UiO : Department of Informatics
University of Oslo

Supporting the Hearing Impaired with Glass Applications

Henrik Janson

Master's Thesis Spring 2017

Supporting the Hearing Impaired with Glass Applications

Henrik Janson

16th May 2017

Abstract

According to the World Health Organization(WHO) over 5% of the world's population – 360 million people – has disabling hearing loss[WHO Media center. Deafness and hearing loss. May 07 2015]. Out of these people, there are those who live with total deafness, and can't get their hearing back with the help of regular hearing aids, nor with operations like cochlear implants. In these situations, we need to look at other ways to aid the hearing disabled. Recent development has made Smart glasses a viable option of becoming a personal tool for automatic audio recognition. By taking use of its augmented reality features, the visualization of important audio happenings around the user can be achieved with a minimal amount of distraction. This thesis investigates how Google's smart glass product, "Glass" can be used as a platform supported by speech, siren and name recognition software to aid the hearing impaired.

The main matter of the work includes the implementation of a glass application in combination with a self-developed Siren recognition system based on machine learning algorithms. Other features looked into was the use of third party speech and name recognition services.

Results show that it is possible to use smart glasses for live speech, siren, and name recognition. However, some problems with the amount of test data needed and lacking integration tests conducted prevent this study for drawing a conclusion on whether the resulting work is a reliable alternative for aiding the hearing impaired. But with further testing and development, it is reasonable to believe that the system could have a positive impact on the hearing impaireds lives.

Contents

List of Figures

List of Tables

Preface

I would like to thank my two advisors Jim Tørresen and Ralf Greisiger for their guidance and support throughout this thesis. As well as the students and administration at Briskeby for putting their school and bright minds at our disposal. My sincere thanks to all my fellow student at Robotics and Intelligent Systems research group, for making this a great learning environment during the Master years. A special thanks go out to Jarle Fosen, who worked as my fellow developer and research partner. And Aleksander Pollen for the valuable discussions regarding the creation of this thesis.
I would also like to thank all my family and friends for all their support and feedback.

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation

In recent years the development of smart-glasses has been a growing field for different major companies. With several proof-of-concept applications being proposed for health care, looking into the usability of the glasses in situations such as operating rooms[1] to home care[2]. This thesis looks into some of the problems that the hearing impaired can see being solved with the help of smart glasses. After results got from research done at Briskeby, School for the hard of hearing(see section 3.2) we decided to focus on the implementation of the three problems:

- Speech recognition.
- Siren recognition.
- Name recognition.

There is a much groundwork needed to be able to implement these three features on the glassware. In this thesis, we will go through the most important parts to make that makes this possible.

1.1.1 Outline

This thesis is divided into four additional chapters, (i) Background, (ii) Implementation, (iii) Experiments and results and (iv) further work and discussion. The first chapter will introduce the glassware that is being used in this thesis and relevant machine learning concepts and services used. The Implementation chapter will explain the different choices made in regards to glassware, development platforms, and machine learning models. It will also go through the process of building the lifecycle of the thesis as shown in figure1.1.

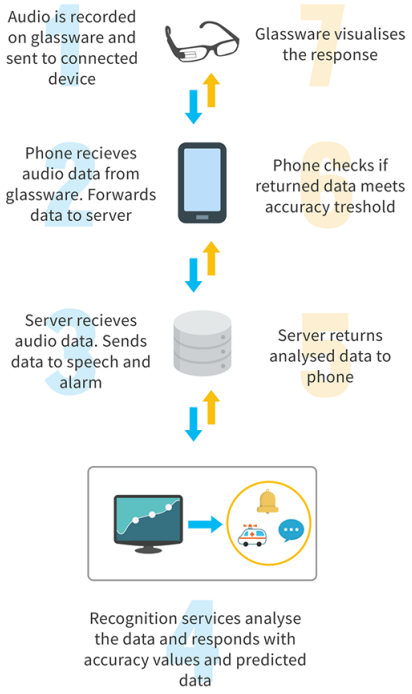


Figure 1.1: Showing an overview of the software lifecycle developed in this thesis

Chapter three outlines the machine learning experiments conducted and analysis done, leading to the final result. The last chapter will go through a general discussion of the thesis as a whole as well as some suggestions for further work.

Chapter 2

Background

2.1 Summary

This chapter will introduce the most relevant theory related to this thesis. The first section contains a short introduction to sound and hearing. The following sections look into different types of new technology and prototypes that can be used to aid the hearing impaired. The last part of the chapter consists of an introduction to the machine learning concepts applied in this thesis, followed by some theoretical insight into machine learning and machine learning algorithms. The last section will discuss some more practical aspects of machine learning.

2.2 Hearing

Sound Can be described as either a psychological or physical phenomena[3]. In physics, sound can be looked at as a type of energy made by vibrations. When an object vibrates, it causes moment in its surrounding particles creating a rippling effect lasting until it runs out of energy. An example of this could be throwing a rock in a lake, the ripples created by the rock are like sound waves moving through the air expanding in any direction. We usually visualize these changes in pressure as sound waves as shown in figure 2.1.

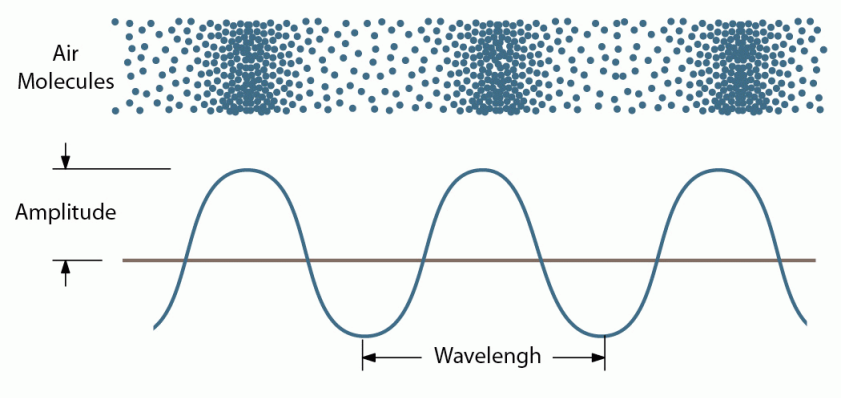


Figure 2.1: Sound waves visualized from air molecules[4]

In a psychological term, a sound is the act of us being able to detect these changes, much like a microphone we interpret these signal by transforming acoustic energy into electrical signals in our brain. This action is also known as hearing. Among other attributes, a healthy ear can detect what is known as frequency and amplitude from a given sound. Frequency being the distance between each wave and amplitude is its force. The frequency of a wave decides its pitch, at a high rate the sound would be perceived as a high piercing sound and at a low frequency would be deep. The amplitude decides how loud the sound would be, the greater amplitude, the louder the sound.

Normal hearing is commonly given as having a hearing range of 20 to 20,000 Hz [5]. Where Hz is the frequency of sound wave cycles per second. Anything above or below this threshold would be too high/low pitched for our ear to detect. The hearing starts with sound entering the ear canal, which causes the eardrum to move. The vibrates from the eardrum moves through the ossicles to the cochlea (see figure 2.2), which causes the liquid in the cochlea to move. The movement of the liquid causes the hair cells to bend creating neural signals that get picked up by the auditory nerve. Hair cells at one end of the cochlea send high pitch sound information and at the other end low pitch information. Finally, the auditory nerve sends signals to the brain where they get interpreted as sound.

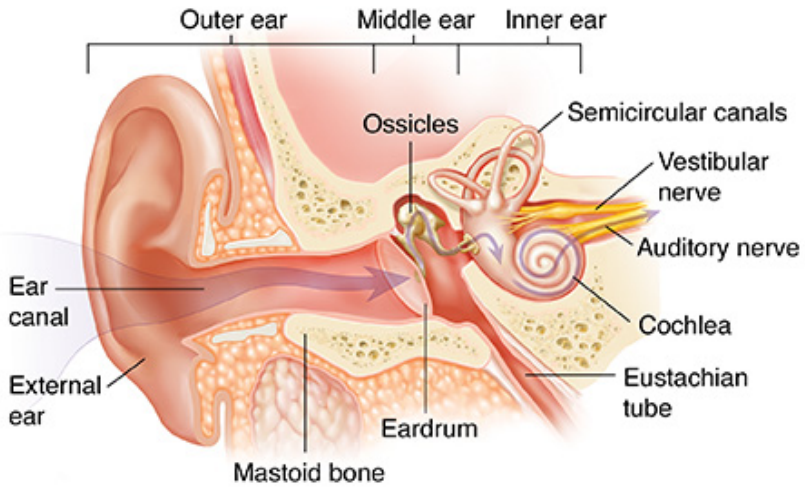


Figure 2.2: Visualisation of the anatomy of the ear[6]

2.2.1 Hearing Loss

Hearing loss can be described by the part of the auditory system that is damaged. There are three basic types of hearing loss: conductive hearing loss, sensorineural hearing loss, and mixed hearing loss. Loss of hearing usually comes from the loss of sensitivity to some but not all frequencies.

Conductive hearing loss Conductive hearing loss occurs when sound is not carried efficiently through the ear canal to the eardrum and the ossicles of the middle ear[7].

Sensorineural hearing loss Sensorineural hearing loss (SNHL) transpires when there is damage to the inner ear (cochlea), to the nerve pathways from the inner ear to the brain[8].

Mixed Hearing Loss Sometimes a conductive hearing loss occurs in combination with a Sensorineural hearing loss (SNHL). In other words, there may be damage in the outer or middle ear and the inner ear (cochlea) or auditory nerve.

2.3 Smart glasses background

One of the most common augmented reality platforms is within smart-glass technology. This section starts with a short introduction to how augmented reality and smart-glasses work. The following sections will introduce Googles smart-glass project; "Glass," since this is the technology used in this thesis.

2.3.1 Augmented reality

Augmented reality (AR) is real life supplemented by some computer generated input. It works by combining real objects with virtual ones in real time. AR is a variation of what is known as virtual reality (VR). VR work by completely immersing the user in a synthetic environment meaning that the user cannot see the real world around him. In contrast, AR allows the user to see the actual world with virtual objects superimposed upon it. Therefore, AR supplements or "augments" the reality, rather than entirely replacing it. Ideally, it would appear to the user that the virtual and physical objects coexisted in the same space. In other words, AR works by enhancing one's reality with the use of sound, video or graphics. An example of AR could be drawing on a piece of paper with your finger, and the using, e.g., a pair of smart glasses or a phone to add color to the screen where your finger moves on the paper.

The phrase `augmented reality` itself was coined in 1990[9], but the technology has been around for quite some time. In 1965 Ivan Sutherland's talks about how the kinesthetic display, one of the earliest AR machines could, in theory, be used to simulate negative mass [10]. Today AR can be found in almost any field. From military applications to greeting cards. There are many different types of AR; a naive version can be as simple as projections of images, to more advanced technologies utilizing object recognition, computer vision or voice recognition to make the users surrounding interactive and digitally manipulable by the user. And like Sutherland thought, this can also be used to capture and visualize real world objects that the eyes can't see, like electromagnetic waves.

2.3.2 Smart-glasses inroduction

As mentioned in the previous section, the thought of creating smart-glasses has been around for quite some time, however, the technology needed to make them viable in a practical sense had been missing. In recent years, however, the development of better batteries and faster CPUs has made smart-glasses more relevant than ever. At the moment there exist many different types of smart glasses. Some under production and other still in the developing phase. Smart-glasses commonly consist of a small head-mounted computer and act like an enhanced pair of regular glasses. They work by showing the user output via a heads up display(VR) or augmented reality(AR). While early models usually performed basic tasks, such as showing a front-end display for a remote system, Modern smartglasses works like smartphones utilizing technology such as Wi-Fi or cellular networks[11].

2.3.3 Google glass

Google Glass (as shown in figure 2.3) was developed by Google X, known as the facility within Google devoted to technological advancements. The glasses had their commercial release in 2014 where they were aimed at developers and early adapters to get used to the new platform. They continued to undergo lots of hardware and interface changes through the coming year until Google in 2015 announced that it would stop producing the prototype. However, Google remained committed to further development of the product. The glasses are intended to give the user all the benefits of a smartphone, without the need to hold the device up where one can see it.

Hardware At current writing the glass hardware consists of:

- 640 x 360 pixels AR display
- 5-megapixel camera
- 16GB of flash storage
- bone conducting microphone
- one microUSB port.

How they work The glasses are controlled by an intractable display with the help of voice and touch commands. They work by using visual, audio and location-based inputs to provide relevant information. For example, upon entering an airport, a user could automatically receive flight status information based on GPS location.

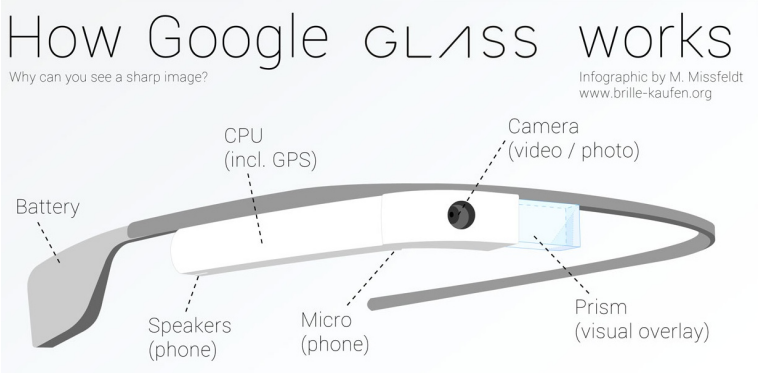


Figure 2.3: How Google Glass works[12]

The frame of the glass is adjustable so that it would fit any face, and got a forward-facing camera and a microphone using bone conduction to transfer sound. Bone conduction works by transmits sound through your bones directly. The headphones decode sound waves and convert them into vibrations that can be received by the Cochlea – so the ear drum is never involved. Meaning that people who are experiencing conductive hearing loss as mentioned in section 2.2, will perceive the sound in the same way as someone with normal hearing.

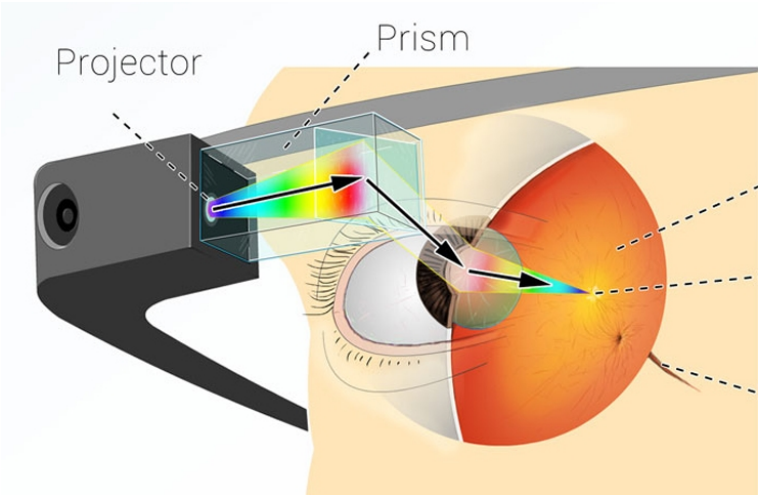


Figure 2.4: Google glass prism[12]

The glasses were designed to be as convenient as possible, allowing one to see the projected screen without it blocking the vision. The display itself consists of a semi-transparent prism that projects the images from the glasses to the user's eye as well as letting background light through(see figure2.4.), giving the effect of Augmented reality.

2.4 Machine learning background

Over the past two decades Machine Learning(ML) has become one of the most central parts of information technology and with that, albeit usually hidden, a part of our life. With increasing amounts of data becoming available there is a good reason to believe that the usage of smart data analysis will become an even more necessary part for further technological progress. In this section, we will explore some of the basic principles of machine learning as well as introduce some of the vital algorithms used in this thesis.

2.4.1 Types of machine learning

Machine learning is a subfield of artificial intelligence(AI) that evolved from the study of pattern recognition and computational learning theory. It is the discipline concerned with trying to create computer software that can learn autonomously[13]. Machine learning is usually split into three broad categories: supervised learning, unsupervised learning and reinforcement learning. Supervised learning is used in cases where the goal is to predict a specific target value based on the data. In unsupervised, we are more interested in trying to finding some pattern or trend. Reinforcement learning is something kind of in between the two former algorithms. It learns from feedback from the user about its performance, but it is not given the answer about what it is supposed to look for.
In an example we can look at the classification of fruit, supervised learning would be trained to know about different classes like: `apple`, or `pear` and so forth. When given data, it would find the closest match and label it respectively. In unsupervised learning, there are no assigned labels, and the algorithm tries to find a pattern in the fruit data, like shapes, size, color, etc. Finally, in reinforcement learning, it would classify it as in unsupervised learning, but then be given an input about its performance, and try to

better the result accordingly. For the rest of this chapter, we will focus on the different aspects and methods of supervised learning since this is the model we are using in this thesis.

2.4.2 Supervised learning

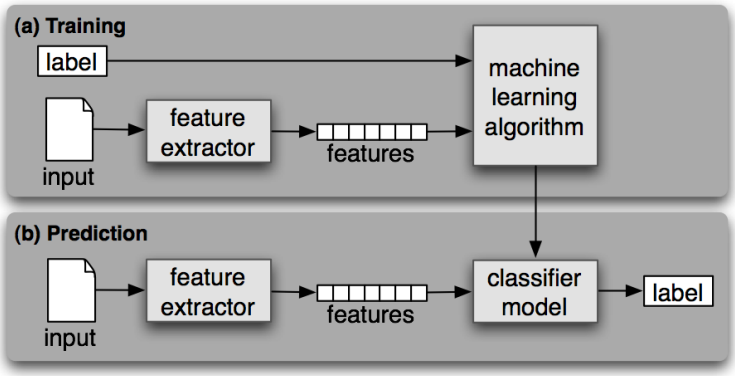


Figure 2.5: A supervised learning model consists of a machine learning algorithm/classifier that takes labeled feature vectors as inputs and produces a model which can predict the label of new feature vectors[14]

As mentioned in the section 2.4.1, the goal of a supervised learning algorithm is to classify some target value y based on some known data x. We hope to manage this by using some machine learning algorithm to help us find some function f that can be utilized on the dataset x to find y. This is usually known as a classification problem where y is a category or a label. An example of such a model can be the one from the figure 2.5 above. In other words, we are looking for the learning algorithm that gives us the best results from $g : x- > y$ where g is defined as the highest scoring value of target y. For now, we can just look away from the feature extraction part of the model as we will have a closer look at that in later sections.

2.4.3 Choosing the models

When choosing an ML model, it is important to look at what type of information one want to get out of the model, and what accuracy is needed to reach this goal. There exist many different types of classification problems, a simple example can be the one mentioned in section 2.4 that look at a binary classification problem between apples and pears, and the goal was to label the fruit as the one or the other. This issue can easily be scaled into a multiclass classification problem when we are either looking at more fruits or just other objects in general. One also need to consider how much data that is available and how good the features one can get from that data is. There exist many types of supervised learning algorithms with their pros and cons depending on what kind of data that gets fed to them. Section 2.5 is going to have a look at some of the different algorithms used later in the thesis, and how these algorithms work.

2.4.4 Cost function

When trying to predict data with the help of a learning algorithm, it is important to note that we can not expect to get the perfect results. However we do want to try to find the best decision function possible, what the decision can be is based on the problem at hand. Some examples of this could be

- Hypothesis problems - where the decision would be to accept or reject some hypothesis
- Classification problems - could be the example of trying to classify pears and apples correctly.
- Model selection problems - where the decision would be to choose one of the candidate machine learning approaches.

Usually, there would be some decision functions available to solve a problem. Say that we wanted to estimate the weight of an apple based on 5 samples. $x = (x_1, \ldots, x_5)$, we could use any of the decision functions $d(x)$:

- the median of the sample: $d(x) = median(x)$
- the average mean of the sample: $d(x) = mean(x)$
- the function always returning one: $d(x) = 1$

Even though having a function always returning one is quite silly, it is still a valid decision function, so we can see that determining which function to use is of high importance. One way to solve this problem is with the use of a cost function also known as a loss or sum of squared error function. The cost functions are used to describe the cost associated with all the types of possible decisions. It works by telling us what types of mistakes made for the decision functions that we should be the more concerned about. The best decision function is the one that yields what is known as the lowest expected loss. When working with supervised learning tasks such as classification or regression the goal for the algorithm is to find the hypothesis $h*$ among the functions H for which the risk $R(h)$ is minimal. In general, the risk cannot be computed because the distribution $P(x, y)$ is unknown to the algorithm. We can, however, get an approximation by averaging the loss function over the training set, this is known as the empirical risk.

2.5 Machine learning algorithms

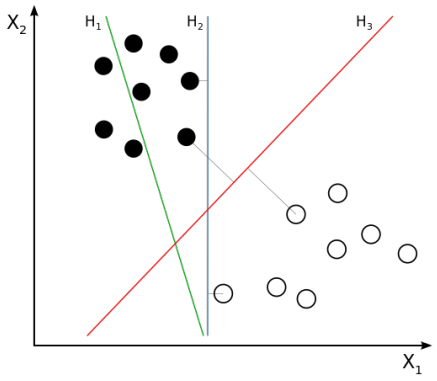
Dependant on the amount of accessible data, a supervised learning algorithm can create highly complex models with low prediction error. The algorithms often work by trying to predict some value v, comparing that value to the correct value l, and then correcting itself accordingly. By doing this many times over, the algorithm is slowly improving itself and would eventually get a hypothesis that would fit the data well. How this improvement works is dependant on the ML algorithm used. This section will have a look at some the of the most commonly used ML algorithms, some of which is implemented in section 3.8

2.5.1 Linear regression

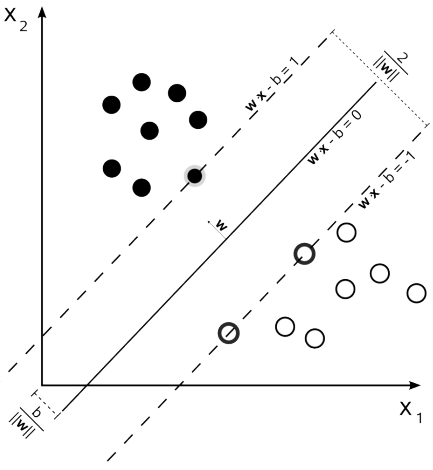
Linear regression might be one of the most well-known and well-understood algorithms in the field of statistics and machine learning. The method works by trying to find some linear function that fit some data x, by minimizing its cost function f . Only being able to use linear relationships between data points may seem like an unnecessary limitation, but it has the advantages of being very quick to implement and run times are fast. In many cases where the data points have a clear difference, the model might perform well and can be used before applying heavier ML models to see if its results are good enough.

2.5.2 Support Vector Machine

A Support Vector Machine(SVM) is one of the easiest to use and hardest to understand algorithms in machine learning. Because of the robustness and capacity of the algorithm to solve a large variety of tasks, it will often find good results even without being optimized for the specific problem at hand. Having these features has made SVM a very popular algorithm used within many fields. The goal of an SVM is to optimally separate points in a p-dimensional vector with a (p-1)-dimensional hyperplane. This is what is called a linear classifier.



(a) Different hyperplane splits, green does not separate the classes. blue does, but only with a small margin. red separates them with the maximum margin.[15]



(b) Maximum hyperplane split[16]

Figure 2.6

A hyperplane is just a generalization of a plane when working in more than three dimensions. Each data point will then be classified based on what side of the hyperplane it ends up on. When visualizing hyperplanes, we often use what is known as a decision boundary. Since a hyperplane always is linear it is impossible to visualize it in higher dimensions; we instead use a projection of the hyperplane onto input space. This is why a decision boundary sometimes can be curved or even discontinuous. So, in other words, the goal of an SVM is to find the best decision boundary to split some data X. Where the best decision boundary is defined as the one that maximizes its distance to all its support vectors. The support vectors are the samples $x \in X$ which lay closest to the decision boundary. The figure 2.6a above shows how different decision boundarys can be set, and figure 2.6b the optimal one.

The kernel trick Since we are working with a linear classifier, what happens if the data we got is not linearly separable? In SVMs this can be solved by something called the kernel trick or kernel method. The idea is to map the data into higher dimensions so that it becomes linearly separable.

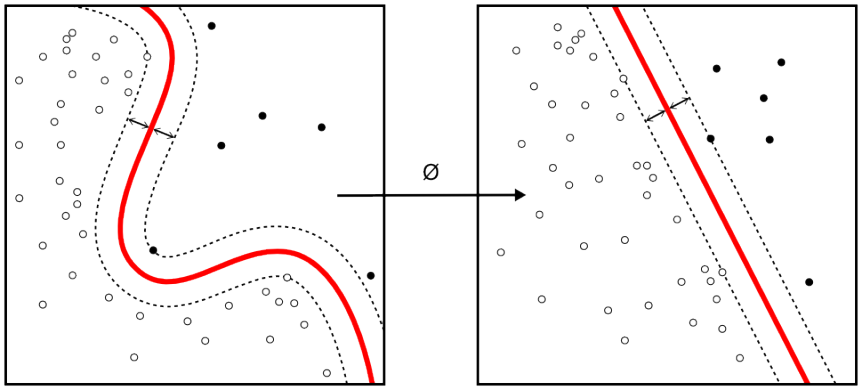


Figure 2.7: The figure is showing the transformation, denoted as Ø transforming the data (dots) into a linear seperable space [17]

A kernel is a similarity function; when given two inputs it returns how similar they are. So instead of using feature vectors, one could use the kernel to look at the difference between the data, and give the labels and kernel to a learning algorithm, getting a classifier. However, since SVM are not built to be used with kernels, we got to use them it in an other way. Luckily Mercer's theorem states that under some conditions, every kernel function can be expressed as a dot product in a possibly infinite dimensional feature space [18]. Moreover, given that many ML algorithms can be expressed as dot products mean that we can take an ML algorithm express it as a dot product and replace that dot product with a kernel. Meaning that we can switch the potentially infinite feature vectors with easy to use high-performing kernels, this is what is known as the kernel trick [19]. The figure 2.7 above shows how the data looks before and after the use of this method.

Tuning the SVM In the previous section, we went through the importance of the kernel function for an SVM algorithm. It is then necessary to note that there are different kernels to choose between, and picking the right one is of great significance to the results you get. Some of the most common used kernels are the polynomial and radial basis function kernel (RBF). In short, the polynomial kernel looks at the vector similarity in a feature space over polynomials of the original variables. Where RBF is a combination of all polynomial kernels of degrees $n \geq 0$. Both of the kernels takes a gamma parameter that defines how far the influence of each training sample reaches. Where using too few support vectors as influence can lead to a overfitted model and too many an underfitted one. The SVMs also has a C parameter that applies what is known as a soft margin (figure 2.8). Since the SVM always separates the labels, this can lead to a poor model in cases of wrongly labeled data or the examples being very unusual. Cortes and Vapnik proposed to solve this by giving the SVM some margin of error so that it could ignore or wrongly place some of the labels[20]. The C parameter is the variable that controls this soft margin cost function; this process involves trading error penalty for stability.

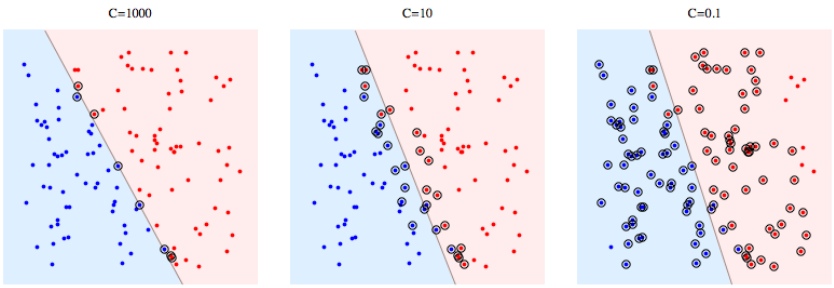
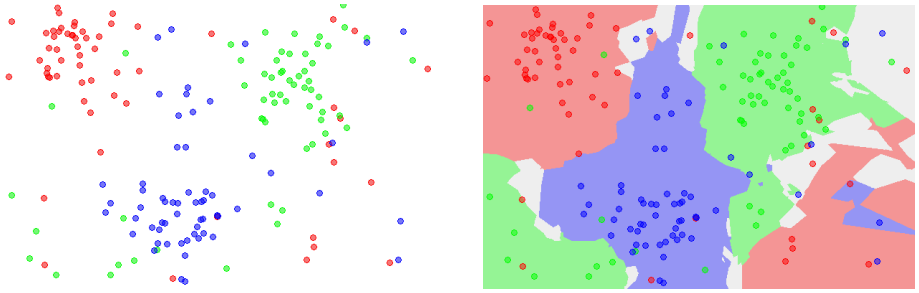


Figure 2.8: Showing how the hyperplane moves with different values of the soft margin parameter C. A higher value of C will choose a smaller margin hyperplane, and a lower value a larger one[21].

2.5.3 Nearest Neighbors

Neighbours-based classification is one of the simplest types of machine learning. It is called an instance-based or "lazy" learning algorithm [22] Because it stores all the training data instances instead of creating some general model. The algorithm works by looking at the k number of predefined training samples (or neighbors) that are closest to the new point. The number of neighbors k can be a user defined (k Nearest Neighbors) or based on local density points (radius-based neighbor learning). The distance between the points can be calculated by any metric measure, where the most common one is the Euclidean distance. In this thesis, we are going to focus on the most common of the algorithms, namely the k Nearest Neighbors (KNN).



(a) KNN dataset[23] (b) 5NN separation[24]

Figure 2.9: The models show how the KNN algorithm would separate a multilabel example of datapoints with looking the the 5 closest points(5NN)

Choosing the correct number of neighbors can significantly affect how the KNN model changes, an example of $k = 5$ is shown in figure 2.9. A small value of k means that noise will have a higher influence on the result. A high value of k will make the algorithm very computational heavy as well as somewhat defeating the philosophy of the algorithm that points who are near each other might belong to similar classes. Finding the best k is very dependant on your dataset.

Nearest Neighbour algorithm When working with the computation of the neighbors, one can understand that the time it takes to calculate the distance between all the data points can quickly become very high. Using the brute force approach to find the distance has a complexity of $O(dn^2)$ [25], for n data points and d dimensions. Meaning that with small datasets, the brute force approach will still be quite fast, but as the data set grows the approach becomes unusable, this problem can, however, be optimized with the use of K-D Tree.

K-D Tree K-D Tree is one of many tree structures that has been applied to optimize the neighbor computation. They usually work with the idea that if some data point T is very far away from point B. Moreover, B is close to F then T is also far away from F without having to calculate the distance between the points. This way we can reduce the complexity to $O(log(n))$ [26]. The K-D Tree will in most cases do very well when it comes to computing the distance between neighbors. It does meet a problem when the number of dimensions becomes too high, which can be solved by algorithms like Ball Tree [27].

2.5.4 Stochastic gradient descent

Stochastic gradient descent (SGD) works by putting together gradient decent and a cost function to create a linear classification model like linear regression or SVM. It has often been proposed to use gradient descent to minimize the empirical risk [28]. The algorithm has become more popular in the recent years mainly because of how fast data sizes have grown, limiting the learning algorithms to computing time instead of the sample size [29]. To understand how SGD works we are going to have a closer look at the gradient descent algorithm. Lets say we have some function $J(\Theta_0, \Theta_1, \ldots, \Theta_n)$, Gradient decent works by trying to reduce this function by changing the $(\Theta_0, \Theta_1, \ldots, \Theta_n)$ values untill it hopefully ends up at a minimum. For simplicity reasons we limit us to the two parameters (Θ_0, Θ_1). The first step is initializing the two parameters to some values.

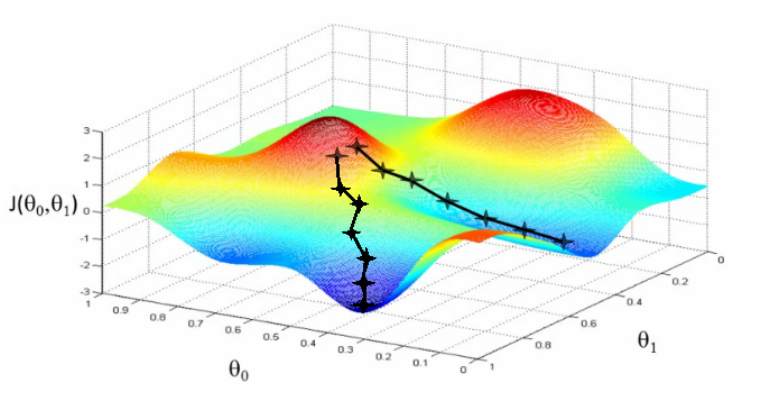


Figure 2.10: Gradient descent example with Θ_0, Θ_1 initialized two times [30]

The algorithm works by trying to find the fastest path to get down from the point it is currently at. From the example figure 2.10 above, one can see that depending on where it started, the outcome can differ a lot. The algorithm can be explained as:

$$\Theta_j <= \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1)$$

Repeated until convergence. In the algorithm above α is the learning rate and $\frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1)$ the derivative of the function. The arrow symbol states that we assign the value to Θ_j. The derivate looks at the tangent slope of the function and moves towards the minimum with the help of α. When putting together with the cost function, we end up with a linear regression model known as batch gradient descent (BGD).

$$J(\Theta_0, \Theta_1) = \sum_{i=1}^{n} (h\theta(x_i) - y_i)^2$$

where $h\theta(x_i)$ is the prediction of a hypothesis on some data x and y_i the actual value and we look for. The difference between BGD and SGT are that where BGD uses all the data from the dataset SGD computes the gradient of the parameters using only a single or a few training examples. We do this to get away from noise local minimums as well as it being computationally a lot faster.

2.6 Machine learning in practice

This section intends to introduce concepts regarding testing of machine learning models. It also looks into some practical challenges that can occur, and some ways to deal with these issues.

2.6.1 Cross validation

In any supervised learning experiment, the set of input/output examples provided is called a training set. Where the inputs are the variables to be predicted and the output are the features. Then we usually have a second set of examples, known as the test set, which is used solely for testing the performance of our model. One of the advantages of supervised learning is that we can use the test sets to get an objective measurement of learning performance since we know what label something should have. For these measurements to make any sense, it is vital that there is no overlap between the training and testing data. It is also of great importance that the data stem from the same source of domain. If they do not, there's no reason to assume that a model built for one will apply to the other. A problem with just using one test set is that estimators can be tweaked to until they perform optimally, meaning that the test set is not longer gives us a general performance of our model. We can handle this problem by using what is known as cross-validation, the three most known cross-validation methods are the Holdout method, k-Fold and Leave One Out.

Holdout method The holdout method is the simplest type of cross-validation, it works by again separating the data into another holdout set that can be tested on the model while optimizing it, using the test set solely for final testing of the model. A problem with this approach is that: (i) Our model's performance will decrease because of the now two datasets that are being held out from training. (ii) The results we get from our estimator will be highly dependent on the data points are chosen to be held out.

K-Fold K-fold is one way to improve the holdout method. It works by dividing the data into k subsets and apply the holdout method k times on our model. This way all the data points gets tested $k-1$ times and trained on k times as shown in figure 2.11. As k rises, this method becomes very computationally heavy, since it has to build the model from scratch k times. The score of this model is calculated by looking at the error rate across all k trials computed. We will have a closer look at how to calculate this in section 2.6.3.

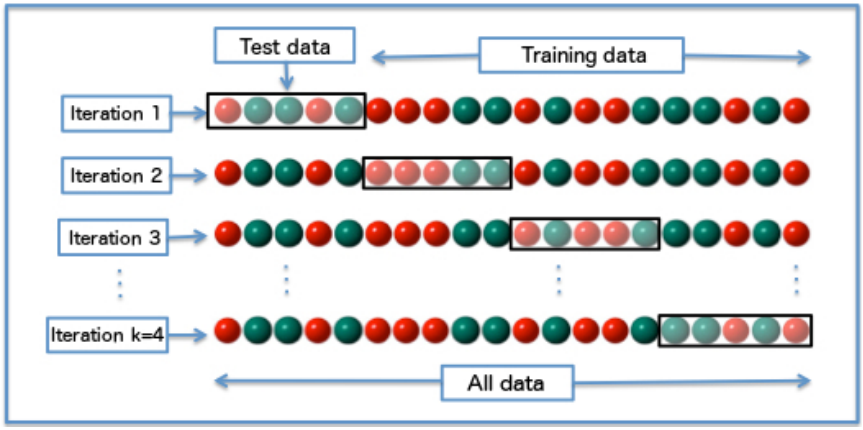


Figure 2.11

Figure 2.12: The figure is showing how K-fold splits and moves the training and test set. [31]

Leave one out Leave-One-Out works just like k-fold except that the subsets only contain a single data point. Meaning that the $k = n$ where n Is the number of data samples, making it heavy to calculate. The goal in cross-validation is to give every example from the original dataset the same chance of appearing in the test and training set. In this thesis, we are going to use what is found to be the most accurate model a variant of k-fold known as stratified K-fold[32], That we will look into in section 2.6.3.

2.6.2 Overfitting vs underfitting

One of the complications with ML is deciding what data is relevant and what is not. Within audio recognition, we try to differentiate between actual sound and just noise. When working with small datasets, there is a higher chance of them containing a large amount of noise that can lead the ML algorithm into believing it has found a pattern it can use to predict the output, where the truth is that the pattern originated only by chance. When building an ML model just having enough data will not suffice to make it useful. One of the most important as well as hardest aspects of ML is developing a model that is sturdy and generalize well. Having created a model that with 100% accuracy can predict its training data is most likely not a good thing. More likely it tells us that something is wrong with either the algorithm or the data. If the model has fitted noise or some pattern that only exist in the training data the model is likely to fail when given new information. This notion is known as overfitting, as shown in figure 2.13. On the opposite end, if the model fails to capture the underlying trend of the data, underfitting occurs.

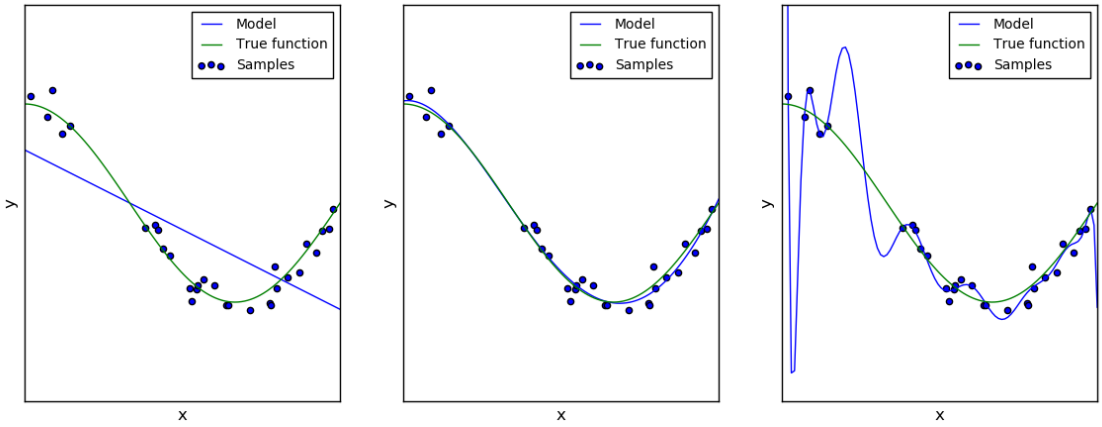


Figure 2.13: Example of problems concerning underfitting and overfitting. Model 1(left) is not sufficient to fit the training samples, therefore under fitted. Model 2(center) fits the data almost perfectly. Model 3(right) learns the noise of the training data and becomes overfitted.[33]

2.6.3 Model performance

In section 2.6.1 we decided to use cross-validation as a baseline to our ML-models performance. It is then important to look at what kind of score we get from the validator, namely how it is calculated.

Accuracy The accuracy of an ML-model is found by looking at all the correct predictions and dividing them by the number of total predictions. The correct predictions are known as the true-positive(TP) and true-negative(TN) values of a model, where false-positive(FP) and false negative(FN) are the incorrect ones, meaning that calculating the accuracy can be described as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

This would give us an indication of how our model is doing and would be a useful mesure in cases where the data is equally divided over the labels. With imbalanced data, however, this could lead to a false sense of security. Consider the binary classification problem of labeling apples compared to everything else; let's say that we train our model with ten apple labels and 9990 not-apple ones, using the accuracy measure here would give us a 99.9 percent score without correctly classifying one apple. In these cases, we can look at what is known as the F-Measure of the model.

F-Measure To find the F-measure of a model we are fist going to find out what is its precision and recall. The precision of a model is the percentage of selected elements that are correct; this can be calculated by

$$Precision = \frac{TP}{TP + FP}$$

And recall being the percentage of correct elements that are selected

$$Recall = \frac{TP}{TP + FN}$$

These two measures tend to depend on each other, so when the recall goes up, the precision usually goes down and vice versa. There might be applications where you want to focus on a high recall; it could, e.g., be when going through the evidence for a case and it is important to get all the data. In cases where it is not that important to know everything, but rather

something specific, e.g., in online advertisement, one would use precision. In our case, we want to look at something in between the two, namely the F-measure or more accurate the average F-measure. The Average F-measure looks at the trade-off between precision and recall, and it can be calculated by;

$$F - Measure = \frac{2PR}{(P + R)}$$

With P being the precision and R the recall. In section 3.9 we will look at how both the accuracy and F-Mesure can be used to score our models.

2.6.4 Feature extraction

To get a generalized model we need to try to optimize the input to make to make it easy for our estimator to separate the different classes. In this section, we are going to look at some of the methods we can use to achieve this goal.

Feature extraction As we talked about in section 2.4 the input data to an ML algorithm is common to referred to as features. Feature extraction works by combining data into a new set of reduced features as shown in figure 2.5, the goal with this action is to get a set of values that separate the classes well. Working with a good set of features will make it easy for the classifier to label instances correctly, with irrelevant features the models could end up worse than what they were by, e.g., finding correlating that do not exist. We will have look closer at how we find this features in section 3.7

Feature selection As with feature extraction, feature selection goal is also to help discriminate between classes. It works by selecting a subset of relevant features from the initial data to be used for construction the prediction model. There are many advantages to applying feature selection to a model: (i) By reducing the amount of data the classifier training time will decrease. (ii) Choosing the best feature can increase the performance of the model, (iii) reducing storage requirements. (vi) Generalizing the model by reducing overfitting [34]. When working with feature selection, there are three main categories, wrapper methods, filter methods and

embedded methods. Wrapper methods look at a set of features with the help of a predictive model to assign them an accuracy score, where different combinations of features are evaluated and compared to each other, meaning that it looks at the "usefulness" of features based on the classifier performance. Filter methods try to find the relevance of the features by looking at univariate statistics instead of cross-validation performance. The embedded methods work like the wrapper method but adding an intrinsic model building metric during training. It is important to note that some features on their own can seem redundant, but be able to improve a model significantly when used with others[35].

Feature scaling A common and good practice in machine learning is to apply feature scaling, also known as normalization, on the dataset. It works by standardizing the range of variables, usually between $[0, 1]$ or $[1, -1]$. The method is applied to:

- Ensure quick convergence for optimization problems, e.g., algorithms like gradient descent converges much faster with normalization than without it[36].
- Avoid too small models weights, for the sake of numerical stability.

Some objective functions in machine learning algorithms may not work properly without normalization. Most classifiers (like KNN) use the Euclidean distance to calculate the distance between two points. So if one of the features has a broader range of values than the rest, the distance will be controlled by this particular characteristic. We normalize the data so that each feature contributes approximately the same to the final distance.

Fourier transform Fourier transforms works by taking in a signal and expressing it regarding the frequencies of the waves that make up that signal. It is used extensively for a wide variety of signal processing applications including spectrum analysis, linear filtering, signal detection audio compression and much more [37]. Later in this thesis, we are going to use what is known as the discrete Fourier transform(DFT) as both a feature vector and data to for feature extraction. DFT is the numerical approximation of the Fourier transform.

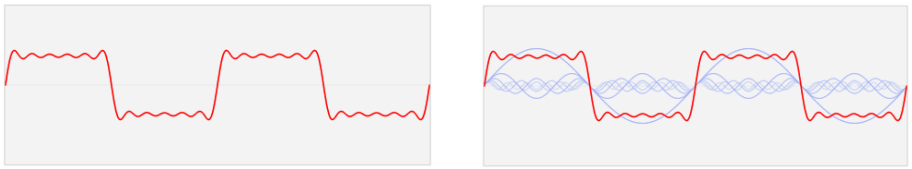
Looking at the Fourier equation

$$f(v) = \int_{-\infty}^{\infty} f(t)e^{-2\pi ivt}dt$$

Over a continuous signal, we can see that we are going to need calculus. Instead, we can generalize the formula into DTF and get

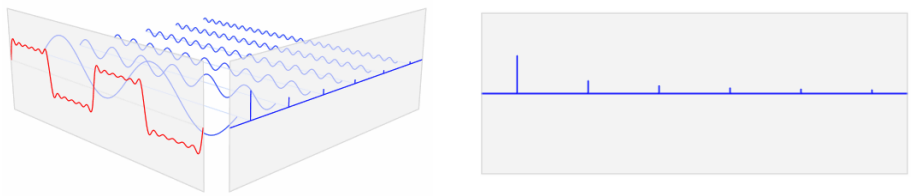
$$F_n = \sum_{k=0}^{N-1} f_k e^{-2\pi ink/N}$$

Replacing the infinite integral with a finite sum. Going deeper into the understanding of these equations is outside the scope of this thesis, but now we know that we use the DFT because it is both simpler mathematically and more relevant computationally.



(a) (b)

Figure 2.14: Figure 2.14a shows what many sound waves would look like together, figure 2.14b with each wave split up in the background [38]



(a) (b)

Figure 2.15: Figure 2.15a shows how we get the Hz from the given waves, and figure 2.15b the result.[38]

The figures 2.14 and 2.15 above show a practical approach to how the Fourier transform work on a different set of waves. Since we changed from regular Fourier defined on a continuous line, to a DFT over a discrete number of frequencies we need to determine which frequencies they are. We do this by sampeling continous function f over a set interval Δ giving us the function:

$$f_k = f(k\Delta)$$

Where k is defined as the sample data and Δ the sample rate. Here the Nyquist–Shannon sampling theorem states that the highest frequency we can represent by something sampled at intervals of Δ is a frequency having the wavelength of 2Δ [39]

2.7 Speech recognition

Historically, automatic speech recognition(ASR) has been driving force when it comes to the development of many ML algorithms; including the Hidden Markov Model(HMM), Bayesian learning, discriminative learning, structured sequence learning and more[40] It has been an active research area for many years, where one of the first models were created in 1930 by the Bell lab Laboratories[41]. In recent years the ML and ASR communities have become closer having a greater influence on each other, which has driven ASR out of controlled environments into our daily life. Applications like Apples "Siri" and IBMs "Watson" may give us the indication that the ASR problem has been solved, but the performance is still too poor for it to be implemented in some situation where it could is of great assistance(e.g., surgery). Currently, Google who is one of the leading companies when it comes to speech recognition has a system that is trained on over 230 billion and being able to understand the context of a one million word vocabulary[42]. Trying to create an ASR model would not be feasible in this thesis, but we are going to look at some of the different open solutions we can use in the coming chapter.

Part II

The project

Chapter 3

Implementation

3.1 Summary

This chapter intends to explain the various choices that had to be made and considered for the creation of the software cycle shown in figure 1.1. The first section will go through the research done at Briskeby School for the hard of hearing, and the conclusions made for further development of the project. Section 3.3 will look into the software tools, libraries, and languages used in the implementation of the lifecycle. The following sections are focused on the creation of a glass application(3.4) and the siren recognition system (3.6). The final part (3.8) is going to look at the experiments done to optimize the ML model for the recognition system.

3.2 Briskeby - School for the hard of hearing

On the 15. October we traveled to Briskeby Upper secondary school for the hard of hearing, to present our thesis as well as interview the students. The goal of the meeting was to gain some insights in what they thought about the concept of smart glasses in general, and in what way the could see the glasses being able to be used as an aid. We held a short presentation about the background of the Google glasses, how they work, and what features they have. Since none of the students had previous knowledge of the smart glass technology, we decided to give them some pointers to how we could see Google Glass being used to aid them, for then to get their input on the matter.

The students attending were in the age range of 16 to 19 years old. After the presentation, we gave each student individual a sheet of paper with the questions:

- Can you see Google Glass becoming useful as an aid for you in some way?
- What kind of situations can be hard for you in your day to day life?
- How do you believe Google Glass can aid your day to day life?

Table 3.1: Briskeby participation

Participants	**25**
Blank answers	4

Figure 3.1: Can you see Google Glass becoming useful as an aid for you in some way?

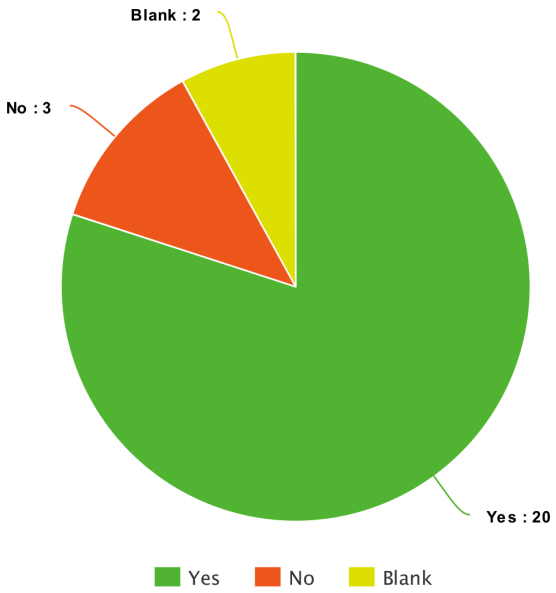


Figure 3.2

Figure 3.2 shows that most of the students saw a potential in the Google glasses for being used as an aid. These results might, however, bias towards the use of Google Glass considering the students already knowing the goal of our study, known as the observer expectancy effect[43].

Table 3.2: What kind of situations can be hard for you in your day to day life?

Topic	**Sum of answers**
It being hard to understand what is said in a crowd	5
Noise	12
Hear traffic and alarms	2
GPS related	2
Understand the origin of a sound	2

The second question is shown in Table 3.2 returned some of the more interesting results considering further development. When asked about general problems occurring in their daily life, the response was not smart

glass-related. Letting us see if any issues could in some way be helped with the use of smart glasses. In figure 3.2, noise refers to sounds that would make focusing/hearing something hard. The students often used talking in groups as an example. The question "How do you believe Google Glass can aid your day to day life?", ended up with a low number of results that were biased towards the examples we came within our presentation.

Table 3.3: Briskeby results

Topic	**Sum of answers**
Speech to text	4
Connect to hearing aid	1
Direction sound is coming from	1

After the individual session, we talked to the students with the results from table 3.2 in mind. The intention was to get a closer understanding of how we could use the glassware to help. They recurring problems were: (i) keeping up when multiple people were talking (ii) not noticing and having difficulties with understanding the direction when someone was saying their name. (iii) The students who had a driver's license also agreed on having issues noticing siren and similar driving-related problems. Solving all of these problems would be quite hard, but we looked into the different options we had and where the Glass could be used. The main issue with (i) and (ii) is that they are hardware dependent on triangulation to find the source of the sound, considering that Google Glass only has one microphone ruled this option out. (iii) Would be feasible to implement, and could be one of the most crucial factors to investigate. Only 3 of the students had a drivers license, but they all reported on problems with picking up on important signals in driving related situations, especially in cases of high noise.

The creation of a system that recognizes horns and sirens, and alerts the user while not having to distract them to look away from the current situation could, in theory, be of great aid to all hearing impaired.

Thus based on this research, we decided to focus on the creation of a siren recognition system, implementation of a speech recognition service as well as looking into the possibility of letting the user train a model to recognize their name. In the coming sections, we are going to look at the technical aspects of creating as well as implementing these models.

3.3 Language and environment

When building the different sections of our software architecture, some requirements was taken into account;

- It had to be able to communicate with each different part of the system to work
- There must be a sufficient amount of libraries to reduce development time
- The languages must be well known to the developers

When implementing the glassware, we were limited to development in Android Studio and the use of Java. Previous knowledge of both Java and the platform was a part of making Google Glass the desired hardware product. The server was written in javascript and had a rather simple task in this thesis of mainly forwarding data to either the glassware/connected device or the recognition services. The siren recognition service is written in Python and based on the Scikit-Learn library. Mainly because of it being high level and well known to both developers. Scikit-Learn is an open source library containing a vast variety of easy to implement ML algorithms. The ML algorithms chosen were:

- Stochastic gradient descent(SGD)
- Support Vector Machine(SVM)
- K Nearest Neigbours(KNN)

The SGD model got chosen because of it being very efficient when it comes to training. Enabling fast testing on larger datasets and speeds up the development process, the SGD model can be built on either an SVM on the linear regression model, meaning that the results are not necessarily worse than other models. SVM got chosen because of its versatility, being able to take usage of the kernel functions mentioned in section 2.5.2, it is also quite efficient when it comes to building the models. Lastly, the KNN was chosen because of it being able to yield very good results in cases where we got an irregular decision boundary and smaller datasets.

Fourier The Fourier signal processing library is used because of sirens normally being within the frequency range of 1000-2000Hz[44] hopefully

enabling the feature extraction to find a pattern in the data.

3.3.1 Project overview

Before going deepening into each part of our project, we will have a rough overview of how our software lifecycle works.

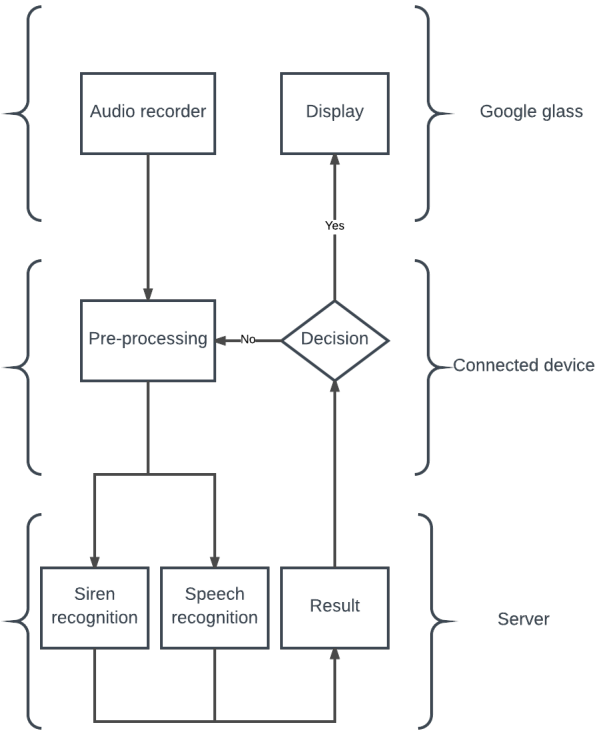


Figure 3.3: Software lifecycle

The figure 3.3 above shows how the various parts of our program correlate. The cycle starts out the in the Google Glass by recording audio that gets pre-processed and sent to our server from a connected device. The connected device serves as a processing medium for the glasses and is usually either a smartphone or a tablet. The server then returns the analyzed data to be visualized for the user of the glass. This is the essence of how our program flow works, and each part will be closer explained in the coming sections.

3.4 Glass implementation

in section 2.3 we went through the basics of how Google glass works. In this section, we will look at the implementation of the project on the glasses and the different steps we went through under the development.

3.4.1 Glass UI

Following Google's development principles for the glass, we created a clean and minimalized UI. The application interface consists of a black(transparent)display. When the application detects sound in the form of speech, it will visualize this as a white text for the bearer of the glasses. The text will scale automatically to fit the screen and remove itself after a given period has passed. If a siren is detected, this will be presented as an alarm symbol for the user. The siren symbol is given priority over text and will always be shown in the top right corner. Any text currently on the screen will then be scaled around the alarm symbol.

Figure 3.4: Google Glass main interface

The application got an options menu given the user the possibility to turn on/off the speech to text functionality as well as exiting the application. The options menu is also transparent. Showing the user if something has happened in the background.

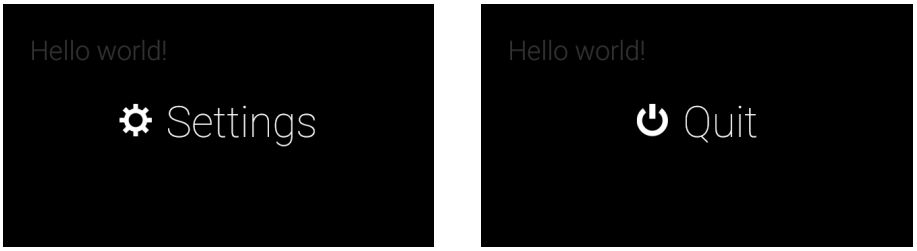


Figure 3.5: Settings

3.4.2 Glass development

As shown in figure 3.3. step 1 we can see that everything starts within the glasses. When initializing and using the audio recorder, there are some important factors to have in mind: one of the first things to into is the sample rate.

Sample rate The sample rate of a microphone decides how many samples of data one wants to record for each second of time. As we talked about in section 2.2 for the human ear, audio is perceived in continuous form. Whereas logic circuits rely on electronic oscillators that sequentially trigger to a specific task. Meaning that an audio signal has to be fed in small pieces to the CPU. The process of reducing this continuous signal to a discrete signal is what we call sampling. The rate of this is how many samples that are then evenly split over a duration of a second. With our current version of Google Glass, these rates can vary between 8000Hz up to 44100Hz. We mainly look at the three different rates in shown in table 3.4

8000Hz	Adequate for human speech, usualy used in telephone transmissions.
16000Hz	Used in most modern VoIP and VVoIP communication products
44100Hz	Audio / CD quality

Table 3.4: Hz values explained

It is important to understand that the choice of sample rate has a lot to do with the usability of the application. The trade-off we look at between is the quality vs. bandwidth/time usage. With the higher rates yielding better results and lower ones having less response time/being easier on the net usage. In other words, a lower rate could e.g., give the user a faster response about an emergency vehicle coming closer but the lack of data

could also prompt a suboptimal speech recognition result. We ended up using a rate of 16000Hz because a lower one would not be supported by speech recognition software, and 44100Hz using to much bandwidth and time. We will have a closer look at how this could be optimized in section 4.1.1.

Audio processing The second part of our lifecycle 3.3 indicates the start of the audio processing done on the connected device. Even though this also can be run on the glass itself, we use the connected devices CPU to save power on the glasses. After the microphone is initialized, it starts recording surrounding audio into an n-size buffer. As mentioned in paragraph 3.4.2 bandwidth is a major factor when it comes to usability, so having the application constantly forwarding data to the server would be suboptimal. One solution to this was pre-analysing the buffers on the glassware itself. Initially, the glassware will transmit audio as fast as possible to the recognition service (usually this is done with a 10ms delay between recordings not to overflow audio calls). Though if the recorded sound does not meet the required decibel level of normal human speech at 60 db[45] within 6 seconds, we assume that there is no vital information to gather at the moment and set a delay between recordings to 250 ms. This delay is again reset to 10 ms if any recording exceeds the dB threshold. Not only does this minimize data usage, but it also reduces the CPU usage of the glassware giving it a longer power lifespan. When n seconds worth of buffers has been filled, we check if the current buffer ends on a high value on the dB scale. If it does, we trace back through the buffer finding the previous low and stopping the sample there. The cutoff part will then be added to the beginning of the next buffer to be processed. We do this to help the speech recognition service by trying not to send words that are split in two. The original part will then be encoded and send to the server. This brings us to the last point of the lifecycle. The response from the server will contain the two accuracy values, one from speech and one from siren recognition. Along with a string containing the speech recognitions highest rated response. If the accuracy value of a siren is met, it will be displayed as a siren on the screen, shown in figure 3.4. Same goes for the analyzed speech. However, if none of the requirements are met, we roll back and add the data to the beginning of the next buffer. Creating one of twice the size giving the recognition services more to work with. This can only be done once for a buffer and is not possible if its next buffer has already been

sent for analysis. We do this mainly to try and improve the speech to text results since they do perform better the more data they get.

3.5 Speech recognition

When implementing a third party speech recognition service to the software lifecycle, some criteria had to be met: (i) The service has to be free. (ii) It had to fit into the flow of our software lifecycle (iii) There could not be a low limit of weakly server calls. (iv) It needs to work for English and if possible Norwegian.

Max Manus We reached out to the Norwegian company Max Manus who delivers ASR software directed towards the medical community[46]. They were interested in the thesis and the possibility of an integration of their system but explained that they mainly focused on the recognition of medical phrases and that using their system as a speech to text service would be futile.

Google The second test was done with Googles speech recognition software. Being one of the leading actors within the ARS field as well as the creator of the Google glass would open the possibility of a seamless integration to our glassware. Googles ASR service had great results individually, but it did not let the developer control the flow of the audio data. Meaning that for our siren recognition to work beside it, one would have to send the recorded data twice over GSM/WIFI. There existed a workaround for this issue, by using their web browser recognition implementation instead. However, this was not the intended use of the system, removing the Google service from the list of possibilities.

IMB The third option was the use of IBMs Watson. IMB created a beta platform called Bluemix that supports several programming languages and services[47]. Bluemix has access to the Watson development cloud that let us implement their ASR service to our software. The ASR had good stand-alone results and was easy to apply to the lifecycle. The service had close to unlimited server calls, making it the best option of the three.

Section 3.4.2 went through some of the approaches made to better the results of the ASR service. Trying to understand natural language is a much heavier process than the classification problem of something being a siren or not. For an ASR system to work properly, it needs full sentences to be able to put words in context, on the other hand waiting, e.g., 4 seconds instead of 1 to be warned about a siren can in many cases be the deciding factor. In this thesis we decided on letting it be a trade-off between the two, focusing on the siren recognition service. The further work section will, however, have a closer look at how this could be improved.

3.6 Siren recognition

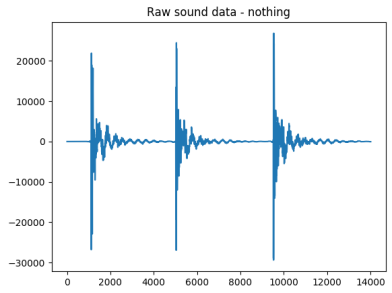
In this section, we will look at the steps taken for our siren recognition service to work. We will start out by looking at the data that we gathered, and then how we proceeded to prepare that data for feature extraction. In the coming sections, we will look at the various features we got from the processed data and how there features fare with our different types of ML models.

3.6.1 Data gathering

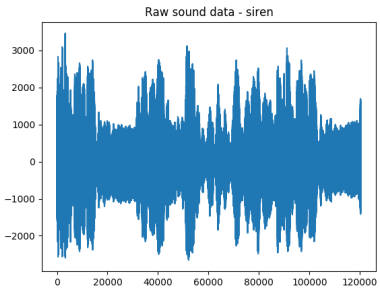
To be able to create an ML model we needed to gather the data to train it on. We started the project with no samples, and as mentioned in section 2.6.2, having enough data is important when it comes to ML. To get an initial database we gathered samples from the various sites on the internet and refactored the data to fit the sound format we set for the glasses. After building a simple model, we started adding audio we recorded ourselves with the glasses/cell phone. With little data to work with we tried to sample sounds, you might regularly encounter that resembles sirens, e.g., birdsong, in hope to not get an underfitted model in real time use. The database built consists of about 100 different audio files.

3.6.2 Preprocessing the data

The next paragraphs will look at two different samples of data and how they are processed before feature extraction occurs. The first sample 3.6a is of someone knocking on a door, and the second one of a police siren 3.6b.



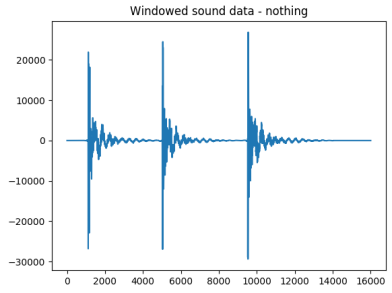
(a) Raw data of a non-siren sound



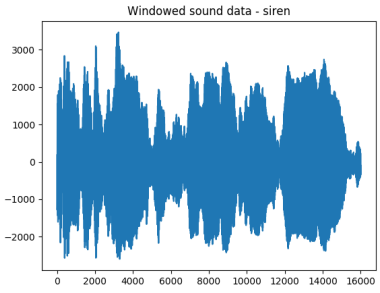
(b) Raw data of a siren sound

Figure 3.6

By looking at figure 3.6a and 3.6b we can see that the data is considerably different when it comes to both power and length. When preparing the data for feature extraction, we want it to be in the same format, so our first step is to window the data into an equal length of 16000 samples (or one second of recorded audio).



(a) (a)Windowed non-siren sound

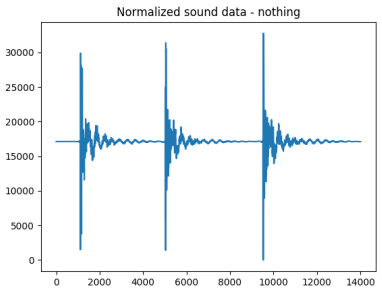


(b) (b)Windoed siren sound

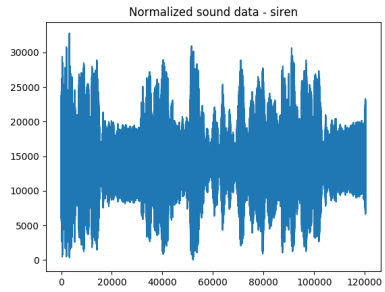
Figure 3.7: Figure showing the windoing of a non siren 3.7a and siren 3.7b datasample.

Note that figure 3.6a contains less than 16000 samples, when this occurs we simply fill the end of the buffer with 0 values so that the data will not have an effect on the outcome of the model. The windowing feature works

by splitting up data into the same length, then traversing over the buffer by moving n samples at the time. Meaning that a given data point can be processed up to $rate/n$ times. We do this to give our model more features to look at for a better understanding of the audio data. After windowing the data, the next step is to normalize it. As mentioned in 2.6.4 many ML algorithms tend to give better results when handling data in the same format, but when it comes to audio classification when and where it is best to apply normalization to the data varies. In our situation, three places could be considered for normalization. The first is at the raw audio level, as showed in figure 3.8a below.

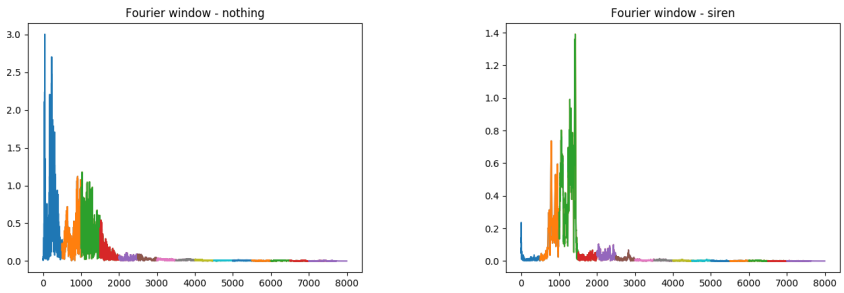


(a) normalized non-siren sound



(b) normalized siren sound

The second part would be during the Fourier transform, which is the last step of audio preprocessing before feature extraction occurs. The last one would be after feature extraction is finished and model is ready to be built. In section 3.9.1 we will have a closer look at the difference it makes. As mentioned in section 2.6.4 Fourier transform works by taking a signal and expressing it regarding the frequencies of the waves that make up that signal. Meaning that using the entire Fourier of an audio signal as single features would be possible and most likely yield good results from an ML algorithm. In this thesis, we instead apply the same method as earlier and window the Fourier into length n with step size s. (1000/500 is used in the figure below, a closer look at how changing these values will affect the overall performance of model is concluded in section 3.9)) Each of these steps will then be used to extract the features used in our final model. Why we do, this is explained closer in the coming section.



(a) fourier non-siren sound

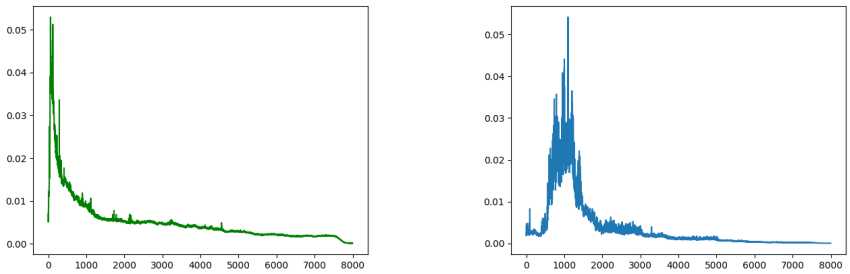
(b) fourier siren sound

Figure 3.9: The figure is showing how the Fourier signal gets split into equal windows, displayed in various color codes.

3.7 Feature extraction

Feature extraction is one of the most important parts to making a decision model with a low prediction error. As mentioned in the section above you could just use all the data points as features, this would, however, end up being quite a power and time-consuming. To speed up the process we apply windowing to split the data into equal lengths, and then use feature extraction of each window. Contrary to using the entire Fourier as features, doing feature extraction of the whole Fourier signal would make little to no sense, since we would not know where in the specter the features would stem from. A valuable part of using all the features is seeing how the model would visualize. This can tell us much about the differences in the data gathered, and show us is if there indeed is a clear difference between sirens and other sounds in the Hz specter. The goal of this section to find what features diverge the most from each other, and if they can be used together to build our ML model. Looking at figure 3.10 we can see that we most likely got a good foundation for our feature selection based on the visible difference in the two models So for the rest of this section, we will look at how we can extract and use features from the Fourier and if we can find any differences between the features extracted.

Feature 1 - Raw fourier The first feature we are going to look at is the raw Fourier of the data. As mentioned in section 3.3 sirens usually lies within the Hz range of 1000-2000Hz, which hopefully will have a visual difference from the non-siren audio data. In the figures 3.10 below, we can see mean of the entire dataset of the siren and non-siren sounds with each point as a part of the 0-8000Hz specter.



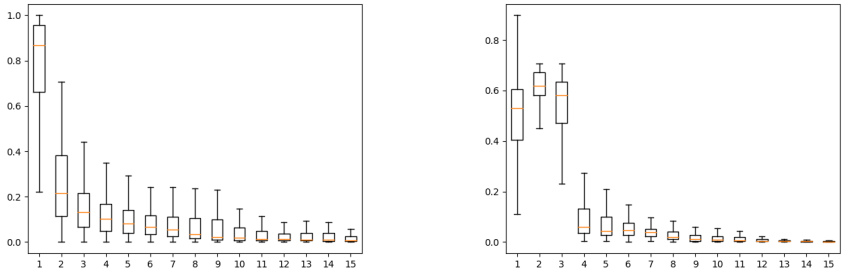
(a) fourier non-siren sound

(b) fourier siren sound

Figure 3.10: The figures are showing the Fourier mean of all non siren(3.10a) and siren(3.10b) audio samples

When using the raw Fourier as a feature vector each data point becomes a feature, meaning that every sample will be compared to the others samples within the same Hz range. Figure 3.10 shows a promising difference between the two labels.

Feature 2 - Magnitude The magnitude of a Fourier is a simple as looking the highest value sample of each buffer. From figure 3.10 we can guess that the non-siren sounds will be starting out high and dropping quickly, and the sirens growing over the first 1000-2000Hz.

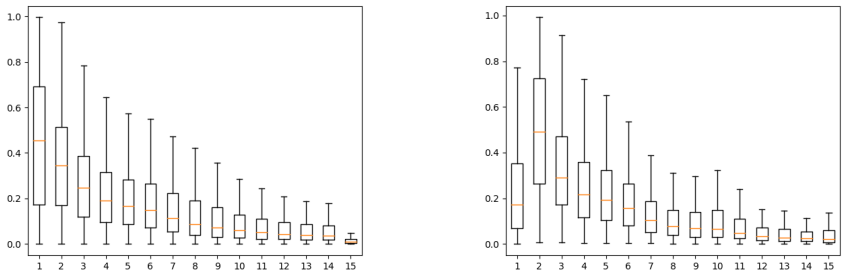


(a) magnitude of non-siren features (b) magnitude of siren features

Figure 3.11: The models are showing 15 features of magnitude for all fourier windows from their respectable label

As we can see in the figures 3.11 above our assessment looks correct, figure 3.11b does start out a bit higher than expected from the overview from figure 3.10, this might tell us something about the outliers of the model being quite high at the beginning of the frequencies or that the first Fourier window catches the start of the spike in 3.10b.

Feature 3 - Minimum The next feature we will look at is the minimum of each window. So far there has been little to no information that can tell us about how this feature will fare.

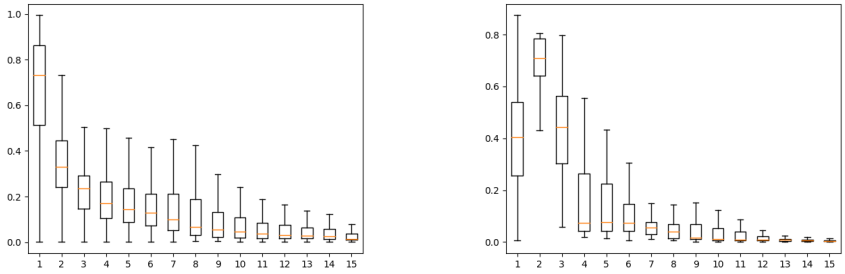


(a) Minimum of non-siren features (b) Minimum of siren features

Figure 3.12: The models are showing 15 features of minium value for all fourier windows from their respected label

The extracted minimum features seen in the figures 3.12, resemble each other much more than the those of the magnitude example 3.7. There is a difference in the 1st window, but the rest seem to follow each other too closely for the minimum value to be used as a type of feature extraction.

Feature 4 - Mean The Mean is calculated by taking the sum of the sampled values and divide it by the number of items in that sample or $\bar{x} = \frac{x_1+x_2+\cdots+x_n}{n}$ in our case $n = 1000$ because of the length of the Fourier buffers mentioned in 3.6 and the samples being The values in its given buffer. Here we can expect a flow more equal to the 3.10. Since we are working with all of the data points.



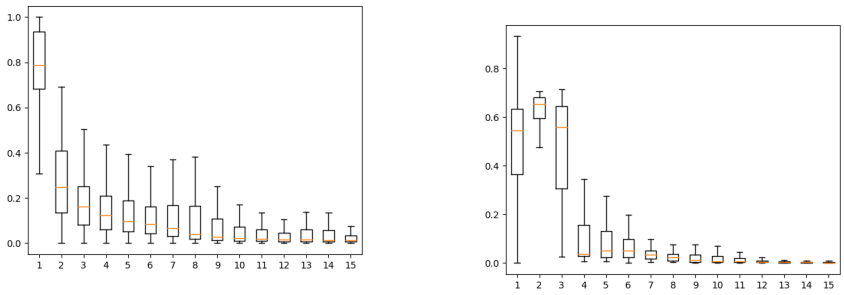
(a) mean of non-siren features

(b) mean of siren features

Figure 3.13: The models are showing 15 features of mean for all fourier windows from their respectable label

We can see that the initial spike from figure 3.11b has been lowered by about 0.1 this tells us that our assessment from the previous feature seems correct. The value has been reduced so that the outliers might be quite high at the beginning of the Hz spectrum, but the value is still above the center meaning that the average probably takes a part of the initial spike.

Feature 5 - Standard deviation Calculation the standard deviation of the data is done by taking the square of the variance of the mean. To find the variance of the mean one can look at the distance from each data point to the mean of the buffer and square the results. or $\sigma^2 = \frac{\sum(X-\mu)^2}{N}$ Where μ is the mean and N is the number of data points in the distribution. After finding the variance, we simply square it to get the standard deviation.



(a) standard deviation of non-siren features

(b) standard deviation of siren features

Figure 3.14: The models are showing 15 features of standard deviation for all fourier windows from their respectable label

3.7.1 Feature extraction discussion

Looking at the initial models of the raw Fourier data 3.10, there was a clear difference in the siren and non-siren sounds. Most of the features extracted from the Fourier in the following analyses also provided promising results when it comes to being used as data for an ML model. In the coming section, we are going to use the three most prominent features, namely the magnitude, mean and standard deviation as well as the raw Fourier to built our ML models.

3.8 Building the model

So far we have gone through how the pre-processing of the audio data has been utilised (section 3.6), and we have seen a trivial example of how the entire Fourier transform of an audio signal could being used as features (section 3.9a) as well how features could be extracted from it (section 3.7). In the coming experiments, we will look at how the three ML-algorithms SGD, SVM and KNN fare when built on different instances of training data.

3.8.1 Splitting the data

Our entire dataset consist of 7024 samples, where 1223 are siren sounds and 5801 are non-siren ones (based on the window size used in section 3.7 and will change if the window size changes). As mentioned in 2.6 Building and testing a model on the same data would be a methodological mistake and could lead to problems such as overfitting. To prevent this, we are going apply a version of the cross-validation method K-fold known as Stratified K-fold as mentioned in section2.6, Like K-Fold, Stratified K-fold works by splitting the training set into k smaller sets. Each of the $k - 1$ sets are used to build a model tested on the remaining k set. The only difference is that the Stratified model tries to even out the labels in each fold if it is possible. This way each fold should represent a fair result, and not be too heavily biased against one label. The validation set of the model consists of 280 siren and 723 non-siren samples. Leaving us with a total of 6021 samples in our training model.

3.9 Testing the features

This sections goal to create the optimal siren recognition model, the development is going to follow the architecture of figure 3.15. Initially, some analysis of normalization and window size, brought to light in section 3.6 is going to be conducted. Followed by:

Experiment one is going to look at the results from builing the ML models on the how the raw Fourier data.

Experiment two will use the chosen feature vectors created in section 3.7 to built the models.

Experiment three will try to find the best solution from the two previous experiments concluded, and create a combination of what is thought to be the best features.

Experiment fourhas the goal of developing the highest scoring model, by tweaking the ML-algorithms parameters based on the results of experiment three.

The ML algorithms use their predefined automatic values, which should give a good indication of how well the models will work. Note that the SVM is using a linear kernel, k = 5 in Nearest Neighbors and SGD is using a linear SVM loss function. For more information about the predefined values see section 4.2.

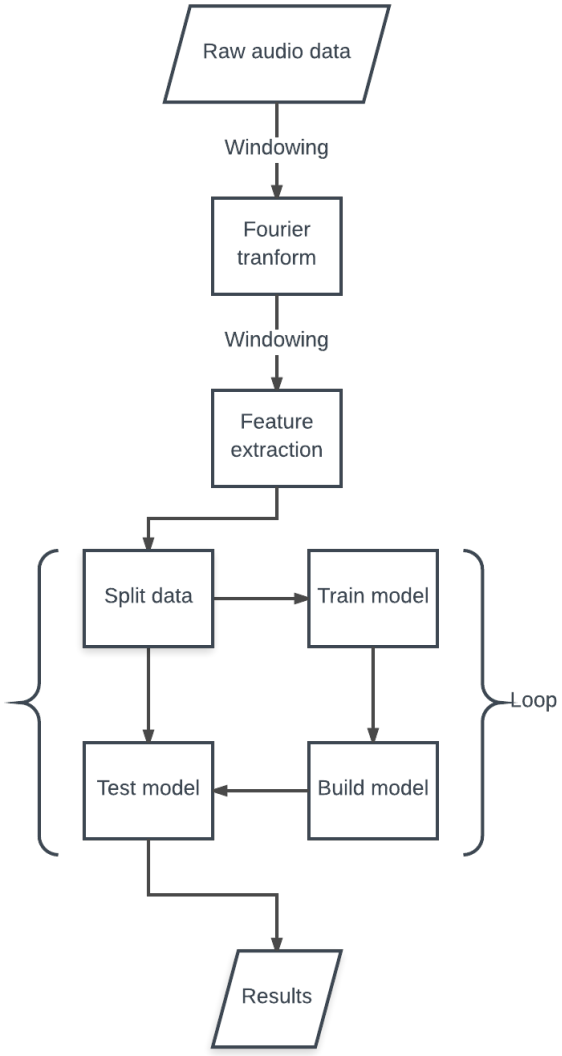


Figure 3.15: The figure shows the steps taken for the creation of a ML model. Square represent processes and parallelograms input/output.

Description of lifecycle

1. **Raw audio data** is the dataset reviced as described in section 3.6.1.
2. **Windowing** implies the splitting of data into equals lengths as done in section 3.7.
3. **Feature extraction** this step recreates one or a combination of the features from section 3.7.
4. **Loop** The following steps will repeat untill all results are found.
   (a) **Splitting the data** Here the data will be split into training and test sets as explained in section 3.8.1.
   (b) **Train model** Takes the $k - 1$ features to train the model.
   (c) **Built model** The model is created and is ready to be tested.
   (d) **Test model** The left out data gets tested on the finished model, and the intermediate score is forwarded
5. **Result** Holds all the intermediate scores from the models tested in the loop

3.9.1 Optimizing the data

Section 3.6 introduced the notion of normalization and windowing to improve the data before feature extraction occures. Before diving further into the creation of optimal ML model, we are going to have a look at how changing these variables can affect the outcome of the results. For simplicity, we are going to use the SGD model built on the raw Fourier data.

Normalization Section 3.6.2 looked into to three different normalization options:

- Raw audio normalization
- Fourier normalization
- Feature normalization

Using cross validation with 5 folds gave the following results:

Table 3.5: Normalization types

Normalization type	**F-measure**
Raw data	0.86
Fourier	0.88
Feature	**0.90**
None	0.79

Table 3.5 Shows the average F-measure over 5 folds, the optimal normalization method is after the features have been created as marked in bold.

Window size This far in the thesis the value of the windowing method has been set to 1000 window size, and 500 step size. Our first test is to see whether even or overlapping windowing is the best option. For simplicity, the coming tests are done on the mean of the Fourier data.

Table 3.6: Even vs overlapping windowing - Mean of Fourier

Windowing	1000/500	1000/1000
F Measure	**0.87**	0.75

Figure 3.6 shows that there is a significant difference between the two. The next step is to see if the optimal window size can be obtained. Figure 3.16 shows the results of the test done on five different windows/step sizes.

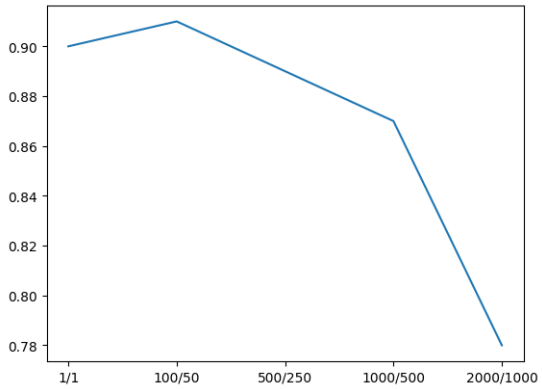


Figure 3.16: The figure shows five different window and step sizes, with their F-measure value built on the mean of the Fourier.

The analysis of normalization a window/step size has concluded that the optimal place to conduct normalization is after the feature vectors have been created and that the overlapping window size of around 100/50 yield the best results. These preprocessing steps will be applied to all the future experiments.

3.9.2 Experiment 1 - Fourier

The first experiment is built with the raw Fourier data points, giving us the feature vector.

$$f = (x_1, x_2, x_3, \ldots, x_n)$$

With n being the length of the Fourier buffer at 8000 samples. The training of the model is going to take significantly longer time than with extracted features, but it should, in theory, give around the best results we can get. We run our k-fold at 5 with the same seed for all the models.

Table 3.7: Experiment 1 - Fourier of learning algorithms

Model 1 **Raw Fourier**			
Algorithm	Fold	Average F-measure	Accuracy
SGD	F1	0.89	0.95
	F2	0.82	0.96
	F3	0.95	0.98
	F4	0.89	0.97
	F5	0.93	0.98
	mean(+/-std)	**0.90 (+/- 0.08)**	**0.97 (+/- 0.02)**
KNN	F1	0.95	0.98
	F2	0.75	0.91
	F3	0.98	0.99
	F4	0.90	0.97
	F5	0.97	0.99
	mean(+/-std)	0.91 (+/- 0.17)	0.97 (+/- 0.06)
SVM	F1	0.92	0.97
	F2	0.84	0.95
	F3	1	1
	F4	0.90	0.97
	F5	0.95	0.98
	mean(+/-std)	**0.93 (+/- 0.10)**	**0.98 (+/- 0.03)**

The 3.7 shows the F-measure and accuracy of our three ML models over five folds. Looking at the difference between the accuracy and the F-

measure mean scores of the models, we can make a calculated guess that all of the models do well when it comes to classifying the non-siren sounds. Based on the dataset having more non-siren labels than siren once, and all algorithms showing an excellent mean accuracy score of 97 / 98 percent. The highlighted values represent the best scores from the different algorithms with, SVM giving the best results accuracy / F-measure results and SGD having the least amount of standard deviation. From the single folds, some noteworthy values come from F2 and F3. In F2 all algorithms get results below 90 percent and F3 the SVM manages to get a perfect score. Based our assumption that our models do well when it comes to labeling non-siren sounds, we can guess that F3 might contain mostly non-siren test data, where F3 is most likely the fold having the largest amount of siren labeled data.

Table 3.8: Confusion matrix of test set built on raw Fourier data

Algorithm	**Confusion matrix**		**Accuracy**
SGD	244	36	0.95
	5	718	
KNN	236	44	0.94
	8	715	
SVM	247	33	**0.96**
	6	717	

The matrixes above shows how the test set got classified when built on the models trained in table 3.7. The top left plots refer to the true positive value, meaning that a siren label was correctly given to a siren sample. Top right is the true negative, or that a siren has been incorrectly labeled as nothing. Down left is the false negative, that a non-siren sound has been labeled as a siren, and down right the true negative, that a non-siren sound has been correctly labeled as non-siren. As guessed based on the scores in table 3.7 the classifiers indeed does very well when it comes to the non-siren classification. Looking at the accuracy of the matrixes 3.8 compared to the training set we see that all the models get a lower score, but the change is so little that it is of not of any great significance.

3.9.3 Experiment 2 - Single mesure feature vectors

The following models are all based on the same feature vector v

$$v = (p(f_1), p(f_2), p(f_3), \ldots, p(f_z))$$

where f contains the fourier data from window z and z can be explaied as

$$\frac{w - (k - s)}{s}$$

where w is the length of the Fourier sample, k equal to the window size and s the step size of the windowing function. f then becomes all the Fourier samples within each window of size k. p describes the measure used on the data. e.g when creating the mean feature vector we look at the model:

$$v = (mean(f_1), mean(f_2), mean(f_3), \ldots, mean(f_z))$$

With the current window size at 1000/500 and Fourier length at 8000, means that the new feature vectors are of size 15 compared to the 8000 in the raw Fourier experiment(3.9.2). For simplicity reasons, we are going to focus on the average score of the following models. However, they all go through the same process as done in model 3.7.

Table 3.9: Single-measure feature vector model building

Feature	Algotithm	F-measure	Accuracy
Magnitude	SGD	76 (+/- 0.13)	0.90 (+/- 0.06)
	KNN	**0.83 (+/- 0.08)**	**0.95 (+/- 0.02)**
	SVM	0.75 (+/- 0.16)	0.93 (+/- 0.03)
Mean	SGD	**0.91 (+/- 0.09)**	**0.98 (+/- 0.03)**
	KNN	0.90 (+/- 0.13)	0.97 (+/- 0.03)
	SVM	0.86 (+/- 0.21)	0.96 (+/- 0.05)
Standard deviation	SGD	0.83 (+/- 0.17)	0.95 (+/- 0.04)
	KNN	**0.91 (+/- 0.08)**	**0.97 (+/- 0.02)**
	SVM	0.76 (+/- 0.10)	0.94 (+/- 0.02)

Table 3.10: Table 3.9 shows us the mean score of each feature when built on the different models. The highlighted values are the best scoring models for each feature

As expected, magnitude has the score with the lower F-measure. Being based on only a single feature from each window makes it more prone to noise than the other features. On average the best performing feature is the mean, but the variance from standard deviation is so small that it is not enough to conclude it as a single best option. Notice that both SGD and KNN outperforms SVM for all the features, even though SVM should, in theory, be doing quite well. The difference in scores between the Fourier feature extraction models and the raw Fourier in table 3.7 are so small that one could argue that the feature extraction model would be the better of the two.

Table 3.11: Matrix of single-mesure feature vectors

Feature	**Algotithm**	**Confusion matrix**		**Accuracy**
Magnitude	SGD	197	83	0.89
		27	696	
	KNN	214	66	0.83
		101	622	
	SVM	189	91	**0.90**
		9	714	
Mean	SGD	229	51	0.93
		17	706	
	KNN	223	47	**0.93**
		16	707	
	SVM	224	56	0.93
		9	714	
Standard deviation	SGD	236	44	**0.92**
		28	695	
	KNN	213	67	0.88
		41	682	
	SVM	197	83	0.91
		4	719	

Table 3.12: The matrixes in table shows how the models in table 3.9 did when run on the test set. Bold values represent the highest value from each measure.

The second experiment indicated that building a model using all three feature measures; magnitude, mean and standard deviation could result in

an optimal model. Magnitude is the weakest of the three features but will be considered because of the possibility of it helping the model improve when used with other features, as mentioned in section 2.6.4. The results from table 3.11 shows us that choosing one of the ML-algorithms to focus on is still to unclear, with results wavering between them.

3.9.4 Experiment 3 - Combining mesure feature vectors

Based on the results from experiment 2, two feature vectors will be tested in this section. One containg the highest scoring features (mean and standard deviation):

$$v1 = (mean(f_1), \ldots, mean(f_z), std(f_1), \ldots, std(f_z))$$

And the other all of the features.

$$v2 = (mean(f_1), \ldots, mean(f_z), std(f_1), \ldots, std(f_z), magnitude(f_1), \ldots, megnitude(f_z)$$

v1 is here containing 30 features and v2 45.

Table 3.13: Multi-mesure feature vector

Features	Aglorithm	F-measure	Accuracy
Mean, Standard deviation	SGD	0.92 (+/- 0.10)	0.98 (+/- 0.04)
	KNN	0.92 (+/- 0.11)	0.98 (+/- 0.03)
	SVM	**0.93 (+/- 0.12)**	**0.98 (+/- 0.04)**
Magnitude, Mean, Standard deviation	SGD	0.86 (+/- 0.05)	0.93 (+/- 0.04)
	KNN	0.89 (+/- 0.11)	0.97 (+/- 0.03)
	SVM	0.85 (+/- 0.04)	0.96 (+/- 0.01)

Table 3.13 indicates that the best combination of features is the mean and the standard deviation. The models are still performing about equally well.

Table 3.14: Matrix of multi-mesure feature vectors

Features	**Algotithm**	**Confusion matrix**		**Accuracy**
Mean, Standard deviation	SVM	247	33	**0.93**
		21	702	
Magnitude, Mean, Standard deviation	SGD	244	36	**0.92**
		31	692	

Table 3.15: Table is showing the two highest scoring values from running the test set on the models built from table 3.13

The third experiment suggests that the optimal model is built on using the measures mean and standard deviation. There is still no great difference in trying to find the optimal ML for the features.

3.9.5 Experiment 4 - Tuning the model

For simplicity, this section is going to look at the highest scoring model from the previous experiments. From experiment three we found that the feature vector:

$$v = (mean(f_1), \ldots, mean(f_z), std(f_1), \ldots, std(f_z))$$

Aided by an SVM resulted in the best scoring model so far. This test look at different variables that can be set for the SVM model as well as how feature selection tools can change the outcome.

An optimization tool is known as Optunity [48] is going to be implemented and run on SVM with the three different kernels.

Table 3.16: Optimized SVM parameters on different kernerls

Kernal	C	Gamma	F-measure	Accuracy
RBF	72.99	0.52	0.93 (+/- 0.10)	0.98 (+/- 0.03)
Linear	12.05	-	0.93 (+/- 0.12)	0.98 (+/- 0.03)
Poly	86.51	0.95	0.92 (+/- 0.15)	0.98 (+/- 0.04)

Table 3.17: The table is showing the optimized SVMs with the use of optunity on three different kernels, with their repsective C and Gamma values.

Results from table 3.16 shows only minor changes from the results in experiment 33.9.4. Looking at the automatic values4.2.2 from the SVM with a linear kernel used in example 1-3 the difference in C value goes from 1 to 12, meaning that the C parameter set was close to optimal one from the beginning. Mentioned in section 2.6.4 we can also try to optimize the model by apply feature selection to the dataset; one approach is the wrapper function SelectKBest, which scores each feature after its importance and removes all but the n best features. Trying with values $n = [5, 10, 20]$ all models resulted both a lower accuracy from 1-3 percent. implying that the there were no redundant features in the current feature vector.

3.9.6 Experiments discussion

The results achieved in experiment 13.9.2 showed that using the raw Fourier of a windowed model would suffice in our current problem situation. There are however two issues to note: (i) When using the raw Fourier as training data is the build time of the models is high. This is not a big issue with our current dataset, but in the case of further development of the database, this solution would most likely be infeasible. (ii) The test data might resemble the training data too much. At the moment it seems like the classifier would perform well in most cases, but there might be issues with certain everyday sounds that has not been a part of the training or test set. Building a large enough database to properly test the algorithms would be out of the scope of this thesis, but for the sake of further development, we opted to continue the optimization of the ML-model in experiment 2-4, with having a larger data set in mind.

Finding the true optimal learning algorithm for this thesis is very improbable, but experiment 4 3.9.5 Gave us a quick and precise model to work with. There are still many different parameters and issues that could have been addressed, some of which are going to be looked into in the further work section 4.1.1

Part III

Conclusion

Chapter 4

Results

4.1 Conclusion

This study aimed to determine whether or not the hearing impaired could see a use for smart-glass technology in their daily life, investigate the possibility of combining the smart-glasses features and machine learning technology, and how the creation and optimization of a machine learning model was applied respectively.

Positive feedback from students at Briskeby, School for the hard of hearing3.2 indicated that the use of smart glasses could be a product used as an aid. Results from the research turned our focus to the creation of a software system with the goal of using the glassware as a platform for:

- Siren recognition
- Speech recognition
- Name recognition

To realize this model, we required a software lifecycle that was able to: (i) create a glass application that could handle input and output from the user, (ii) implement siren, speech and name recognition. (iii) Set up a server that could bind (i) and (ii) together. (iv) create a database of audio samples for the purpose of building and testing the recognition software model. Section 3.4 went through how the simple setup of a glass application would suffice for recording and visualizing data for the user, and how different features were applied to better the user experience. There are still features and tweaks that could be implemented to better the application, some of which

we are going to have a closer look at in the further work section4.1.1.

The central development focus in this thesis besides the glass application was the creation of a siren recognition model. The experiments completed in section 3.9 showed us that the data recorded from the Google glasses would suffice in recognition of a siren. The best models effectiveness managed an F-measure classification of about 93 percent with a 98 percent accuracy. It is possible that the amount and type of audio data used to build this model are lacking when it comes to showing a good generalization of the model. However, with limited time creating a sufficient database would not be possible.

The creation of a speech recognition service was way out of the scope of this thesis. But section 3.5 showed how the use of third party software enabled the integration of speech recognition into our platform, and how the different models work.

For name recognition, we initially looked at a third party software known as Wit. Wit is an open platform that is intended for the training and recognition of single words or sentences. One of the problems encountered was user based feature training, meaning that building the model would require the user to get people to say their name multiple times. We decided to drop this part of the project and instead rely on the response from the ASR service since their name recognition works better by default. There are some improvement options to this solution in section4.1.1.

4.1.1 Future work

For this work to become a viable option for aiding the hearing impaired, future research is needed. This section looks at some of the steps that could have been taken to improve the quality of the service.

Google glass One of the issues faced in this thesis was the amount of GSM/WIFI data it would take to run the application. This field has much room for improvement, where some of the features looked into was: (i) Saving the siren/speech recognition models locally, (ii) Giving the user options to decide the sampling frequency. Under development, there was no official way to export the ML-models to work with Java. Different tests had been done with manually editing the files to fit Java the format, but

we decided not to follow it any further. Getting a local speech recognition model would be hard unless the Google ASR system changed into letting the developer control more of the data flow. The decision to focus on a 16000Hz Sample rate was made, mainly because of the GSM usage a higher option would require. Letting the user, e.g., decide a higher rate would in cases significantly improve the ASR results and would incorporate flexibility to the application when on WIFI contra GSM.

Name recognition Mentioned in section 4.1 ASR became the soultion to the name recognition problem. A simple implementation thought of further developing this feature was to let the user type in their name and do string search over the ASR data for matches, then notify the user if a match was found.

Siren recognition

- Database

Further developing the database would increase the generalization of the classifier. Being able to test the data on more samples could reveal issues with the models or features that at the moment are unknown.

- Features

This project has not investigated the use of continuous features over the Fourier or audio signal. Meaning that sounds that resemble one second of a siren would most likely be classified as one. Adding this feature would probably not increase the accuracy of the models by a lot, but would improve the overall usability of the final work.

- Machine learning

The models were built on what was thought to be a good windowing size of audio and Fourier data, but the optimization of these values should still be assessed. Ensemble learners have not been tested on the dataset. Ensemble learners work by combing classifiers and predict based on what the majority of the classifiers predicted.

- Speech recognition

Many ASR services were not tested in this work, trying other options as well as the optimization of the services should also be a point of interest

Appendix

4.2 Tools

4.2.1 Android Studio

Android studio[49] is the official platform for development of Android applications. It can be used on platfroms as Ubuntu, Windows and MAC. The glassware is built on Googles glass development kit(GDK), which is a stand-alone development kit built on top of the android studio framework.

4.2.2 Scikit learn

Scikit learn is a free software machine learning library for the python programming language.

SVM default values are set as:
`C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None.`
When testing the models in experiment 1-3 a linear kernal was chosen because of times issues related to the creating of the model in experiment 1. The probabillity was set to true, enabeling a possabillity to get a performance mesure from the model. More information about the model can be found at their page[50]

KNN default values are set as:
`n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=1, **kwargs`

No changes were made in any of the experiments. More information about the model can be found at their page[51]

SGD default values are set as:
`loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, n_iter=5, shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight=None, warm_start=False, average=False`
More information about the model can be found at their page [52]

Bibliography

[1] O. J. M. M. L. C. Z. M. B. J. Kübler, 'Google glass in pediatric surgery: An exploratory study', 2014.

[2] L. Engelen, 'Is google glass useful in the operating room?', 2013. [Online]. Available: https://www.linkedin.com/pulse/20130815203138-19886490-google-glas-in-or?trk=mp-author-card.

[3] C. Plack, *The sense of hearing*. Lawrence Erlbaum Associates Publishers, 2005.

[4] sound proofing. (21 June 2016). What is sound?, [Online]. Available: http://www.soundproofingcompany.com/soundproofing101/what-is-sound/.

[5] S. Rosen, *Signals and Systems for Speech and Hearing (2nd ed.)* BRILL, 2011.

[6] (2016). Ear anatomy, [Online]. Available: http://gothing.info/ear-anatomy/.

[7] M. S. S. M. Hussain, 'Synopsis of causation - conductive hearing loss', 2008. [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/384492/conductive_hearing_loss.pdf.

[8] B. E. Schreiber, C. Agrup, D. O. Haskard and L. M. Luxon, 'Sudden sensorineural hearing loss', *The Lancet*, vol. 375, no. 9721, pp. 1203–1211, 2010, ISSN: 0140-6736. DOI: https://doi.org/10.1016/S0140-6736(09)62071-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140673609620717.

[9] K. Lee. (March 2012). Augmented reality in education and training, [Online]. Available: http://www2.potsdam.edu/betrusak/566/Augmented%20Reality%20in%20Education.pdf.

[10] I. E. Sutherland, 'The ultimate display', 1965. [Online]. Available: http://worrydream.com/refs/Sutherland%20-%20The%20Ultimate%20Display.pdf.

[11] Google. (April 04, 2016). Google glass faq, [Online]. Available: https://sites.google.com/site/glasscomms/faqs.

[12] M. Missfeldt. (2013). Google glass (infographic) - how it works, [Online]. Available: https://www.brillen-sehhilfen.de/en/googleglass/.

[13] W. L. Hosch. (May 12 2016). Machine learning, [Online]. Available: http://global.britannica.com/technology/machine-learning.

[14] E. K. Steven Bird and E. Loper., *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[15] Z. Weinberg. (26 November 2012). Svm separating hyperplanes, [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svm_separating_hyperplanes_(SVG).svg.

[16] ——, (16 February 2008). Svm max sep hyperplane with margin, [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svm_max_sep_hyperplane_with_margin.png.

[17] (2011), [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Kernel_Machine.svg.

[18] J. Mercer, 'Functions of positive and negative type, and their connection with the theory of integral equations', *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 209, no. 441-458, pp. 415–446, 1909, ISSN: 0264-3952. DOI: 10.1098/rsta.1909.0016. eprint: http://rsta.royalsocietypublishing.org/content/209/441-458/415.full.pdf. [Online]. Available: http://rsta.royalsocietypublishing.org/content/209/441-458/415.

[19] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, ser. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, Dec. 2002, p. 644.

[20] C. Cortes and V. Vapnik, 'Support-vector networks', *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995, ISSN: 0885-6125. DOI: 10.1023/A:1022627411411. [Online]. Available: http://dx.doi.org/10.1023/A:1022627411411.

[21] J. M. Girard. (2016). What are c and gamma with regards to a support vector machine?, [Online]. Available: https://www.quora.com/What-are-C-and-gamma-with-regards-to-a-support-vector-machine.

[22] M.-l. Zhang and Z.-h. Zhou, 'Ml-knn: A lazy learning approach to multi-label learning', *PATTERN RECOGNITION*, vol. 40, p. 2007, 2007.

[23] Agor153. (30 January 2013). Data classes, [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:Data3classes.png.

[24] ——, (30 January 2013). Map 5nn, [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:Map5NN.png.

[25] J. Chen, H.-r. Fang and Y. Saad, 'Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection', *J. Mach. Learn. Res.*, vol. 10, pp. 1989–2012, Dec. 2009, ISSN: 1532-4435. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755852.

[26] J. L. Bentley, 'Multidimensional binary search trees used for associative searching', *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975, ISSN: 0001-0782. DOI: 10.1145/361002.361007. [Online]. Available: http://doi.acm.org/10.1145/361002.361007.

[27] S. M. Omohundro, 'Five balltree construction algorithms', 1989.

[28] R. D. E. H. G. E. and R. J. Williams. (1896). Learning internal representations by error propagation.

[29] L. Bottou. (2010). Large-scale machine learning with stochastic gradient descent.

[30] (8 March 2017). Introduction to gradient descent algorithm (along with variants) in machine learning.

[31] wikipedia. (2016), [Online]. Available: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#/media/File:K-fold_cross_validation_EN.jpg.

[32] R. Kohavi, 'A study of cross-validation and bootstrap for accuracy estimation and model selection', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95, Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995,

pp. 1137–1143, ISBN: 1-55860-363-8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1643031.1643047.

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

[34] R. P.-W. A. S. C. H. I. R. H. C. A. F. W. J. F. W. F. A. P. N. C. S. Haley, 'Application of high-dimensional feature selection: Evaluation for genomic prediction in man', 2015.

[35] I. Guyon and A. Elisseeff, 'An introduction to variable and feature selection', *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Mar. 2003, ISSN: 1532-4435. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944968.

[36] S. Ioffe and C. Szegedy, 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167.

[37] J. O. Smith, *Mathematics of the Discrete Fourier Transform (DFT)*. http://www.w3k.org/books/: W3K Publishing, 2007, ISBN: 978-0-9745607-4-8.

[38] (2013), [Online]. Available: https://en.wikipedia.org/wiki/Fourier_transform#/media/File:Fourier_transform_time_and_frequency_domains_(small).gif.

[39] A. J. Jerri, 'The shannon sampling theorem; its various extensions and applications: A tutorial review', *Proceedings of the IEEE*, vol. 65, no. 11, pp. 1565–1596, Nov. 1977, ISSN: 0018-9219. DOI: 10.1109/PROC.1977.10771.

[40] L. Deng and X. Li, 'Machine learning paradigms for speech recognition: An overview.', *IEEE Trans. Audio, Speech Language Processing*, vol. 21, no. 5, pp. 1060–1089, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/taslp/taslp21.html#DengL13.

[41] L. Huffman. (February 17, 2014.). Leopold stokowski, harvey fletcher and bell laboratories, [Online]. Available: http://www.stokowski.org/Harvey%20Fletcher%20Bell%20Labs%20Recordings.htm.

[42] J. Schalkwyk. (2011), [Online]. Available: https://research.googleblog.com/2011/07/google-north-american-faculty-summit.html.

[43] R. R., 'Experimenter effects in behavioral research.', 1966.

[44] D. G and M. B, 'Siren actuated warning device for automobiles', Dec. 1961, US Patent 3,014,199. [Online]. Available: https://www.google.com/patents/US3014199.

[45] T. N. I. on Deafness and O. C. Disorders. (October 2010). How loud is too loud.

[46] (). Talegjenkjenningsprodukter, [Online]. Available: http://www.maxmanus.no/Produkter/Talegjenkjenningsprodukter.

[47] IMB. (). Bluemix, [Online]. Available: https://www.ibm.com/cloud-computing/bluemix/.

[48] (2017). version 1.1.0, [Online]. Available: http://optunity.readthedocs.io/en/latest/.

[49] (). Android studio, [Online]. Available: https://developer.android.com/studio/index.html.

[50] (2017), [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[51] (2017), [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors.

[52] (2017), [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.