newLine();
                                writer.flush();
                                writer.close();
                                text_area.setBackground(Color.LIGHT_GRAY);
                                } catch (FileNotFoundException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                } catch (Exception e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                }
        }



    public static void main(String[] args)
    {
        new ptixiaki();
    }


}
```

Raspberry Pi-Bash Script

```
#!/bin/bash
# Unattended podcast record process
# Author: Raisa Angelidou
# File Recording and upload to dropbox then removes original file from pi and shuts down

DROPBOX_UPLOADER=/home/pi/Dropbox-Uploader/dropbox_uploader.sh

for i in {1..5}
do
{
FILENAME=$(date +"%Y%m%d_%H%M")

#record for 5 minute
sudo arecord -r 16000 -d 300 -f S16_LE /home/pi/${FILENAME}.wav

[ $? -eq 0 ] || exit $?

sudo $DROPBOX_UPLOADER -f /root/.dropbox_uploader upload /home/pi/${FILENAME}.wav ${FILENAME}.
    ↪ wav

[ $? -eq 0 ] || exit $?

sudo rm /home/pi/${FILENAME}.wav

sleep 10
}
done

sudo shutdown -h now

exit
```