Identifying Environmental Sounds with an IoT Device in Smart Cities

Zaw Moe Htat

Supervisor: Joshua Leask

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) (Software)

School of Electrical Engineering and Computing

The University of Newcastle, Australia

15th November 2019

Word count: 11,750

Signed: 

Student: Zaw Moe Htat

Signed: 

Supervisor: Joshua Leask

Abstract

Sound classification is the type of supervised machine learning techniques to predict the category that the sound belongs to. It has been used in various types of applications such as speech detection and music analysis. As this area of research grows, researchers have been working on identifying urban noises with the Internet of Things (IoT) technology as part of the smart city projects. The application of which can be used as a surveillance method throughout the cities. The main objective of this project is to train a classification model that is deployed on Raspberry Pi as an IoT edge device to detect environmental sounds (which are dynamically set from the web application) correctly and send the signal to the cloud server when an event is triggered. In this project, two popular supervised machine learning algorithms, Support Vector Machine (SVM) and k-Nearest Neighbours (KNN), are used for sound classification. To extract audio features, we use Mel-frequency cepstral coefficients (MFCCs). In our experiments, we use a dataset of 8732 samples with 10 different classes (such as car horn, dog bark, jackhammer, siren and so on) from UrbanSound8K to train the classification models. During the experiment, we explore the parameters of both SVM and KNN to get the best parameter values for the classification of the dataset. We achieve accuracy of 56.8% and 49.98% for SVM and KNN respectively. We design the architecture that allows a web application to set the dynamic sound for Raspberry Pi to listen to and send the signal to the cloud server when the event is detected.

Keywords: Machine Learning, Internet of Things (IoT), Support Vector Machine (SVM), k-Nearest Neighbours (KNN), Mel-frequency cepstral coefficients (MFCCs), Web Application, Cloud Server, Raspberry Pi

List of Contribution

My contributions to this project includes the following:

- Provided theoretical review of Support Vector Machine and k-Nearest Neighbours
- Provided a literature review of related works on urban sound classification using IoT units
- Trained the model to classify environmental sound
- Experimented with two machine learning algorithms to study the performance
- Designed and implemented the system for IoT based urban sound classifier including web application development

Acknowledgement

I would first like to thank my project supervisor Joshua Leask of Ampcontrol for his time and support throughout the project. His help and explanation on every problem I had make this project completed successfully.

I would also like to thank University of Newcastle for providing me an opportunity to pursue my passion in Software Engineering and guiding me to the right direction. Also, without the efforts of all the staffs at University of Newcastle to help me through all difficulties I had throughout my undergraduate degree program, my study in University of Newcastle would not have been easy.

Finally, I must express my very profound gratitude to my parents for their unconditional love and support. Especially to my father for making my study in Software Engineering at University of Newcastle, Australia possible. Without my parents, I would not be able to achieve this accomplishment. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In order to monitor the road events and keep the vicinity safe, video camera recording, such as CCTV, has been widely used as a surveillance method across the cities. Most of the time, it is required for a person to monitor the screen to identify an event. It sometimes can be quite inefficient when there is more than one area to focus on. Having a machine to monitor an event without the need of people is possible with the machine learning technology. Machine learning techniques have been used in many visual recognition applications such as objects detection in images and videos for computer vision. However, it requires higher computational power to train the images and videos, and process the detection of an event in real-time with the footage from the video camera.

Recently, researchers have been focusing on audio recognition using machine learning methods which have been applied in speech and music processing. Using an audio recognition technique on environmental sound analysis has been challenging for researchers in the area of research to be practically applicable because the environmental sound can be described in neither music nor speech [21]. However, there are a number of environmental sound datasets available in different types of events such as car horns, gunshots, and siren, which can be used to improve the performance of environmental sound recognition. Therefore, it has been motivated to apply this technique in smart cities to identify the event by detecting the environmental sound as an audio surveillance system, which requires less computational power in contrast to surveillance system with video camera

The Internet of Things (IoT) has been a popular technology in many smart cities projects.

It may also be used in an audio surveillance system for reporting to a responsible organisation such as the police department, emergency department, and maintenance units when a particular event has been detected.

1.2 Objectives

Machine learning techniques are used efficiently and effectively to classify urban sounds. The main goal of this project is to explore machine learning techniques and Internet of Things applications.

Major objectives of this project are listed as follows.

- To explore machine learning classification algorithms, mainly focus on Support Vector Machine and k-Nearest Neighbours.
- To completely train the machine learning model for classifying urban noise correctly.
- To successfully deploy machine learning inference and integrate IoT functionalities on Raspberry Pi such that it can listen to a dynamic array of sounds and send the signal when the sound is detected.
- To develop a cloud server and web application for the management of the dynamic list of sounds by the administrator.

1.3 Approach

In this project, our approach to the solution of the problems is by following the methods found in the research paper "*A Machine Learning Driven IoT Solution for Noise Classification in Smart Cities*" [1] to train the machine learning model. We extend the paper by implementing IoT functionalities on Raspberry Pi. We then develop the cloud server and web application to set the sound for Raspberry Pi to listen to and test the IoT functionalities to simulate the communication between edge device and end-user such as sending messages to an organisation when an event is triggered.

1.4 Organisation of the Thesis

This thesis shows how machine learning techniques and Internet of Things applications can be used in smart cities for environmental sound detection. The organisation of this thesis is as follows. Chapter 2 covers the literature review on works that are related to this project. The literature review breaks down into four main different topics.

Chapter 3 introduces the technical background of the project. We first explain the machine learning concepts with three different types of learning methods. Then, we describe the hardware specification of Raspberry Pi for computation and Mic-Hat for microphone recording. In the last section of Chapter 3, we explain the overview concept of the Internet of Things.

In Chapter 4, we describe the details of how sound is classified using machine learning algorithms. We explain how feature extraction works on audio. We also describe the dataset used for training the model.

Chapter 5 explains the development methodology used for this project. In this chapter, we describe a high-level overview of system design and architecture. We then explain the hardware and software used in the development and experiment. We also describe the detail implementation of a sound classifier, deployment on Raspberry Pi and the development of a cloud server and web application.

In chapter 6, we investigate the classification results obtained from the experiments. We also walk through the demonstration of the project with example screenshots. Lastly, chapter 7 gives a summary of the Thesis as well as the future works that can be done as an extension of this project.

Chapter 2

Related Work

2.1 Sound Classification with Machine Learning Approach

Sound classification has been successfully used in many applications, such as identifying frog sound [12] and urban noise [1]. In both [12] and [1], two popular supervised machine learning algorithms, k-nearest neighbor (KNN) and support vector machine (SVM), were used as a classifier with Mel-Frequency Cepstrum Coefficient (MFCC), which is widely used in sound classification. The additional two tree-based methods, bootstrap aggregating and random forest, were used to experiment in [1]. Other machine learning techniques such as unsupervised learning and semi-supervised learning are also used in sound classification [40]. Salamon et al [28] proposed unsupervised feature learning using spherical k-means algorithm and log-mel-spectrograms to learn the feature. The experiment was done using tree-based classifier, random forest (500 trees), with learned features. Apart from traditional classification algorithm, deep learning techniques such as Convolutional Neural Network (CNN) has been used in environmental sound classification in the last couple of years [6]. As a result, it is observed that CNN has outperformed common classification methods.

In approaches of the related work, Convolutional Neural Network is generally deep and heavy. Therefore, it requires higher computational power. Since low computational power device such as Raspberry Pi is used for this project, our approach to training classification model is using KNN and SVM methods by complementing the approach done in the study of IoT solutions for noise classification in smart cities.

2.2 Audio surveillance in Smart Cities

As part of smart cities project, audio monitoring has been conceptualized and developed around the cities of Europe in order to enhance the life style of citizens [33]. Urban noise affects the health risk and the quality of life of citizens [1]. Therefore, acoustic monitoring system has been developed to control and manage the environmental noise pollution which is generated by road traffic, loud music, airplanes and so on [42]. To reduce the cost, Zappatore et al [42] proposed Android based Mobile Crowd-Sensing system to monitor urban noise and have it sent to server for further management. Due to the legality of geolocation, sound audio surveillance has been a challenging project to monitor the area for security purposes. Recently, System on Chip (SoC) has been used in surveillance cameras with in-built video and audio analytics to classify sounds and detect potential threats or events for public security such as gunshot, screams, glass breaks, explosion and so on [13].

Apart from noise pollution measurement and cities security monitoring, sound surveillance has been used in smart cities for traffic flow monitoring [7]. The EAR-IT project funded by EU FP7 [33] has successfully been deployed over the city to detect the traffic incidents. With the IoT technologies, EAR-IT devices surveilling across the city area are able to send the signal to other applications such as when the siren is detected, the signal is sent to the traffic light controller to change the traffic light for ambulance to pass through, and the devices can also be used to detect the street events such as concerts and send the message to end-user's applications [7].

Therefore, surveillance using sound classification with computational intelligence has been a growing area of research for various types purposes and applications. Similar to related work, our project aims to listen to urban sound in the city to detect the events happening in the area with IoT solutions. In contrast to related work, the edge devices are able to monitor the area for different purposes by dynamically assigning them with the type of sounds we wish for devices to listen to.

2.3 Sound Identification with IoT Unit

Internet of Things (IoT) has been widely used for many sectors of smart cities applications such as eHealth and environmental noise pollution monitoring system [1]. Alsouda et al [1] presented the approach to classify urban noise in smart cities using low-cost computational

IoT unit, Raspberry Pi Zero W which has Wi-Fi capability with 1GHz CPU and 512MB RAM. The device uses ReSpeaker 2-Mic Pi hat microphone board which has two microphones to collect environmental sounds. The machine learning inference is deployed on the device for classification. The sound sensed by the microphones is fed into classifier to predict the categories of sounds. The EAR-IT edge devices [33] use IoT unit to identify the type of events and trigger the required actions.

We extend the related work by using IoT solutions to communicate with cloud server in classifying environmental sounds, which including having ability for each Raspberry Pi to dynamically identify the event by listening to the list of sounds assigned on cloud server.

2.4 IoT Web Application

IoT application can be used with web services by following standards of Internet Engineering Task Force (IETF), European Telecommunication Standards Institute (ETSI) [41]. In [9], administration web application is developed to monitor the noise emission in real-time, visualize and manage the noise-related public process by using MQTT (Message Queue Telemetry Transport) for exchanging data and communication. Tsao et al [38] developed cloud-based website written in PHP which is used to monitor the noise pollution by using the data sent from distributed sound sensing system (DSSS) via Internet.

In contrast to the related work, we are building web application where administrative users are able to manage the list of sounds and actions when the event is triggered. The overall architecture and protocol are similar to approaches done in the related work.

Chapter 3

Technical Background

3.1 Machine Learning

Machine learning is one of the capabilities of artificial intelligence [27] that allows a machine to learn to produce a particular outcome by feeding the dataset into an algorithm. The machine learning algorithms automatically alter or adapt the model's architecture by repeating the process to achieve the desired task without having to hard code like traditional programming. It has been applied in various fields such as finance, pattern recognition, computer vision, medicine and so on [8].

Since the invention of machine learning, scientists have formally defined machine learning in a different way. Tom Mitchell defined machine learning in his textbook as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [17]. In the textbook of Ethem Alpaydin, "Machine learning is programming computers to optimize a performance criterion using example data or past experience" [2].

Machine learning is mainly categorised into three types of learning, which are supervised learning, reinforcement learning, and unsupervised learning.

3.1.1 Supervised Learning

Supervised learning is the technique of machine learning which learns to produce the outputs based on labeled instances of the dataset where the features may be continuous, categorical or binary [14]. Table 3.1 shows an example of a labeled dataset. In supervised learning,

the sequence of input-output pairs is fed into a learning algorithm in the form $< x_i, y_i >$, where y_i is the output associated with input x_i. The supervised learning algorithm learns a function f that takes a possible input x_i to produce output y_i, such that $f(x_i) = y_i$ for all i [5].

Dataset					
No.	Feature 1	Feature 2	...	Feature n	Category
1	xxx	xx		x	1
2	xxx	xx		x	2
3	xxx	xx		x	3
...					...

Table 3.1: Example of known set of data

There are two learning problems in supervised learning which can be divided into classification and regression.

Classification Learning Problem

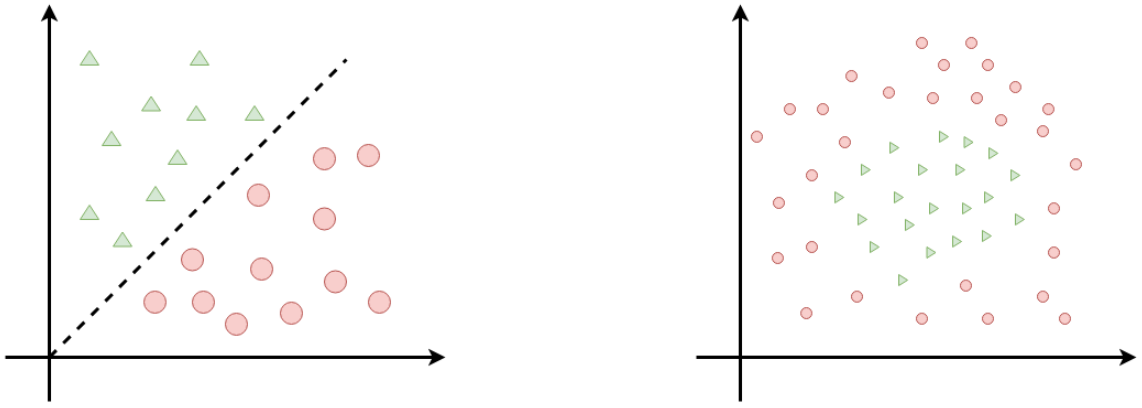
Classification is the supervised machine learning technique that is used to differentiate between k different classes by observing multiple variables on a set of features described in the dataset. The algorithm in the classification model, also known as *classifier*, can classify new objects based on the specified dataset [18].

The classifier is trained to learn the boundary that separates different classes [2], for examples circle and square. That boundary is called a decision boundary. In a classification problem, if given data can be separated by the linear decision boundary or linear separator, such a problem is called linearly separable (see Figure 3-1a). Otherwise, it is called non-linearly separable [27] (see Figure 3-1b).

In classification, there are several types of algorithms can be used to train the classifier. Some examples of classification algorithms are Naive Bayes Classifier, K-Nearest Neighbour, Support Vector Machine, Decision Trees, and Neural Networks.

Regression Learning Problem

Regression is the supervised machine learning technique that is used to continuously predict the estimation of the target variable [36] (see Figure 3-2). In the regression model, the output is a numeric value as opposed to the classification model which generates Boolean



(a) Linearly Separable (b) Non-linearly Separable

Figure 3-1: Two different types of classification learning problem

output [2]. Therefore, the task of the regression model is usually to predict the possible numerical outcomes of the future event, such as the population of the world and the price of company stocks. There are several types of regression model, which are divided into simple linear regression, polynomial regression, support vector regression, decision tree regression, and random forest regression.

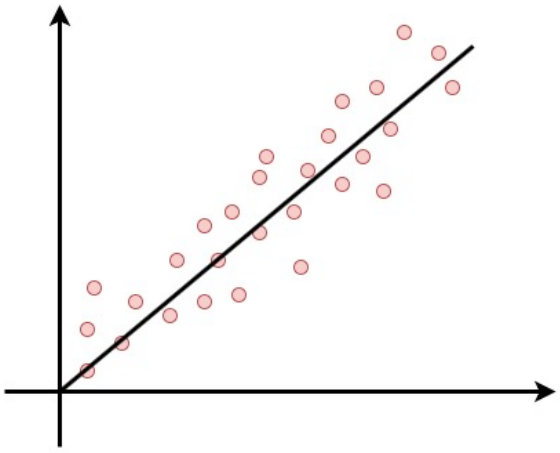


Figure 3-2: Linear regression model.

3.1.2 Reinforcement Learning

Reinforcement learning is the type of machine learning technique that is used to learn optimal actions [27] based on estimating value functions from experience, simulation or search [34]. This technique is known as dynamic programming [35]. The reinforcement learning method

is used to solve problems such as controlling a mobile robot, optimizing the operations in factories, and playing board games. In reinforcement learning, the decision maker, the agent, performs an action in its environment and it will be rewarded in each state. The learning algorithm aims to choose the sequences of actions that will reward the agent in the greatest amount (see Figure 3-3) [17]. There are several algorithms can be used for reinforcement learning. Among them, Q-learning is the most widely used reinforcement learning algorithm [11].

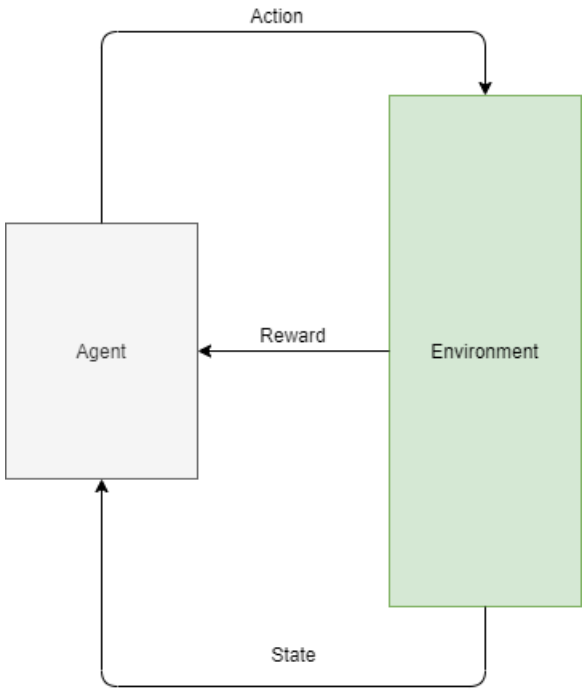


Figure 3-3: Reinforcement Learning

3.1.3 Unsupervised Learning

Unlike supervised learning and reinforcement learning techniques, unsupervised learning model learns patterns in the input data without supervised target values are given, nor feedback is supplied [10]. There several methods of unsupervised machine learning. Clustering [27] is the most common task of unsupervised machine learning. In cluster analysis, the knowledge is gained by separating all data points into different groups such that similar data points are in their own cluster [6] as shown in Figure 3-4.

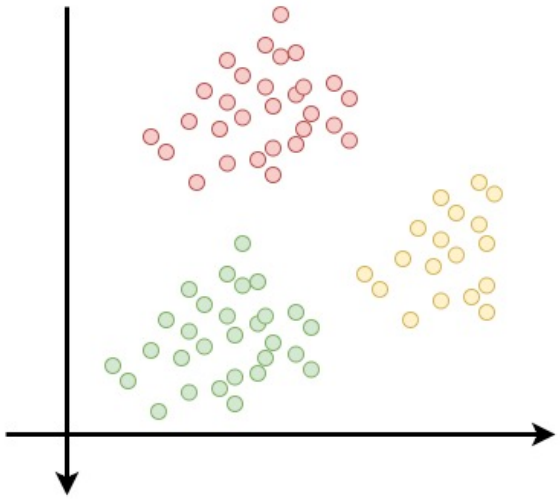


Figure 3-4: Three clusters of different data points described in three different colours.

3.2 Raspberry Pi and ReSpeaker

Raspberry Pi [24] is an inexpensive small device that can compute like a desktop computer. It requires a computer monitor or TV, keyboard and mouse. It has been used widely for many projects in various areas such as robotics, music, IoT, and smart devices. It is also a device that is capable of learning how to program in many languages such as Scratch and Python. In this project, a Raspberry Pi 3 Model B (see Figure 3-5a) is used for the computation and classification process. The specification of the Raspberry Pi 3 Model B is described in Table 3.2. Along with the Raspberry Pi, ReSpeaker 2-Mics Pi HAT [30] (see Figure 3-5b) is used for sound sensor. There are two microphones on the board which collect data. The ReSpeaker 2-Mics Pi HAT is popular in building sound applications.

(a) Raspberry Pi 3 Model B

(b) ReSpeaker 2-Mics Pi HAT

Figure 3-5: Environmental sound classification hardware with a Raspberry Pi Zero W and ReSpeaker 2-Mic Pi HAT.

Property	Value
SoC/CPU	Quadcore 64-bit 1.2GHz ARM Cortex A53
GPU	Broadcom VideoCore IV @ 400 MHz
RAM	1GB (LPDDR2-900 SDRAM)
Storage	MicroSD
USB	4 USB socket
Ethernet	1 Ethernet socket
Wi-Fi	802.11n Wireless LAN
Bluetooth	Bluetooth 4.0
Video output	HDMI/Composite via RCA Jack
Audio output	3.5 mm jack
GPIO	40 pins
size	85.6mm x 56.5 mm x 17 mm

Table 3.2: Specification of Raspberry Pi 3 Model B

3.3 Internet of Things (IoT)

The Internet of Things (IoT) is a communication method between computational intelligent devices such as home appliances, monitoring sensors, surveillance cameras, vehicles and so on. [41]. With IoT, smart devices can exchange the data over the Internet without the need for any interaction between humans and humans, or humans and devices. Therefore, in many smart city projects, the application of IoT has been adopted for development in many different areas, such as traffic management, home automation, energy management, medical aids and so on.

As the research in IoT grows, different researchers have proposed many layer architectures for IoT applications. The simplest and most basic architecture for IoT is three-tiers layer architecture, which are the perception layer, network layer, and application layer. However, three layers are not sufficient for developing a complete IoT application. Thus, five layers architecture has been used (Figure 3-6) [31]

The responsibility for each layer are described as follows:

- **Perception Layer**: It is a physical layer which has sensors integrated. Therefore, the sensors in this layer sense and collect data.

- **Transport Layer**: It is responsible for communication between the perception layer and the processing layer. It sends data collected by sensors in the perception layer to the processing layer and vice versa. The data transfer is via a network such as LAN, wireless, or Bluetooth.

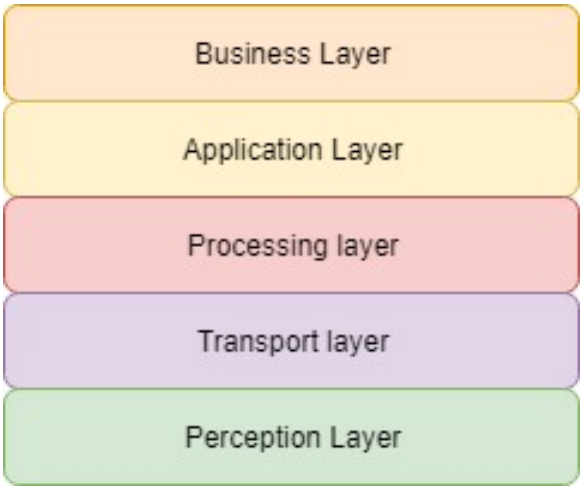


Figure 3-6: Five layer architecture of IoT.

- **Processing Layer**: It serves as the middleware where it stores and computes a large amount of data sent from the transport layer, for example, cloud server with database.
- **Application Layer**: It is responsible for utilising the data processed in the processing layer to various types of applications such as smart devices, smart cities, smart vehicle and so on.
- **Business Layer**: It is responsible for the management of the whole IoT system.

Chapter 4

Environmental Sound Classification

Machine learning technique has been proven to be effective in classification recently and the classification algorithm has been successfully applied to identify events based on sounds. In this chapter, we explain the dataset of environmental sounds, audio feature extraction for sound modeling and how sound is classified with two supervised classification algorithms: support vector machine and K-Nearest Neighbours.

4.1 Dataset

The environmental sounds are categorised into several classes of events such as car horns, gunshots, jackhammers, siren, dog bark and so on. There are several popular datasets that have been used in environmental sound classification, which are ESC-10, ESC-50, ESC-US [21] and UrbanSound8k [39].

4.2 Feature Extraction

In all sound classification system, the first step is to extract all features from the audio, which is then followed by classification. Audio features are a set of properties that describe a sound signal in such a way that it is informative and discriminative. In machine learning, feature extraction is the process of selecting the most informative and descriptive set of features from the audio. Therefore, the initial raw input data can be reduced into manageable sets for processing. The choice of features is important as it can affect the efficiency of classification algorithms.

There are several audio features that have been used in sound classification systems such as Zero-Crossing rate, signal bandwidth, spectral centroid, signal energy and Mel-Frequency Cepstral Coefficients (MFCC). Among all these features, it is observed that the performance of MFCC feature sets in audio classification is better [3]. In this project, we mainly focus on MFCCs for audio feature extraction.

4.2.1 Mel-Frequency Cepstral Coefficients

MFCCs are defined as short-term spectral-based features. MFCCs are the most popular features and widely used for sounds relating projects such as sound classification because they represent the amplitude spectrum of the sound signals in a compact form [16]. They are similar to what humans can hear as MFCC is based on the human hearing system. In MFCC, sound waves are analysed linearly at the frequency of below 1kHz and logarithmically at the frequency of above 1kHz [37]. The MFCCs can be produced by following the process as shown in Figure 4-1.

Framing

The first step of MFCC feature extraction is to divide the audio signal into small frames by applying a windowing function. Each frame has a length between 20 ms and 40 ms. There are a few window functions that can be used to divide the frame. However, in audio, Hamming and Hann window functions are the most effective window function to remove edge effects. The following equation yields the coefficients of Hamming and Hann window:

$$Hamming : w(n) = 0.54 - 0.46\cos(2\pi\frac{n}{L-1}), \quad 0 \leq n \leq L-1 \tag{4.1}$$

$$Hann : w(n) = 0.5(1 - \cos(2\pi\frac{n}{L-1})), \quad 0 \leq n \leq L-1 \tag{4.2}$$

Where L is the window length.

Discrete Fourier Transform

In this step, the frequency components of a discrete signal from each frame are extracted by using a Discrete Fourier Transform (DFT). The DFT is defined as:

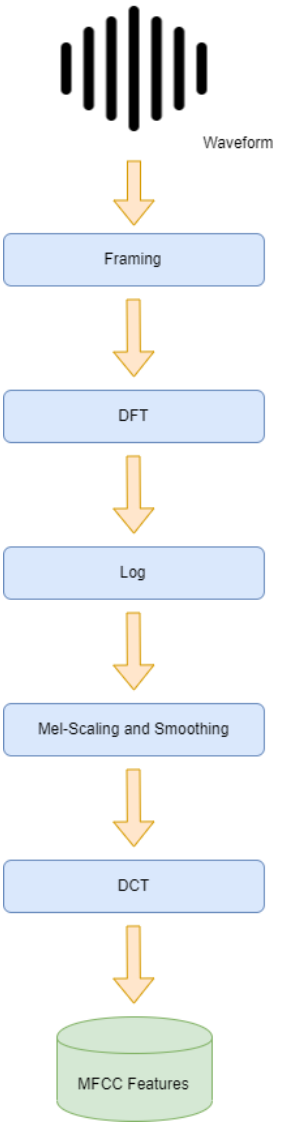


Figure 4-1: The process of creating MFCC features

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \quad 0 \leq k < N \tag{4.3}$$

Mel Filterbank

The next step is to obtain a mel-scale power spectrum by applying Mel filterbank to the power spectrum obtained from DFT. The mel function is computed as follows:

$$M(f) = 1127ln(1 + \frac{f}{700} \tag{4.4}$$

Then, the bank of triangular filters is applied to collect energy from each frequency band. The formula to calculate these filters are given as follows:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{4.5}$$

Where m is the number of triangular filters, $1 \leq m \leq M$, f is mel-scale frequency, which can be calculated from:

$$f(m) = (\frac{N}{F_s})M^{-1}(M(f_l) + m\frac{M(f_h) - M(f_l)}{M+1}) \tag{4.6}$$

Where M is mel function as described in Equation 4.4, f_l is the lowest frequency of the filter and f_h is the highest frequency of the filter.

Logarithm

The next step is to compute the logarithm of the power spectrum at the end of each filter. By taking the logarithm of the amplitude spectrum, it reduces noises that are not significant for speech recognition.

Discrete Cosine Transform

In the final step, Mel-frequency cepstrum is obtained by applying a Discrete Cosine Transform (DCT) to M-filters outputs. The following DCT equation yields the coefficients of the Mel-frequency cepstrum.

$$c(n) = \sum_{m=0}^{M-1} S(m) \cos \frac{\pi n(m+0.5)}{M}, \quad 0 \leq n \leq M \tag{4.7}$$

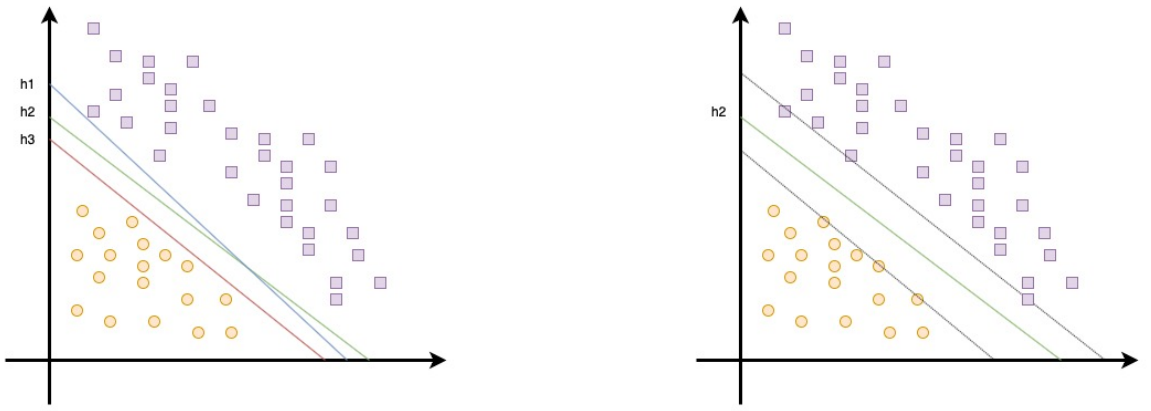
Where M is the number of coefficients. The value of M is usually 13 for sound classification [16].

4.3 Algorithm

There are several classification algorithms that can be used for classifying sounds. Choosing the best algorithm is not always easy as there are different characteristics in a different model, such as the complexity of the model, the accuracy of the prediction and the time it takes to train the whole model. In this section, we explain two supervised algorithms that are used for sound classification, which are support vector machine (SVM) and k-nearest neighbours (KNN).

4.3.1 Support Vector Machine

Support vector machine (SVM) is a machine learning framework [27] that has been widely used in pattern recognition and classification due to its computational efficiency [20]. In SVM, the model is designed to find the separator, which is known as a decision boundary, that separates two different data points. The decision boundary is mathematically described as a hyperplane. There is more than one hyperplane that separates two different data points in SVM. However, the best hyperplane for the SVM model is the hyperplane with the maximum possible distance to data points. For instance, as shown in Figure 4-2a, there are three hyperplanes (*h1*, *h2*, and *h3*) that separate two classes (circle and square). However, *h2* is chosen to be the best hyperplane because it leaves the maximum margin as shown in Figure 4-2b.



(a) Two different data points (circle and square) with three hyperplanes.

(b) The hyperplane with the maximum distance between two closest points.

Figure 4-2: An illustration of SVM classification

SVM is also used for classifying non-linearly separable data points. Using the kernel

trick, the classifier can separate non-linearly separable data points. There are several kernel functions to train the SVM classifier. Choosing the right kernel function can improve the SVM classifier. Among all available kernel functions, radial basis function (RBF) is widely used for SVM classifier, which can be described as:

$$K(x, x') = e^{-\gamma||x-x'||^2} \tag{4.8}$$

Where x and x' are two feature vectors, and γ sets the spread of the kernel. In the RBF kernel function, there are two main parameters to be considered, which are C parameter and Gamma parameter.

In RBF, the C parameter adjusts the simplicity of the decision boundary and errors of the classification. When the C parameter is low, the model will highly likely misclassify data points, whereas, when the C parameter is high, it will achieve a perfect classification however it increases the complexity of the decision boundary. The γ parameter controls the decision boundary spread of the kernel. When a high value is given to the γ parameter, the decision boundary will be small and thus there will be less misclassification. When the *gamma* value is low, the decision boundary becomes so broad and thus the classifier will misclassify data points.

In this project, the SVM classifier is trained using the environmental sound dataset with a range of C parameter values and γ parameter values. The experimental results are discussed in the later chapter.

4.3.2 k-Nearest Neighbours

KNN is an instance-based method and it is considered as the most basic machine learning algorithm [17]. In KNN, the test point, an instance, is classified as the group of its nearest training points that appear the most frequently. The nearest neighbours of that instance are defined by the minimum distance such as Euclidean distance.

As shown in Figure 4-3, there are two training data set, which are positive and negative. An instance x is the data point to be classified. In this example, k is given a value of 3. And, thus, the instance x is tested against its 3 nearest neighbours. The most frequent data point near the instance x is a negative data point. Therefore, the instance x is classified as part of the negative data point group.

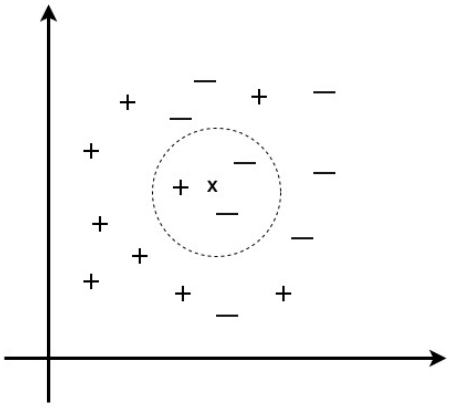


Figure 4-3: 3-Nearest Neighbours for a set of positive and negative training example along with an instance x to be classified.

There are two important things to consider in KNN to improve performance. The most important key element is the value of the nearest neighbours k. When k is small, the KNN model becomes complicated with rough decision boundary and thus the accuracy in classification is higher. However, if the value of k is too small, it can lead to over-fit. Whereas, when k is large, the decision boundary is smooth and thus it creates a simple KNN model. The downside of it is that the misclassification rate is higher. With the large value of k, the model will be over-fitted and all testing points will be classified as the same dominant class.

The second consideration for KNN is the choice of the distance metric as it is one of the key elements that can impact the accuracy of KNN. Among all available distance function, there are three widely used distance functions for KNN, which are described as follows:

$$Euclideandistance: \quad d(q,p) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{4.9}$$

$$Manhattandistance: \quad d(q,p) = \sum_{i=1}^{n}|q_i - p_i)|) \tag{4.10}$$

$$Chebyshevdistance: \quad d(q,p) = max_i(|q_i - p_i|) \tag{4.11}$$

Where d is the distance between two data points: $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$

Chapter 5

Design and Implementation

In this chapter, we explore the method used in the development of the project. We also describe the detail design and implementation of the project.

5.1 Development Methodology

In this project, there are three separate major developments, which are training classification models, Raspberry Pi integration, and cloud-based web application development. These three developments are done one after another with the classification model first, then followed by Raspberry Pi integration, and finally cloud-based web application. Therefore, in order to appropriately manage each development, the iterative agile development method is used for this project as shown in Figure 5-1. In each development, there are five phases, which are the planning and analysis phase, design phase, implementation phase, testing phase, and deployment phase. In iterative development method, the development begins with the planning and analysis phase. It finishes in the deployment phase and then, the next development begins. In this section, we elaborate on each phase in general for each development iteration.

The planning and analysis phase is important in each development because it is the starting phase of the development to which all other phases are depending on. In this phase, we collect all requirements for development such as dataset for training model, hardware and software for Raspberry Pi integration, tools for web application development and so on. When the requirements are defined in this phase, we begin to work on the plan for the development. The planning includes a development plan, a time plan, a testing plan, and a

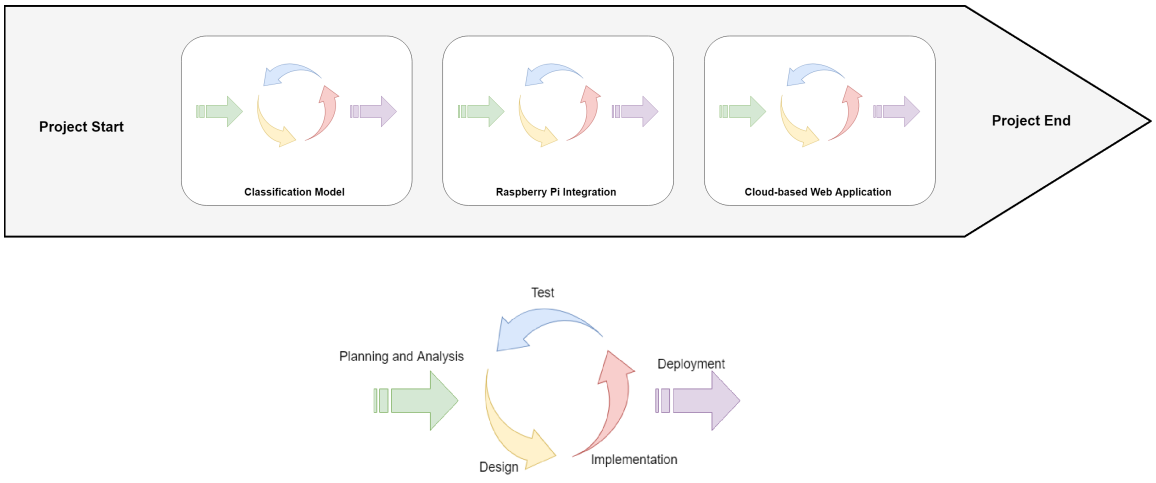


Figure 5-1: Iterative agile development method approach of the project

deployment plan.

In the design phase, we focus on system design and architecture for development solutions. The system architecture involves the details of the design of how each component in the system should communicate. The system design is important as it gives a better vision of how the system works, for instance, the communication between edge devices, IoT cloud server, and end-user applications. In this phase, we work on how the system should be implemented, which includes researching and applying suitable software architecture and framework. When the design is done, we analyse the design which consists of testing the system using a black box and white box testing techniques. It is important to ensure the design does not have an error before moving to the next phase.

After completion of the design phase, the implementation is taken in action. In this phase, we implement the design obtained in the previous phase. We follow the plan from the planning and analysis phase in order to keep the development on the right track. Generally, this phase is the longest phase of development and it is the most important part of the project. During this phase, we constantly revisit and review the design to ensure it is practical. At the end of this phase, we complete the implementation.

As we get the final implementation done, it is required to test it thoroughly before moving to the next stage. At the beginning of this phase, the list of test cases is generated. For example, in the classification model and Raspberry Pi, it is tested with random sound to ensure it predicts the sound event as expected and for the cloud server and web application,

it is tested by observing if the correct messages or signals are sent by the Raspberry Pi and so on. In the final development of this project, we focus on testing Raspberry Pi to identify dog barks, siren, and car horns. When the tests are failed or not met the desired outcomes, we revisit the design phase to make the changes.

When all tests are passed, the development is marked completed. In the deployment phase, we work on deploying the complete implementation on the system which is used in the next development iteration. For instance, after the model is trained, the machine learning inference is deployed onto Raspberry Pi to begin the development on Raspberry Pi and which then is used for the cloud server and web application development.

Therefore, with this development methodology, we are able to manage the flow of the development and keep each development on track, which leads to success and completion of the project requirements.

5.2 Overview System Design

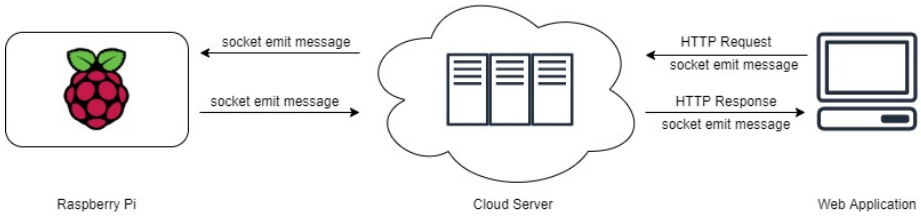


Figure 5-2: System overview design

Our system consists of three main subsystems, which are Raspberry Pi, cloud server and web application as shown in Figure 5-2. Both Raspberry Pi and the web application communicate with each other via the cloud server. The cloud server is responsible for socket connection and API requests. Each subsystem waits for the message emitted from others. In order for one subsystem to communicate with another, it needs to emit a message which other subsystems are listening to. For instance, if the server is listening to the message "*detected*" from Raspberry Pi and if Raspberry Pi detects an event, it sends a data together with the message "*detected*", so that the server can proceed to notify web application by emitting a message to the socket. The server accepts API requests via HTTP and responds to the web application. Figure 5-3 shows the details of how each component in the system communicates with each other.

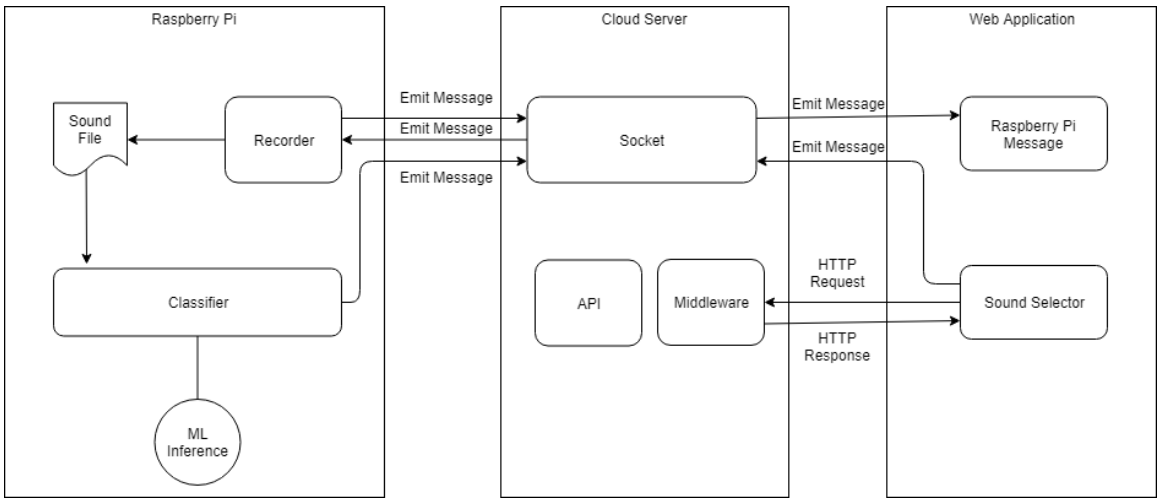


Figure 5-3: System components.

5.3 Hardware and Software

In this project, the training experiments for sound classification models are done on the computer with NVIDIA GTX 1080 GPU. Generally, classification models such as SVM and KNN does not require high computational power for training. However, the GPU improves the duration of training when the grid search with numerous range of parameter values is performed on thousands of sounds dataset. For environmental sounds classification device, the Raspberry Pi 3 Model B and ReSpeaker 2-Mics Pi HAT (Figure 5-4) are used. The ReSpeaker sensor records the sound in the environment and Raspberry Pi processes the sound using the machine learning inference to predict the event.

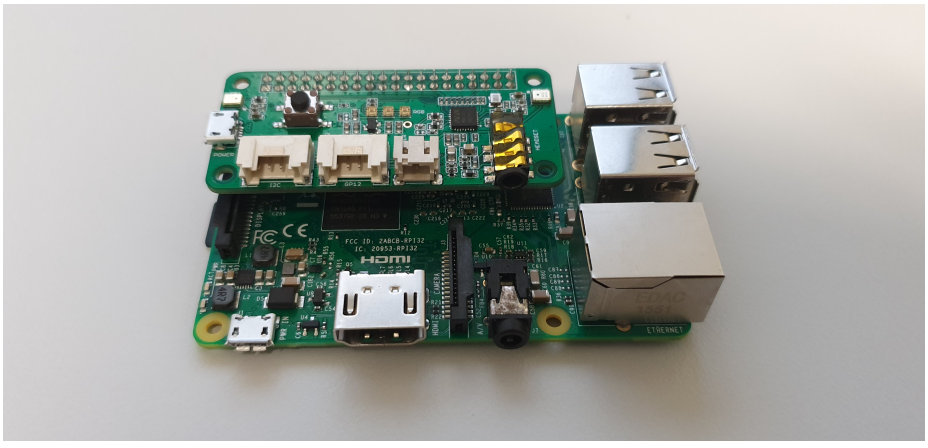


Figure 5-4: ReSpeaker 2-Mics Pi HAT mounted on Raspberry Pi 3 Model B for environmental sound classification device.

The SVM and KNN models are implemented in Python using scikit-learn [29], which is an open-source Python library for classification, regression and clustering algorithms. We implement feature extraction by using python audio processing library called librosa [15]. Nodejs [19] is used for cloud server implementation. For the web application, one of the most popular JavaScript framework called Reactjs [25] is used.

Application for Raspberry Pi is implemented in Python. In order for Raspberry Pi, Cloud Server and Web Application to communicate, we use Socket.io [32] to send and receive the signals and messages from one end to another. We use seeed-voicecard [26] driver to utilise the sensors on the ReSpeaker board with PyAudio [23] to record the sound.

5.4 Environmental Sound Classification

Training for sound classification models is the first development iteration of the project. In this development, we mainly focus on exploring and experimenting with SVM and KNN models to identify environmental sound events. In this section, we explain the dataset used for training in SVM and KNN. We also describe the implementation details of the two models.

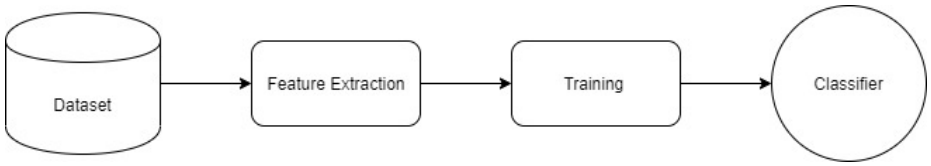


Figure 5-5: Audio classification process

5.4.1 Dataset for Training

UrbanSound8k [39] sound dataset is chosen for training both SVM and KNN model. There are 10 classes (see Table 5.1) in UrbanSound8K which are all categorised taxonomically. In this dataset, it contains 8732 labeled urban sounds. Each sound has a duration of an average of 4 seconds. All samples are distributed into 10 folders. We choose 9 out of 10 folders for training and test with the remaining folder. Table 5.1 shows the number of samples and duration for each sound event in the dataset.

Class	Samples	Duration (min:sec)
Air Conditioner	1000	66:35
Car Horn	429	17:34
Children Playing	1000	66:02
Dog Bark	1000	52:29
Drilling	1000	59:08
Engine Idling	1000	65:36
Gun Shot	374	10:17
Jackhammer	1000	60:11
Siren	929	60:33
Street Music	1000	66:40
Total	8732	525:04

Table 5.1: Classes of Samples from UrbanSound8K Dataset

5.4.2 SVM Classifier

Our first experiment started with exploring the SVM model. Before training the SVM model, we first extract the features from each sound sample of the dataset. All samples are then labelled correctly with the metadata provided by the UrbanSound8K dataset. There are several parameters to adjust for training SVM model. Since our problem is non-linearly separable, we use the Radial Basis Function (RBF) kernel to perform the classification. The main parameters for training SVM are the C parameter and the γ parameter. In order to get the best parameters, we first run a grid search with 13 values for C and γ parameters. The range of parameter values used in the experiment is as described in Table 5.2.

C	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10	10^{2}	10^{3}	10^{4}	10^{5}	10^{6}	10^{7}	10^{8}
γ	10^{-11}	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10

Table 5.2: C parameter values and gamma parameter values for SVM grid search

When the grid search is done, the best parameters are taken and used to train the SVM classification model. In training the model, 10-fold cross-validation is performed to improve the accuracy. As mentioned above, there are 10 folders in the UrbanSound8K dataset. In order to perform 10-fold, we choose 9 out of 10 folders for training and the other one folder for testing. The process is repeated with different testing folder number for 10 times. The accuracy of the model is then calculated by taking an average score of the prediction. When the training has completed, the model is then saved into the disk. The details implementation of the SVM model is shown in Algorithm 1 (Appendix A).

5.4.3 KNN Classifier

Similar to training SVM, we first extract all features from sound samples of the dataset. In KNN, there are two main parameters to be considered, which are the value of k and *metric*. In this experiment, we perform a grid search with three metric values, which are *euclidean*, *manhattan* and *chebyshev*. The k values we use in the grid search are ranging from 1 to 50. When the grid search is done, the training for KNN is begun with the best parameters obtained from the grid search. Algorithm 2 (Appendix B) shows the implementation of the KNN model training. Similar to SVM, 10-fold cross-validation is performed in the training process.

5.5 Raspberry Pi Integration

After the first development iteration is completed, the model we obtained from training is deployed on Raspberry Pi. As shown in Figure 5-6, We record the environmental sound using ReSpeaker for 4 seconds and save it in wav format. Once the recording is done, MFCC features are extracted from the recorded sound. Those features are then used for predicting the sound event with the trained model.

First, the Raspberry Pi is connected to the cloud server for communication. Once the connection is successful, it starts listening to the cloud server for the message about what sound to detect. Then, the Raspberry Pi will only listen to that particular sound. If the target sound is detected, it sends a signal to the cloud server. Algorithm 3 (Appendix C) shows the detail implementation of how environmental sound is classified on Raspberry Pi.

5.6 Cloud Server and Web Application

In order to experiment with the IoT application with Raspberry Pi, we implemented a simple cloud server and web application that are able to communicate with the device for setting a specific sound on the device to listen to and getting signals from the device when the target sound is detected. The messages are mostly exchanged via socket. However, there is one API can be requested via HTTP to get the list of sound classes.

When the Raspberry Pi is connected to the server via socket, the server will start listening to the signal. There are two signals from Raspberry Pi to the server which are shown in

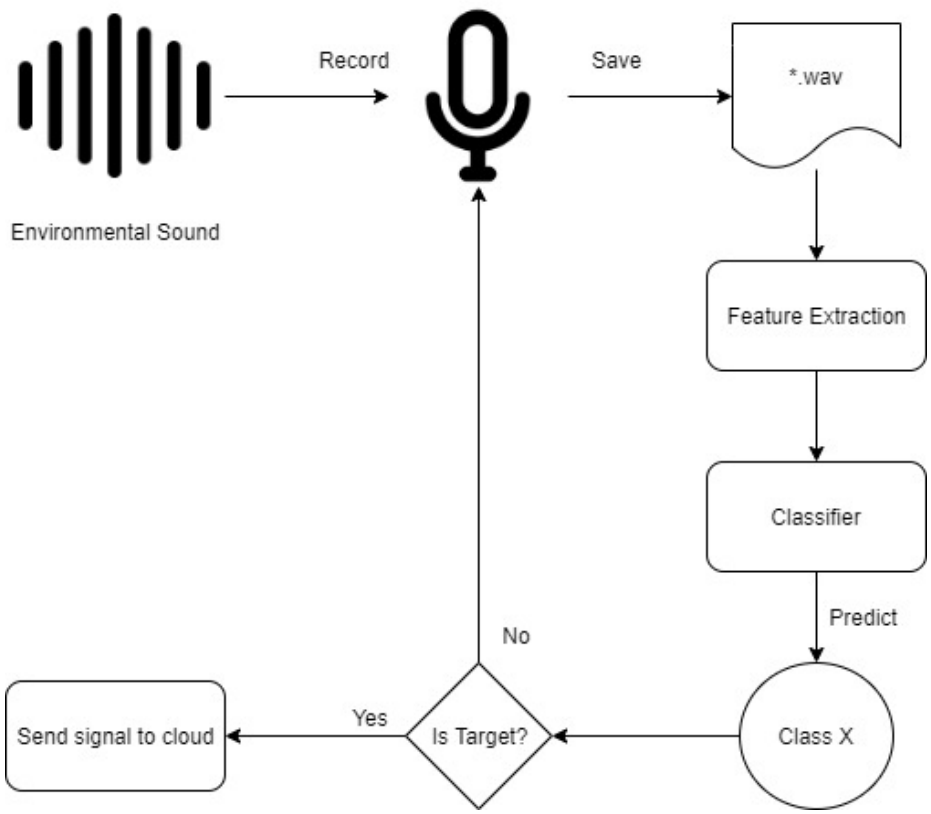


Figure 5-6: Environmental sound classification with Raspberry Pi and ReSpeaker

Table 5.6. The server sends only one signal to the Raspberry Pi to set the target sound (see Table 5.4).

Signal Name	**Data**	**Description**
pi_status	Sound ID	The status of current device such as what sound is the device listening to.
pi_detected	sound ID	The sound ID that the device detected.

Table 5.3: Socket signals from Raspberry Pi to Server

Signal Name	**Data**	**Description**
pi_setsound	Sound ID	Set the target sound to the given sound ID.

Table 5.4: Socket signals from Server to Raspberry Pi

The web application is also connected to the server via socket to communicate with the Raspberry Pi. The web application contains one dashboard as shown in Figure 5-7. The dashboard is used to set the target sound for the Raspberry Pi. The messages of Raspberry

Pi sent from the cloud server are displayed on the dashboard such as the status of the device and the notification message when the target sound is detected. Therefore, the cloud server sends two signals to the web application via a socket which is described in Table 5.5. Likewise, when the web application sets the target sound, it sends a signal to the server to set the target sound on the Raspberry Pi (see Table 5.6). Moreover, the cloud server serves one API to retrieve the list of sound available for the web application to set the target sound (see Table 5.7).

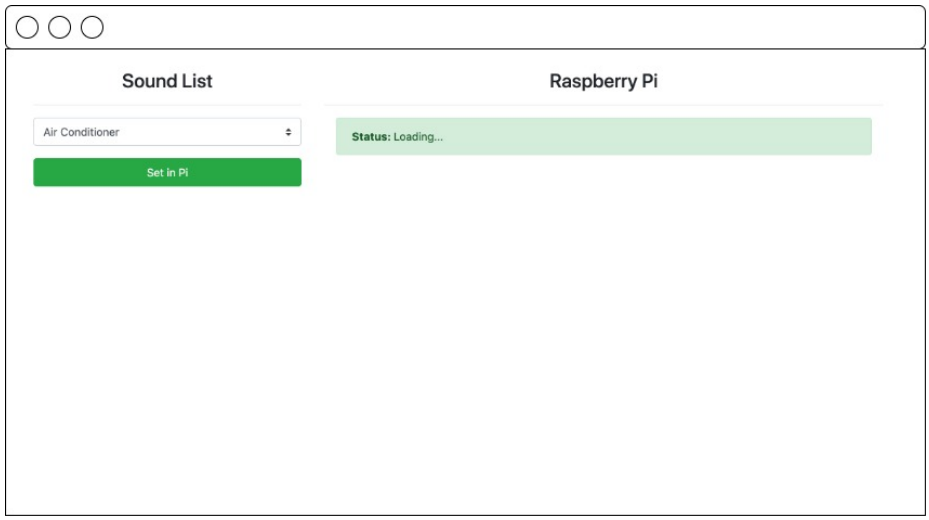


Figure 5-7: Web application dashboard that has a list of sound classes to set for the target sound on Raspberry Pi and current status of Raspberry Pi.

Signal Name	**Data**	**Description**
webapp_status	Sound ID	The status of current device sent from Raspberry Pi.
webapp_detected	Sound ID and Sound Event Name	The sound that the Raspberry Pi detected.

Table 5.5: Socket signals from Server to Web Application

Signal Name	**Data**	**Description**
webapp_setsound	Sound ID	Notify server to set the target sound on Raspberry Pi.

Table 5.6: Socket signals from Raspberry Pi to Server

Method	**Request**	**Response**
GET	/soundclass	List of sound class

Table 5.7: Available API of cloud server

Chapter 6

Results and Discussion

6.1 Classification Outcomes

In this section, we investigate the performance of two classification algorithms, SVM and KNN. We then compare the performance of two algorithms.

6.1.1 SVM Parameters Exploration and Training Results

In order to get the best performance of SVM model, we performed grid search to find the best parameters combination of C and γ. As a result, it is discovered that the highest accuracy we achieved is 56.8% for C and γ values of 10^3 and 10^{-6} respectively (see Table D.1). The accuracy for each combination of C and γ parameters obtained from grid search is plotted in heat map as shown in Figure 6-1. We then take the best C parameter value and plot the accuracy graph against 13 range of γ parameter values as shown in figure D-1. Likewise, we plot the accuracy graph with 13 range of C parameter (see Figure D-2) to explore the changes in accuracy with different parameters.

With the best parameters from the grid search, we then train the model to be deployed on Raspberry Pi. Figure 6-2 shows the confusion matrix of SVM classifier with predicted and true sound labels. From the confusion matrix, it is observed that the SVM classifier cannot identify the sound of the air conditioner and jackhammer correctly, resulting in a score of 0.05 and 0.19 respectively. Meanwhile, the performance of the classifier on other sound events is significantly better with a score of more than 0.5. The most accurate predictions that are observed from this experiment are the sound of a car horn, dog bark, and siren with

a score of around 0.73.

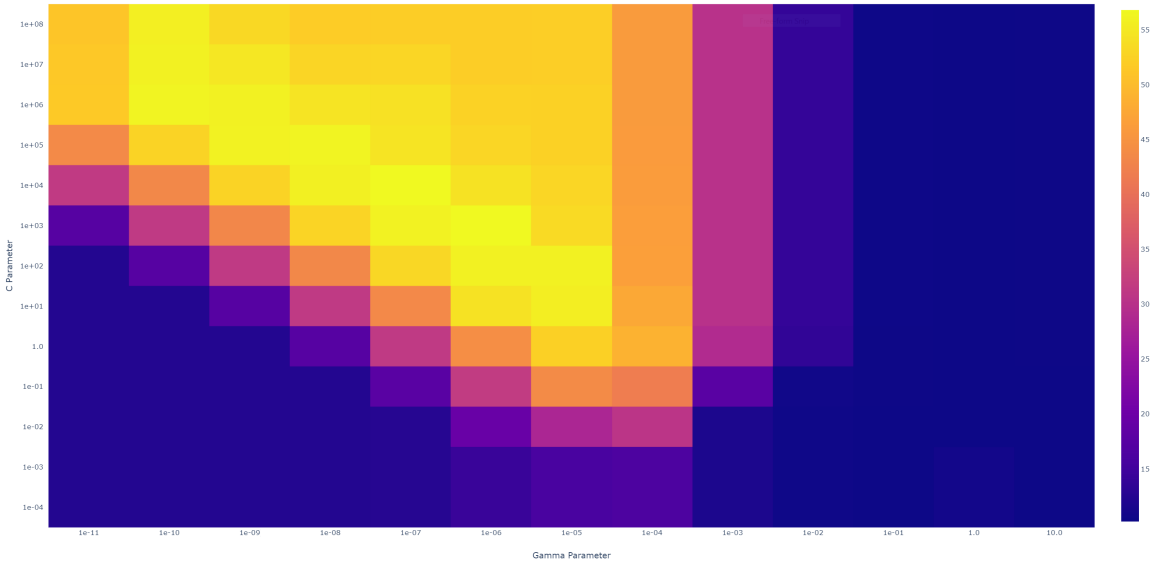


Figure 6-1: Heat map of SVM accuracy for the combination of C and γ paramters.

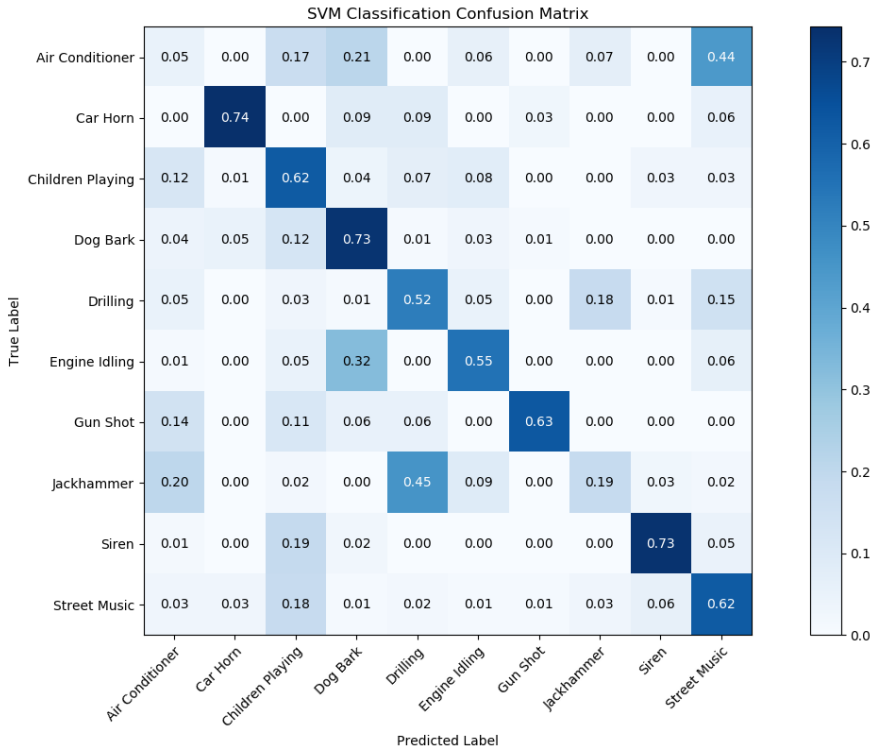


Figure 6-2: Confusion matrix of SVM environmental sound classifier.

6.1.2 KNN Parameters Exploration and Training Results

To find the best parameters for KNN, we perform the grid search for the value of k from 1 to 50 for three different distance functions: Euclidean, Manhattan, and Chebyshev. As a result in Figure 6-3, the Manhattan distance function seems to perform better than the other two distance functions. It is also observed that the three distance functions tend to have the same behaviour on a different number of nearest neighbours k. The accuracy is high with less nearest neighbours and it starts to decline as the number of nearest neighbours increase. In this experiment for KNN, we achieved the highest accuracy of 45.98% for $k = 6$ with Manhattan distance function. Therefore, We used 6 nearest neighbours and Manhattan distance function on training KNN classifier.

Figure 6-4 shows the confusion matrix of KNN classifier on 8 different events after training. In which, it is observed that four events, namely "Air Conditioner", "Engine Idling", "Jackhammer" and "Street Music" are scored lower than 0.5 while the rest are above 0.5. The highest score of the KNN classifier is 0.69 which is "Car Horn".

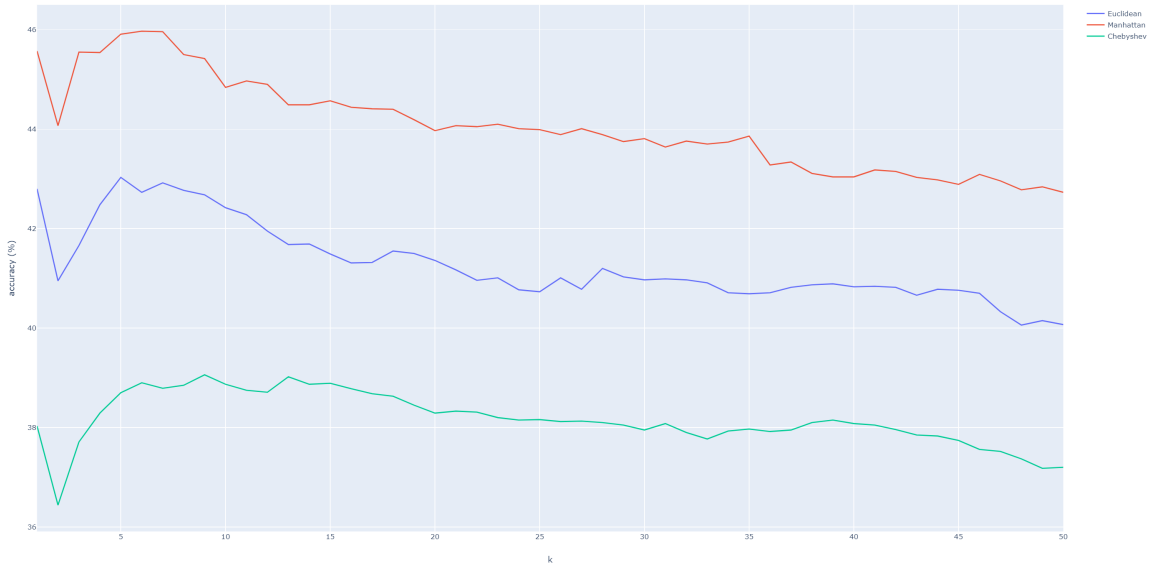


Figure 6-3: Performance of KNN classifier for k values from 1 to 50 and Euclidean, Manhattan, and Chebyshev distance functions.

6.1.3 Performance Comparison of Classifiers

When choosing the algorithm for classification, there is no best one that works for all applications. Throughout this experiment with SVM and KNN, both models have trade-offs

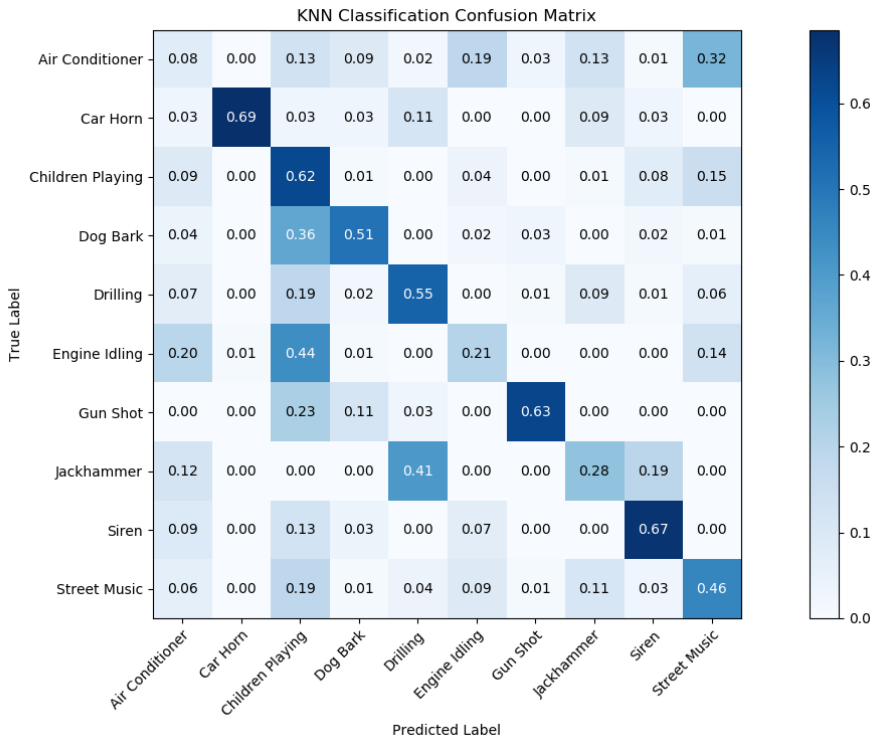


Figure 6-4: Confusion matrix of KNN environmental sound classifier.

between different performance, such as time and accuracy. In this section, we compare the performance of SVM and KNN in terms of accuracy for detecting the event based on environmental sounds and time required for training on computer and predicting the event on Raspberry Pi.

From the experiment, it is observed that the best parameter values for SVM and KNN are as follows:

- **SVM:** $C = 10^3$ and $\gamma = 10^{-6}$
- **KNN:** $k = 6$ and $Distance function = Manhattan$

Comparing the model training time, we observed that the SVM classifier is significantly slower than KNN. When testing the model on Raspberry Pi, it is observed that SVM predicts the sound events a lot faster than KNN. However, the accuracy of SVM is a lot higher than that of KNN. Table 6.1 shows the performance of both SVM and KNN classifiers.

Regarding the model accuracy, both SVM and KNN can identify the sound of a car horn and siren very accurately. However, it is observed in Figure 6-2 and Figure 6-4 that KNN tends to misclassify more sound events than SVM classifier.

Classifier	Accuracy (%)	Training (s)	Testing (ms)
SVM	56.80	69.34	9.55
KNN	45.98	17.87	38.29

Table 6.1: Performance of SVM and KNN in accuracy for identifying environmental sound and time taken for training on NVIDIA 1080 GTX GPU and testing on Raspberry Pi 3 Model B.

6.2 Project Demonstration

Based on the performance results during the experiment, we selected the SVM model as a classifier for Raspberry Pi. We successfully implemented IoT application by building a simple cloud server and web application which are used for communication between Raspberry Pi. In this section, we walk through the demonstration of the web application and Raspberry Pi. There are three scenarios to be covered in this demonstration which are described as follows:

- **Scenario 1:** Using a web application to select the sound for Raspberry Pi to listen to.
- **Scenario 2:** Raspberry pi starts listening to the environmental sound to identify the event.
- **Scenario 3:** Raspberry pi detects the target sound and web application display the message.

6.2.1 Selecting the target sound

When the web application is started, the sound list can be seen on the left side of the screen and the status and message from Raspberry Pi are on the right side as shown in Figure 6-5. There are in total 10 different sound events can be selected. In this demonstration, we select the target sound as "Dog Bark".

When the Raspberry Pi is started, it prepares all configuration for ReSpeaker to be ready for recording. Then, it waits for the signal from the cloud server as shown in Figure 6-6. If the Raspberry Pi is already in the process of identifying the sound, it will continue with the current process. When it is done, it will continue with the next target sound that is set from the web application.

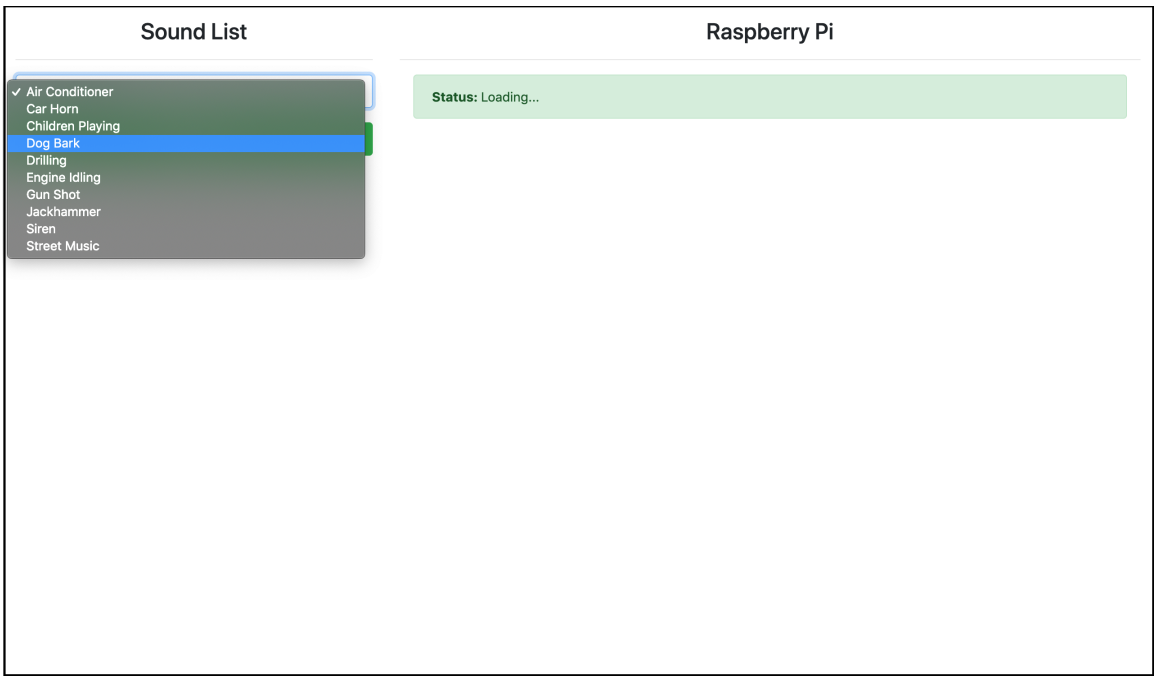


Figure 6-5: Selecting "Dog Bark" on web application.

```
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
ALSA lib pcm_oss.c:377:(_snd_pcm_oss_open) Unknown field port
ALSA lib pcm_oss.c:377:(_snd_pcm_oss_open) Unknown field port
ALSA lib pulse.c:243:(pulse_connect) PulseAudio: Unable to connect: Connection refused

ALSA lib pulse.c:243:(pulse_connect) PulseAudio: Unable to connect: Connection refused

ALSA lib pcm_a52.c:823:(_snd_pcm_a52_open) a52 is only for playback
ALSA lib conf.c:5014:(snd_config_expand) Unknown parameters {AES0 0x6 AES1 0x82 AES2 0x0 AES3 0x2  CAR
D 0}
ALSA lib pcm.c:2565:(snd_pcm_open_noupdate) Unknown PCM iec958:{AES0 0x6 AES1 0x82 AES2 0x0 AES3 0x2
CARD 0}
ALSA lib pcm_usb_stream.c:486:(_snd_pcm_usb_stream_open) Invalid type for card
ALSA lib pcm_usb_stream.c:486:(_snd_pcm_usb_stream_open) Invalid type for card
ALSA lib pcm_dmix.c:1043:(snd_pcm_dmix_open) The dmix plugin supports only playback stream
ALSA lib pcm_dsnoop.c:575:(snd_pcm_dsnoop_open) The dsnoop plugin supports only capture stream
ALSA lib pcm_dmix.c:1043:(snd_pcm_dmix_open) The dmix plugin supports only playback stream
ALSA lib pcm_dsnoop.c:575:(snd_pcm_dsnoop_open) The dsnoop plugin supports only capture stream
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
```

Figure 6-6: Raspberry pi is waiting for the message from the cloud server.

6.2.2 Listening to the target sound

When the button "Set in Pi" is clicked, it sends a signal to the cloud server together with the sound ID. The cloud server then sends a signal to the Raspberry Pi to listen to the selected

target sound. Once the Raspberry Pi receives the message, it starts recording as shown in Figure 6-8. Before it proceeds to the recording, it sends a status message to the cloud server that it has already started recording. The cloud server then notifies web application and the message is displayed as "**Status:** Listening to Dog Bark", as shown in Figure 6-7.

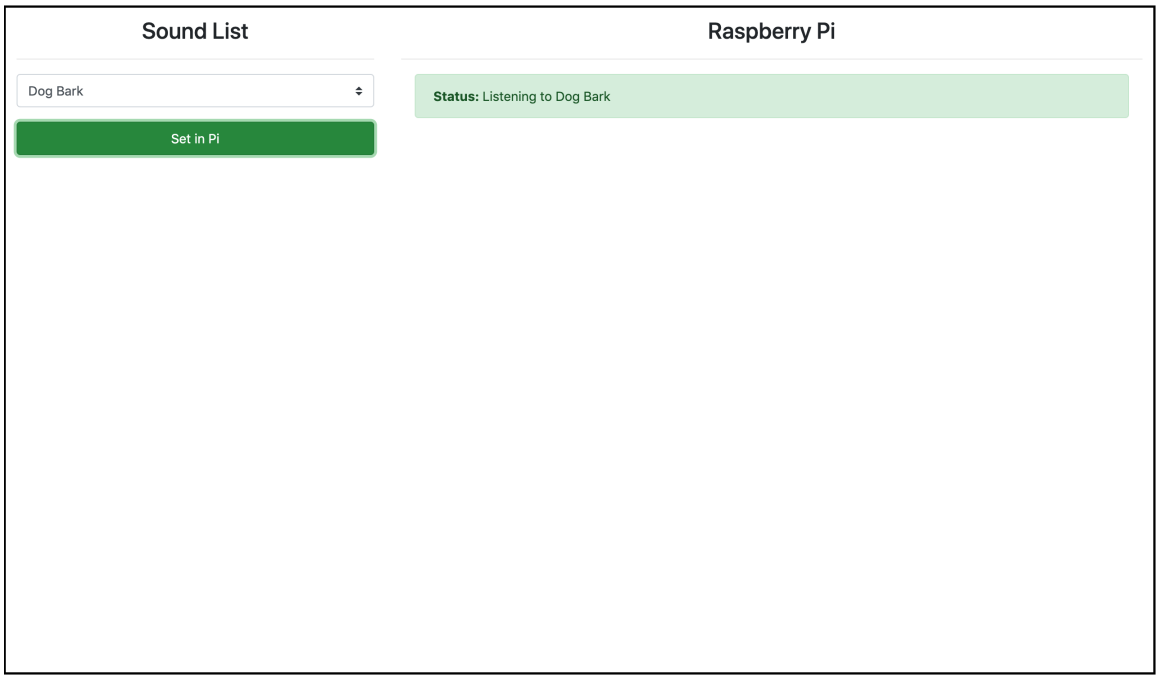


Figure 6-7: Setting Raspberry Pi to identify "Dog Bark" on web application.

6.2.3 Detected the target sound

Figure 6-10 shows the Raspberry Pi has detected the target sound "Dog Bark". When the target sound is detected, it sends the message to the cloud server. In this experiment, the cloud server sends a signal to the web application in which the message of "Dog Bark Detected !" is displayed as shown in Figure 6-9. If the Raspberry Pi detected sounds other than the target sound, it will just ignore it by not sending messages to the cloud server. On every detection, Raspberry Pi continues recording and identifying the target sound until the target sound is changed.

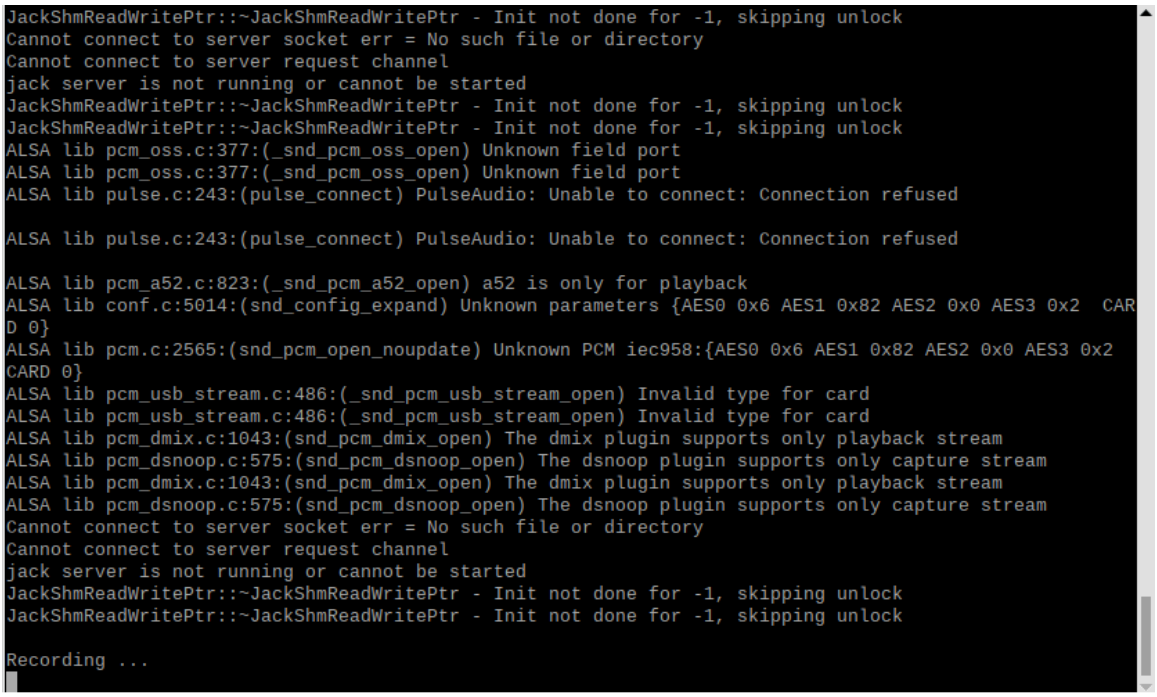


Figure 6-8: Raspberry Pi starts recording to identify "Dog Bark".

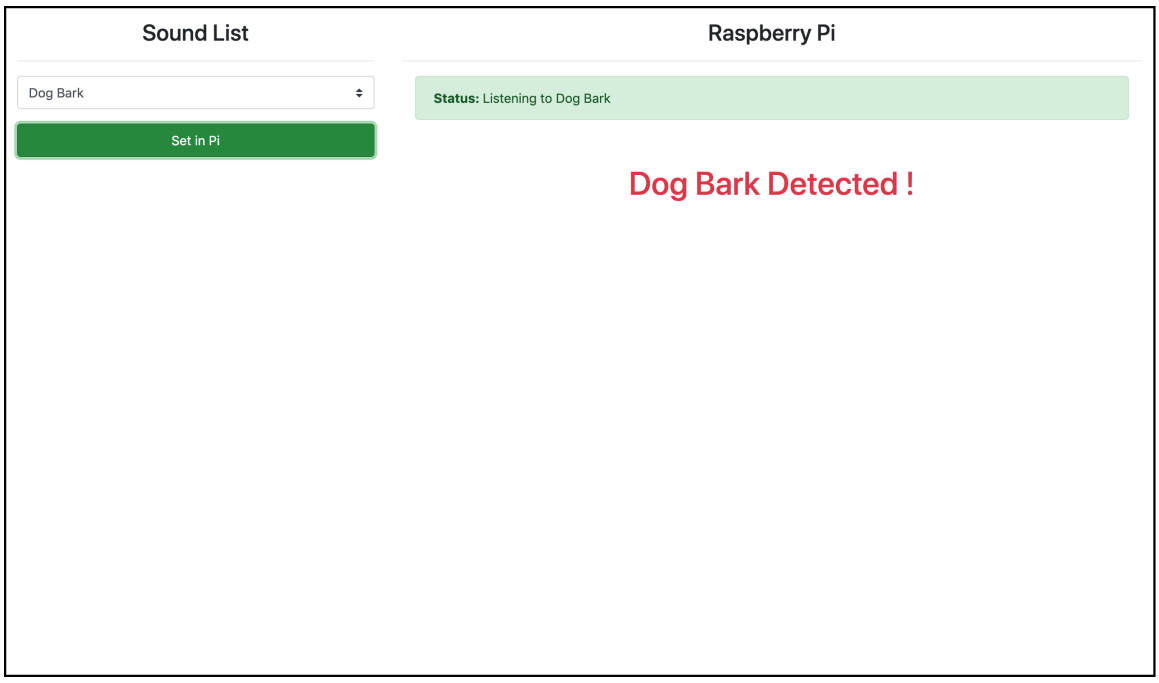


Figure 6-9: Web application displays the message when "Dog Bark" is detected.

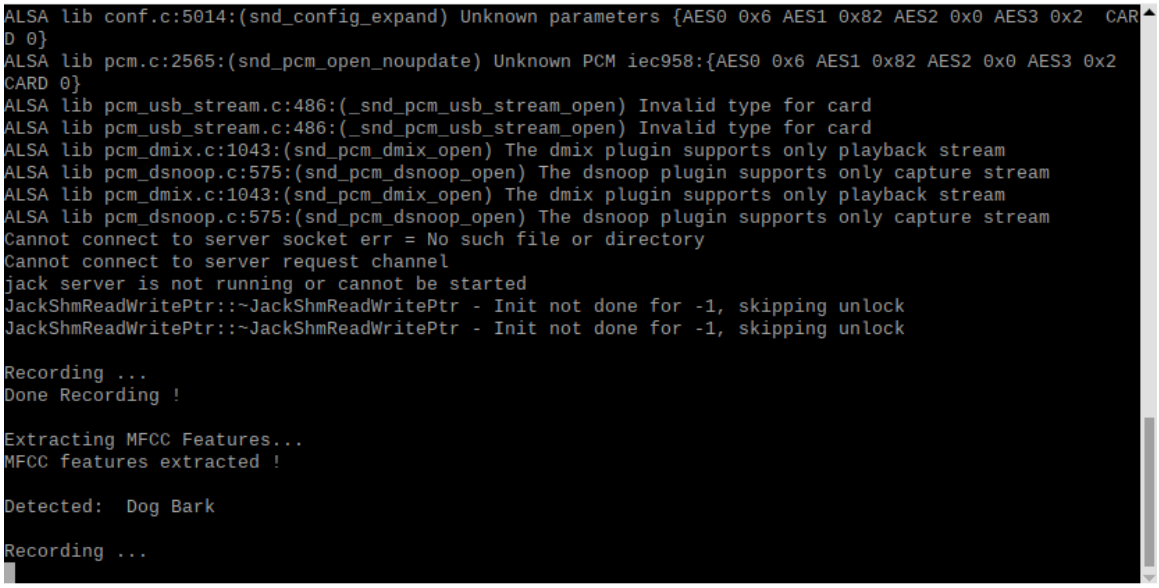


Figure 6-10: Raspberry Pi has detected "Dog Bark".

Chapter 7

Conclusion

In this thesis, we have covered a solution for smart cities to dynamically identify different events based on environmental sound using Raspberry Pi as an IoT Unit and machine learning techniques.

For project development, we followed the proposed iterative agile development method to achieve the goal of the project. We first developed and experimented with machine learning models, then we worked on Raspberry Pi followed by cloud server and web application.

Regarding our machine learning approach for environmental sound classification, we used MFCC feature extraction for audio features and two supervised machine learning algorithms which are SVM and KNN, to classify various types of environmental events. In our model training, we used the UrbanSound8K dataset which contains 8732 samples with 10 different classes such as car horn, dog bark, siren, jackhammer and so on. During the experiment, we have observed that the SVM classifier performs better than the KNN classifier. We achieved the accuracy of 56.8% for SVM with the parameter values of $C = 10^3$ and $\gamma = 10^{-6}$, whereas KNN classifier yielded the accuracy of 49.98% for $k = 6$ with the Manhattan distance function. We did not achieve the results as described in the paper [1] because the dataset used in the paper are handpicked from the different dataset, whereas we used the dataset provided by UrbanSound8K. Although the duration of training for KNN is significantly shorter than SVM, we have observed that the SVM classifier can predict sound faster than KNN. With the accuracy of close to 60% and the prediction time of less than a second, we chose the SVM classifier to test with Raspberry Pi.

For environmental sound classification, we deployed the machine learning inference on

Raspberry Pi 3 Model B for classification and used ReSpeaker 2-Mics Pi HAT to record the environmental sound. We then implemented a web socket to communicate with the cloud server and web application. In order to test and demonstrate the project, we built a simple cloud server and web application that can communicate with the Raspberry Pi. With the cloud server and web application, we achieved the goal of the project to set the sound dynamically for Raspberry Pi to listen to.

7.1 Future Work

In current work, Raspberry Pi records the sound for 4 seconds and processes the recorded sound. Then, it starts recording again once the process is done. Therefore, there is a pause between the first 4 seconds and the next recording. Future work will investigate the following things:

- Making Raspberry Pi to continuously listen to the environmental sound and detect the event in real-time.
- Being able to manage multiple Raspberry Pi devices on the web application.
- Integration of other IoT applications such as sending short messages to a mobile phone of a responsible organisation for the event.

Bibliography

[1] Alosouda, Y., Pllana, S., & Kurti, A. (2018) *A Machine Learning Driven IoT Solution for Noise Classification in Smart Cities.*

[2] Alpaydin, E. (2014). *Introduction to machine Learning.* Cambridge: MIT Press.

[3] Breebaart, J., & McKinney, M. *Features for Audio and Music Classification.*

[4] Brownlee, J. (2019). *Difference Between Classification and Regression in Machine Learning.* [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/ [Accessed 29 Sep. 2019].

[5] Dean, T., Allen, J. and Aloimonos, Y. (1995). *Artificial intelligence.* Menlo Park, California [etc]: Addison-Wesley.

[6] Ding, C., & He, X. *K-means Clustering via Principal Component Analysis.*

[7] EAR-IT: Using sound to picture the world in a new way - Digital Single Market - European Commision. Retrieved from https://ec.europa.eu/digital-single-market/en/news/ear-it-using-sound-picture-world-new-way.

[8] El Naqa I., Murphy M.J. (2015) *What Is Machine Learning?.* In: El Naqa I., Li R., Murphy M. (eds) Machine Learning in Radiation Oncology. Springer, Cham.

[9] Goetze, M., Peukert, R., Hutschenreuther, T., & Toepfer, H. (2017). An open platform for distributed urban noise monitoring. 2017 25th *Telecommunication Forum (TELFOR).* 10.1109/telfor.2017.8249339

[10] Ghahramani, Z. (2004). *Unsupervised Learning.*

[11] Hasselt, H., Guez, A., & Silver, D. *Deep Reinforcement Learning with Double Q-Learning.*

[12] Huang, C., Yang, Y., yang, D., Chen, Y. (2009). *Frog classification using machine learning techniques. Expert Systems With Applications*, 36(2). 10.1016/j.eswa.2008.02.059

[13] Kong, P. *The Benefits And Challenges of In-Camera Audio Analytics For Surveillance Solutions.* Retrieved from https://www.securityinformed.com/insights/understanding-camera-audio-analytics-surveillance-co-9381-ga.1528791285.html

[14] Kotsiantis, S. *Supervised Machine Learning: A Review of Classification Techniques*

[15] librosa/librosa. (2019). Retrieved 5 November 2019, from https://github.com/librosa/librosa

[16] Logan, B. *Mel Frequency Cepstral Coefficients for Music Modeling.*

[17] Mitchell, T. (1997). *Machine learning.* New York: MacGraw-Hill.

[18] Nakhaeizadeh, G. and Taylor, C. (1997). *Machine learning and statistics.* New York: John Wiley Sons.

[19] Node.js Foundation. (2019). Node.js. Retrieved 5 November 2019, from https://nodejs.org/en/

[20] Pao, T., Chen, Y., Yeh, J. and Li, P. (2019). *Mandarin Emotional Speech Recognition Based on SVM and NN.*

[21] Piczak, K. *ESC: Dataset for Environmental Sound Classification.*

[22] Piczak, K. (2015). Environmental sound classification with convolutional neural networks. *2015 IEEE 25Th International Workshop On Machine Learning For Signal Processing (MLSP).* 10.1109/mlsp.2015.7324337

[23] PyAudio: PortAudio v19 Python Bindings. (2019). Retrieved 5 November 2019, from http://people.csail.mit.edu/hubert/pyaudio/

[24] Raspberrypi.org. (2019). [online] Available at: https://www.raspberrypi.org[Accessed 28 Oct. 2019]

[25] React – A JavaScript library for building user interfaces. (2019). Retrieved 5 November 2019, from https://reactjs.org/

[26] respeaker/seeed-voicecard. (2019). Retrieved 5 November 2019, from https://github.com/respeaker/seeed-voicecard

[27] Russell, S., Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.).

[28] Salamon, J., & Bello, J. (2015). Unsupervised feature learning for urban sound classification. *2015 IEEE International Conference on Acoustics, Speech And Signal Processing (ICASSP).* 10.1109/icassp.2015.7177954

[29] scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. (2019). Retrieved 5 November 2019, from https://scikit-learn.org/

[30] Seeedstudio.com. (2019). *ReSpeaker 2-Mics Pi HAT.* [online] Available at: https://www.seeedstudio.com/ReSpeaker-2-Mics-Pi-HAT.html [Accessed 28 Oct. 2019]

[31] Sethi, P., Sarangi, S. (2017). *Internet of Things: Architectures, Protocols, and Applications. Journal Of Electrical And Computer Engineering*, 2017, 1-25. doi: 10.1155/2017/9324035

[32] Socket.IO. (2019). Retrieved 5 November 2019, from https://socket.io/

[33] Stahlbrost, A., & Sallstrom Danillo hollosi, A. (2014). *Audio Monitoring in Smart Cities - A Privacy Perspective.*

[34] Sutton, R. *Generalization in Reinforcement Learning: Successful Example Using Sparse Coarse coding.*

[35] Tan, M. *Multi-Agent Reinforcement Learning: Independent vs Cooperative Agents.*

[36] Tan, P., Steinbach, M. and Kumar, V. (2015). *Introduction to data mining.* Dorling Kindersley: Pearson.

[37] Tiwari, V. (2009). *MFCC and its applications in speaker recognition*

[38] Tsao, Y., Su, B., Lee, C., & Wu, C. (2017). An implementation of a distributed sound sensing system to visualize the noise pollution. 2017 *International Conference On Applied System Innovation (ICASI).* 10.1109/icasi.2017.7988503

[39] Urban Sound Datasets. (2019). *Urban Sound Datasets*. [online] Available at: https://urbansounddataset.weebly.com [Accessed 28 Oct. 2019]

[40] Virtanen, T., Plumbley, M., & Ellis, D. *Computational Analysis of Sound Scenes and Events* (pp. 13-40).

[41] Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of Things for Smart Cities. *IEEE Internet Of Things Journal*, 1(1). 10.1109/jiot.2014.2306328

[42] Zappatore, M., Longo, A., & Bochicchio, M. (2017). Crowd-sensing our Smart Cities: a Platform for Noise Monitoring and Acoustic Urban Planning. *Journal Of Communications Software And Systems*, 13(2). 10.24138/jcomss.v13i2.373

Appendix A

SVM Training Algorithm

Data: Dataset, C, Gamma
Result: SVM Model and accuracy
begin
  $mfccs \longleftarrow featureExtraction(dataset)$
  $acuracy \longleftarrow 0$
  $SVM \longleftarrow initialiseSVM(C, Gamma)$
  for $i \longleftarrow 1$ *to* 10 **do** `// 10-fold cross validation`
    $trainData \longleftarrow$ Get all samples from $mfccs$ where fold is not i
    $trainLabel \longleftarrow$ Get all labels from $mfccs$ where fold is not i
    $validationData \longleftarrow$ Get all samples from $mfccs$ where fold is id
    $validationLabel \longleftarrow$ Get all labels from $mfccs$ where fold is i
    Train SVM model with $trainData$ and $trainLabel$
    Validate SVM model with $validationData$
    $accuracy \longrightarrow$ Add accuracy of the prediction
  end
  Save SVM model into output folder
  $accuracy \longleftarrow accuracy/10$
end

Algorithm 1: Training SVM Model with 10-fold cross validation

Appendix B

KNN Training Algorithm

Data: Dataset, K, Metrics
Result: KNN Model and accuracy
begin
  $mfccs \longleftarrow featureExtraction(dataset)$
  $acuracy \longleftarrow 0$
  $KNN \longleftarrow initialiseKNN(K, Metric)$
  for $i \longleftarrow 1$ *to* 10 **do** `// 10-fold cross validation`
    $trainData \longleftarrow$ Get all samples from $mfccs$ where fold is not i
    $trainLabel \longleftarrow$ Get all labels from $mfccs$ where fold is not i
    $validationData \longleftarrow$ Get all samples from $mfccs$ where fold is id
    $validationLabel \longleftarrow$ Get all labels from $mfccs$ where fold is i
    Train KNN model with $trainData$ and $trainLabel$
    Validate KNN model with $validationData$
    $accuracy \longrightarrow$ Add accuracy of the prediction
  end
  Save SVM model into output folder
  $accuracy \longleftarrow accuracy/10$
end

Algorithm 2: Training KNN Model with 10-fold cross validation

Appendix C

Raspberry Pi Implementation

```
   Data: ServerURL
 1 begin
 2     Connect server at ServerURL
 3     Load classification model
 4     while True do
 5         Listen to the signal from server
 6         if signal received then
 7             Target ⟵ Set target sound
 8         end
 9         Start recording for 4s
10         Save recording to .wav
11         soundFile ⟵ Load recording
12         mfccs ⟵ featureExtraction(soundFile)
13         data ⟵ Get samples from mfccs
14         Feed data into classifier to predict the class
15         if Target is detected then
16             Send signal to cloud server with predicted class
17         end
18     end
19 end
```

Algorithm 3: Raspberry Pi environmental sound classification

Appendix D

SVM Parameters Exploration

C \ γ	10^{-11}	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10
10^{-4}	12.19	12.22	12.22	12.21	12.46	14.17	15.76	16.21	11.58	10.39	10.2	10.76	10.22
10^{-3}	12.19	12.22	12.22	12.21	12.46	14.17	15.76	16.21	11.58	10.39	10.2	10.76	10.22
10^{-2}	12.19	12.22	12.22	12.21	12.46	19.37	28.16	30.6	11.58	10.39	10.2	10.2	10.22
10^{-1}	12.19	12.22	12.22	12.21	17.6	31.76	43.67	41.83	17.62	10.53	10.29	10.24	10.22
1	12.19	12.22	12.22	17.29	31.34	44.15	52.27	48.75	29.0	13.5	10.25	10.21	10.27
10	12.19	12.22	17.26	31.33	43.41	54.25	55.57	47.5	30.14	13.77	10.31	10.21	10.27
10^{2}	12.19	17.25	31.35	43.27	53.02	55.87	55.93	46.5	30.14	13.77	10.31	10.21	10.27
10^{3}	17.25	31.38	43.22	52.67	56.0	56.8	53.39	46.27	30.09	13.77	10.31	10.21	10.27
10^{4}	31.41	43.28	52.64	55.81	56.79	54.31	52.98	45.98	30.12	13.77	10.31	10.21	10.27
10^{5}	43.64	52.65	56.05	56.26	54.44	52.89	52.43	45.89	30.12	13.77	10.31	10.21	10.27
10^{6}	51.53	56.23	55.9	54.53	54.15	52.45	52.44	45.85	30.12	13.77	10.31	10.21	10.27
10^{7}	51.44	55.92	54.82	52.87	52.92	51.92	52.08	45.85	30.12	13.77	10.31	10.21	10.27
10^{8}	50.99	55.74	53.2	51.82	51.95	51.92	52.08	45.85	30.12	13.77	10.31	10.21	10.27

Table D.1: Accuracy table of SVM with 13 C and γ parameter values.

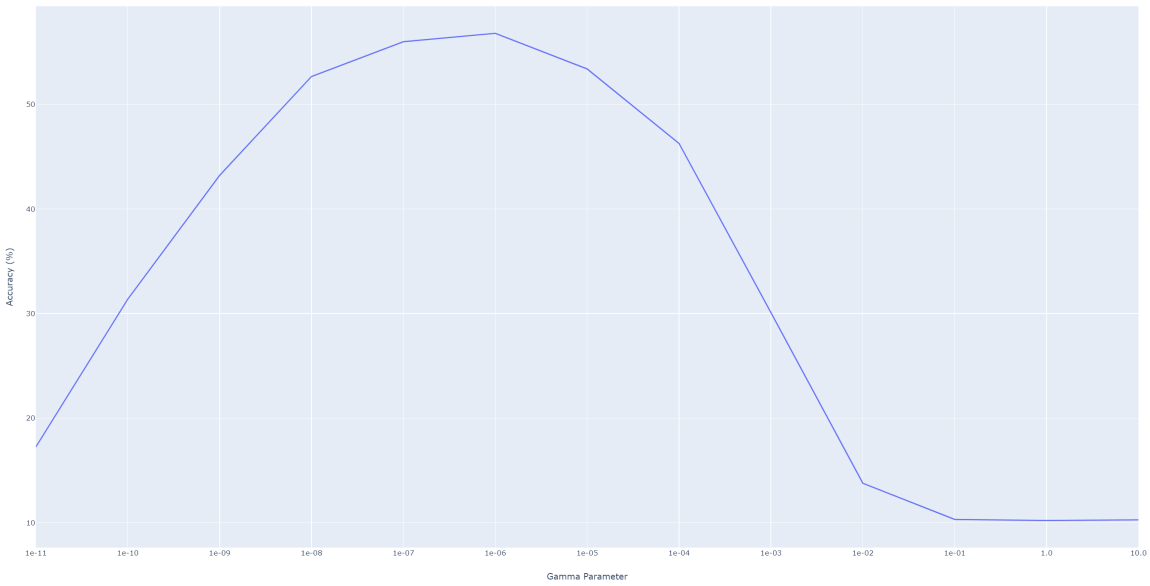


Figure D-1: SVM performance on γ parameters.

Figure D-2: SVM performance on C parameters.

Appendix E

KNN Parameters Exploration

k \ Distance	Euclidean	Manhattan	Chebyshev
1	42.81	45.57	38.03
2	40.95	44.07	36.44
3	41.66	45.55	37.71
4	42.48	45.54	38.29
5	43.04	45.92	38.70
6	42.74	45.98	38.90
7	42.93	45.97	38.79
8	42.78	45.58	38.85
9	42.69	45.43	39.06
10	42.42	44.84	38.87
11	42.28	44.97	38.75
12	41.95	44.91	38.71
13	41.68	44.49	39.02
14	41.69	44.49	38.87
15	41.49	44.57	38.89
16	41.31	44.44	38.78
17	41.32	44.41	38.68
18	41.55	44.4	38.63
19	41.51	44.19	38.45
20	41.36	43.97	38.29
21	41.17	44.07	38.33
22	40.96	44.05	38.31
23	41.01	44.12	38.20
24	40.77	44.01	38.15
25	40.73	43.99	38.16
26	41.01	43.89	38.12
27	40.78	44.01	38.13
28	41.22	43.89	38.10
29	41.03	43.75	38.05
30	40.97	43.81	37.95
31	40.99	43.64	38.08
32	40.97	43.76	37.90
33	40.91	43.73	37.77
34	40.71	43.74	37.93
35	40.69	43.86	37.97
36	40.71	43.28	37.92
37	40.82	43.34	37.95
38	40.87	43.11	38.10
39	40.89	43.04	38.15
40	40.83	43.04	38.08
41	40.84	43.18	38.05
42	40.82	43.15	37.96
43	40.66	43.03	37.85
44	40.78	42.98	37.83
45	40.76	42.89	37.74
46	40.74	43.09	37.56
47	40.33	42.96	37.52
48	40.06	42.78	37.37
49	40.15	42.84	37.18
50	40.08	42.74	37.20

Table E.1: Accuracy table of KNN for number of nearest neighbors k from 1 to 50 and Euclidean, Manhattan, and Chebyshev distance functions.

Appendix F

Git Project Repository

This project is developed under the name "NUesc - Newcastle University Environmental Sound Classification". The complete implementation of this project can be found at https://www.github.com/Susros/NUesc. The instructions and requirements to setup the project are all documented in the repository page.