VOIP Spam Counter Measures

Michael Pantridge

Kongens Lyngby 2006
IMM-MSC-2006-85

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-MSC: ISSN 0909-3192

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the MSc degree in Computer Systems Engineering.

The thesis deals with different approaches in tackling the future problem of redundant, SPAM-like internet telephony calls.

The thesis was written during the period February - August 2006. Lyngby, August 2006

Michael Pantridge

Acknowledgments

I thank my supervisor Dr. Robin Sharp for his continual encouragement and provision of direction and critical thought.

Additional tribute is paid to the many enthusiastic participants during the testing procedure, who so willingly contributed samples of their own voices.

Michael Pantridge August 2006

Contents

Chapter 1

Introduction

1.1 Problem Definition

The modern phenomenon of spam mail is widespread throughout the internet. Most email systems have inbuilt mechanisms to detect and handle such mail automatically before the culprit mails are even presented to the user.
Spam is not limited to just email. It is certainly to be found in the standard telephone networks in the form of automated advertisements and unsolicited sales offers. As broadband networks develope and voice conversation becomes more prevalent over the internet through the medium of Voice-Over-IP (VOIP), the possibility of generating and delivering voice spam messages increases rapidly. Undoubted consequences include frustration and inconveniencing of the user and cluttered bandwidth. More sophisticated techniques will need to be used in comparison to the existing email spam counter mechanisms. The message medium changes from plain text to audio samples.
In addition, vocal correspondence demands real-time operation in order to be effective. This places strong limitations on what operations can actually be performed on incoming voice packets with regards to the speed and complexity of the analysis.
The aim of this project will be to investigate suitable counter-measures which may be utilized to detect voice spam and deal appropriately with it before the user is inconvenienced. Within the context of VOIP, appropriate voice recogni-

tion technology and pattern matching techniques will be examined, resulting in the construction of a prototype voice spam filter.

1.2 Document Organisation

Many common existing spam detection methods are to be found in the email model. Therefore, an analysis of what form they take was performed. With this established it was considered which techniques could be carried over into the realm of voice audio as a means of content filtering. Foundational to the project will be an understanding of typical VOIP structures so that the contextual position of any prototype voice spam filter may be isolated. Chapter 2 will detail both of these preliminary steps.
By the conclusion of Chapter 2 it should be possible to see how the proposed solution could be generalised. The solution was basically divided in two parts - Voice Recognition Feature Extraction and Pattern Matching Classification. Chapters 3 and 4 respectively discuss these topics. Chapter 5 provides an overview on how the chosen methods in Chapters 3 and 4 could be implemented in practice. Chapter 6 describes how the resulting implementation was tested and concluding remarks are drawn together in Chapter 7.

Chapter 2

Analysis

2.1 Existing Approach - Email Case Study

2.1.1 What is SPAM?

For the basis of this project one categorises everything which finds its way into one's email inbox which one does not want and is only very loosely directed at the recipient as SPAM. This includes not just the latest mortgage offers from such-and-such a company, but also viruses (sometimes from acquaintances), chain letters and monetarily or religiously persuasive correspondences which are unwanted. [28, 11]
It is a pest, is generally recognisable and ultimately turns the everyday process of email checking into an often troublesome and painstaking experience of sifting through SPAM emails to separate them from legitimate correspondences. In 2004 it was estimated that 60 % of all email messages sent were SPAM.[41]

How does one maintain an open channel of communication with a variety of sources in the outside world, whilst at the same time protecting the inbox from this flood of unsolicited, unwanted trash?

Before introducing strategies which continue to be used to combat SPAM emails,

it is necessary to establish a couple of integral concepts and terms.

- **HAM** - The opposite of SPAM, namely the good emails which are legitimate and come from trusted sources with genuine purposes. [42]

- **False Positives** - The proportion of HAM (ie, the valid, trusted emails) which are wrongly discarded or identified as SPAM by any SPAM protection technique. In many ways, false positives pose a greater problem than SPAM mail which gets through any defence put in place - it results in a loss of information.

- **False Negatives** -The SPAM mails which somehow breach any counter measures employed. Their consequences can range from being harmless to reasonably serious, should virii exist within. Generally, though False Positives are of greater concern than False Negatives. [11]

  "a filter that yields false positives is like an acne cure that carries a risk of death to the patient" [11].

2.1.2 Counter Measures - Preliminary Steps

A common sense approach is to make sure that the email address is not publicly on display for all to see.[28]
If the user has to transmit their email address to another party then some kind of conventional encoding of the address could provide some limited obscurity.

ie. **johnsmith@NOSPAMhotmail.com**

or **"johnsmith at hotmail dot com"**

The user could also use several "throw-away" addresses. Transactions could occur through these for a temporary time period and then the address could be disposed of, taking accumulated SPAM with it.
This may be suitable for people whose online activities are confined to online purchases, Message boards, Usenet and mailing lists, but for those who actively promote a business or enterprise where their email address has to be displayed as a formal communication portal, it is most impractical.

2.1.3 Basic Text Filtering

It is possible to detect a large proportion of incoming SPAM by using a simple string matching technique to scan the header fields or the body [28, 9].This is based on the premise that the appearance of certain character stings is common to certain SPAM messages. Unfortunately this can lead to a high false positive rate - the variety of human communication and expression across a number of communication scenarios makes this inevitable.
There are however a few basic steps one can take where this kind of filtering can be used:

1. Identify and accept peculiar headers which are known to be from legitimate sources though may otherwise be rejected by other rules.
2. Identify and reject known bad senders
    - This may be known email addresses from whom correspondence is not wanted
    - "<>" in the From: field. (ie empty)
    - "@<" in the header (typical of a lot of SPAM)
    - Certain content types which could contain virii and one would wish to avoid regardless.
    - Certain strings associated with certain countries (A lot of SPAM comes from certain areas/countries in the world - Korea, China).
3. Accept messages where the contents of the *From* field are known (ie perhaps cross referenced with a contact list or list of previous correspondences).

2.1.4 Blacklisting

These are semi-static lists of known SPAM hosts which are maintained on a local host. The sender information from the incoming mail is checked against the list, and if a match is found, the message is labelled as SPAM.

2.1.5 Real-Time Black Holes (RBLs)

These work in much the same way as Blacklists however, RBLs are dynamic lists which contain more up to date information on SPAM hosts. They there-

fore must be downloaded from a trusted third party before being able to be effectively used [42, 9].
Black lists and RBLs allow for SPAM emails to be blocked before they even enter the infrastructure of a company. An advantage is that it could be configured for the SPAM host to receive a reject message, leading to the possible removal of the recipient's address from their list of targets. Extra processing time could be required however for connection to RBL databases.
RBLs are effectively a kind of distributive and adaptive Black List. A further way they can be extended is every time someone deletes a SPAM message, notification can be sent to a distributed authority which would maintain a digest of known SPAMs. Minor or automated mutations of what is essentially the same SPAM message can be easily detected also. Razor (http://razor.sourceforge.net) and Pyzor (http://pyzor.sourceforge.net) are two such services currently in operation.
Should someone report a valid message to the digests in this way, it would only be effective for that person since legitimate mail characteristics vary from person to person. Because the distributed blacklist servers are maintained and managed regularly, misreported legitimate messages could be easily detected and corrected.
False positives are therefore unlikely with this technique, though false negatives are still likely to persist suggesting that further techniques should be used in conjunction with this one.

2.1.6 White Lists

Essentially the inverse of a Black-list, a white-list is a list against which incoming mail can be checked and if the sender credentials are found on the list then that sender will always be respected as legitimate. Mail will always be accepted from senders on the whitelist regardless of whether they might also appear on a black list or RBL.

2.1.7 Automated Verification

A more aggressive addition to just using a whitelist would be to only accept mail messages where the sender is on the whitelist - and nothing else. Otherwise the whitelist filter would send a challenge to the sender, with instructions on how the sender can be added to the whitelist. This challenge would also contain a Hash ID of the original mail and the response back from the sender should also contain this ID for the sender to be added to the whitelist. When a challenge is successfully answered, the sender is added to the whitelist and any future

messages are automatically allowed through [28].

Since nearly all SPAM messages contain a forged return address, the challenge will not arrive anywhere. If the SPAMMER does provide a usable return address, he is unlikely to respond anyway for several reasons:

1. should the SPAMMER respond, he would make himself more easily traceable.

2. SPAMMERs are using automated delivery systems which do not cope with incoming mail - they exist solely to produce outgoing spam mail [11].

This method is likely to be quite effective against SPAM, however it has the following drawbacks:

- Places an extra burden on legitimate senders. They may not respond to a challenge for a variety of reasons and in doing so, effectively raise the false positive rate - this could be down to an over-draconian firewall, unreliable ISP, or apathy or ignorance on the part of the original sender.

- This simply would not work for non-human correspondents (ie valid automated response systems from online sales, online registrations etc.)

- Ultimately, where timeliness of response and fluidity of correspondence matters, an unexpected hurdle is placed at the onset of a new dialogue.

2.1.8 Sender Policy Framework (SPF)

This is a different mode of validating whether a sender really is who they say they are, by querying the SPF record of the domain which is listed in the From: field (Most SPAMMERS use forged addresses) [9].
It is a community effort which is rapidly gaining support, and it requires that each company which supplies email addresses supply an SPF record. The SPF record may be queried to ascertain if the sender's address and the machine used are credible values according to the email company.

2.1.9 Header Filtering

Inconsistencies or errors in the header information of the email may be searched. Normal email clients by default send complete header information with each mail, but it is not difficult for a SPAMmer to alter the header information which might in some way disguise where the mail has been sent from, or the real identity of the mischievous sender.
A common tactic employed is to make the FROM: and TO: field identical so that it appears that the recipient has received the SPAM message from his/herself.
Also contained within the email header is the Route through the internet the message took to arrive at the user agent of the recipient. Suspicious route information would therefore sometimes be included should a SPAMmer wish to conceal their identity by masquerading as taking a certain, false path.
The email distribution agent is also included in the header information which is sent. It gives some idea as to the origin of the email by highlighting its construction method. SPAMMERs using an automated approach of sending out thousands of emails to generated addressed are probably not using a standard email client such as Outlook or Mozilla Thunderbird to do so.
Shouting text (where the subject line may be in CAPITALS) is a further characteristic of SPAM mail. The SPAMMER ultimately wants to catch the attention of the person who has been lumbered with their unsolicited offers.

Further anomalies which could be detected directly from the header alone, are:

- An empty or malformed MIME from: field
- A large recipient list included. (one could use a threshold value to determine the email as SPAM should there be more than X recipients). These tend to be joke or chain mails.
- Messages containing links to remote images - Spammers are using image only mails to circumvent text filters.
- Invalid domain listed. A simple DNS lookup would easily detect an invalid domain.
- The first part of the mail address listed in the subject line - SPAMMERs do this to "personalise" the SPAM mail.

All of the above give initial pointers as to the nature of the received email without even looking at the content of the email. In fact, an option could be for the user to just download the header information from an email without the content, and still have good indicators as to its credibility.

2.1.10 Content filtering - Bayesian Filtering

Bayesian filtering is an artificial intelligence technique which can be used to perform filtering on emails [11, 12, 28, 41]. It can be used to filter both normal and abnormal patterns of actual mail the individual receives.
The method works on the premise that an event is more likely to occur in the future if it has occurred in the past, and subsequent re-occurrence increases this likelihood. One could extend this premise to the arena of filtering emails for familiar textual patterns. It would be possible to use a database of known patterns which are typically found in spam messages and normal messages (known as "Ham"), and assigning each word or token a probability value based on how often that word appears in Normal or spam messages. To obtain these probability values and the actual tokens, a typical system must be trained to read and analyse the user's mail for some period of time. In a way the discernment of the system becomes tailored for the actual user themselves.
The words which have been tokenised and extracted from the two corpus-es will each be assigned a probability based around how their presence in a message is likely to contribute to the message being actual SPAM or HAM. These probabilities may also be weighted according to the profile of the person concerned.
An overall probability rating is ultimately assigned to the message after all the words have been analysed. Words which are not normally associated with SPAM actually contribute to the validity of the mail as much as more notorious occurrences (such as "sex" or "viagra") contribute to a SPAM designation. All the evidence is considered together so that isolated occurrances of problem words do not adversely determine the outcome.
The two corpus-es can be kept up to date by moving the most recent messages into them (a "delete-as-spam" button would be one option which would remove SPAM from the inbox, but hold it in the SPAM database for future analysis of incoming mail). This enables the filter itself to evolve as SPAM evolves.

In addition to this self-balancing attribute, a further advantage is that it is self-adapting. It is capable of learning from new SPAM and new valid outgoing emails. For example, substitution of letters in words would be easily detected - "5ex" instead of "Sex" is commonly found in SPAM but there is very little likelihood that a Bayesian filter would ignore this.
The system is also language independent and international - because it is self-adaptive it can be used to any language however it would require keyword lists to be set up in the language concerned.

2.1.10.1 Reservations

This style of Bayesian filtering is used by both Microsoft Outlook's Spam filter and the internet message filter in Exchange Server. They do not require initial learning but they do rely on a publicly available Ham file which could possibly be hacked and bypassed. Because it is not tailored to the user the chances of getting more false positives increases.
The success of such a Bayesian filter is ultimately reliant upon the SPAM file being kept up to date. A recent BBC article outlines the success of Bayesian filtering, with a 99.7 percent success rate and low false positive rate [40]
But tests have shown that despite a high success rate, infiltrating spam emails may be used to fool the filter in the future. Continued false acceptances over time may subtly and gradually convince a filter that the reappearance of the same random words is to be permitted. Therefore, the filter learns to accept a sub-series of emails which it has accepted as valid in the past but which are actually not [41].

A more recent algorithm aside from Bayesian filtering is the use of the Chung-Kwei algorithm to detect patterns of letters in messages which would suggest the message is SPAM. (The Bayesian method uses patterns of words with each word being a kind of token). This uses pattern recognition in the same way as is used with DNA analysis and identification. Similar to Bayesian Filtering, the system would required training from a known set of SPAM and HAM patterns, though tests show that it would only take up to 15 minutes to train a system using 22000 known SPAM messages and 66000 HAM messages. The rate of successful identification has been found to be around 97% [38].

With the Bayesian case, contextual training is ultimately crucial to the success of the system, and the recommended 2 week setup period is a necessary if irksome procedure.

2.1.11 Content Filtering - Heuristic Rule-based

Where the Bayesian method trains a system with SPAM "signatures" an heuristic rule approach analyses the content and context of the message qualitatively. The artificial intelligence used is if you like, more "artificial". It assumes that there are consistent standardised traits employed within SPAM which can be detected [28]. Regular expressions are used and category specific attributes allow the user to tune their preferences to differentiate between SPAM and accepted

communications.

Where the heuristic techniques excel is at identifying traits of SPAM which have been previously unreferenced (in other words previously unseen and not stored in any database - a signature of unknown validity). As SPAMMERs learn new techniques to overcome identification through signature analysis, heuristic techniques may provide an effective second line of defence.

The weakness with Heuristic rules is that the methodology behind the rules is entirely subjective and the rules must be tuned according to the user's preferences. If not properly configured false positive rates could increase because the system is not comparing fingerprint against fingerprint if you like - there is never an explicit pre-existing reference. As SPAM itself evolves, so too must these rules for detecting it - Viagra may be a popular theme now, but it may change to something else in a few years.

In general though, the Heuristic Rules should check for both legitimate traits and illegitimate traits in the message. Some of the latter could be:

- Message header follows non-standard formats and hop patterns.
- Excessive use of punctuation, including multiple interjections or exclamations.
- Unsubscribe footer follows conventional formats to appear legitimate.
- Text syntax consistencies between paragraphs appear normal.
- Self-executing Javascript contained within.

2.2 Voice Spam Vs Email Spam

2.2.1 Introduction

There is no reason to suggest why the White-list and Black-list ideas used with normal email could not be extented into the VOIP realm - communication is from one point to another and both parties should, to a certain extent be presented with some information pertaining to each other prior to the start of the vocal communication. Towards the end of the previous section the methods employed as content-based filtering were discussed - continually up-to-date Heuristic Rules and Bayesian Filtering, tailored to the individual concerned

and performed on the textual email body. The question is now considered, what could be learned from these techniques in relation to a message body of a different kind - namely speech data? Three issues will be addressed:

1) How does Voice SPAM compare in nature with Text SPAM and how well could the above methods be adapted?
2) What are the practical considerations of detection which would need to be taken into account in order to make any Voice SPAM content filter usable?
3) What other inherent characteristics of a voice message, unrelated to textual analysis, can be monitored and therefore used to predict the credibility of a message?

2.2.2 Statistical analysis

In many cases a collection of letters can be pronounced in some way, even if that collection of letters is not recognised as a given word in a particular language. However, some collection of letters are extremely difficult to pronounce, for example "xgrdyb". Furthermore, a certain collection of letters may be pronounced in a different way, depending on

- intonation,
- accent,
- speed of speech,
- syllable emphasis.

This outlines one of the problems of dealing in spoken word terms as opposed to textual terms. Text is much more universal in that it does not undergo the same degrees of variation. A word typed by one person will always be the same as the same word typed by someone else as long as the spelling is correct. By studying a word, one can objectively tell if it is spelled correctly. It is also unambiguous as to where one word starts and another finishes - whitespace takes care of this. Words are easily compared and easily parsed and tokenised.
With the Bayesian example described earlier, advantage is taken of this attribute through the ability to assign re-occurring tokens a qualitative value. This occurs after the trivial process of detecting and deciphering what the "word" is - even if it is merely a group of letters. Words or groups of letters are unambiguous, discrete, and easily compared.
Unfortunately with speech, words and sentences are not transferred as unambiguous symbolic information, rather as compressed audio. After decompressing

the audio it is possible to obtain a waveform representation of a sequence of speech. However, it is not always clear where one word starts and another ends. There is also no guarantee that a 100% reliable 1-1 mapping from waveform onto word entity could be made. There is a high degree of subjectivity, which a trained human brain can accommodate but with which a computer has major difficulties. Word analysis by a computer would require the following steps:

1. decompression of speech into waveform representation
2. tokenisation of waveform into separated words
3. mapping of spoken word onto known word entity
4. statistical manipulation of collected word entities & comparison against known SPAM.

[45] Steps 1 to 3 are not present in the textual model, but yet in the vocal model they are essential and present the greatest hurdles.
Step 4 would be as straight-forward as the textual model if it were possible to maintain large repositories of SPAM and HAM voice messages, or rather some kind of discrete representation of what known SPAM and HAM are which can be referenced against incoming, "interpreted" waveforms. This assumes that SPAM voice messages are something which is a well established, everyday occurrence and can be consequently learnt over time by a machine. Herein lies a further problem. Whilst the existence of such SPAM is not in question, its propagation is perhaps less evident at this moment in time. Consequent problems are therefore presented in relations to the acquisition of SPAM and HAM examples.

2.3 Technology Background - VOIP

Continuing the discussion of content based filtering, it would be useful to know from where within existing VOIP models, content itself can be captured. Before any content analysis can happen, the speech of the calling party has to be collected from somewhere. Without performing any great technical assessment on the types of VOIP available, some background on the sort of system likely to be used will be needed to determine where and how any filtering of VOIP traffic can occur. This section will concentrate mainly on the communication aspects of various kinds of VOIP network, and at the same time suggest possible junctures at which any kind of filtering could happen. Because the project is centred around what solutions are available for the management of incoming

SPAM, the area of interest extends only to the boundaries of a network which has allowed VOIP protocols to be passed through its perimeter firewalls, and once inside, how and where the data flow can be checked.

An in-depth coverage of the entire scenario in relation to the two main VOIP technologies (SIP and H.323) is provided in **??**.

2.3.1 Isolating the point of interest

After consultation with Appendix A it should be clear that any kind of Voice SPAM filtering system in either VOIP architecture would at some stage receive RTP packets from the calling party. It is from these packets, and after subsequent reconstruction and decoding that the proposed system would be presented with a wave of an incoming speech pattern.[16]

It was decided then that this should become the entrance point for the project - a proposed system should be presented voice information in the form of a waveform and then subsequent content-based filtering would be implemented (**figure 2.1**).

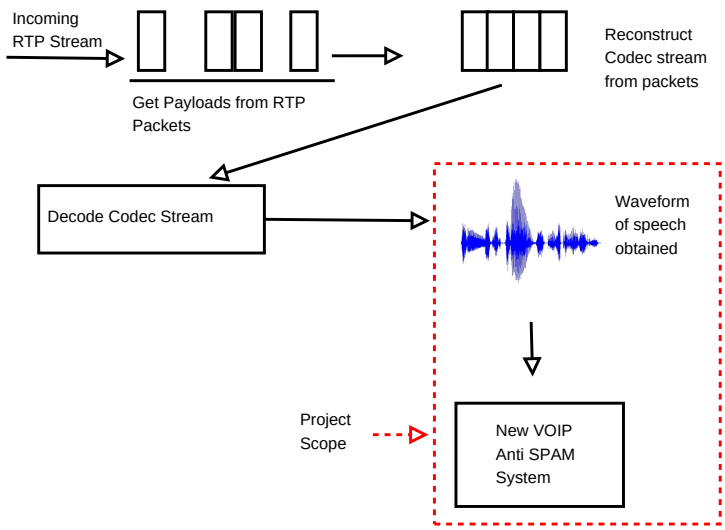

Figure 2.1: General scope of the project

2.4 Shaping the solution

Any possible overall solution must take into consideration the context of a VOIP conversation, at least in comparison with a standard telephone conversation. A temporal context of the problem is shown in **figure 2.2**, the red arrow indicating at what stage any sort of content filtering would have to be performed.

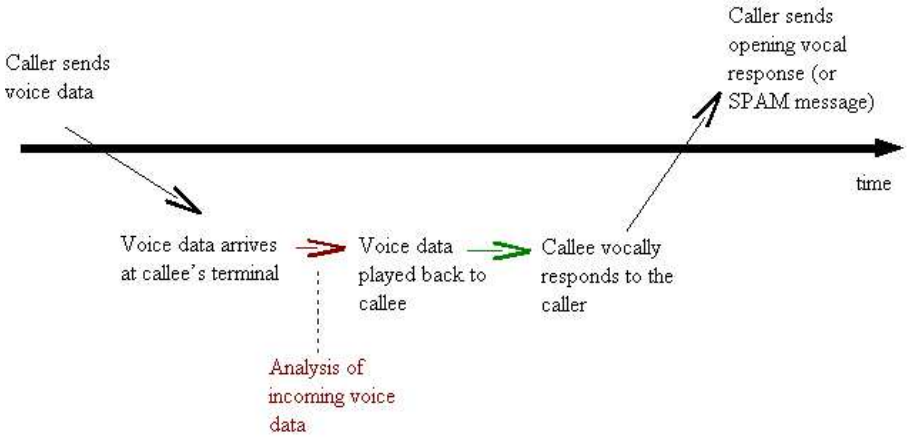

Figure 2.2: Temporal Context of the problem

2.4.1 Conversational Conventions

One of the differences between a normal phone call conversation and that of a VOIP conversation is that with the former, convention dictates the conversation is started by the callee. It is almost as if the presence of the callee on the other end of the line, indicated by the callee's initial greeting (ie "hello, this is xxx") is a final acknowledgement of the callee's acceptance of the incoming call.
Sometimes if one called an answering machine, a voice would appear at the other end of the line, but the semantics of the response are different, since the answering machine is telling the caller that their call can not be taken right now. In a way, this is an acknowledgement of the call getting through, but further analysis of this "acknowledgement" would show that it is hardly akin to a kind of "200 OK" response. In this sense, it is more like a 3xx response - "yes, you have reached the correct person, but he's not here".
Often, when one encounters a typical SPAM message over the standard telephone lines, it consists of merely a recorded message, which either begins im-

mediately with the caller picking up the phone, or there is a short pause before the useless message begins. In any event, a sane person can usually tell very quickly whether there is any cognisance at the other end of the line. It is with this in mind that one may be tempted to reduce the problem to a determination of intelligence on the part of the caller. However whilst it may be true that most SPAM messages currently in operation presently are "dumb" and are solely recorded, SPAMMER technology could improve in the future to incorporate more sophisticated methods. One is hesitant to consider a solution which merely monitors for continuous, uninterrupted "babble" whilst does little to actively identify the caller, which is much more useful.
The setting of a context therefore would be a most desirable scenario both for collecting a meaningful and sufficient sample of the caller's voice and also to establish cognisance from the calling party.

2.4.2 The Secretary Model

Faced with the burden of voice sample collecting in real time, the proposed solution is based around providing an automated context for voice sample procurement, namely a kind of secretary interface which could function independently of the callee and could query the caller appropriately whilst performing an analysis any caller voice samples which have been possible to collect. Largely akin to the role of a secretary in an office, the role here of this additional mechanism is to provide a further authentication stage, via the content channel (**figure 2.3**). When the RTP Session is set up, instead of being directly involved in the conversation, the callee would pass control of events to a Secretary interface which would begin by issuing some kind of automated challenge to the caller (**figure 2.4**). This challenge could be defined in content by the callee and could be in the form of:

- A simple question with an elementary answer.
- A request to repeat a one-time password challenge.
- A request to the caller to quote their name or the current date.

2.4.2.1 The Challenge

There are a number of options available here for the parameters governing the challenge message itself.

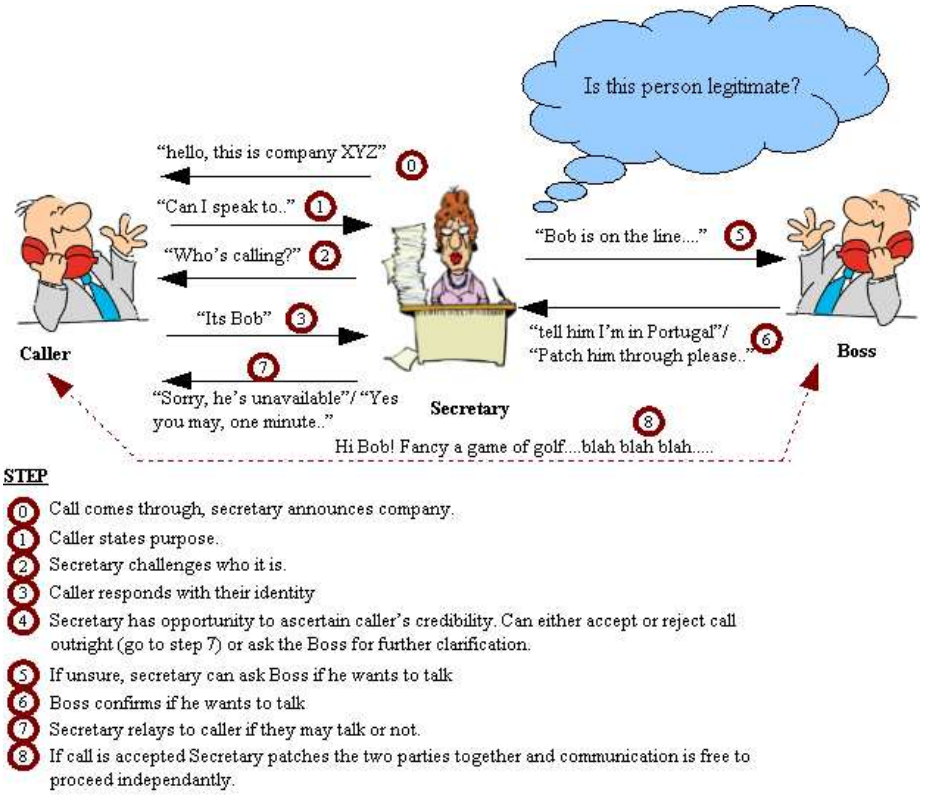

Figure 2.3: Standard Secretary Operation

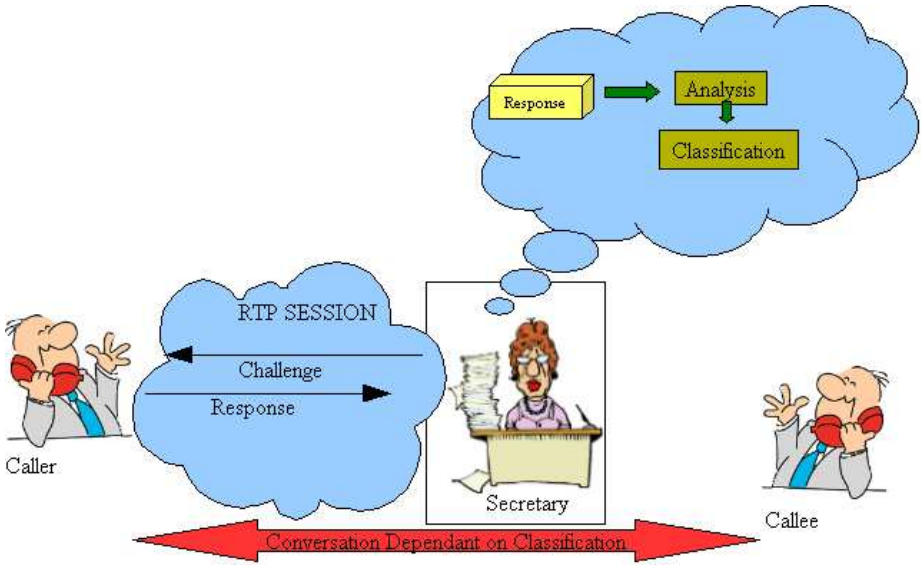

Figure 2.4: Secretary Challenge/Response

Does one produce the recorded challenge message oneself?
Does one produce a number of different challenges over time?
If there are a number of different challenges, how are they alternated?
Do they change monthly/weekly/daily?
Are they selected at random each time they are activated?
What should the challenge consist of?
How much information should it require the caller to give? (this will ultimately determine the amount of data one will have to perform analysis on)
How long should the challenge be (in seconds)?
How much information should the challenge message give about the callee?

2.4.2.2 The Response

Since the system aims to establish a meaningful context between the callee and a credible caller, any response received during the Secretary phase should be monitored for both its timing and for its actual content. Also, how much time should be allowed for the caller to give a response - should there be a time-out phase for this? The amount of time given for this should be balanced by what is the expected amount of data to be returned and what is a reasonable amount of time to be allowed for such a response.

2.4.2.3 Timing of the Response

Much then depends on any SPAM message trying to get through and its response to the challenge. Is there a built in delay or does the message begin immediately? The actual timing of the packets coming in from the calling party becomes an issue. It could then be possible to detect typical SPAM messages by scanning packets sent with timestamps indicating creation before the Secretary has finished issuing the challenge. In effect the Secretary would be challenging the caller in some way and the caller SPAM message would interrupt the secretary. In the event of a "dumb" SPAM message, the interruption would also be a fairly prolonged affair, the message is just sent with no regard for any kind of interaction.
But perhaps the SPAM message could use some kind of delaying tactic so that the message would play after the secretary has greeted/challenged on the other end of the connection. In response then, the Secretary challenge could be made long enough to render the delay useless. With reference to **figure 2.5**, responses

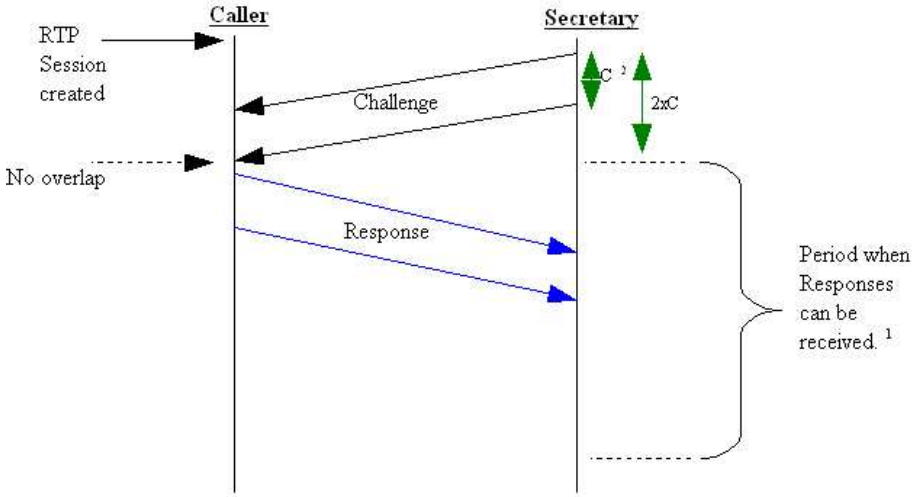

Figure 2.5: Ideal Challenge/Response Flow

from the caller should be expected within a certain period. A response arriving before this period indicates that the caller could not have fully listened to the challenge. A response arriving after this period indicates the message response is probably junk as it has taken an unreasonably long time to be transmitted in its entirety.
C is the length of time taken to transmit the challenge message. In effect it corresponds to the playback length of the challenge in seconds. The period when responses can be received begins 2xC seconds from when the challenge began

to be transmitted. This is the minimum time (assuming negligible propagation delays) after which a response could be received.

Figures 2.8,2.7, and 2.6 help to describe the aforementioned scenarios based around the timing of the messages sent from the Caller to the Secretary. They explain what traits might be discovered from typical SPAM messages in relation to each of the scenarios. **Figure 2.5** shows how a credible caller might use

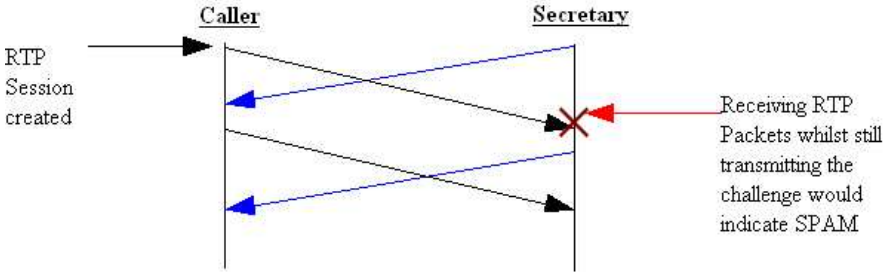

Figure 2.6: Immediate Response - Challenge interruption

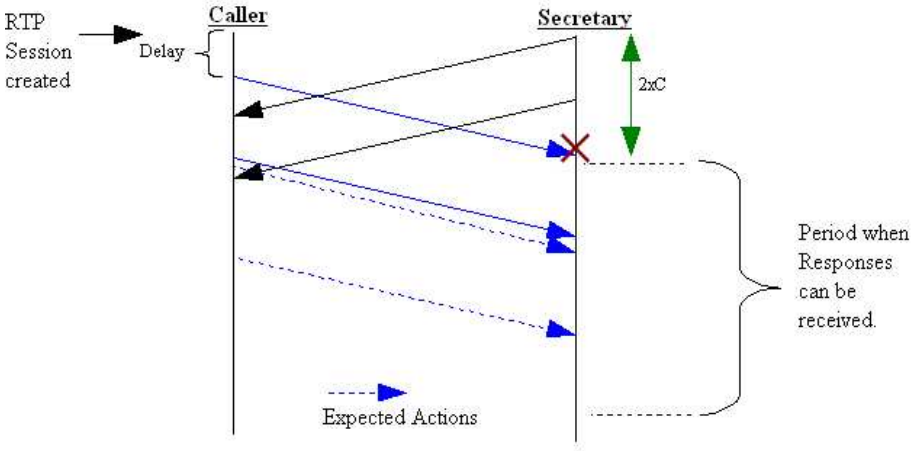

Figure 2.7: Delayed response arrives at an unacceptable time

the system. **Figure 2.6** illustrates the case when the SPAM message is sent immediately and appears to collide with the secretary as the challenge is being made. **Figure 2.7** is similar, only the SPAM caller waits for a short time and then sends the message. In any case, even though the secretary has finished sending the challenge by the time the first packets of the response arrive, it is still too early based on a calculation of when a valid response is likely to have

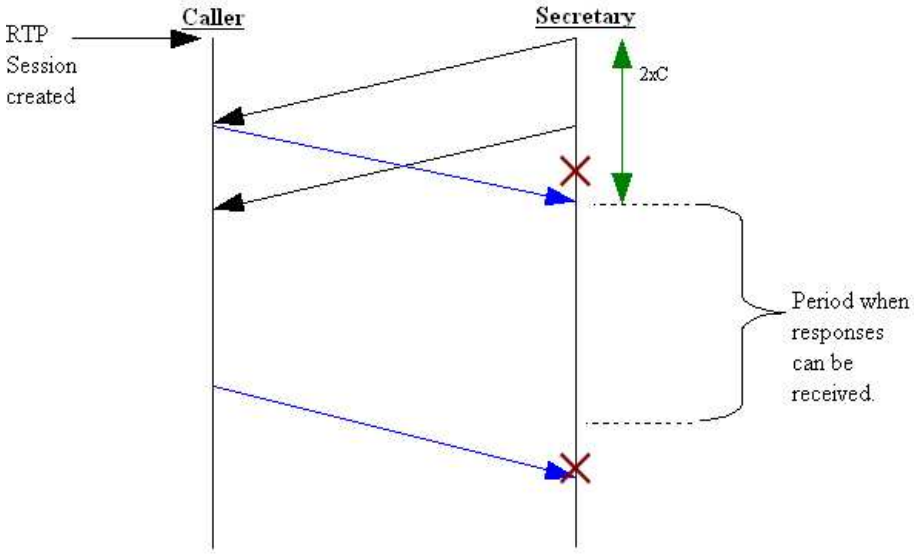

Figure 2.8: Response too long in duration

been sent. **Figure 2.8** demonstrates two means of detection, namely receiving RTP packets too early and too late. This example shows when a caller sends the SPAM message as soon as it receives RTP packets from the Secretary, but it also shows how the Secretary can provide some kind of safeguard against long response streams.

2.4.2.4 Examining the caller

By having access to a recorded sample of the caller's speech, some kind of feature extraction could take place, upon which subsequent classification of the caller could be made. Some initial assumptions made about typical SPAM callers and methods of detecting them may be found in **??**.
It was decided that the goal should be to identify the caller in relation to who the system may have had experience with before. So, the chosen approach was to utilise existing voice recognition techniques to first of all reduce the speech signal into a compact acoustic representation [32] and then try to identify patterns which may be specific to a certain caller. Modern voice recognition systems may be summarised as shown in **Figure 2.9** [45, 8, 37]. This however assumes an intended tokenisation and recognition at the word level. In this project it was decided that word recognition may not be necessary if it is identities rather than semantic attributes which are sought after. This theme is expanded in

the next section and a more generalised view of a modern speaker recogniser is shown in **Figure 2.10** [5, 14].

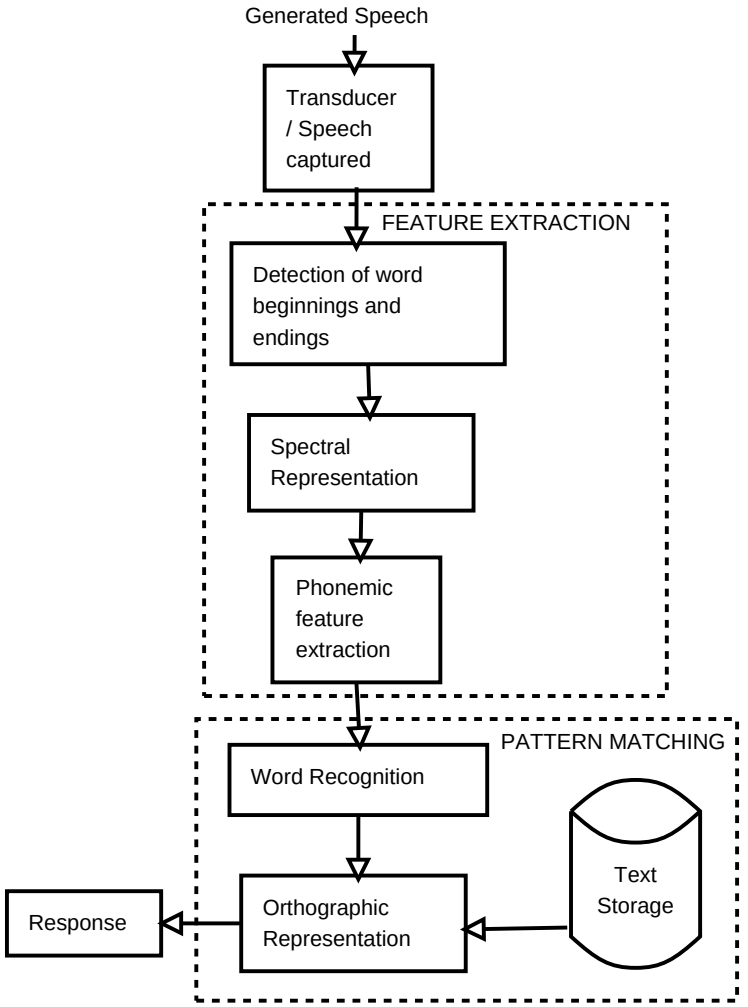

Figure 2.9: Modern Automatic Speech Recognisers

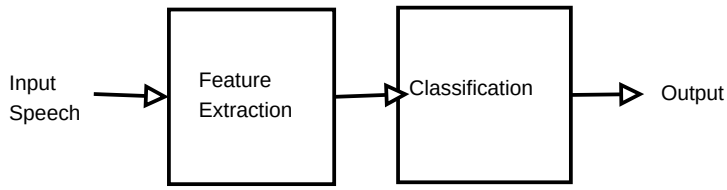

Figure 2.10: Modern Speaker Recognisers

2.4.3 Expanding the Solution - A text independent speaker problem

Each speaker voice sample would be statistically reduced to a vocal "fingerprint" or model using existing feature extraction methods.[5] The examination would stop short of extracting phonemes, words and phrases from the capture. Instead of separating the voice data into a series of states, the entire utterance would be examined as a single state effecting the creation of a speaker model for each caller sample. The choice was a straight-forward one after consideration of the following general Operational and Practical constraints.

2.4.3.1 Practical Constraints

1. **Lack of VOIP spam sample data**
   Some experience of SPAM telephone callers may be wide spread in relation to regular phone services, but precious little is known of the same through VOIP networks. One can only make assumptions on what likely characteristics are exhibited based on the regular phone equivalent. Acquisition of VOIP spam calls would require the release of the constitutive data packets from an ISP. ISPs are typically reluctant in this respect.

2. **Language**
   Tokenisation of words is computationally impractical if one should try to copy the standard email model. Speech evolves and does not inherit the same rigid boundaries and syntax as text. Some other alternative representation of the speaker's voice is necessary.

3. **Processing time**
   Telephone calls are a real time activity and any undue delay will have an adverse effect on the quality of the service. The secretary interface affords the user a context and some additional processing time for the monitoring of incoming calls. Any solution with a low processing time overhead is still highly desirable.

2.4.3.2 Operational Constraints

1. **Placement of proposed system**
   The ideal placement of such a content based filtering system is outlined in **figure 2.1**. This assumes access and familiarity to existing VOIP technology and infrastructure. The technical nature of VOIP as detailed in appendix ... would no doubt require much work in adapting existing structures to incorporate any proposed filtering system.

2. **Sample gathering time**
   A larger captured sample will require more processing time, whilst a shorter captured sample may prove cost efficient with regard to time but prove insufficient in regard to meaningful content. A balance is necessary.

3. **Sample quality**
   A standard audio format used by most of the codecs in VOIP is an 8000hz, linear 16 encoded audio file [18]. Codecs may introduce some quantisation noise. Therefore, the above format as a WAV file should be assumed to be the (unreachable) upper limit for quality in regard to the type of reconstructed sample file the system will deal with. Codecs are only available under licence thus restrictions will be placed on how accurately one can re-produce VOIP speech samples. Estimations based on the above audio file quality are really the only viable option in this respect.

4. **Project Complexity - Word Recognition**
   The time allotted for the project as a whole will simply not allow for the creation of an entire voice recognition system (including the creation of Hidden Markov models for bottom-up state-wise word discernment from phonemes [8]).

With reference to the previously described timing mechanisms and existing speaker recognisers, an overview of the proposed solution is presented in **figure 2.11** which will act as a guide to how the remainder of this document is structured and discussed.
The next chapter discusses the Feature Extraction which is performed upon the captured response from the caller.

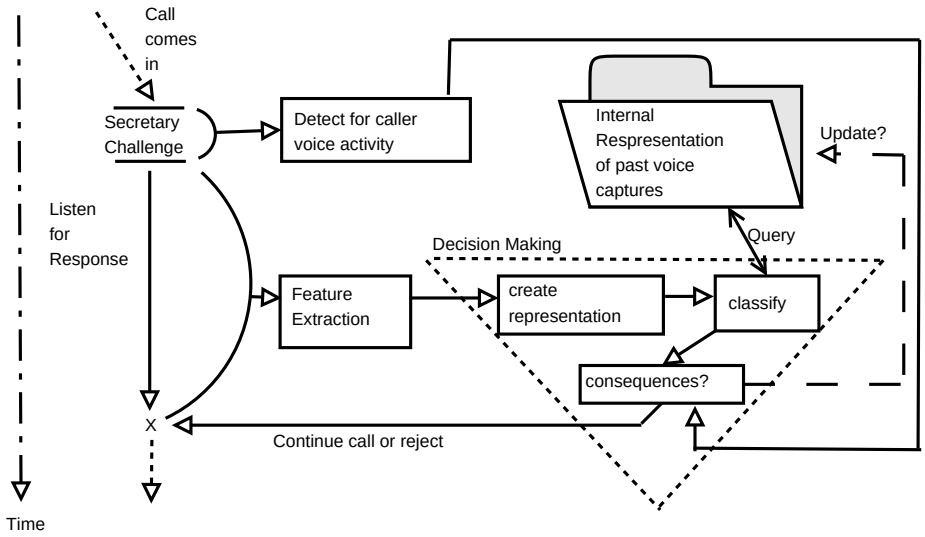

Figure 2.11: Project Overview

Chapter 3

Feature Extraction

3.1 Introduction

For each captured WAV sample which is made from incoming caller utterances, it should be possible to create a low cost statistical representation of the speech activity. Such representations will now be referred to as **features**. They will form the basis of any model construction for any given caller. Focusing in on the Feature Extraction area of **Figure 2.11**, time-based vectors of features should be extracted from the speech as shown in **Figure 3.1**. In the interests

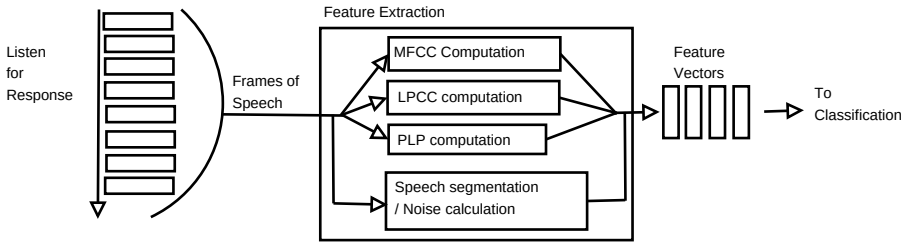

Figure 3.1: Feature Extraction

of robustness and capacity for successful identification, any extracted features should exhibit the following characteristics [26, 20].

1. Large variability between speakers - Features should contrast between speakers to a reasonable degree.
2. Small variability within one speaker - Features from a single individual should not alter much with different words, accents or emotional emphases.
3. Ease of measurement - Feature gathering should take place within a reasonably short period of time. They should be representative of naturally occurring in human speech.
4. Low Complexity - The feature sets one extracts from a certain piece of speech should be very small relative to the amount of actual speech data which they strive to model.
5. Resilience to Noise/Interference - Any speech data traveling through a low bitrate channel will be subject both to additive noise and linear distortion [1]. The former is that which is introduced by anything in the real environment which happens to be picked up by the recording equipment.The latter is caused by distortion in the spectral realm caused by the quality of the recording equipment, the construction of room, or the quantisation in the transmission channel.

One would hope that a chosen feature set would inherit as many of these attributes as possible, thus avoiding several undesired scenarios:

1. A valid speaker is refused because their speech bared no resemblance to that which the identification system was used to.
2. A SPAM caller would gain entry because their speech bared no resemblance to that which the identification system was used to, in a negative sense.
3. A valid speaker might be refused entry because noise prevented a pure recording from taking place.
4. A SPAM caller might try to disguise their voice by changing their utterance or through mimicry. For example, in a text independent scenario, were a SPAM caller to yield a speech utterance which is closer textually to what a perceived valid caller might, he is more likely to score higher in a whitelist matching scheme than otherwise.

And finally a very serious issue must be considered which is also of concern to standard email SPAM filters. That is, the enrollment by accident or otherwise of a SPAM caller in the system as a valid caller. This would have critical implications for the future discernment of callers coming into the system [41].

3.1.1 Existing Measures

When one considers existing techniques used in text independent speaker identification, there are several approaches to transforming the speech input from its high-content source data to a smaller, feature-set representation. Two notable procedures are:

1. Mel Frequency Cepstral Coefficients (MFCCs) [1, 8, 7, 20, 33, 31, 17, 13]
2. Linear Prediction Coding Coefficients (LPCCs) [8, 22, 43, 4, 27, 29, 32, 20, 33, 31, 17, 13]

During the course of this project it was decided these two be explored. In addition an enhanced extension to the latter was also implemented and considered as a third alternative. This involved the use of warped spectral filterbanks, incorporated to reproduce the actual human perception of speech. This is why it is called Perceptual Linear Prediction Modeling (PLP) [15].

3.1.1.1 The Human Vocal Tract

All of the above are based on the principle that it is possible to obtain information pertaining to the size and shape of the human vocal tract and the mass of the vocal chords, by analysis of the frequency spectrum of a speech sample [45, 8, 1]. The power spectrum of an observed speech signal is the product of the Physical Excitation mechanisms (the lungs and vocal chords) and the Vocal Tract (those passages above the vocal chords through which air passes during voiced speech - includes the oral and nasal cavities).

$P_x = P_v P_h$

Where P_x is the observed power spectrum, P_v is the excitation power and P_h is the vocal tract filter.

The vocal tract varies greatly in size and shape between speakers and so feature sets based on estimations and approximations of its effects are most desirable. Voiced sounds are those where the vocal chords vibrate creating the initial sound (and determining the fundamental frequency). As the sound travels upwards, the vocal tract influences the creation of other resonant frequencies (formants). The resultant speech passes out through the mouth and nose, however the vocal

tract closes or is obstructed at certain points in time, and in different places corresponding to the creation of consonant sounds. All vowel sounds are voiced sounds where there is no obstruction in the vocal tract, but not consonant sounds require the vocal chords to vibrate (for example, "p", "t") [45].
The above proposed methods examine the power spectrum and model the human physical speech mechanisms based on the formant and fundamental frequency trajectories.

3.2 Mel Frequency Cepstral Coefficients

This method attempts to isolate features from speech in a way that is akin to the human ear. It uses perceptual weighting principles to warp the frequency spectrum of the sound data in a way that is non-uniform and conducive to accentuating speech signal features which are influenced by the shape of the human vocal tract [1, 7]. Generally, this scheme seeks to extract a set number of coefficient values from an input speech sequence which may be subsequently elongated according to delta and delta-delta time, thereby including speed and acceleration movements.
The typical number of coefficients which may be extracted from the speech waveform is 12-14 [1].Of course, coupled with the delta coefficients, this can rise up to 24-28 or even 36-42. In any case, this is a significant representational reduction given that the input speech may be up to 8khz in bandwidth over several seconds.
A diagrammatic overview of the procedure is shown in **Figure 3.2**.

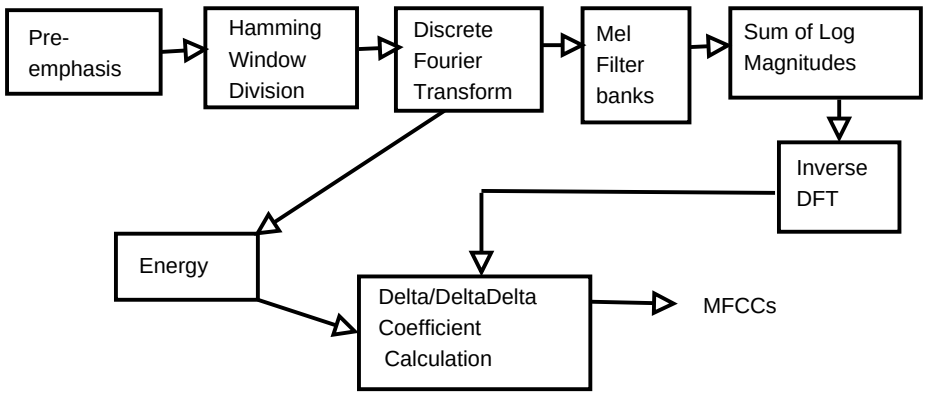

Figure 3.2: MFCC Calculation Overview

3.2.1 Pre-emphasis

High frequency formants have a smaller amplitude with respect to lower ones. One would wish to have a similar amplitude across all formants [1, 31, 17, 13]. Therefore, the first stage in the process is to filter the speech signal so as to achieve this. the function then is:

$$H(z) = 1 - a * z^{-1}$$

In the above, a is the pre-emphasis factor and has a typical value of 0.95. Consequently, the high frequencies are amplified by about 20dB [1].

3.2.2 Windowing

As a speaker talks the spectral response varies according to the different words, syllables and phonemes spoken. The phoneme is the contents of the shortest time interval over which articulatory stability happens [8, 45]. Therefore, it may be considered to be the shortest interval during which meaningful vocal activity occurs. A reasonable expectation is for this interval to last in the region of 20-30ms [1]. This approach makes the assumption a phoneme, should be able to be encased in one window.
The speech signal could then be split up into a continuous string of these regions (or windows), each containing what is called the short time spectrum - the power spectrum for that particular window, or interval [8] (**figure 3.3**).

But the windowing function is not just a matter of arbitrarily splitting a

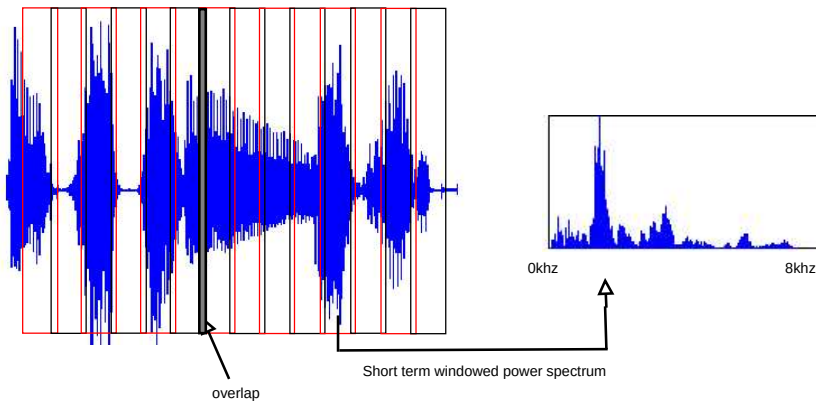

Figure 3.3: The short term spectral window

sound wave into consecutive short-time periods. The very presence of a window introduces a distortion on the whole picture - this is called leakage and happens when adjacent windows swap energy from different spectral frequency values [1]. To reduce this, a Hamming window function was chosen and windows were set to overlap with each other. The size of the overlap is of course a separate debate, but for the basis of this thesis, a length of 10ms was chosen. The hamming function itself is given as:

$$w(n) = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{N-1}), & n = 0, ..., N-1 \\ 0 & \text{Otherwise} \end{cases} \tag{3.1}$$

There are alternatives to the Hamming window (for example "Hanning") but the guiding principle in the application is that the side lobes are lower than with a purely rectangular implementation, which, combined with the overlap reduces the leakage problem. **Figure 3.4** further describes the typical application of a Hamming window in this context.

3.2.3 Spectral analysis

Spectral analysis follows the windowing of the sound sample in that each window of speech generated in step 2 undergoes a Discrete Fourier Transform [8]. This has the effect of translating each window into an M-point histogram where each interval on the x axis represents an approximated frequency band (from 0 up to 8khz) and the y-axis indicates the magnitude for that frequency.
Taking the square magnitude of each individual DFT value results in the power spectrum.

$$powerspectrum = |DFT(w(n))|^2 \tag{3.2}$$

The resolution of the power spectrum is determined then by the number of points specified at the Discrete Fourier transform stage. In this thesis, 256 was specified as a compromise between accuracy and complexity.

3.2.4 Filterbanks

Filterbanks are applied to the short term power spectrums, emphasising human-observable characteristics of the sound. The human ear uses around 24 different filter banks to process the sound information which it receives. This method

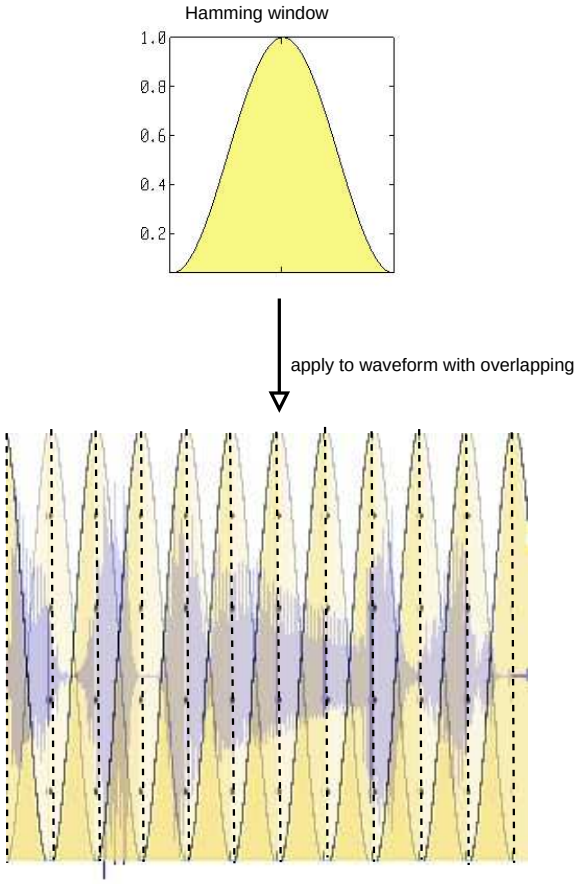

Figure 3.4: The hamming window

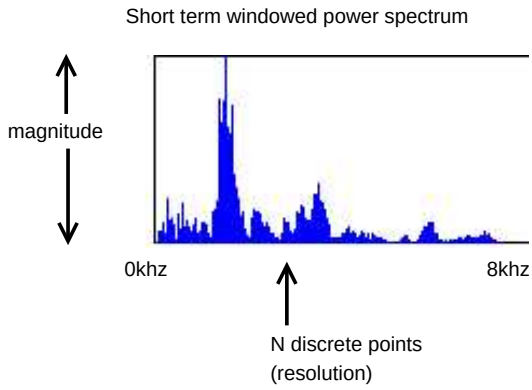

Figure 3.5: A Power Spectrum for an 8Khz range

aims to replicated this processing though a kind of perceptual weighting process [1].
Mel Frequency Cepstral Coefficients get their name from the Mel Scale Filter banks through which the power spectrum is passed. These filter banks are spaced such that the spectrum below 1Khz is processed by more banks since it contains more information on the vocal tract shape. Then, moving over 1Khz the band frequency gets higher, so too does the bandwidth of the filter bank (logarithmically)(**figure 3.6**). This increasing nature helps to allow for temporal resolution of bursts [1]. Each filter is a triangular window which then

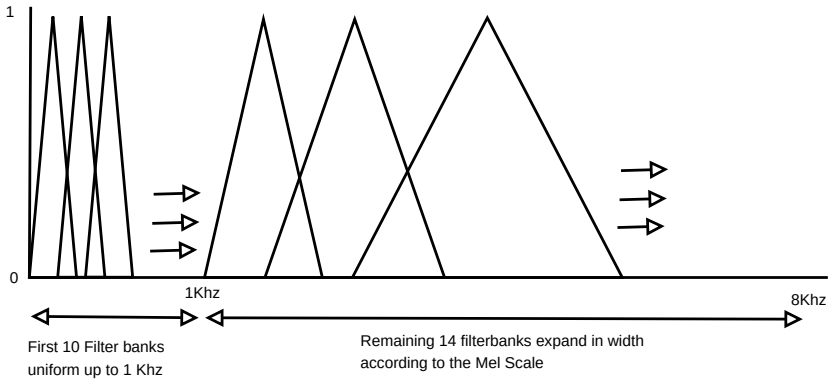

Figure 3.6: Mel Scale Filter Banks

gets shifted and warped according to the frequencies it covers. The construction of these filter banks is constructed by first determining the mid point, or the point of the triangle in each case. The entire range of filterbanks can then be

constructed using the following formula:

$$U_{\triangle_m}(k) = \begin{cases} |midpoint - f| < \triangle_m \rightarrow 1 - (|midpoint - f|/\triangle_m) \\ |midpoint - f| \geq \triangle_m \rightarrow 0 \end{cases} \tag{3.3}$$

where $\triangle_m$ is $\frac{1}{2}$ the width of the m-th triangular filter bank.

The width of each triangular filter is chosen so that there are 10 uniformly spaced filters from 0-1Khz. The triangular mid points can be calculated as being the sum of the previous midpoint and half the current triangle width:

$$b_m = b_{m-1} + \triangle_m \tag{3.4}$$

from 1Khz up to 8khz then, the width of the next triangle is approximately equal to 1.2 * $\frac{1}{2}$ the current triangle width [1]:

$$\triangle_m = 1.2 \times \triangle_{m-1} \tag{3.5}$$

The output of each filter bank is then merely the sum of the magnitudes when the filter bank is applied to the window in question (see **figure 3.7**).

$$Y_t(m) = \sum_{k=b_m-\triangle_m}^{b_m+\triangle_m} X_t(k) U_{\triangle_m}(k + b_m) \tag{3.6}$$

3.2.5 Log energy calculation

The output of each filter bank together yield N impulse values. By taking the logarithm of these values, a dynamic compression is applied which has the effect of making feature extraction less sensitive to dynamic variation - similar to the ear's processing [1].

3.2.6 Mel Frequency Cepstrum Calculation

It is profitable in terms of inter-speaker robustness if one could further decorrelate the filterbank outputs. This is achieved by performing an inverse

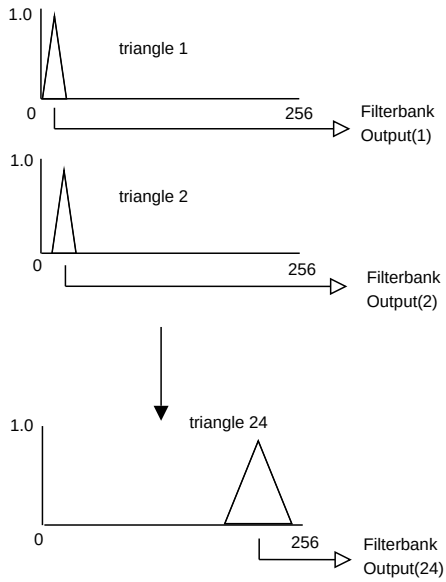

Figure 3.7: Mel Filter Bank Outputs

Discrete Fourier Transform upon them.

$$Y_t^{(m)}(k) = \sum_{m=1}^{M} \lg\{|Y_t(m)|\} \cdot \cos(k(m - \frac{1}{2})\frac{\pi}{M}), k = 0, ..., L \qquad (3.7)$$

The IDFT in effect reduces to a Discrete Cosine Transform (DCT) which is very good at procuring uncorrelated values and creates what one refers to as the cepstrum [1, 8]. Its also useful because it means that the complexity of the representation is further reduced from 24 down to somewhere in the region of 9-13. In most modern Speech Recognition Systems the number of coefficients calculated is less than 15, and in this project 12 was chosen [1].
In practice the total number of coefficients to be used is the number of cepstral coordinates +1, because the logarithm of the total energy for the frame in question is appended to the start of the coefficient set. It is a useful additional value because it gives a more complete picture of the stress a certain speaker might place on different sounds.

3.2.7 Delta Coefficients

The feature values by this stage do not take into consideration much about the dynamic variation of the speech. In other words, the speed and acceleration of the speech as perhaps characteristic of that speaker.
Such information can be represented by taking the first and second order differences with respect to time. The delta and delta-delta coefficients may be obtained from windows in the same locality of the entire window sequence:

$$\begin{aligned} \triangle^0\{u_t\} &= u_t \\ \triangle^i\{u_t\} &= \triangle^{i-1}\{u_{t+1}\} - \triangle^{i-1}\{u_{t-1}\} \end{aligned} \tag{3.8}$$

where u_t is a given feature vector at time t and i is the order of the difference.

Inclusion of the delta and delta-delta coefficients produces an overall feature vector of size 3*$NoOfCoefficients$. Applying this to a windowed sound sample is equivalent to reducing several seconds of speech into N*(3*$NoOfCoefficients$), where N is the number of windows decided at the outset in accordance with the sample length,window length and window overlap, and NoOfCoefficients is 13.

3.2.8 Summary

Certainly if one is looking for a way to transform an input speech sequence in a more simplified, uncorrelated and dynamically-aware representation, the MFCCs offer a most viable option. It is a popularly used method in many modern speech recognition systems [7] though it will require the setting of several parameters.

Below is an overview of the inputs, outputs and parameter choices:

Inputs:
A sequence of speech (T seconds long)
No of pts in Fourier Transforms (256)
The length of the windows ($30ms$)
the length of the window overlap ($10ms$)
the windowing function ($Hamming$)
the pre-emphasis factor (0.95)
the number of Mel filter banks (24)
the number of coefficients to extract (12)

Outputs:
N*(3*$NoOfCoefficients$) $(39 * N)$
ie. N 39pt feature vectors

3.3 Linear Predictive Cepstral Coefficients

Rather than performing operations on a set number of filterbank outputs, Linear Predictive Coding seeks to produce an approximation of the all-pole model of a spectral envelope by weighting the sum of previous speech samples [29].
The same preliminary step of converting the sound wave into a windowed, spectral representation is used as with the MFCC calculations, thereby yielding a series of short term, spectral windows which can be further analysed individually. Of particular significance again is the pre-emphasis step which is required to even out the spectrum in preference of higher frequency characteristics which tend to have a lower amplitude.
The said approximation is made possible through the premise that each sample can be calculated as a weighted sum of past samples, which is intuitive given that adjacent speech samples are usually highly correlated. To begin with then, the following equation presents this notion (taking into account a window of length N) [27]:

$$\hat{s}_n \approx \sum_{i=1}^{p} a_i s_{n-i}, i = 1, ..., p \tag{3.9}$$

where p is the order of the predictor.

The goal is to compile a list of these "a" values or linear prediction coefficients such that at each stage in the prediction, the error of the predictor is minimised. The error (e) of a certain predicted value can be calculated from the corresponding actual value:

$$e_n = s_n - \hat{s}_n \tag{3.10}$$

and then from this one can find the summed squared error (E)over a window of length N would be:

$$E = \sum_{n=0}^{N-1} e_n^2 \tag{3.11}$$

OR

$$E = \sum_{n=0}^{N-1} (s_n - \sum_{k=1}^{p} a_k s_{n-k})^2 \tag{3.12}$$

The most accurate predictor value for the a value is then one where the Squared Error value is a minimum and this is obtainable by solving the equation when the derivative with respect to a is 0.

$$\frac{\partial E}{\partial a_j} = 0 \tag{3.13}$$

Differentiating equation 3.12 with respect to a_j gives:

$$\begin{gathered} 0 = -\sum_{n=0}^{N-1} (2(s_n - \sum_{k=1}^{p} a_k s_{n-k}) s_{n-j}) \\ 0 = -2\sum_{n=0}^{N-1} s_n s_{n-j} + 2\sum_{n=0}^{N-1}\sum_{k=1}^{p} a_k s_{n-k} s_{n-j} \\ \sum_{n=0}^{N-1} s_n s_{n-j} = \sum_{k=1}^{p} a_k \sum_{n=0}^{N-1} s_{n-k} s_{n-j} \end{gathered} \tag{3.14}$$

One may define the covariance matrix Φ with elements $\phi_{i,k}$

$$\begin{gathered} \phi_{i,k} = \sum_{n=0}^{N-1} s_{n-i} s_{n-k} \\ \phi_{i,0} = \sum_{k=1}^{p} \phi_{i,k} a_k \end{gathered} \tag{3.15}$$

This can be re-written in matrix form which looks like this [27]:

$$\begin{Bmatrix} \phi_{1,0} \\ \phi_{2,0} \\ \phi_{3,0} \\ \cdots \\ \phi_{p,0} \end{Bmatrix} = \begin{Bmatrix} \phi_{1,1} & \phi_{1,2} & \phi_{1,3} & \cdots & \phi_{1,p} \\ \phi_{2,1} & \phi_{2,2} & \phi_{2,3} & \cdots & \phi_{2,p} \\ \phi_{3,1} & \phi_{3,2} & \phi_{3,3} & \cdots & \phi_{3,p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_{p,1} & \phi_{p,2} & \phi_{p,3} & \cdots & \phi_{p,p} \end{Bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_p \end{Bmatrix} \tag{3.16}$$

or just $\Phi_0 = \Phi_a$

and therefore, the a values can be simply retrieved by multiplying the toepliz by the matrix inverse:

$$a = \Phi^{-1}\Phi_0 \tag{3.17}$$

At this point the window of speech is considered once again. By refining the above equations to take into consideration the boundary edges the following is sufficient to describe the toepliz matrix in relation to the window:

$$\phi_{i,k} = \sum_{n=0}^{N-1-(i-j)} s_n s_{n+(i-j)} \tag{3.18}$$

Thus, $\phi_{i,k}$ can be written in terms of the auto-correlation function r_{i-j} since it is only dependent on the difference $i - j$ [8, 4]:

$$r_k = \sum_{n=0}^{N-1-k} s_n s_{n+k} \tag{3.19}$$

Consequently the application of the autocorrelation matrix looks something like this [8, 27]:

$$\begin{Bmatrix} r_1 \\ r_2 \\ r_3 \\ \cdots \\ r_p \end{Bmatrix} = \begin{Bmatrix} r_0 & r_1 & r_2 & \cdots & r_{p-1} \\ r_1 & r_0 & r_1 & \cdots & r_{p-2} \\ r_2 & r_1 & r_0 & \cdots & r_{p-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{p-1} & r_{p-2} & r_{p-3} & \cdots & r_0 \end{Bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_p \end{Bmatrix} \tag{3.20}$$

Inverting the matrix to help calculate the a parameters is achieved via the recursive Levinson-Durbin approach [27]:

1. Assume $a_k^{(i)}$ denotes the values of the predicted parameters at iteration i.
2. Assume $E^{(i)}$ denotes the residual energy at iteration i (and when i is 0, $E^{(0)}$ is just the total energy for the frame).
3. $k_i = (r_i - \sum_{j=1}^{i-1} a_j^{(i-1)} r_{(i-j)})/E^{(i-1)}$
4. $a_i^{(i)} = k_i$
5. $a_j^{(i)} = a_j^{(i-1)} - k_i a_{i-j}^{(i-1)}, 1 \leq j < i$
6. $E^{(i)} = (1 - k_i^2)E^{(i-1)}$

The k's which are calculated are intermediate values which are otherwise referred to as *reflection parameters* [27].

The output from the Levinson Durbin recursions which are of most interest are the a parameters, the prediction parameters for the spectral window. The number used is open to experimentation but the greater the number, the more adept the prediction model (**figure 3.8**) [17]. As a general rule, one parameter

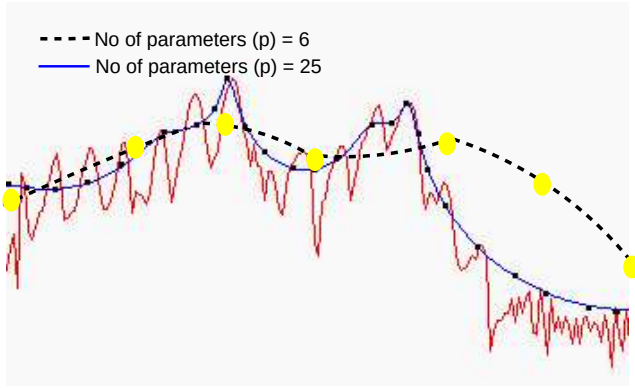

Figure 3.8: The Power Spectrum with approximations by different sizes of predictive parameter sets

is selected for each khz in the range, plus 2-4 in addition to model lip and glottal effects. This project is concerned with source audio of 8khz. Therefore, 12 was chosen as the number of parameters to estimate.
A feature set with highly correlated values is not of much use as it will not ensure high inter speaker variability. A further step is involved in order to convert the LP parameters into cepstral coefficients [7].

$$c_k = a_k + \frac{1}{k}\sum_{i=1}^{k-1} ic_i a_{k-i}$$
$$1 \leq k \leq p \tag{3.21}$$

The end product was therefore set as a feature vector of 12 cepstral points, which could be extended to include the same delta and delta delta sets as before, given that the processing is occurring on a window by window basis.

3.3.1 Summary

The nature of this kind of analysis is somewhat different than that of the Mel Filter bank outputs. It is an all pole modeling technique taken from a least squares comparison and resulting in a "best possible" estimation of the frequency curve, the resolution of which is determined by the user's specification of the order of the linear predictor.

Inputs:
A sequence of speech (T seconds long)

No of pts in Fourier Transforms (256)
The length of the windows (30ms)
the length of the window overlap (10ms)
the windowing function (Hamming)
the pre-emphasis factor (0.95)
the order of the linear predictor (12) (also equivalent to the number of coefficients)

Outputs:
N*(3*NoOfCoefficients) (39*N)
ie. N 39pt feature vectors

3.4 Perceptual Linear Prediction - (PLP)

Perceptual Linear Prediction is an extension of the Linear Predictive Model in that it aims to provide a cepstral coefficient feature set based on the predictive all pole modeling of the windowed spectrum. However, it is enhanced by the exploitation of known perceptive/acoustic properties incorporated by the human ear when it processes sound. This takes the form of the perceptual weighting of the audio spectrum and helps the overall process isolate more clearly distinct features within it [15].
Moreover, when the Linear predictive model is created under the previous procedure, it ignores the fact that the spectral resolution of human hearing decreases with frequency. Hearing is also more sensitive to the middle frequency range in human conversation.
The pictorial overview for this method is shown in **figure 3.9**.

3.4.1 Preliminaries - Windowing, FFT, Pre-emphasis

These functions are performed in same way as with MFCC's and LPCC's. They are foundational to any of these feature extraction methods.

3.4.2 The Bark scale

The bark scale is a psycho acoustical scale which ranges from 1-24. That is, there are 24 critical bands, so 1 bark corresponds to a an area or range within the spectrum where some kind of unique activity may be humanly perceived.

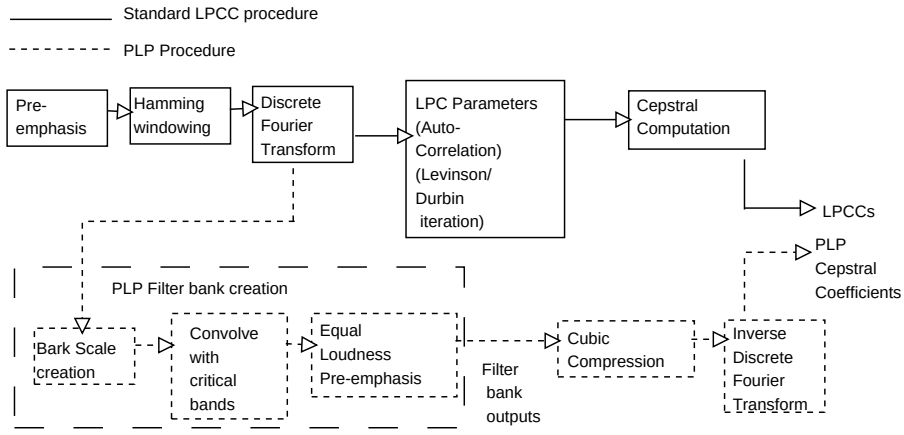

Figure 3.9: PLP Overview

It is possible to convert a Hz value into a Bark value by applying the following formula:

$$\begin{gathered}\Omega(f) = 13\arctan(0.00076f) + 3.5\arctan((f/7500)^2) \\ OR \\ \Omega(f) = 6\ln\{f/1200\pi + [(f/1200\pi)^2 + 1]^{0.5}\}\end{gathered} \tag{3.22}$$

where f is the frequency in hz.

The bark scale is fairly reminiscent of the mel scale, though the filter banks used in PLP are shaped differently.
The frequency axis of the short term spectral window is converted from hz to Bark and the bark index for each amplitude is translated accordingly. The filter banks are constructed by determining the mid point of each. These are distributed at roughly 1 bark intervals throughout the spectrum. For each filterbank, the total range of bark values are surveyed and subtracted from the filterbank midpoint. The difference value is then used to determine the value for that index point in the critical band filters. The following formula provides a further illustration:

$$\Psi(\Omega) = \begin{cases} 0 & \text{for } \Omega\prime < -1.3 \\ 10^{2.5(\Omega\prime+0.5)} & \text{for } -1.3 \leq \Omega\prime \leq -0.5 \\ 1 & \text{for } -0.5 < \Omega\prime < 0.5 \\ 10^{-1.0(\Omega\prime-0.5)} & \text{for } 0.5 \leq \Omega\prime \leq 2.5 \\ 0 & \text{for } \Omega\prime > 2.5 \end{cases} \tag{3.23}$$

where $\Omega\prime = \Omega(f)$ - $\Omega(i)$ and i = 1,...,24, denoting the midpoints of the filterbanks. (the above representation is a more generalised approach to the convolution of the critical band filter definition and periodic function provided in [15]).

3.4.3 Equal loudness curve

The next step is to apply equal loudness pre-emphasis. The human ear is non-equally sensitive to different frequencies, so in order to adapt the spectral model appropriately an equal loudness curve is applied to the array of filter banks. The curve seeks to simulate the sensitivity of the human ear to sounds at around the 40dB level.
The equation of the curve itself is given as:

$$E(w) = [(w^2 + 56.8x10^6)w^4]/[(f^2 + 6.3x10^6)^2 \times (f^2 + 0.38x10^9)] \tag{3.24}$$

It is applied to the filter banks in this way:

$$\Xi(\Omega(w)) = E(w)\Psi \tag{3.25}$$

ie the hz equivalent of each bark value from the critical band filters is multiplied by the Equal loudness curve. By applying this curve to the filter banks the structure in 3.10 is created:

3.4.4 Cubic compression

The resultant filters banks are then applied to the actual spectral window of the speech which means that a spectral sum is taken of each filter bank, incorporating the weighting which the filter bank imposes on its area of the spectrum. Mathematically this means:

$$\Xi(\Omega(w_i)) = \sum_{w=w_{i1}}^{w=w_{i24}} w_i(w)P(w) \quad \text{where } i = 1....24 \tag{3.26}$$

where P is the power spectrum and w_i is the weighting function performed by the filter bank concerned.

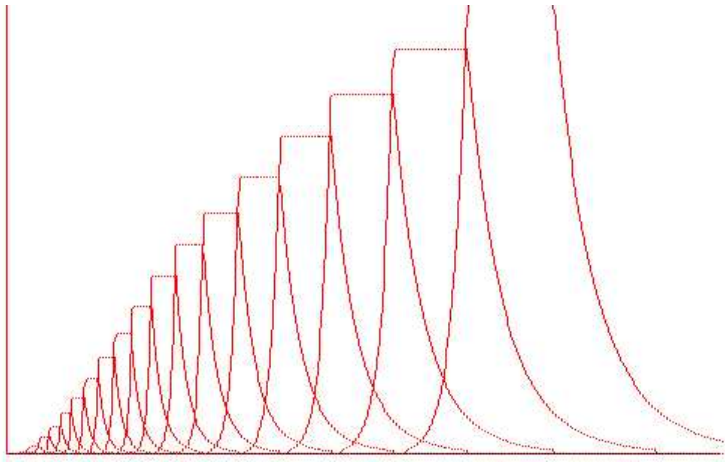

Figure 3.10: PLP Filter Banks

24 filter bank summations are then collected, but in order to further replicate the non-linear relationship between perceived loudness and sound intensity, a cube root operation is performed on each filter bank output.

$$\Phi(\Omega) = \Xi(\Omega)^{0.33} \tag{3.27}$$

3.4.5 Auto regressive modeling

The cubic compressed filterbank outputs can be transformed using an Inverse Fourier Transform to yield the auto correlation values which are same parameter approximations used in the LPC model - the a's in equation 3.9. As with the LPC model one need only choose the first p auto correlation values from the IDFT. The value of p was set to be the same as the LPC procedure.

3.4.6 Cepstral Transformations

The same de-correlating procedure as with the standard LPCC calculation is repeated, providing cepstral coefficients according to the number of parameters [7].

$$c_k = a_k + \frac{1}{k}\sum_{i=1}^{k-1} i c_i a_{k-i}$$
$$1 \leq k \leq p \tag{3.28}$$

3.4.7 Summary

PLP cepstral coefficients can be further extended to include speed and acceleration information in the form of delta and delta delta coefficients.
The PLP method shows a good appreciation for the the psychoacoustic realities of human hearing. In tests it is shown to have competed on a par with MFCCs and better than LPCCs under noisier conditions [39]. It would certainly appear to be a more involved and informed approach to the standard LPCC computation.

Inputs:
A sequence of speech (T seconds long)
No of pts in Fourier Transform (256)
No of pts in Inverse Fourier Transform (34?)
The length of the windows ($30ms$)
the length of the window overlap ($10ms$)
the windowing function ($Hamming$)
the pre-emphasis factor (0.95)
Number of critical band filter banks (24)
the order of the linear predictor (12) (also equivalent to the number of coefficients)

Outputs:
N*(3*$NoOfCoefficients$) ($39 * N$)
ie. N 39pt feature vectors

3.5 Noise / Speech Segmentation

Regardless of any recording conditions, noise is almost always present in any audio sample, and in VOIP especially it is unavoidable. This presents a number of challenges in relation to accurately extracting meaningful and unbiased data from the speech. However, it may also provide an extra point of leverage in spam detection.

Spoken word is not a constant stream of vocal activity. Human speech is characterised by pauses and gaps between words, syllables and sentences. The entire utterance can therefore be divided up into those time intervals which contain speech and those which do not (**figure 3.11**). In other words, it is necessary to run a speech segmentation stage in parallel with the standard feature extraction techniques (**figure 3.1**). In other words

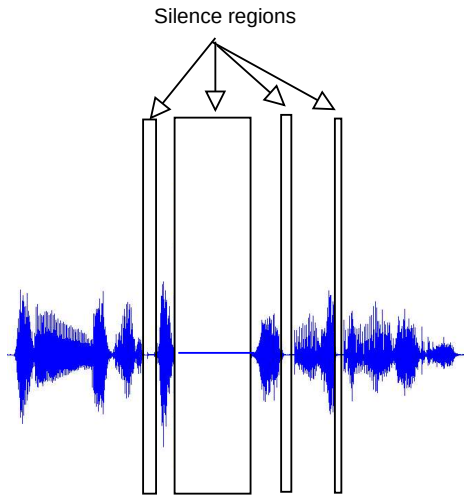

Figure 3.11: A waveform consisting of audio and silence

$$Entireutterance = spokenpart + silencepart$$

3.5.1 Audio codecs

A related technique used in low bandwidth audio codecs (eg G.729) is the use of Voice Activity Detection (VAD) [18]. In order to preserve bandwidth, the codec will only code those areas of the input speech pattern which it can ascertain as speech. The remaining pauses are not encoded, but instead the codec merely waits until it detects the voice again. In the meantime some of these codecs introduce their own noise at the decoding end to fill the gap.
In such a situation as this project it would be most helpful if the spoken part of each voice sample could be gathered together and the silence part merely discounted from playing any further part in feature extraction. This has the following benefits:

1. Only that which is relevant to feature extraction is collected together. Normalisation at various stages would be adversely affected by silences.

2. The cost of processing the speech part would be less than processing the entire duration of the input sound sample.

3. Quantifying the amount of actual speech determines an estimate of overall vocal activity. In a situation such this VOIP SPAM problem, one indication of SPAM is a seemingly unintelligent stream of speech, unbounded by cognisance on the part of the calling party.

The above objectives are achievable through the following techniques.

3.5.2 Standard activity detection

Assuming the entire sample of the input utterance can be split up into a series of overlapping windows, there is a discrete time division already built into the audio source which is to be analysed (**figure 3.12**). The totality of the sample

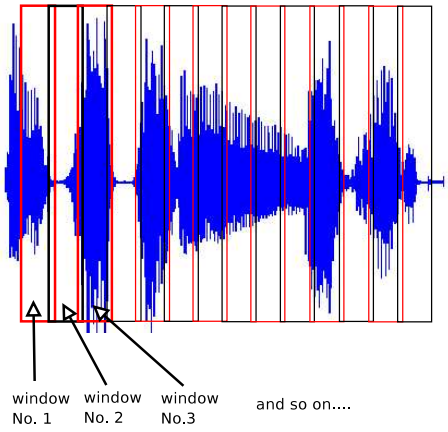

Figure 3.12: Waveform windowing gives a frame-based timing dimension

could be defined as a sequence of windows designated as either pure noise or pure speech. Distinguishing which is which is found by examining the frame-wise signal energy from the short term power spectrums [36].
For each window, the total observed signal can be said to be $X(w_{all}, t)$. At time t one is concerned with the corresponding spectral window and w is each individual frequency within the range of the spectrum. Speech and Noise are

assumed to be additive and independent, therefore it follows that

$$X(w,t) = S(w,t) + N(w) \tag{3.29}$$

This equation assumes that the background noise energy is stationary throughout the input signal.
The signal energy of each window, X(w,t), will either be greater than or equal to the noise energy, N(w,t). The degree of how much larger the energy from X(w,t) is in comparison with N(w,t) decides if the frame is a designated "noise" frame or not. N(w,t) is a continual estimate based on previous designated noise frames with the initial frame defaulting as noise and thereby the base estimate. Calculating the status of the current frame is done via the following formula:

$$N(w,t) = \begin{cases} N(w,t-1) & \text{if} XNR(t) > \alpha \\ (1-\beta)N(w,t-1) + \beta X(w,t) & \text{otherwise} \end{cases} \tag{3.30}$$

where: $XNR(t) = \sum_w X(w_{all},t) / \sum_w N(w_{all},t-1)$

α is a threshold for the signal to noise ratio.
β is a parameter determining the adaptation speed of the noise estimation.

The above formula produces an estimation of the noise spectrum for each window in the input signal and also a strict designation as to whether the given window/frame is pure speech or pure noise (in effect, a noise "masking" array).

3.5.2.1 Limitations

The above method is useful as long as there is a reasonable definition between the noisy regions and the speech regions. Much depends on the setting of the XNR and adaptation parameters. One cannot assume that all incoming speech signals are the same in regards to their expected noise levels.
However, for basic tests with a controlled sample procurement environment it is adequate and cost effective.

3.5.3 Quantile Based Noise Estimation

A more realistic view of the noise situation within a given speech signal is possible by moving from frame-wise noise estimation to frequency based estimations [36, 2]. Not all frequency bands contain significant magnitudes during real speech. In fact, even when a subject is talking some bands still remain around the base noise level. A more accurate approximation of the overall noise

spectrum may be had if each window is sorted according to each frequency magnitude. In the form of an equation this would be:

for each w sort $X(w,t)$ such that

$X(w,t_0) \leq X(w,t_1) \leq ..... \leq X(w,t_T)$ where $T =$ no of Windows

Plotting a graph of the resulting sort according to a qth-quantile of time ($0 \leq q \leq 1$), would resemble **figure 3.13**.

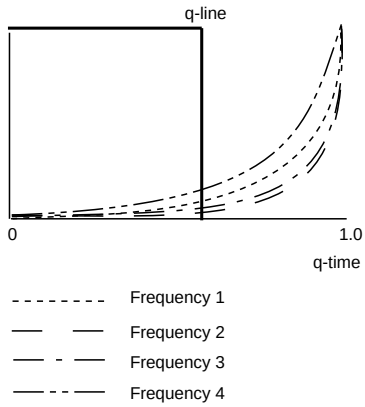

Figure 3.13: QBNE approach to sorted frequencies

q is a quantile of the total length of the sorted signal. The graph only shows a handful of sampled frequencies from a set of possibly 256, though it is immediately apparent that each frequency does not contain much information for a significant portion of the overall time. One could say that each frequency band carries pure noise for at least the qth-fraction of time. Expressed as a formula this translates to:

$N(w) = X(w, t_{\lfloor qT \rfloor})$

Thus, the setting of the q value is tantamount to the setting of a time-influenced noise threshold for each frequency. A recommended value for q is the median (0.5), though this value is open to experimentation and is somewhat dependent on the expected noise level of the incoming signal.
Ultimately the estimated noise spectrum can be found on a frequency by frequency basis by taking the highest magnitude value occurring before the q-time bounded cutoff in the sorted collections.

3.5.4 Advanced Quantile Based Noise Estimation

This method seeks to take into consideration the non-uniformity of the curves in **figure 3.13**. For example, using 0.5 as a q-time cutoff value may be suitable for some frequencies but not optimal for all, even when the same basic shape is made by each. To cater for the variety of speeds and accelerations with each frequency curve subsequent to the q-cutoff point, AQBNE proposes that this point should not be a single static boundary in q-time. Instead it should be curved itself to allow for a more realistic estimation of noise levels [2].
Figure 3.14 demonstrates how the QBNE may be improved upon through this modification.

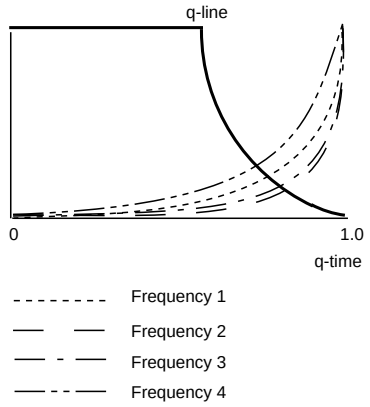

Figure 3.14: Improved AQBNE approach

In this improved model there is a *qmin* parameter which designates the original point of the curve, where it begins to slope downwards. Furthermore, the specification of the r parameter determines the steepness of the slope the curve makes. Typical values for *qmin* are 0.3-0.45 and for r, 10-15. Both are open to experimentation. In the same way as before, an estimation of the noise spectrum is merely a case of taking the sorted frequencies arrays and for each, using the magnitude at the point of intersection with the q-time curve.

3.5.5 Weiner Filtering/Noise removal

Assuming a varying degree of additive noise contained within every incoming speech signal, it may prove beneficial to try to remove it, thereby leveling the playing field so to speak, in relation to each caller candidate prior to feature

extraction.
A well known method for carrying out this task is Wiener Filtering [36, 8] which is defined as:

$$H(w,t) = (X(w,t) - N(w,t))/X(w,t) \tag{3.31}$$

$$S(w,t) = H(w,t)2X(w,t) \tag{3.32}$$

where the intermediate result ($H(w,t)$) is calculated from the original spectrum ($X(w,t)$) and the estimated Noise Spectrum ($N(w,t)$). It is then used to find $S(w,t)$ which is the noise-removed spectrum.
Often, *equation 3.31* is modified to

$$H(w,t) = max(X(w,t) - N(w,t), 0)/X(w,t) \tag{3.33}$$

taking consideration of the case where the long term estimated noise spectrum is larger than the instantaneously observed power spectrum. It has been observed that better speech recognition results are obtained if a small fraction of the noise spectrum is actually left in the signal. The formula then becomes:

$$S\gamma(w,t) = max(S(w,t), \gamma N(w,t)) \tag{3.34}$$

where $\gamma = 0.04$ (experimentally obtained).

3.5.6 Summary

Noise analysis allows the measurement of noise quantity, noise spectrum and the separation of speech from silence. The noise spectrum may be then used to remove the noise from the signal.

Inputs:
A sequence of speech (T seconds long)
No of pts in Fourier Transform (256)
The length of the windows ($30ms$)
the length of the window overlap ($10ms$)
the windowing function ($Hamming$)
the pre-emphasis factor (0.95)
β - the adaptation speed parameter to the Noise spectrum estimation
α - the XNR threshold value
q - the QBNE quantile-time cutoff value
$qmin$ - the first q-time point of the AQBNE cutoff curve

r - governs the steepness of the AQBNE cutoff curve
γ - the Wiener filter noise remnant fraction

Outputs:
A mask array of length T showing which windows are noise and which aren't (T = No of Windows)

Frame-wise noise estimation:
An array of noise spectrum estimates, the final one being the most accurate.(length T)

QBNE/AQBNE :
a single noise spectrum estimate for the whole input signal.

Chapter 4

Speaker Classification

4.1 Introduction

Feature Extraction procures a number of different quantities. But how may these quantities be manipulated into a form which is useful when the desire is to compare the similarity between different groups of extracted features? The focus of this chapter will therefore be on techniques used to order or cluster the extracted features into a more generalised representation. In effect these techniques will shape a model or "audio fingerprint" based on the raw features, paving the way for pattern-matching of different identities.
If at the start of feature extraction, the input speech signal had been split into N windows then one can expect to have the following available:

- N feature vectors, made up of cepstral coefficients from one of the 3 listed methods (MFCC, LPCC or PLP). Each vector containing up to 39 constituent values.
- a noise mask array of N values, determining the noise value for that window ("1" for noise, "0" for speech).
- An estimated Noise spectrum and estimated values for the overall noise energy - using AQBNE and QBNE.

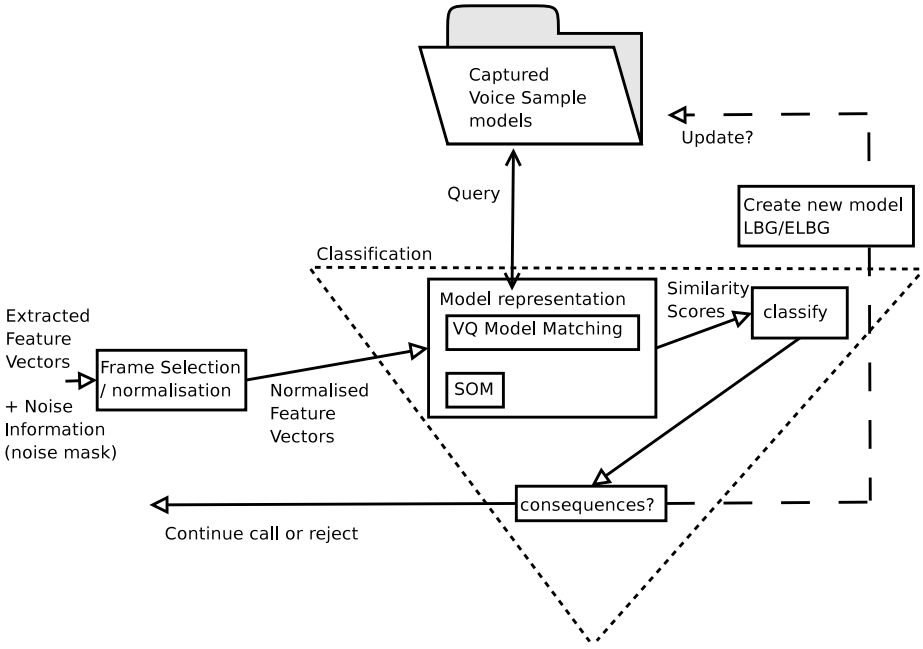

Figure 4.1: Classification Overview

Figure 4.1 is an overview of the different components described in this chapter. It is subset of figure 2.11 and shows how selected features are normalised, and then passed to one of two pattern matching algorithms. The first is Vector Quantization with reference to trained speaker models. The second is a Self Organising Map. Both will return an similarity score or some kind of measurement as to how the observed features correspond to existing knowledge within the system. These scores provide a basis for subsequent categorisation.

4.1.1 Frame selection / Normalisation

The feature vector set for one speaker needs to be normalised to allow effective comparison with other stored examples. However, the normalisation process should be only concerned with the pure voice data from the captured sample. Otherwise silences have an unessessary influence on the normalisation. Therefore, using the noise mask obtained from the Feature Extraction phase, it is possible to normalise the extracted coefficients on a frame-wise basis - with reference to whether the frame contains speech or not. The resultant normalised vectors are set to values between 0 and 1 and passed on for modeling to take place.

4.2 Modelling - An audio fingerprint

For dichotomisation of feature vectors two main approaches were considered at the outset.
The first is the *parametric* model where one makes an assumption on the feature vectors - that they can be fitted to a certain distribution (eg. Gaussian) according to the maximisation of certain inherent criteria. Some structure is assumed and that can be described through parameterisation [14, 20]. The advantages of this approach are that different parametric models are more succinctly discriminative, and the dichotomisation from feature vector to "speaker model" is more efficiently performed. The disadvantages of this approach are that the assumed structure of the data may be too rigid and in a situation where the range of sample data is more fluid and less predictable, it may be inappropriate to apply such restrictive assumptions.
Taking into account the scope of this project, and considering the variety of sound data which may be possible to transmit over IP telephony an alternative approach to parametric modelling was preferred. (Speech, and in particular a prompted spoken response is but one type of source which could be used as input to the system. Any kind of aural information - music, quantised digi-

tal tones, pure noise, silence, shouting, white noise - could be used to counter, frustrate or confuse the decision making process). If one could be assured that every time a call is received the caller plays by the rules by supplying a spoken word response, then parametric modelling would be more appropriate. Such a guarantee is not automatic.

The second, and preferred approach was to apply $non-parametric$ modelling, where no structure is assumed present in the source sound sample [14]. Within this approach, there are several options in regard to performing a comparison. A Nearest Neighbour approach involves the calculation of a similarity score from the individual distance measurements between each point in the feature vector and their nearest correspondents in a separate, trained sample [32]. For example:

$$dNN(x, R) = min|x - r_j| \quad \text{where } r_j \in R$$

However, such a method is often open to distortion from outliers, and it may be computationally expensive both in terms of training data and testing and comparing newly observed samples against known samples
Vector Quantisation (VQ) [32, 14, 19] can be used instead and it was the chosen method for this project. Instead of directly comparing the feature vector data itself directly, the vectors are transformed in a representative form of lower complexity. It involves the construction of what is referred to as a codebook - a collection of centroids calculated from the feature vectors which are also known as codewords. This could be viewed as the clustering of input feature vectors where each feature vector falls into the catchment area of one cluster (or centroid, or codeword) (**figure 4.2**). The number of codewords to be used is not always clear however. Of course, it depends very much on the data itself which is to be clustered. In this project therefore, the exact amount will be left as an exercise during the testing phase. Another acknowledged shortcoming with this approach is that there can be no overlap between cluster areas - each feature vector therefore relates to one, and only one centroid. Depending on the type of data analysed this may be prove to be too rigid.
In any case, what is then required is that codebooks be created for each training sample. Then it is the codewords of each known sample which are compared to the extracted feature vectors of the observed utterance and the distances calculated in much the same fashion as the Nearest Neighbour method, namely:

given a codebook C
where $C = c_0, c_1, c_2.....c_{K-1}$ (K = number of codewords)

and given an input feature vector set X
where $X = x_0, x_1, x_2, ...x_{T-1}$ (T = number of Feature Vectors)

$dVQ(x_i, C) = d_{min}(x_i, c_j)$ (where $c_j \in C$)

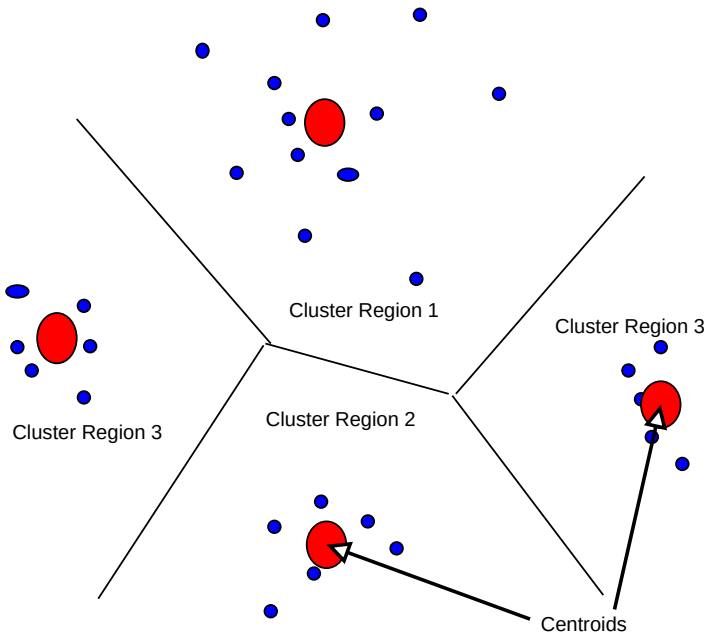

Figure 4.2: Cluster Regions and Centroids

Furthermore, it is possible to obtain the average quantisation distortion (AQD) from the individual distortions, across the entire input range. This essentially gives an estimate of how well a set of feature vectors fits a codebook and an effectual similarity score:

$AQD(X, C) = \frac{1}{T} \sum_{i=1}^{T} d(x_i, C)$

4.2.1 Creating the codebook and learning methods

If a system such as the proposed one is to be capable of comparing new, previously unseen feature vector sets to some kind of existing, experiential knowledge, this knowledge must take the form of a set of codebooks gained from training data.
A system which relies on training data to postulate the identity of new observations will only be as effective as the training data allows it to be. Therefore the choice of how to train the system and how to manipulate known samples for this is crucial.
Training can be supervised or unsupervised [32, 20]. The former relies on an allowance of inter-dependencies between all the possible sets of data used to train the model. The latter implies the creation of some kind of dichotomised representation, independent of other feature sets. In other words, where there exists a feature vector set, it may be transformed into a clustered representation based purely on its own overall characteristics. With the supervised model, the cluster set would be identified in relation to prior experience with other feature vector sets.
In this project it was decided that both be explored as a means of comparing codebooks. The well-known *LBG (Linde-Buzo-Gray)* algorithm [24] was implemented along with Patane and Russo's enhanced version [30], as an unsupervised approach. Tuevo Kohonens *Self Organising Maps* provided a supervised alternative [21].

4.3 LBG (Linde-Buzo-Gray)

4.3.1 Introduction

The LBG, or Generalised Lloyd algorithm works by selecting an initial codebook of size K and then iteratively finding a new quantiser where the overall distortion is less than or equal to the previous amount. **Figure 4.6** gives an overall picture

of the general procedure.
The first step is the initialisation of the codebook itself into K codewords. K

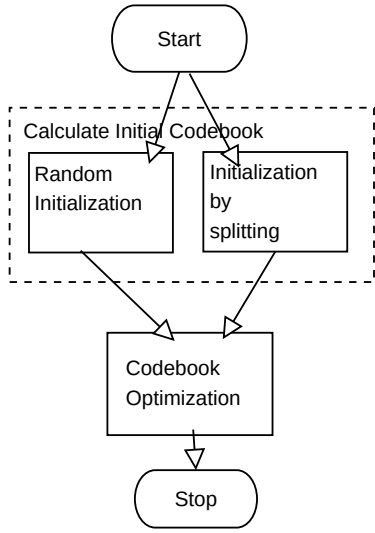

Figure 4.3: Basic LBG overview [30]

is only chosen experimentally, and effective values of K are largely dependant on the data to be clustered.
The initial codewords are chosen by either randomly selecting K different vectors from the input set, or by choosing an initial vector from the set and splitting it according to a value e which is a pre-calculated, fixed perturbation vector. This is repeated m times so that the final number of codewords is equal to 2^m, where each stage involved the splitting of each previously generated codeword.
Once the initialisation of the original codebook is complete, the remainder of the LBG algorithm is concerned with optimising the codewords so that the overall distortion measure in relation to the feature vector set is reduced on each iteration.
The general procedure is as follows:

1. Find the Voronoi partition of the current codebook Y in relation to the input set X. This consists of assigning the nearest codeword to each input vector in the set:

   $P(Y) = S_1....S_{Nc}$ (where Nc = Number of Codewords)
   and
   $S_i = \{x \in X : d(x, y_i) \leq d(x, y_j), j = 1...Nc, j \neq i\}\ i = 1...Nc$
   $(y \in Y)$

On a side note, $P(Y)$ is in effect an optimal partition because

$D(Y, S) \geq D(Y, P(Y))$

2. Calculate the quantiser distortion of the Voronoi set $P(Y)$ set in relation to the current codebook. This is given by:

   $D_m = D(Y_m, P(Y_m))$ (where m is the iteration number)

   then,

   if $|D_m - D_{m-1}|/D_m > \epsilon$ (where $\epsilon > 0$ and is a precision measure)
   constitute new Codebook Y_{m+1} from $P(Y_m)$ (step 3)
   return to step 1 and begin a new iteration
   else stop $\rightarrow Y_m$ is the optimal codebook

   (where $\epsilon > 0$ and is a precision measure)
   (generally $\epsilon = 0.001$ - 0.1)

3. Calculating the new codebook is the same as calculating a new centroid $x(A)$ whereby the Euclidean Squared Distances between it and any of the input vectors which reference it as their Nearest Neighbour (set A) is a minimum:

   $d(x, x(A))|\ x \in A = d_{min}(x, u)|x \in A$
   ($d()$ is the Euclidean Squared distance measure)
   (u is any vector in the set referencing the centroid as its NN)

   so in other words, the new centroid is calculated merely through:

   $x(A) = \frac{1}{N} \sum_{n=1}^{N} x_n$ (where there are N elements in A anxd $x \in A$)

   Figure 4.4 provides an overview of the entire scheme.

   The end product of the process is an optimised codebook whose codewords are effectively the optimised cluster centroids with respect to the input data set.

4.3.2 Limitations of LBG - Enhanced LBG

The LBG algorithm is essentially an approximating algorithm with limitations imposed by the positioning of the initial codebook. The initial codebook plays a

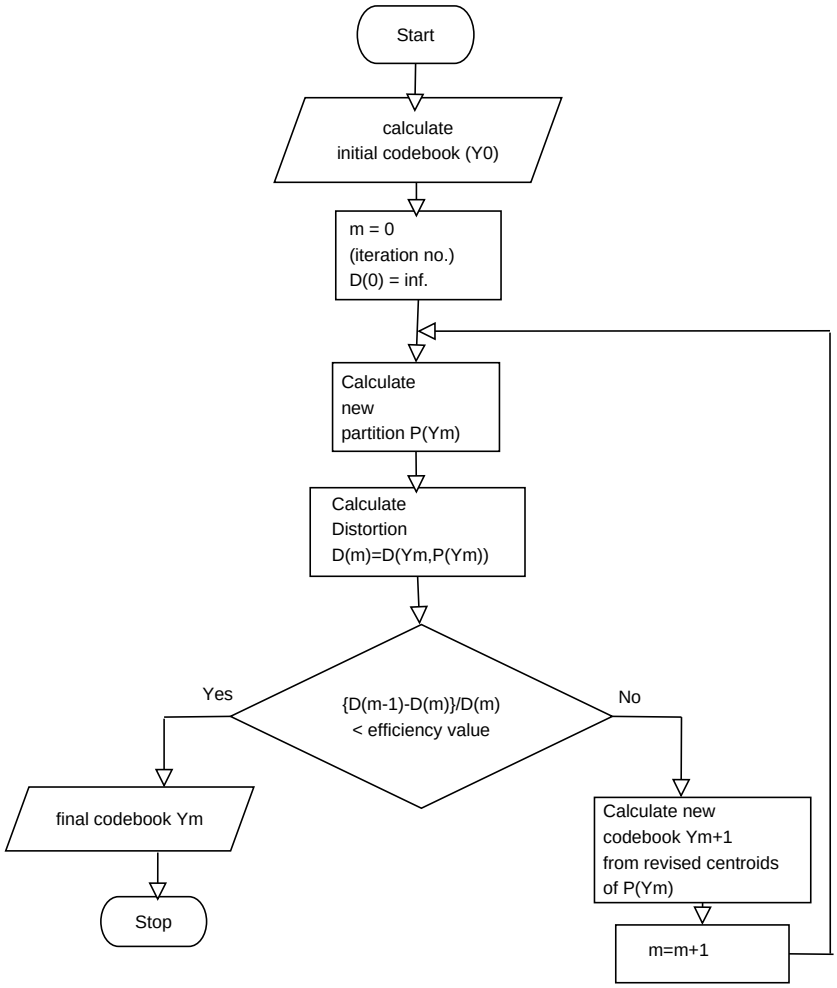

Figure 4.4: LBG Flowchart

large part in determining just how effective the overall method is and badly positioned codewords at the beginning simply may be recovered from, [30]. Moreover, badly positioned initial codewords slow the algorithm down somewhat [23]. **Figure 4.5** gives a basic picture of the possible problems represented in just two dimensions.

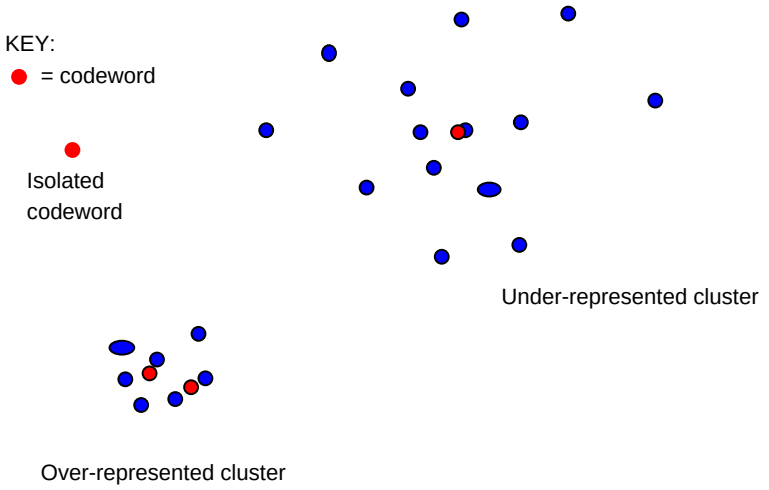

Figure 4.5: Typical LBG Clustering problems

Namely three particularly undesirable effects may be brought to pass [30] :

- Over-representation of a cluster. A relatively small cluster can be made to have a large number of centroids.
- Under-representation of a cluster and consequent loss of precision. A relatively large cluster can be made to have a small number of centroids.
- Isolated clusters never influenced. If an initial codebook contains codewords which are never referenced during the calculation of Voronoi partition on each LBG iteration calculation, then they will never actually be moved closer to any of the other clusters. They will remain in their isolation because all the feature vectors in the input set will continually reference other centroids.

The LBG algorithm is improved if the 3 situations above can be identified and appropriate action taken in each instance.

Patane and Russo's enhanced version includes the addition of an ELBG block to **figure 4.4** just prior to the calculation of the new codebook. Its purpose is to locate poorly positioned codewords and to take appropriate action, but only if a repositioned codeword will have a positive effect on the continuing iterative process of diminishing the overall quantiser distortion. **Figure 4.6** resembles the ELBG block.

The flowchart in **figure 4.7** gives some idea of where it is placed.

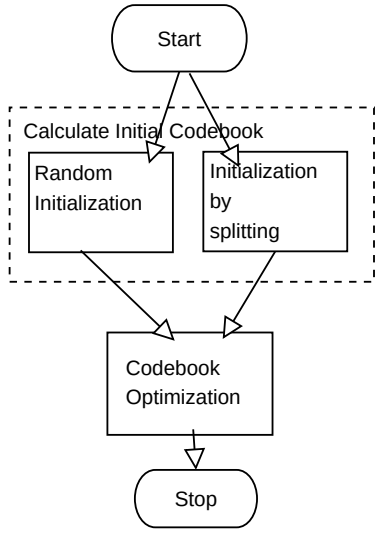

Figure 4.6: Basic ELBG overview

4.3.2.1 Enhanced LBG

Gersho states that "each cell makes an equal contribution to the total distortion in optimal vector quantisation with high resolution" [30].The method relies on the computation of what is called a "Utility" index U_i for each of the codewords. This is in effect a measure of how much distortion a given codeword is contributing overall.
It is defined as the normalisation of the total distortion of the codeword with respect to the mean distortion of all the codewords:

$D_{mean} = \frac{1}{Nc} \sum_{i=1}^{Nc} D_i$

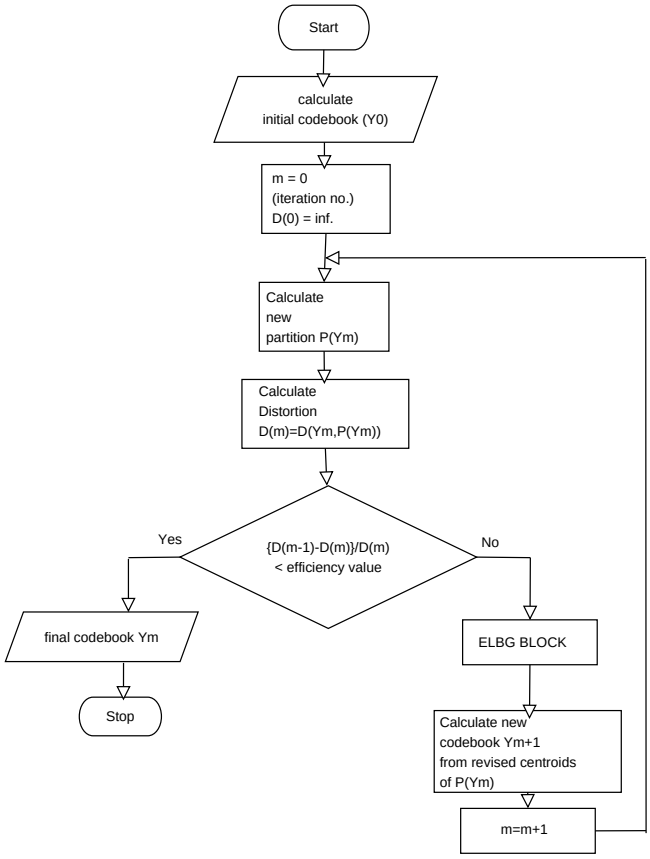

Figure 4.7: ELBG Flowchart [30]

$U_i = D_i/D_{mean}$, for i = 1...Nc

With respect to Gersho, it is apparent that the equalisation of distortions is equivalent to the equalisations of Utilities.
In practise then, a codeword with a low utility value (where $U_i < 1$) is one where there would appear to be more centroids or codewords representing a relatively small cluster area. Conversely, a codeword with a higher utility value is one where its own distortion is quite high and would lead to the assumption that it may be a single centroid representing a more disperse cluster.
The aim of the ELBG block then is to identify codewords with low utility values and try to move them into the general vicinity of codewords with high utility (but of course governed by the condition that the projected mean distortion must be lowered for such a codeword shift to be viable). It is an iterative process of shift attempts or SoCA's (Shift of Codeword attempts) which then ends when all the low utility valued codewords have been considered.
The actual workings of the ELBG algorithm will now be described.

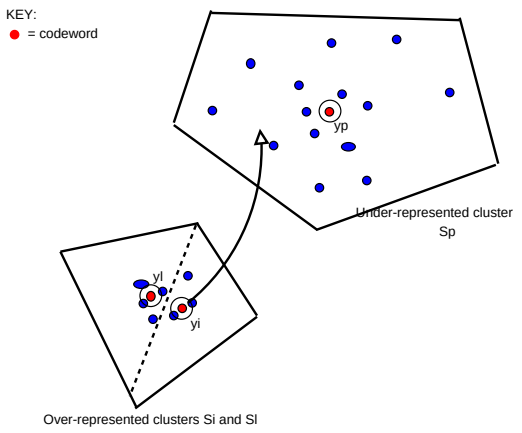

Figure 4.8: Initial Cluster situation (REF Patane, Russo)

step 1 - Utility calculations: The Utility value of each codeword is calculated and two groups are formulated:
1)those codewords with a Utility value $< 1 \rightarrow U_i$
2)those codewords with a Utility value $> 1 \rightarrow U_p$
Two candidate codewords are then selected, one from the U_i group and the

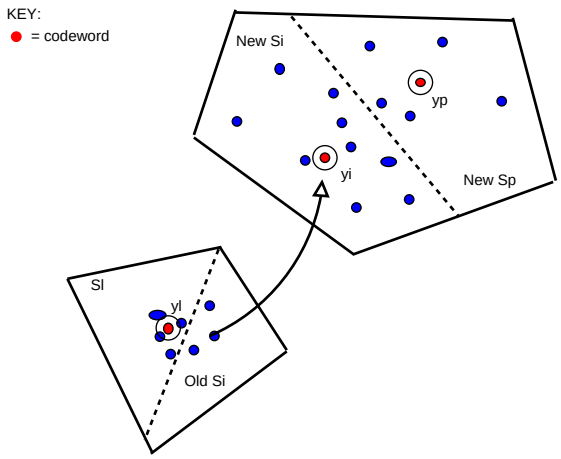

Figure 4.9: Moving the low utility centroid (REF Patane, Russo)

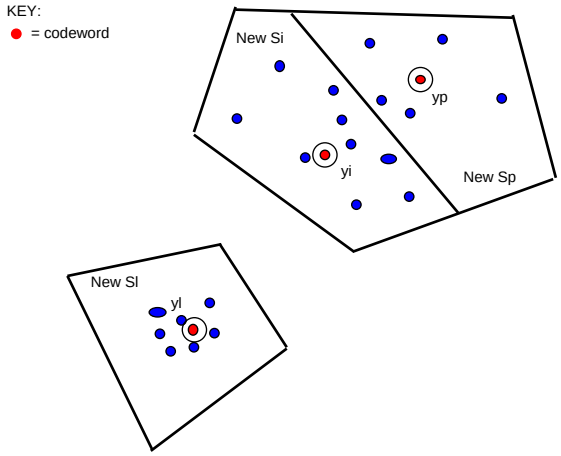

Figure 4.10: Reapportioning the cluster regions (REF Patane, Russo)

other from the U_p group. Typically the candidate from U_i will be selected in a sequential manner, whereas the U_p candidate is selected stochastically from the following probability measure:

$P_p = U_p / \sum_{h:U_h<1} U_h$

step 2 - Codeword shift: Shifting a codeword not only affects the values the codeword represents. It also means that the old cluster of a low utility codeword must have a new representative centroid to reflect its movement away. This would involve the merging of the old cluster with the next nearest cluster (which should be close-by because of the low utility).
The centroid of the cluster to which the low utility codeword is moved, is a high utility codeword, but it must also be recalculated in order to cater for the fact that the new cluster must be split to accommodate the new codeword.
Notice in the previous illustrations, the high utility codeword is called y_p and its cluster S_p. The low utility codeword is called y_i and its cluster S_i. The closest codeword to y_i is y_l and its corresponding cluster is S_l.
Therefore when y_i is moved to S_p, S_p must be re-split because y_i and y_p both reside within it. y_p's coordinates must change to reflect this and y_i's new coordinates must be calculated in accordance.
The two new sets of coordinates are determined by placing them on the principal diagonal in the h-dimensional hyper-box in which the cluster resides. (h is determined by the number of coordinates there are in each feature vector). One is placed at a point corresponding to $\frac{1}{4}$ of the range of each coordinate set whilst the other, $\frac{3}{4}$ of the range. The following formula better describes this idea:

h-dim Hyper-box = $[(x_{1m}, x_{1M}) \times (x_{2m}, x_{2M}) \times (x_{3m}, x_{3M}) ..... \times (x_{hm}, x_{hM})]$
m = the minimum value in that coordinate range
M = the maximum value in that coordinate range

So, for example:

y_i becomes $[\frac{1}{4}(x_{1M} - x_{1m}) \times \frac{1}{4}(x_{2M} - x_{2m}) \times \frac{1}{4}(x_{3M} - x_{3m}) \times ..... \times \frac{1}{4}(x_{hM} - x_{hm})]$
y_p becomes $[\frac{3}{4}(x_{1M} - x_{1m}) \times \frac{3}{4}(x_{2M} - x_{2m}) \times \frac{3}{4}(x_{3M} - x_{3m}) \times ..... \times \frac{3}{4}(x_{hM} - x_{hm})]$

This is not exactly the same procedure as given in Patane and Russo, but is rather a reasonable approximation of what is admittedly a suboptimal solution. The outline given in Patane and Russo is of a 2 dimensional solution. The above approximation was considered appropriate given the 13-39 dimensions likely to be used in this project.
In order to fine tune the new y_i and y_p points, a local LBG is used with a high precision value(ϵ = 0.1-0.3), and this usually only adds a couple of iterations to

the overall complexity. Considering y_i 's old cluster S_i, it is merged with S_l and y_l is adjusted by recalculating its centroid from the union of S_i and S_l.

Step three - Recalculating the distortion: The Voronoi partition is recalculated using the new codewords (including the shifted ones) and the overall distortion is compared against the previous value to see if the shift has actually been advantageous.
But the calculation of the Voronoi partition is perhaps the most computationally expensive part of the process. Again, a slightly suboptimal alternative is preferred. If one considers the three new cluster regions after shifting then their combined distortions may be said to be:

$D_{inew} + D_{pnew} + D_{lnew}$

It follows then that if this combined distortion is less than the combined distortion when considering the 3 regions in their previous state, the overall distortion of the entire codebook will be less.
In other words,

if $D_{inew} + D_{pnew} + D_{lnew} < D_{iold} + D_{pold} + D_{lold}$
then Overall Distortion is less, and thus, shift was beneficial
else shift was counter productive, and should be discarded.

Generally, this approximated intelligent shifting is a balance between precision and computational complexity. It does not offer an optimal solution but given timing and computational constraints, offers a reasonable compromise for gaining an accurate codebook.

4.3.3 Kohonen Self-Organising Maps (SOM)

A Self Organising Map [21, 44, 10, 35, 6] produces a 2D or 3D representation of each 13-39 pt feature vector which has been extracted from the speech. This is not merely a cut-price dimensional reduction on its own - it is done with reference to what the system understands to be the universe of feature vectors at that point in time. The Map stores topological relationships inherent to training data in network form.
For simplicity's sake only the 2 Dimensional representation is considered for the time being, though implementation of both 2-D and 3-D models was planned. The map design very much mirrored the existing C++ implementation from [44]
As a preliminary step, the size of the map must be determined. It can be con-

sidered as a 2 dimensional grid shape or "lattice" of height y and width x and therefore containing x*y constituent nodes. Each node will contain a set of weights, the number of which is determined by the size of feature vectors which will be presented as input (**figure 4.11**).

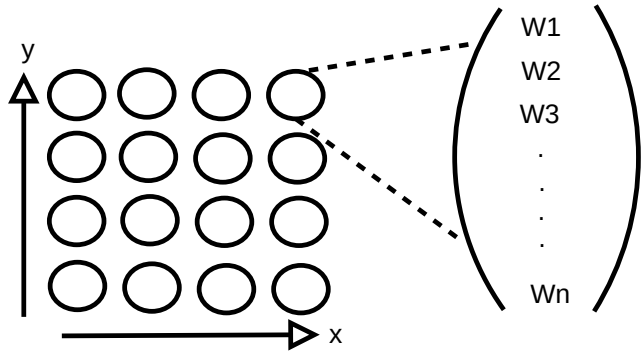

Figure 4.11: Basic 4x4 SOM configuration

The end goal will be to have a lattice weighted according to every possible input in the training data so that from the subsequent presentation of future sample vectors, a 2-D equivalent can be returned corresponding to the best fit point on the grid for that vector.
The categorisation of any input vector is relatively trivial. It is just a case of finding the node on the lattice whose weight set is closest to the vector. The categorisation is therefore a single 2-D point for each vector input.
However, this point determination is made trivial because the map will have undergone training at a prior stage.

4.3.3.1 Training the SOM

Each node in the x*y lattice will contain weights according to the size of the input vectors. The weight of each node is initialised as a random number between 0 and 1. (The assumption is made that every training vector consists of normalised values within the same range).
For each training vector which is presented to the SOM, the best matching unit (BMU) is calculated by iteratively finding the Euclidean Distance from the training vector to the weights at each node.

$$Dist_{EUC} = \sqrt{\sum_{i=1}^{NoWeights} (v_i - w_i)^2} \tag{4.1}$$

(where $v \in V$, an input/training Vector
and $w \in W$, the set of weights for a given node)

Once the BMU has been calculated for a particular input vector, the neighbourhood of the selected node is then discovered. This neighbourhood is a set area, the residents of which will have their weights adjusted to a greater or lesser degree depending on how close they are to the BMU. This adjustment itself occurs over a number of iterations, which is decided prior to the creation of the SOM.

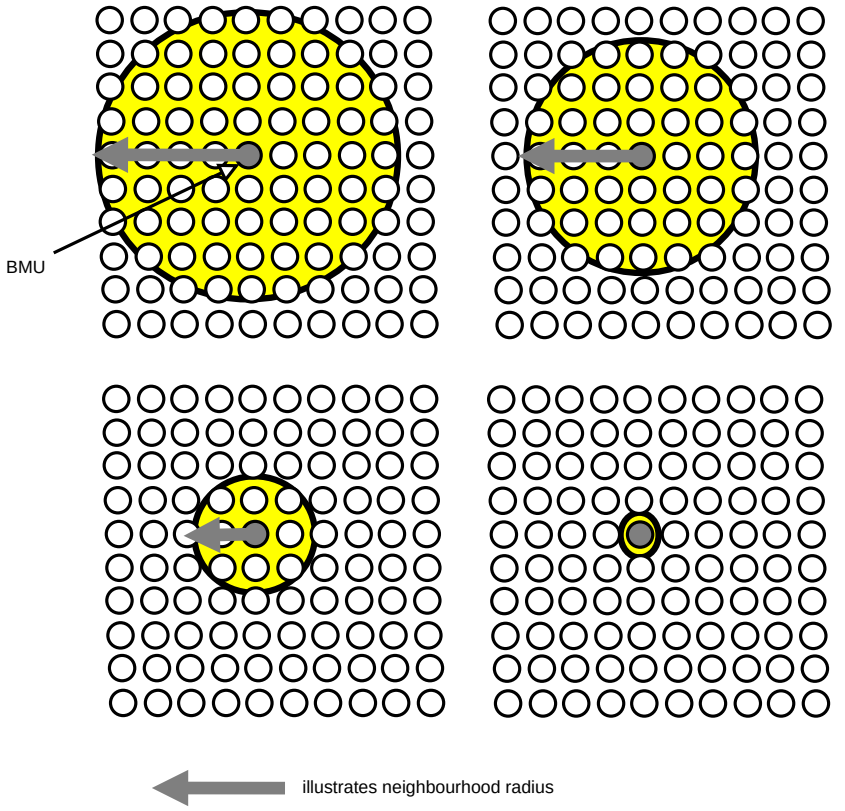

Figure 4.12: The ever-shrinking BMU Neighbourhood of a 10x10 SOM

In **figure 4.12** it can be seen that the neighbourhood takes the form of a circle, the size of which is adequately described by its radius. The radius is set to shrink over time according to the iteration number. The formula for the radius is as follows:

$$radius_t = radius_0 \times e^{(-t/\lambda)} \tag{4.2}$$

where t is the iteration number (1,2,3...$maxNoIterations$),
λ is a time constant given as

$maxNoIterations/log(SOMradius)$

As the learning takes place and the iterations pass, the size of the neighbourhood reduces until it consists of merely the BMU on its own.

4.3.3.2 Adjusting the weights

When a neighbourhood is defined of course, the constituent member nodes must have their weights adjusted. This is an iterative process and is done by adding a fraction of the difference between the old weight and the input vector to get the current weight for that iteration.

$$W_{t+1} = W_t + L_t(V_t - W_t) \tag{4.3}$$

where W_t is a given weight at time t,
L_t is the learning rate at time t
V_t is the corresponding value in the training vector at time t

The learning rate is set with an initial value (0.1) which then also decays with each iteration. It is defined as:

$$L_t = L_0 * e^{(-t/\lambda)} \tag{4.4}$$

Equation 4.3 would be fine if all the neighbourhood nodes were to have their weights adjusted by the same amount, but as previously mentioned, this needs be done in proportion to how far a neighbourhood node is from the BMU. Amending equation 4.3 to include this dynamic provides the following:

$$W_{t+1} = W_t + \theta_t L_t(V_t - W_t) \tag{4.5}$$

where θ_t is a time dependant value (actually decays over time itself) governing

how much influence a node's distance from the BMU has on its learning. θ_t is defined as:

$$\theta_t = e^{(-dist^2/2\times SOMradius^2)} \tag{4.6}$$

4.3.3.3 Querying the SOM

Each training vector is passed through to the SOM and the weights of all the nodes in the lattice are representative of the topological relationships present in the entire training set.
This allows a new feature vector set from an observed speaker to be passed through to the SOM and the BMU can be queried for each vector. Of course, this will give no indication of how well the observed feature vector fits as a typical member within a set but it will provide a means of determining the closest identity to the observed vector.
A further requirement is that the SOM be continually retrained using retained feature vector sets of all previously observed identities (from both training and recorded samples). For each identity, there will then be created a new disperse group of 2 dimensional pts which can then be compared against the similarly calculated equivalent of the input vector. Some kind of nearest neighbour comparison could then be made and the winning candidate from the identities in the knowledge base is that with the lowest distortion measurement.

4.3.3.4 Practical Considerations

The unsupervised LBG algorithms requires less overhead and less dependence upon a continual renewal of the knowledge base. However, the supervised nature of the SOM demands a retained and ever-widening knowledge base as new feature set observations are presented.
It will be necessary to manage a feature vector "repository" which is used to retrain the system with each iteration.
The continual retraining of the SOM would certainly be a time-consuming activity but nonetheless unavoidable if the map is to be in any way effective. However, any inconveniences introduced by complexity of retraining are offset by the fact that the application of the system is infrequent and allows an ample period in between use where re-training could be easily implemented.

4.4 Categorisation

Comparisons and distortion measures will lead to the creation of a similarity score which should give some idea of how well the observed data fits known identities. The aim of this section is to describe how the ultimate SPAM or NON-SPAM designation is made using these results.

4.4.1 Nature of the comparison

The similarity score is given meaning by measuring it against some kind of threshold. The threshold setting ultimately determines the designation made. In regard to the comparisons there are two modes of operation [32, 5]:

Closed Set : the collected features from a new observation would be compared against all the previously observed examples, and the one with the highest matching score is chosen as the identity. The newly observed sample would be expected to be contained within the group of known samples. As the size of the known group grows, so does the difficulty of finding an exact match.

Open Set : the observed speaker might not be contained within the known group. In this case, it is not so much a matter of finding the closest matching example, it more a case of deciding on if the observed speaker is "known" with reference to the others as a whole.

With inspiration from the standard email model a closed set White-list comparison was used in tandem with an open set Black-list comparison. The latter would first determine the close-ness of an observed identity to a SPAM corpus treated as a single block entity. The former would then determine candidates for known identities from the observation data - one assumes there ought to be a strong favourite amongst the candidates if the observed speaker is in fact a member.
For the White-list the individual average distortion measurements could be normalised since the comparison isn't checking for membership of the group, instead its trying to find a possible identity within the group. If the highest normalised similarity score rises above a set threshold, then the designation of the observed speakers can be made with reference to the highest scoring model. In other words, the comparison against the whitelist would serve to discover if there is a succinct, clear and well matching known candidate which stands out from the others.
Two conditions could then be said to govern the overall SPAM designation:

1. The observed identity is sufficiently close to the blacklist set - (the Black-list condition)

   OR

2. there is a known whitelist entry which is sufficiently close to the observed sample in relation to it peers, and the sample is sufficiently far from the blacklist group (the White-list condition).

If neither of the above are fulfilled then one assumes that the analysis has been insufficient for a conclusion to be reached. In effect, such samples would fall under a "grey list" category - one simply cant categorise the data.
Some kind of effective strategy to deal with these types of situations has still to be formulated. Some possible solutions are listed below, though any kind of automatic designation using these techniques would be left to the user to choose as their own preferred supplementary option, rather than including them as part of the main decision making process - this is to acknowledge the premise that the "recorded characteristics" used in these methods are not as reliable as the feature vectors one gains from MFCC/LPCC/PLP methods.

1)Do nothing to a "grey list" observation. Leave it up to the users to listen to the recording and determine its validity themselves.
2)Employ some kind of neural net extension to the Whitelist, which includes as inputs:

AQBNE/QBNE values
Average pitch
Pitch standard Deviation.

The above would be calculated for each observed sample which makes it onto the White-list. A nearest-neighbour calculation could then give some kind of clue as to the validity of the caller.
3)Automatically default grey list observations to being blacklisted or whitelisted, depending on how paranoid the system is to be.

Figure 4.13 presents an overview of the chosen designation process.

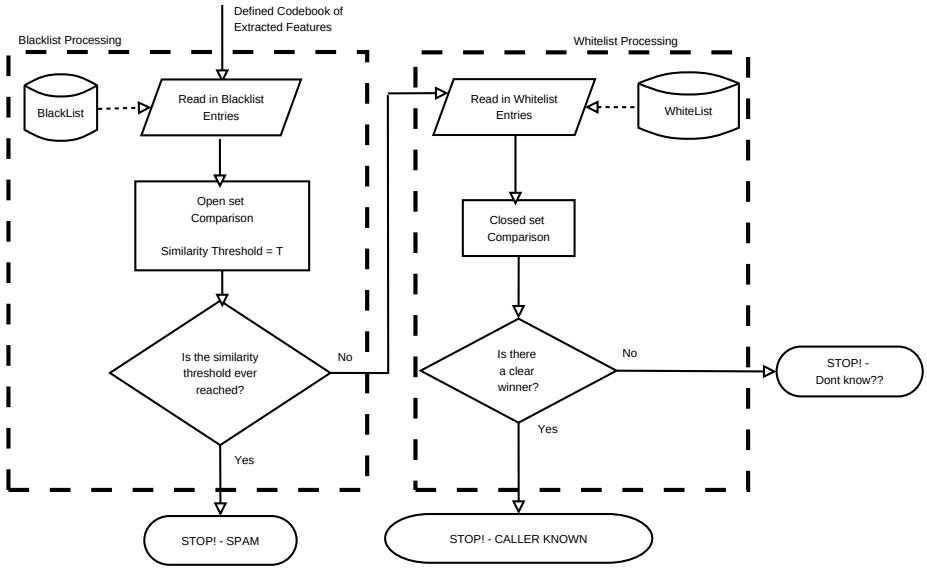

Figure 4.13: Proposed Whitelist/Blacklist Approach

4.4.2 Consequences of designation

The continual success of such a system is dependant on its ability to update itself. Machine learning implies that the system will not merely rest on the laurels of the training data alone - but will continue to hone its knowledge based on continuing samples of real life data.
Where an observation sample has been designated "white" or "black" it would be of much use to retain the data and add to the systems knowledge base.
Therefore, if the system has decided that the incoming caller is SPAM, the user should be at least prompted for the addition of a further codebook to the blacklisted models, based on the extracted feature vectors. In the same way, if an incoming caller is deemed to match a model on the whitelist, the user should be at least prompted to add a further calculated codebook of the caller to the already existing codebooks in the whitelist.
The term "at least prompted" is used because the system should offer the user the final say in the matter. Although the system should be capable of automatic designation and thus update, in the early stages of development, it would be helpful to add the functionality whereby SPAM messages can be heard back to the user before they submit any models for future determinations.

There are then three types of data which can be retained and their re-use is summarised in **figure 4.14** and **figure 4.15**. The former explains what should

happen if the observed utterance matches a blacklist model (ie, it is within a threshold range of the open set). The latter explains the procedure when the observed utterance produces no match in the Blacklist but strongly matches one of the known speakers in the Whitelist.

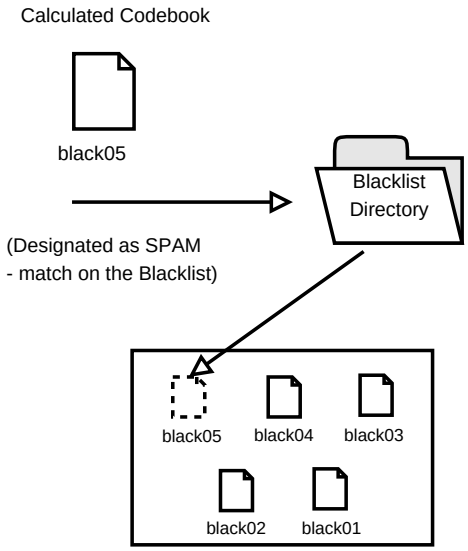

Figure 4.14: Consequences of a Blacklist Match

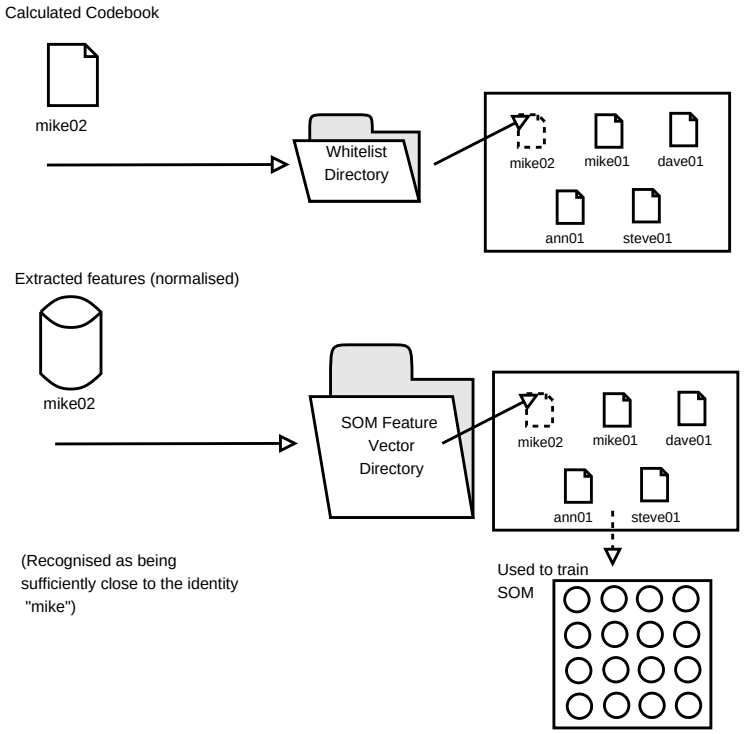

Figure 4.15: Consequences of a Whitelist Match

Chapter 5

Implementation Details

5.1 Introduction

The basic practical foundation upon which the project was built will now be discussed. For a more involved description of the transition from the aforementioned design to the actual working implementation, Appendix ... may be consulted. The system was built as self contained "product", an interactive solution which allowed for model training and testing against real-time voice captures and procured test sample audio. The **TkSnack** interface [34] was chosen in conjunction with the Python programming language (www.python.org). After brief descriptions of their involvement, this section will conclude with a presentation of the graphical interface, designed to show the system at work and how it may be trained and configured.

5.2 Foundational technologies

5.2.1 TkSnack

Initial trials of the discussed formulae may be easily implemented in MATLAB (www.mathworks.com). There exist many libraries for this which can read and write WAV files, perform Fourier transforms, filterbank scale conversions, and give the ability to view the results of such functions graphically [3]. (Whilst the viewing of graphical interpretations of the captured speech data and its manipulations is not critical to the actual working of the system, it gives a most valuable visual verification that the data concerned is meaningful.)
In a signal processing / laboratory environment this is sufficient, however practically it is not especially helpful when one prefers an actual system which can prompt for a response, record the response, perform an analysis on the response and give a conclusion all automatically. This implies that one needs a handle on the recording and playback of sound samples, together with instantaneous access to the data within. It also means that the timing of when different processes get activated is in accordance with the desired intervals of speech playback and speech acquisition. (see section 2.4.2.3)

Kåre Sjölander from the department of speech, music and hearing at the Royal Institute of Technology in Sweden has developed an interface to the Python/Tkinter programming languages, which fortunately solves this issues in a most helpful way. His "Snack Sound Toolkit" provides extensions to the above scripting languages which ultimately would allow the creation of a standalone Secretary system - ie something which could be called from the command line and executed. The Snack Toolkit, (or "**TkSnack**") is cross platform, has commands for basic sound handling (recording, playback, file IO, socket IO) and also provides visualisation functions (waveforms, spectrograms) which can be used on any digital sound file. It is also completely open source and one existing online example of its use is the capture of streaming audio from the internet. In general the main advantages and reasons for choosing this as a foundation to this thesis project are as follows:

- Sound procurement: **TkSnack** allows for the capture of sound in many formats, encodings, and bitrates. (Very useful in replicating real-life speech channel qualities).
  It is also possible to capture audio directly from a known sound source (ie sound card output) in a computer system. Or it is even possible to take sound data directly from a socket connection.

- Sound representation: **TkSnack** is able to maintain its own representation

of the captured sound in memory by using its Sound() object. Sound() objects can be created from known files resident on the system, or may be created as empty and subsequently used as container to hold whatever may be captured via a connected source. Their contents may be subsequently written to files on the system.

- Sound manipulation: **TkSnack** Sound() objects may be queried and manipulated themselves. It is possible to extract ranges of values from the Sound() object together with the all important power spectrum directly. Pitch values may also be obtained in a similar way.

- Sound visualisation: Given any Sound() object it is possible to view a graphical representation of its waveform and dBPower spectrum on top of a TKinter canvas window.

- Interface embedding: Because **TkSnack** is effectively an extension of both python and Tcl/Tk, the latter may be used in conjunction with it to not only display evidences of how the system is performing its calculations, but also to construct a user interface around the main working processing. This yields a more real life solution and more of a prototype "feel" about the whole thing.

- Extensible Assurance: By using **TkSnack** in conjunction with both Tk and Python, one therefore has access to all of the provided resources inherent to these scripting languages. Not only that, but it should allow the final solution or at least parts of it to be embedded itself within other structures and projects (for example a Python software driven IP telephone)

Considering these given attributes it was decided that **TkSnack** provided the best available entrance point to the implementation of the project.

5.2.2 Python

The fact that **TkSnack** is essentially provided as an extension to Python and that the author had previous experience with it as a scripting language made its choice an inevitability. Python whilst offering extensive standard libraries further provides threading and object orientated functionality making it an attractive and widespread language in the realm of software development. It also offers strong support for integration with other programming languages and platforms.

5.3 Programming the Interface

TkSnack could be developed in conjunction with both python and tcl/tk. Aside from the obvious benefits of the audio processing toolkit, there existed an extensive graphical toolkit for the display of various functions and operations together with the basic primitives for building a windows-style interface. The development of this interface for the project is therefore not mission critical so to speak, in that the basic building blocks of the timing, feature extraction and decision making are altered. However, the use of such graphics helps to augment the system by:

- providing a tangible, visual demonstration of how it should work overall
- illustrating the interaction between the calling and called party and prompting the latter appropriately.
- providing a means of visual verification, that the data is actually being captured in a sensible way. The mathematics used in this project, when considered as a whole paints a very complicated picture and is virtually impossible to make sense of or even validate, given merely the numbers themselves.

5.3.1 General Operation

Running the *Secretary.py* file commences the system which immediately begins the challenge to the would-be caller. The corresponding interface window is shown in **figure 5.1**. Interestingly the flashing of the secretary's eyes in the main interface window indicates that sound data is actually being recorded, even during the greeting. This continues until the greeting is finished and the response listening period begins (**figure 5.2**). The eyes remain flashing until the response gathering time window terminates (incidentally, users may alter this as they wish). The system then performs its feature extraction and decision making procedures as it enters a thinking state (**figure 5.3**). Once it has come to a decision there are two scenarios which may present themselves.

1) The secretary has determined that the captured sample sound is SPAM.
2) The captured sample audio does not pertain to SPAM and in fact a suggestion for who it may be is given.
In both cases, the main secretary window expands to present two further options to the user:

Figure 5.1: Greeting/Challenge Window

Figure 5.2: Caller Response Capture Window

Figure 5.3: Data Processing Window

- Submit Model : This commences the process of actually writing the codebook (which the secretary has just extracted and used in comparisons) to a new model file in the appropriate model directory. User parameters (in particular the feature extraction method used) determine the exact directory but not before the user has been prompted further to ascertain whether the new model should be added to the Whitelist or Blacklist and what identity it should be given (shown in **figure 5.4**). In the above it

Figure 5.4: Submitting a new speaker model

can be seen clearly that the user has 3 different naming options:

1. the suggestion which the system has come up with after the comparisons
2. an already existing model (listed) (the user may wish to update a

certain identity).

3. a new identity, the naming of which is offered to the user.

- Show the Graphics of the captured Sound. (see next section - "Graphics and Visualisation")
- Close the program.

If the captured sound data has been determined as SPAM a popup alert appears immediately offering the user more options (**figure 5.5**).

Figure 5.5: Spam Prompt

- Another Submit Model Query. In this case, if the user chooses "Yes" then the extracted codebook model is added to the designated blacklist directory (which accords to the feature extraction method used).
- Playback all of the captured sound from the calling party. This allows a further user verification step.

5.3.2 Graphics and Visualisation

Displaying the findings of the feature extraction and decision making is not aided by the graphical display. In fact, it plays no part in any of the real processing of the system. It just regurgitates what has been extracted from the initial sound objects in a more tangible and human readable way.
The captured sound information can be displayed on a larger canvas window upon the activation of the "Show Graphics" button from the main secretary interface:

DIAGRAM TO BE INSERTED HERE

The graphics window contains the following:

- A Spectrogram of the period of the caller's speech during the greeting phase.
- A Spectrogram of the period of the caller's speech during the response phase.
- A plot of the power spectrum of a certain window (including any filtered power spectrums carried out).
- A plot of the dB power spectrum of a certain window.
- A plot of the normalised coefficients extracted from the window
- Statistical Information about the window concerned and the overall captured response.
- Buttons to show the noisy frames of the response capture.

By including both a spectrogram for the vocal activity of the calling party both during the greeting and response phases it is possible to visually observe just how much (or how little) the calling party says.
The buttons at the bottom of the screen are useful when one wishes to see which parts of the response speech are being included in the normalisation and subsequent codeword creation (since noisy frames are effectively discarded from this process). Windows estimated as noisy are effectively blanked out from the captured response spectrogram, on the display. This also provides the user with some kind of visual feedback on how well the noise estimation settings are working.
The caller response spectrogram also has an event handler attached to it whereby a mouse click results in the x coordinate being used to determine a window from the entire sound.(the x coordinate is effectively a fraction of the spectrogram width. The window from the collection of frames corresponding to this x position in the graph is then set as the window of interest). In other words, the user clicks on the spectrogram and the corresponding window in the sequence lights up and immediately the power-spectrums and extracted coefficients are displayed on separate regions of the canvas. A section of the statistical information display also updates with respect to the selected window.
This is only achieved by keeping track of all the frames created with the windowing process, and also their respective filtered equivalents, gleaned coefficients and statistics. This is all merely a case of overall record-keeping during the feature extraction process. Once feature extraction is performed then, all these collections of windowed data may be sent straight to the graphics functions for

this very purpose.
The same is true for the noise display upon the spectrogram. The noise mask array from section 3.5.2 is calculated in advance and then is passed on to the implementation of the graphics.

5.3.3 Whitelist/Blacklist Management

When speaker models are compared against existing models from the systems knowledge base, it is imperative that the two parties in any comparison were created using the same parameter settings. Different settings may be used in the calculation of feature vectors, different methods may be used to calculate feature vectors. Different numbers of codewords may be used to represent the clusters of a model. Different settings may be used to estimate noise. Therefore, there must be consistency across the entire range of speaker models under consideration. The process by which the models were arrived at must be identical if they are to be compared in a fair manner.
An aim of the project in general was to see how different feature extraction techniques might fare in comparison to one another. Therefore, the only variable which was allowed to change with the speaker modeling was the actual feature extraction method itself. Of course, a model created from the PLP coefficients of a certain sound sample is going to be different from an MFCC model created from the same source. Therefore, provision must be made to store and handle models according to their feature extraction technique. It would be beneficial that for a given captured sound sample, models are created independently using the three feature extraction techniques.
It then follows that when a captured sound sample is analysed, the chosen feature extraction technique should be kept in mind when deciding with which portion of the knowledge base to perform comparisons - ie only those existing models created in the same fashion.
It was decided that models be stored in a simple directory structure according to how the model was fabricated and also whether the model itself is a Whitelist or Blacklist model. The chosen structure is best described in **figure 5.6**.

5.3.3.1 Codebook Model structure

Each model is stored as a basic file. With the Whitelist models the files are merely named after the identity of one who's voice has been recorded. More than one model may be created of the same person. This is totally acceptable so long as the name of the model filename is unique. In such a case, multiple models of the same person could have numbers appended to their titles, for

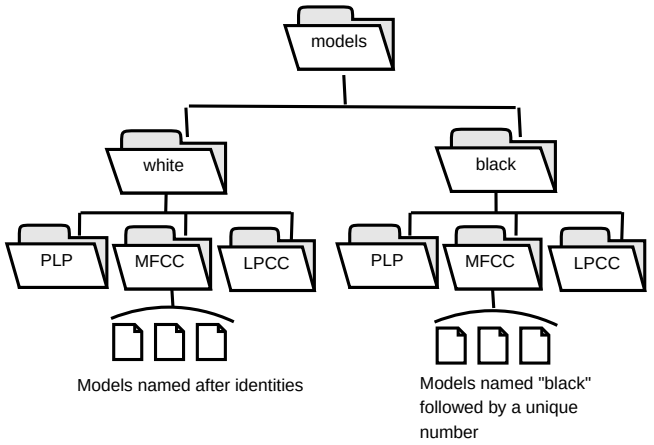

Figure 5.6: Blacklist/Whitelist directory structure

example, dave1, dave2, etc....
The Blacklist by its very nature is an open set problem - the identity of the nearest model is not important; what is important is the measure of how well the sampled speaker fits into the group as a whole. Therefore the naming of the Blacklist models is an automated process, done entirely arbitrarily.
The structure of any model however is entirely consistent across the whole system. Any model will consist of K codewords each of length N (where N is the size of an extracted feature vector). Within the model files, each individual feature vector value is a floating point number separated from its neighbours by a new line and each codeword is separated by a single "." character on a new line. The end of file marker is simply "END" on a new line. This simplifies the task of reading from and writing to model files. This structure is exemplified in **figure 5.7**. This all means that when the secretary system needs to become aware of existing models, it merely needs to go to the appropriate model directory based on the chosen feature extraction method and whether it is the Whitelist or Blacklist. It then can read from these model files, populating its own codebooks by reading from the model files in accordance with their given structures.

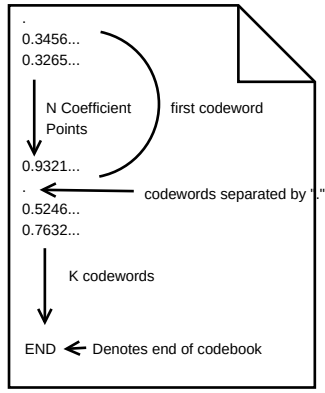

Figure 5.7: Example Codebook Model File

Chapter 6

Testing

6.1 Introduction

The system would only give out useful responses if was trained with the voices of people. A small number of trained samples offered the advantage of a low time cost in terms of training, and for those who's identities were already trained to the system, correct designation would be probable given the relatively high inter-speaker variation present in smaller sets. A large training group would take longer to train but would afford the system greater selectiveness in terms of the number of people it could recognise - even though the inter-speaker variation reduces as the training group enlargens. The final number was also partly determined by availability and willingness of individuals to take part. Initially 21 people agreed to offer their voices as sample data and it was generally though that this would provide a basis from which reasonable deductions could be made. In terms of the training data itself, the 21 identities said their own name 10 times. The microphone on a Dell Axim X50 Pocket PC was used to record each individual. (The author frequently uses this hand-held device with the VOIP application Skype - with good results due to its quietness and lack of hard disk noise). This allowed the convenient procurement of voice samples in WAV form and afforded portability which enabled as many as 21 voices to be captured.
Once the voices were obtained, each was resampled to 8Khz, the target format of the system. Extracting each utterance for each voice then allowed speaker

models to be created using the methods discussed in chapters 3 and 4.
The secretary system provided its own interface to train new models into the system. The user could train more than one WAV sample at once as figure 6.1 demonstrates. The user inserts each WAV sample to be trained into the "*to_train*" folder and the trainModel.py class is called which provides the interface. The fact that the speakers name was chosen as the text was because

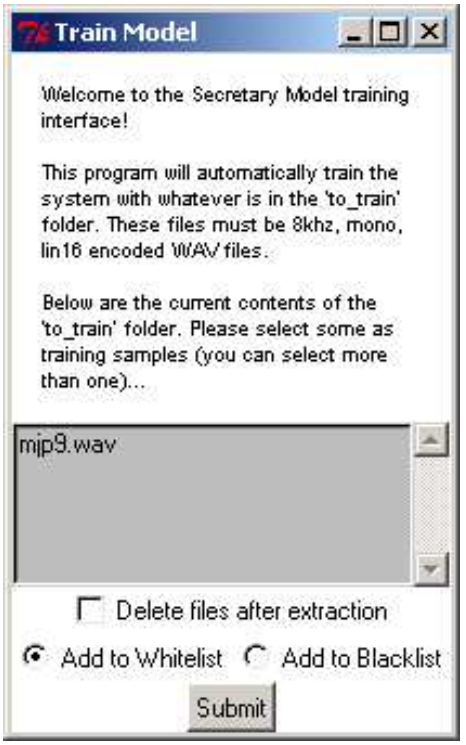

Figure 6.1: The Training Window

it was uncomplicated and offered something reasonably individualistic for each speaker to say. Although the system should support text-in-dependency, the quoting of one's own name provided an inherent individualism.

6.2 Test 1 - Optimal Configuration

More than one type of model was created for each voice utterance using the different feature extraction methods in conjunction with different codebook lengths. PLP, Mel Frequency and Linear Predictive coefficients were extracted from each utterance and then moulded into models with codebook lengths of 10, 15 and 20. This gave 9 speaker model permutations.
The first set of tests was design to ascertain the strength of each of these 9 configurations in regard to merely identifying a speaker from the system. Those permutations which performed badly in this way could be discarded thus saving time for future testing with the effective configurations.
Each of the 21 identities was tested by selecting at random 2 models from each, removing those from the systems knowledge base and then observing the systems ability to identify the speaker based on the remaining models. Each time a test was run, the system would make a decision as to the identity of the sample based on the AQD / similarity scores described in Section 4.2. Repeating this for each Feature Extraction/Codebook Length configuration provided the results found in figures C.1 to C.9 in Appendix C. A summary of these results is shown in figure 6.2. It was clear to see that the LPCC feature extrac-

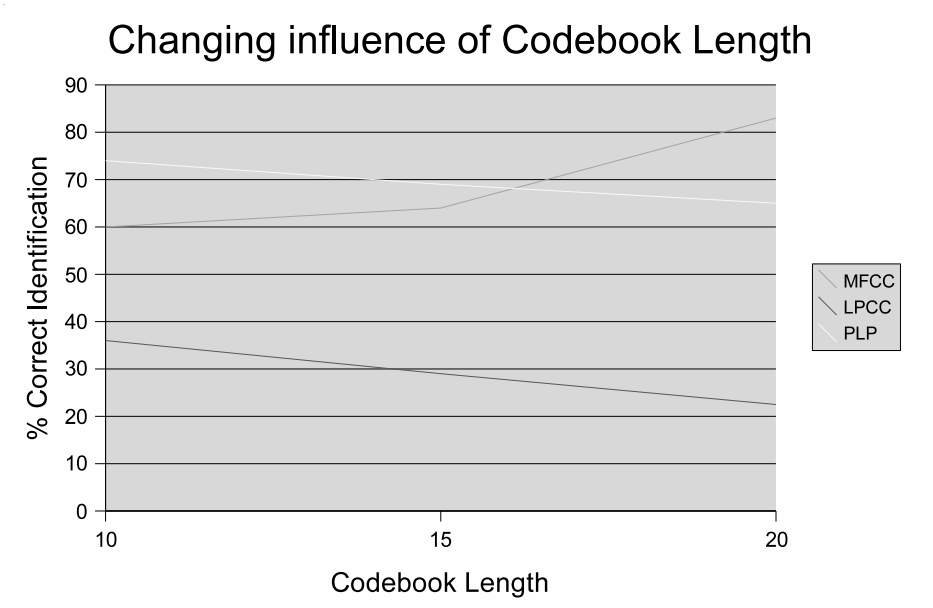

Figure 6.2: Summary of different codebook lengths

tion was largely ineffective regardless of whatever codebook length was chosen.

Both MFCC and PLP performed better with MFCC improving as the codebook length increased. Changes in codebook length were less influential with the PLP coefficients. Therefore, for the remaining tests, LPCC was discarded as a feature extraction method. It was decided that the MFCC and PLP methods should be retained with codebook lengths of 20 and 10 respectively.

Furthermore, by analysing these two configurations in greater detail it was possible to obtain a picture of how well each performed in regard to correct vs. incorrect designations. By creating a histogram of the performance of each (figures 6.3 and 6.3), it was possible to make an estimation of good threshold values for later tests. The MFCC configuration would appear to provide a more clear

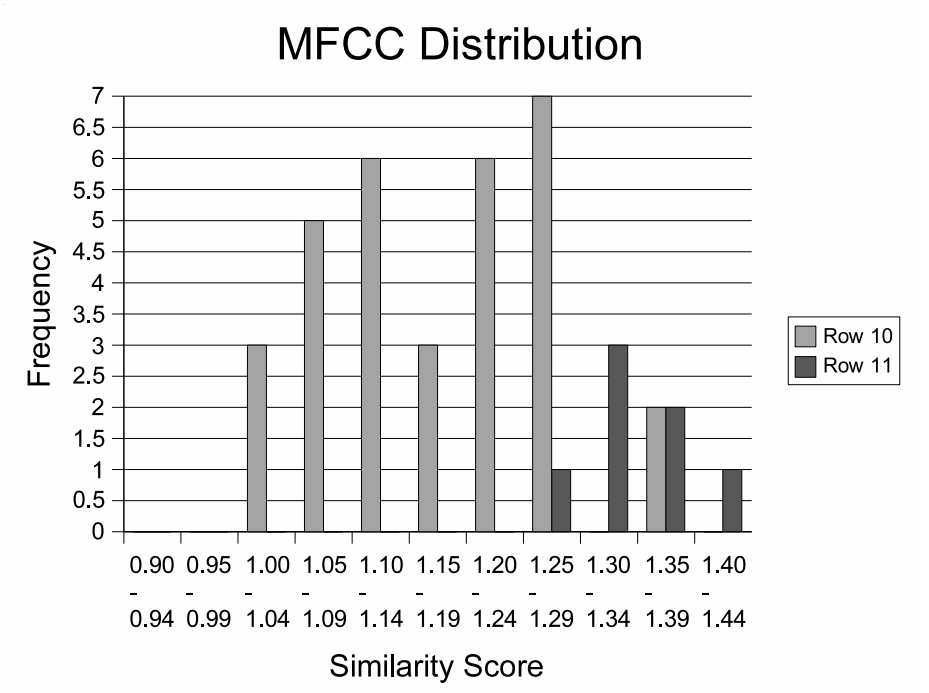

Figure 6.3: MFCC performance with 20 codewords

cut scenario when determining a reliable threshold for designations. From figure 6.3 threshold values of 1.23 (standard value) and 1.3 (pessimistic) were chosen for future tests.
The PLP configuration was less clear cut in this respect, and threshold values of 0.95 (standard) and 1.05 (pessimistic) were settled upon, though with less confidence.

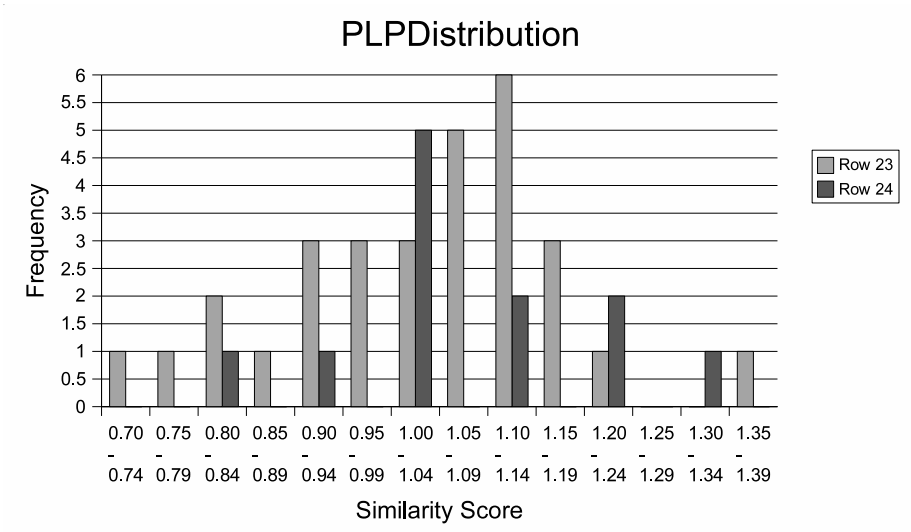

Figure 6.4: PLP performance with 10 codewords

6.3 Test 2 - Blacklist/Whitelist Trials

In order to mirror the intended whitelist / blacklist operation detailed in section 4.4 the stored identities were then split arbitrarily into two groups. Half of the identities were designated as the blacklist contents whilst the remainder were placed on a whitelist. The system was tested again, using the two retained configurations and by using the model removal principle employed previously. The system was monitored for its ability to designate utterances based on its now split knowledge base. The results of this may be viewed in figures C.10 and C.11 in Appendix C and are summarized in the following tables. The aforementioned standard and pessimistic threshold values were used in each instance.

STANDARD THRESHOLD	(PLP / 10)
Percentage	**Description**
4.55	*Incorrectly Blacklisted*
18.18	*Incorrectly Whitelisted*
9.09	*Grey*
68.18	*Correct*

PESSIMISTIC THRESHOLD	(PLP / 10)
Percentage	**Description**
18.18	*Incorrectly Blacklisted*
18.18	*Incorrectly Whitelisted*
9.09	*Grey*
54.55	*Correct*

STANDARD THRESHOLD	(MFCC / 20)
Percentage	**Description**
22.73	*Incorrectly Blacklisted*
4.55	*Incorrectly Whitelisted*
13.64	*Grey*
59.09	*Correct*

PESSIMISTIC THRESHOLD	(MFCC / 20)
Percentage	**Description**
77.27	*Incorrectly Blacklisted*
4.55	*Incorrectly Whitelisted*
9.09	*Grey*
9.09	*Correct*

6.4 Test 3 - Spoofing

In a bid to see how well the system performed purely in terms of its resilience to text-in-dependency, 3 of the blacklisted speakers were recorded saying each of the names of the other whitelisted participants, twice. The MFCC/20 codeword and PLP/10 codeword configurations were retained as before. The standard threshold values were used, after the experiences of Test 2. The results (figures C.12 to C.17, Appendix C) are summarized in figure 6.5.

6.5 Test 4 - Unknown Speakers

The previous tests demonstrate how the system copes with identities which are known to it. However, they demonstrate nothing in relation to how the system reacts to new identities with whom no contact has been made in the past. In an

Identity	Method	Blacklisted Correctly	Blacklist identity correct
Ruben	PLP/10	75.00%	5.00%
	MFCC/20	85.00%	65.00%
Michael	PLP/10	40.00%	65.00%
	MFCC/20	45.00%	70.00%
Esther	PLP/10	10.00%	75.00%
	MFCC/20	85.00%	100.00%

Figure 6.5: Spoofing results

effort to learn more in this regard, 3 new identities were presented to the system. Each identity was then tested 4 times with each test run resulting in Blacklist and Whitelist similarity score and corresponding suggested identities. Since the standard threshold values were retained, each test also meant that each new speaker was given a designation. Figure 6.6 shows the collected results.

6.6 Test Case limitations

In order to obtain as many training samples as was possible, the actual recording could not occur under laboratory conditions. Often, significant background noise was present depending on where the sample was taken and it was notable during the testing that those samples which contained less noise, were much more easily identified on the basis of their peers.
Unfortunately some of the training identities whose voice samples were recorded under noisier conditions, simply did not train. The implemented clustering methods appeared to become stuck in an infinite loop. It was assumed that this was due to the noise in the sample which masked the extracted feature vectors to the extent that it was not possible to create 10, 15 or 20 cluster centroids. Certainly the problem became more prevalent with the noisier samples as the codebook length increased. The cleaner identity samples did not have this problem in training.
All of the training and test samples used in the project were recorded using the same microphone setup. Had time allowed, an examination of the systems performance using different recording equipment would have been a worthwhile pursuit.
But generally, the test scheme described in section 6.2 seemed to be the only practical one, given that voice samples are much more easily captured than human subjects (in person!).

Unknown Speakers	MFCC / 20				
	BL Person	BL Score	WL Person	WL Score	Identified as?
Sevan02	Ruben	1.2	Christian	1.24	Black
Sevan03	Michael	1.2	Carol	1.36	Black
Sevan04	Trina	1.19	Christian	1.18	Black
Sevan05	Ruben	1.18	Christian	1.15	Black
Martin01	Lisa	1.2	Carol	1.14	Black
Martin02	Lisa	1.18	Carol	1.08	Black
Martin03	Lisa	1.19	Carol	1.18	Black
Martin04	Michael	1.13	Carol	1.06	Black
Navar01	Ruben	1.2	Christian	1.19	Black
Navar02	Ruben	1.32	Anne	1.35	White
Navar03	Ruben	1.24	Christian	1.29	White
Navar04	Michael	1.13	Carol	1.2	Black

Unknown Speakers	PLP / 10				
	BL Person	BL Score	WL Person	WL Score	Identified as?
Sevan02	Michael	0.93	Christian	1	Black
Sevan03	Michael	1.01	Christian	1.09	White
Sevan04	Michael	1.01	Christian	1	White
Sevan05	Ruben	1.02	Christian	1.04	White
Martin01	Mareta	1.05	Anne	1.03	White
Martin02	Michael	1	Anne	1	Grey
Martin03	Mareta	1.02	Christian	0.94	White
Martin04	Michael	0.92	Christian	0.93	White
Navar01	Michael	1.09	Christian	1.04	White
Navar02	Michael	1.06	Christian	1.07	White
Navar03	Michael	1.04	Colin	1.08	White
Navar04	Michael	0.86	Christian	0.91	Black

Figure 6.6: Test Results - Tests with unknown identities

6.7 Test Conclusions

Test 1 demonstrated that with low bandwidth audio samples, MFCCs and PLP Coefficients are reasonably effective at verifying the identity of a known speaker. LPCCs grossly underperformed. Increasing the codebook length was beneficial when using MFCCs but marginally detrimental with PLP coefficients.
Test 2 showed that MFCCs had the capacity of performing as well as PLP Coefficients when a Blacklist / Whitelist scenario was created. However, the former is much more sensitive to threshold settings - when the more pessimistic setting was used around half of all samples became re-designated from being correct to blacklisted. The PLP method appeared to be more stable in this regard and maintained a correct designation success rate of 54-68% (whereas MFCCs at best was 59%).
Test 3 demonstrated one advantage of using MFCCs over PLP, namely that it was less easily fooled by spoofing callers. Several additional interesting things came out of **test 3**. Upon closer inspection of the results in in Appendix C, it is apparent that each time the blacklisted identity spoke a different name, the closest name on the whitelist remained fairly constant. Even though it is of course incorrect, there was some consistency. In addition, from all the names on the blacklist, more often than not, the correct once was chosen as the closest from that set. The value of placing the blacklist check before the whitelist check (see section **??**) was demonstrated also. Often the whitelist score was qualitatively better than the blacklist score, even though the blacklist score actually referred to the test model.
The results of **Test 4** were largely inconclusive. Really the only conclusion was that the methods used in this project, given the context could not procure models which had enough inter-variability from which accurate designations could be reached.

6.7.1 General Conclusions

The importance of obtaining clean training samples could not be underestimated. The ease of which these could be identified in relation to the more noisy samples emphasised this. It gave cause for satisfaction that the feature extraction mechanisms on their own could reduce the voice data into an effective and compressed acoustical representation. The success ratings of MFCC and PLP in relation to speaker identification in this project show that they are viable feature extraction options for lower bandwidth applications like VOIP telephony.
The results suggest that it is possible to construct such a voice spam filter, akin to a normal email filter. However, on the basis of the tested system, the benefits are limited and by no means guarantee as effective an operation as the email

equivalent. The feature extraction techniques can produce feature sets which are robust to an extent but are still prone to mis-designation of both SPAM and non-SPAM identities, and have great difficulties when unknown identities present themselves.
It would appear that the MFCC / 20 codeword combination was better employed in Blacklist identification - it performed better at closed-set threshold distance based designations. The PLP / 10 codeword configuration on the other hand appeared more effective at open-set comparisons between known identities. In each of these, the setting of threshold values is highly influential and should be reviewed as any such system adds to its knowledge base. Some consideration could be given towards decision making based on aggregated scores of a group of most similar identities.

Whilst not directly examined through the other test initiatives, it was noticed that when the speaker model under consideration *was* included in the knowledge base, it nearly always produced the correct identification from the system and in a most unambiguous way - the resultant similarity score by doing this was always far out on its own in terms of closeness. This apparent "resonance" when an exact known sample is observed provides hope in countering more intelligent spammers who might try to artificially create responses to the challenge this system gives out. If a spammer sends out crafted responses from a database of his own examples - they could probably be easily detected if the secretary system has heard them before.
But herein lies the problem for the secretary model - that is the natural variability of credible, cognisant callers. The tests run here show that whilst an individual may be easily recognised in relation to others, difficulties occur when the number of speakers in the knowledge base becomes larger. There are also problems when unknown speakers try to pass through the system. Much therefore depends on the quality and selectiveness of the training phase itself. Admittedly, the methods for designation were limited to rather coarse thresholding but their implementation was all that was possible in the time provided. Future improvements could undoubtedly include methods to weight observed identities in relation to their similarity to both whitelisted and blacklisted models (like those used in the Bayesian filtering of email).
Possible use of pitch values in tandem with the other techniques may enhance the system. Such information was available through the TkSnack interface but was not used due to time restraints. They would certainly aid situations where a male speaker is mis-identified as female or vice versa.

Chapter 7

Review and Conclusions

7.1 Overall Conclusions

It was felt that the experiments performed were fraught with subjectivity. The testing part of the project was probably of more use in seeing the objective merits of the feature extraction techniques in relation to each other. It was a disappointment that the Self Organising Maps could not be incorporated into the system, as it would have provided an interesting counter procedure to the Vector Quantization approach of Pattern Recognition.
The full implementation of the Weiner Filter as a counter measure to noise was also not possible due to time. In the future this could be used in both the training procedure to improve noisy samples or just prior to the feature extraction stage to ensure a higher quality spectral representation is available.
Somewhat bewildering was the sheer scope for which parameters could be tweaked and tuned, the author really felt forced to just choose arbitrarily on a number of occasions. The inherent variability of voice data, recording conditions and recording equipment further added to the subjectivity.
The choosing of test scenarios was also limited to a certain context and could not possibly cope with the multitude of likely scenarios which could be presented to such a system in practice.

On the positive side, it was clear that the implemented techniques functioned

adequately as a whole. They didn't provide much evidence that a robust solution to VOIP spam was in the offing, at least not in the same way that email filters function today. They did however show that some intelligent verification of known entities was possible. However, great care was needed in relation to the setting of threshold values. This is again highly dependent upon the nature of the model knowledge base.
The challenges which remain were demonstrated well in the testing section. Perhaps the main one is coaxing such a system into a way of operation where it is able to weigh up positive and negative evidence and make a better informed decision. There must be better methods available for categorisation than those used in this project and had time allowed. This would certainly have been a primary area for improvement.

7.2 Personal Reflections

The project as a whole provided a most interesting insight into Voice Recognition and Artificial Intelligence technologies. It certainly built upon the author's experience of signal processing and pattern matching techniques. The biggest challenge the project presented was really the sense that it was being built from the ground up - related material was not especially easily available. Much time as required just to inform oneself on the subject matter so that subsequent experiments were at all possible.
Once the learning material had been found, the need to digest complicated mathematics and then make an informed judgment on the direction of the solution was crucial.

Appendix A

VOIP Context

A.1 VOIP Technology Overview

The contents of this Appendix are a summary of existing VOIP technologies drawn from [25] and [16].

A.1.1 SIP - Session Initiation Protocol

Probably the most simple variant of VOIP is the SIP protocol which is an application layer protocol using basic textual commands. It is possible to run it over both UDP and TCP, however the former is more often used.
SIP endeavours to make use of the following ideals:

- **User Location**: Association of end users with an end point which is identified via technical parameters (eg. IP address)
- **User Availability**: Can the end user be reached and are they willing to participate in a conversation?
- **Endpoint capabilities**: Determination of media types and system functions which can be used with them

- **Session setup**: The prospect of being able to call a remote device and be able to set up a session at both points.
- **Session Management**: Session parameters could be managed, other services invoked, and sessions terminated or transferred.

From an overall Perspective the system consists of a SIP Network server whose job is to perform name resolution. It may consist of the following 3 components:

1. **Proxy Server** - Acting as a kind of firewall with security and authentication capability, the job of the proxy server is to manage SIP requests from an internal network and pass them onto other proxy servers through which internetworked SIP communication can take place.
2. **Redirect Server** - Simply redirects users to help them find the sought after address.
3. **Registrar Server** - Registers users and maintains an internal directory of telephone numbers and other internal addresses mapped to actual IP addresses.

The call setup uses a three way handshake - The caller will send an INVITE request to try and setup a connection. Within the INVITE message there will exist a "To:" field which will identify the callee, usually in the form of an email address. The request arrives at the SIP network server which, through its registrar component will try to resolve the provided identifier to an internal IP address.
The registrar server may not be up to date with its mappings. In which case the INVITE message is passed on to the mapped IP regardless, but the actual recipient is able to reply with a transitive response demanding some form of redirect to the appropriate recipient. This helps the registrar server maintain its mappings and thereafter, the INVITE message is resent to the correct internal recipient. The recipient is then able to send an indirect response via the proxy to the initial caller, signifying that the request for correspondence has been accepted (or rejected). If the invitation has been accepted, then a final confirmation from the caller to the callee is sent. A direct channel is then opened, independent of the proxy and audio communication can flow back and forth within. **Figures A.2, A.3 and A.4** provide an illustration of what is actually happening.

The advantages of SIP over H.323 are in its simplicity and efficiency. The setting up of calls is a less arduous process, therefore it should be easier to pinpoint areas of interest for SPAM detection techniques. Unfortunately because PSTN networks have become more complicated, SIP lacks many of the features

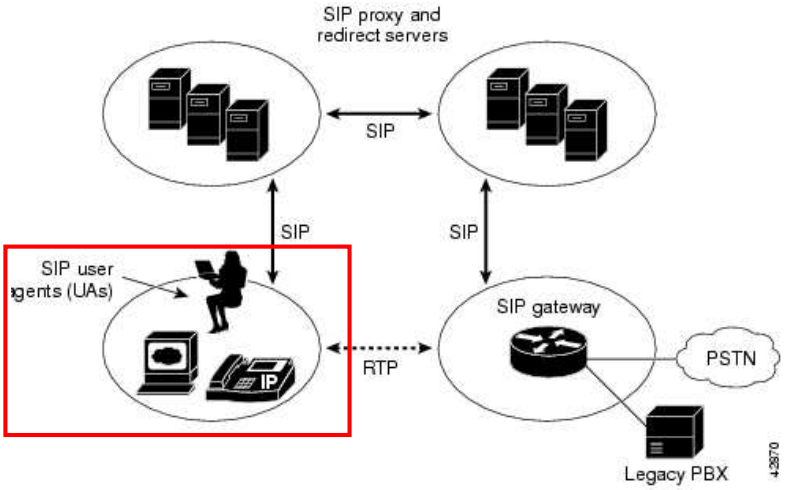

Figure A.1: SIP Proxy Network:Overview

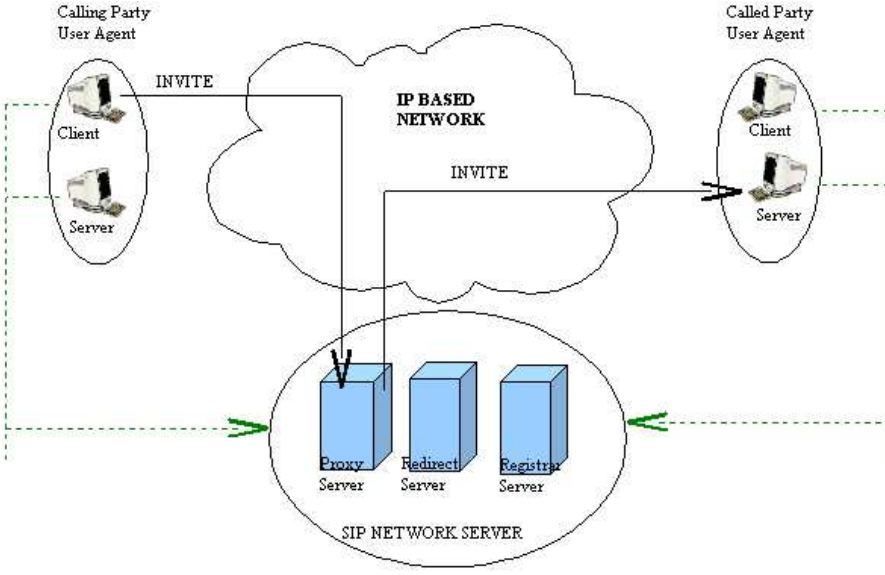

Figure A.2: SIP Network: Calling Party Invite

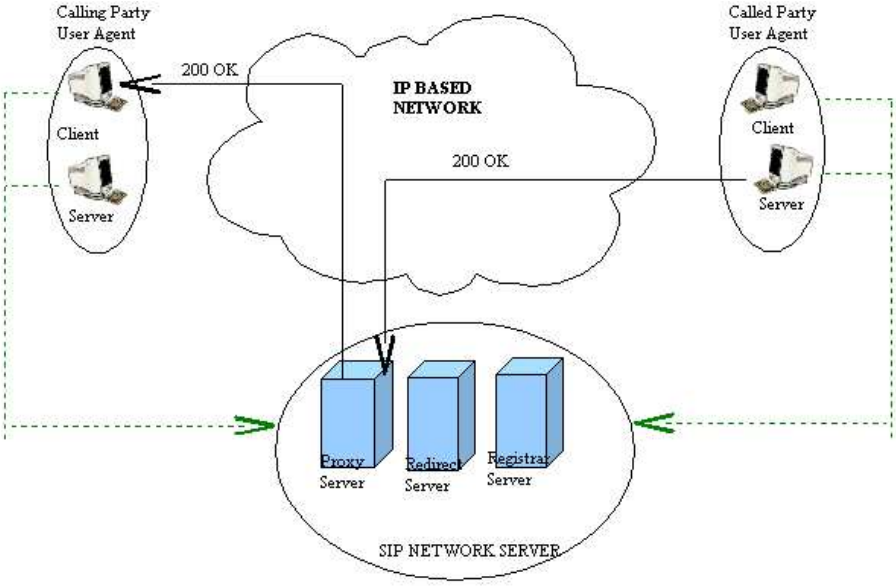

Figure A.3: SIP Network: Called Party Acknowledgement

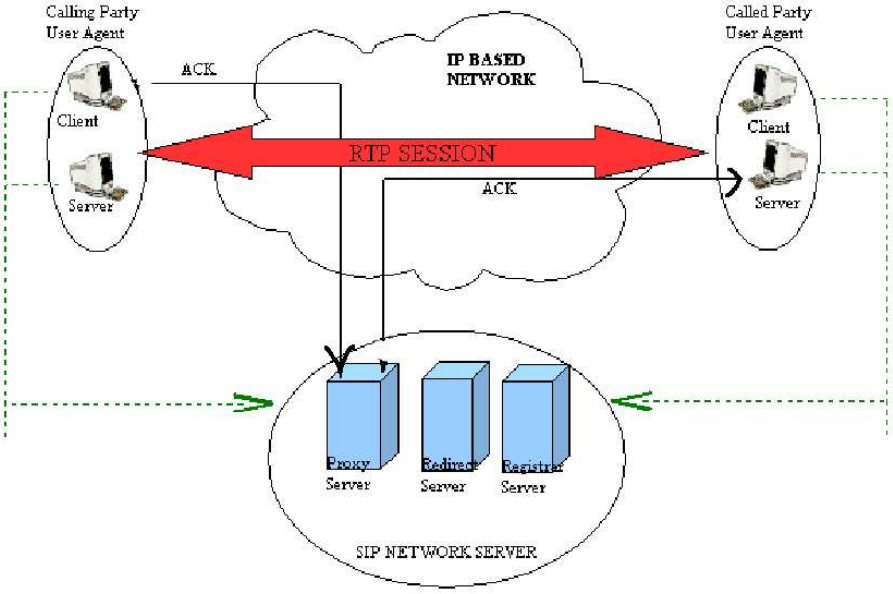

Figure A.4: SIP Network: RTP Protocol Established

needed to incorporate it. Therefore, whilst SIP is becoming more and more popular due to its ease of use, H.323 is much more widespread because it has been more easily adapted for PSTN networks.
Thus SIP is more likely to gain popularity when considering communication between computers.

A.1.1.1 Implementation Details

Endpoints are typically identified by a URL or URI for example

$sip : mike@192.34.53.4$

There are many possible types of SIP addresses but generally they look similar to web addresses, only that the scheme is either "sip:" or "sips:" (a secure variant) followed by a hostname. The hostname can be either in dotted IP format or its DNS name equivalent.
SIP uses a kind of client/server model where the caller becomes the client and callee the server. Communication messages between the two are seen as transactions where the client makes a request from the server and then receives an acknowledgement describing in what way the request was processed.

A basic example of a SIP call is shown in **figure A.5**. The contact between

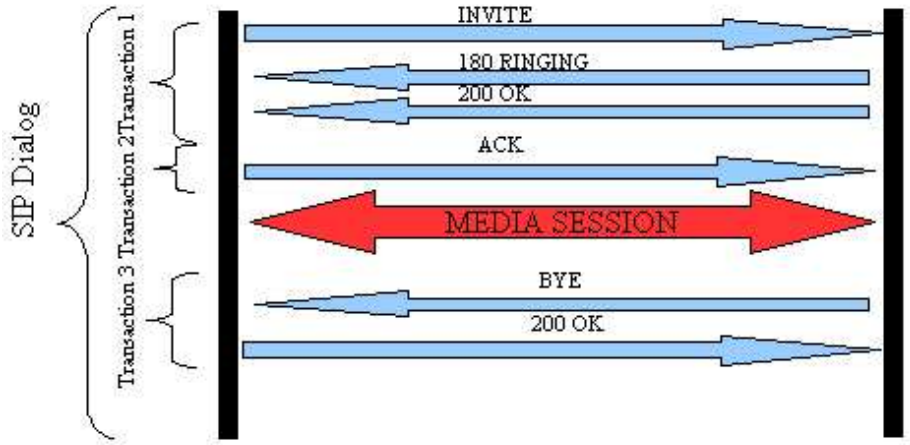

Figure A.5: SIP Dialogue Communication Pattern

the two parties extending over the period of time between a call is requested

and the same call is cleared is called a SIP dialogue. Each dialogue has three Transaction phases.

1. In the first transaction phase, the caller tries to establish a connection with the callee. Once the callee receives the INVITE request a 180 response message is sent back from the callee to signify that the INVITE message has arrived and that the caller must now wait for a further response from the callee, signifying if the call is to be accepted. A 200 OK message is subsequently sent if the call is to be accepted. This is the end of the First Transaction Phase and the caller should now be aware of whether or not their request for a conversation has arrived and the callee has accepted it.
2. The second transaction is a more simple affair and consists only of an ACK message from the caller back to the callee to signify that the caller acknowledges the wishes of the callee and is ready to send the media, if the call is to be accepted.
   A Media Session is then set up where the voice conversation is sent back and forth between the two parties.
3. This ends (and the third transaction phase begins) when one of the parties sends a BYE message which means the call is to be terminated. The other party on receipt of the BYE message then returns a response message to acknowledge the BYE and the call is cleared.

None of this of course takes into account that messages may be lost in transit because it is assumed that SIP is using the UDP protocol for transport. However, this perhaps goes beyond the scope of interest since the purpose here is not to explain in full the technical permutations of the system - rather perform an analysis of the architecture involved from the point of view of the recipient, since it is here where any SPAM filtering of any kind will occur.

A.1.1.2 The Invite Message

As well as alerting the callee to the presence of a caller, the INVITE message contains information which will allow the callee to see what media capabilities the caller has. A typical INVITE message might resemble the following:

INVITE $sip : john@192.190.132.31SIP$
$Via : SIP/2.0/UDP10.11.12.13; branch = z9hG4bK776asdhds$
$Max - Forwards : 70$
$To : "John" < sip : john@192.190.132.31 >$
$From :: "Mark" < sip : mark@10.11.12.13 >; tag = 1928301774$

$Call-ID$: $a84b4c76e66710$@10.11.12.13
$Cseq$: 314159 **INVITE**
$Content-Type$: $application/sdp$
$Content-Length$: 228

$v = 0$
$o = mark114414141122 14INIP$410.11.12.13
$s = -$
$c = INIP$410.11.12.13
$t = 00$
$m = audio$49170$RTP/AVP0$
$a = rtpmap$: $0PCMU/8000$
$m = video$51372$RTP/AVP31$
$a = rtpmap$: $31H261/90000$
$m = video$53000$RTP/AVP32$
$a = rtpmap$: $32MPV/90000$

As can be seen, the message is split into two parts - Firstly the typical mandatory SIP headers are listed, then there is a blank line followed by a media description using the Session Description Protocol (SDP) which is used to describe the media capabilities of the sender.

The first line of the INVITE message indicates the type of message (in this case, INVITE) followed by a URI identifying the user to which the message is addressed. Finally comes the SIP version being used.

The second line is the "Via" header which (for the purposes of this example) is used to signify the address to which responses should be sent. If the message was sent directly from the client to the server, then this will be the same as the actual address of the sender. However, requests like this may be sent via one or more proxies. In which case the address of the most recent proxy will be given. When SIP proxies are used, each proxy adds its own via header to the received request before forwarding it on. This allows the route of the SIP request to be traced.

The third line is related to the second in that it contains the maximum number of times a message can pass through a proxy on its way to the destination. This is useful to prevent routing loops.

The fourth line contains the From: header which identifies the caller. It is quite similar to an email address format with a display name given in quotes and the actual email address/URI enclosed within "<" and ">". If no display name is to be used then the "<" and ">" are not required. Alternatively, the sender can

also used the keyword "Anonymous" if they wish to remain just that. There is also a mandatory Tag identifier which identifies the relevant existing dialogue.

The fifth line is the To: field which identified the party to whom the request is being sent. It uses the same syntax as the From: field but without the Tag identifier.

The sixth line is the CallID header which is a globally unique identification value for that particular call. Line seven is the CSeq value which is a sequencing value used to match the Request and Responses. (this is important due to the unreliability of the underlying UDP transport layer and the consequent non consecutiveness this may induce.).

Finally, the Content Type and Content Length headers follow, which describe the type of content the SIP message is carrying (in this case SDP data) and its size in bytes.

The SDP content is in human readable form and defines the media carrying capabilities for the whole session. It always begins with a "v=" part (which defines the SDP version) and is followed by various other global fields. Finally there are several media description sections (beginning with "m=").
The following table describes the meaning of each field, plus several others of interest.

Session Level Field Types	**Description**	**Example Format**
$v =$	Protocol Version	"$v = 0$"
$o =$	Owner/Session identifier	o=<username><sessionid> <version><networktype> <address type><address>
$s =$	Session Name	s=<session name>
$c =$	Connection Information	c=<free text description>
$t =$	Time the session is active	t=<starttime><endtime>
$e =$	Email address	e=<email address>
$p =$	Phone Number	p=<phone number>
$b =$	Bandwidth Information	b=<modifier> <bandwidth(kbits)>
$z =$	Time Zone Adjustments	
$u =$	URI Description	u=<URI>
$m =$ (zero or more)	Media Name	m=<media><port> <transport><formatlist>

Media Description Level Field Types	**Description**
$i =$	Media Title
$c =$	Connection Information
$b =$	Bandwidth Information
$k =$	Encryption Key
$a =$	Other Media Attribute

From the header fields, the To:, From:, Via and Call Id tags together identify a particular dialogue. They are sent with the initial INVITE message and subsequent Server/callee Responses, ACK and BYE messages. In general the format of each SIP message passed back and forth between two parties does not really change that much. The 200 OK message sent from the callee back to the caller will be nearly the same, other than having the From: and To: fields reversed and the SDP section will contain the callee's capabilities. The ACK will be almost identical to the INVITE apart from having no SDP payload and being labelled "ACK" rather than "INVITE" in the appropriate places.

The BYE message, which signals the termination of the media correspondence, will also contain the same headers as the INVITE and ACK, but of course the From: and To: fields will depend on which party has taken the decision to end the call.

Perhaps of particular interest here is how does the callee reject a call? For example, should any SPAM detection system glean from the INVITE that the calling party is undesirable, what are the options for dismissing the attempted call?

When a call is accepted, the callee returns a "200 OK" response back to the caller. However, if something should appear to be amiss or suspicious about the INVITE message, there are a number of other responses which are possible and ensure that the session goes no further. Similar to other high level protocols, a 3xx response may be given which indicates some kind of re-direction (in which there would be included an alternative address if the callee had moved temporarily or permanently or if a PROXY needed to be used). A 4xx response notifies an error on the part of the client (caller) and some examples are:

400 BAD REQUEST
401 UNAUTHORIZED
403 FORBIDDEN
404 NOT FOUND
406 NOT ACCEPTABLE

410 GONE
485 AMBIGUOUS
486 BUSY HERE

It would really be up to the callee's discretion, which of the above should be used as a SPAM detected response.

A.1.1.3 Concluding Remarks

Due to the simplicity of SIP, any analysis of headers for suspicious contents can really only be done in 2 places - the INVITE message and the ACK. Both of these should be very similar, with the INVITE containing the SDP payload describing the media session options. These media options may give clues as to SPAM characteristics, particularly after some learning by any filter.
It is likely that many of previously mentioned textual techniques used for Text Spam may be implemented with SIP, because of the textual nature of the SIP messages. It would be easy to carry across many of the header checking mechanisms, together with various keyword searches, blacklists and whitelists.

Aside from this, the only other option is to filter based on content of the stream between the two parties. This will take place through the ports and protocols (for voice conversation, likely RTP and RTCP) included in the SDP description. Therefore, the contents can begin to be analysed as soon as RTP packets are received after the second transaction phase is complete.

A.1.2 H.323

An earlier implementation than SIP, the ITU-T standard is more of an effort towards transmission of multimedia in general as opposed to just a voice conversation. Around 95% of all VOIP messages at the moment use this protocol and most VOIP phones and other equipment support H.323. The standard specifies:

- terminals
- Gateways between a H.323 network and other voice networks.
- Gatekeepers which are the control servers of the H.323 network. They take care of registration and admission procedures.

The Standard includes the use of the Real-Time Transport Protocol and Real-time Control Protocol (RTP and RTCP respectively). These allow for the inclusion of timestamps and sequence numbers together with the other standard TCP/IP provisions so that jitter and packet loss can be easily detected.

Its latest version (5) is the most reliable, and flexible yet, although there is no support for QoS. The single biggest difference between H.323 and SIP is, where SIP uses a textual approach, the H.323 standard is implemented using the ASN.1 (Abstract, Syntax, Notation One) scheme which is a binary protocol. Furthermore, where SIP is an application level protocol in its own right (similar to HTTP), H.323 is more of a suite of other protocols each fulfilling a different purpose (as shown in **figure A.6**). Those which are relevant to this project are:

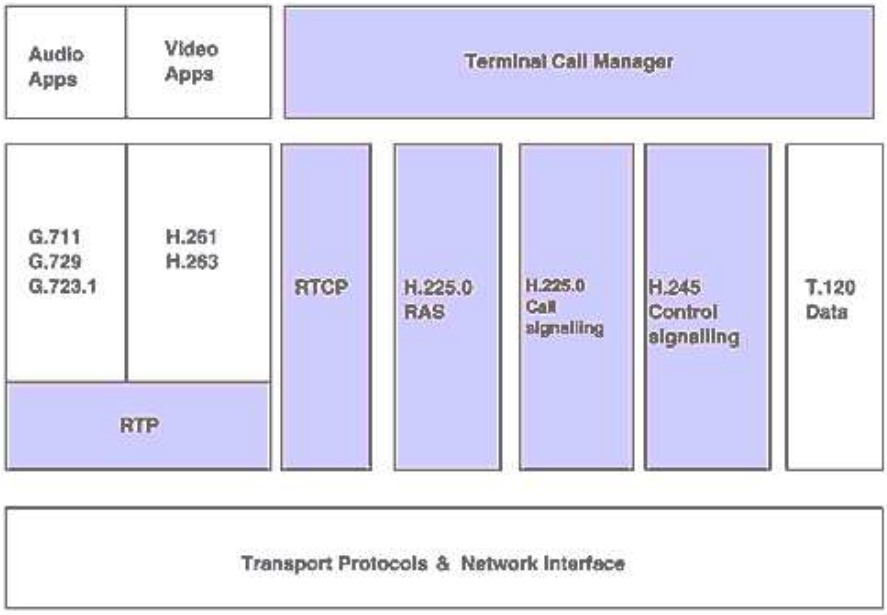

Figure A.6: H.323 Block Diagram (*www.iec.org*)

- **H.225.0 (Q.931)** - This is the part which controls call signalling and setup. (The Q.931 standard used here is also used in ISDN call signalling.).
- **H.225.0 (RAS)** - The Registration, Admission and Status channel is used for communication with a gatekeeper. Through it is passed registration, admission, address resolution and status messages.

- **H.245** - Known as Conference Control, this channel is the means through which the actual multimedia is passed from one host to the next. As has been mentioned, H.323 is used to transfer multimedia of all kinds, however this thesis is only concerned with the transfer of vocal communication via this channel. The H.225 protocol negotiates the call setup, and H.245 negotiates the communication transfer which includes the setting up of the vocal compression to be used.

These protocols may be used together in different modes, depending on the application. As with the proxy server in SIP, the gatekeeper is performing a kind of intermediary role, aiding the setting up of calls and registering of users. However, the extent of its involvement is determined by the chosen mode.
Direct Signalling Mode dictates that only the H.225.0 RAS messages are passed through the gatekeeper, while everything else is passed directly back and forth between caller and callee.
Gatekeeper routed call signalling mode means that only the H.245 messages are passed directly between caller and callee (more similar to SIP).
Gatekeeper Routed H.245 Control Mode goes further in that the only thing which is passed directly between the two terminal end points is the media stream with H.245 control messages going through the gatekeeper.
With *Direct Signalling Mode*, load is taken off the gatekeepers as they only play the minimal role of call admission. In other words, they have little part to play and are largely ignorant of the connected communications. For the purposes of this project, this would therefore suggest that SPAM would have to be effectively handled at the client rather than at the gatekeeper. It could be assumed that whilst the gatekeeper might have some control over who is allowed to register through it, this is really of little protection. Some kind of blacklist could be used at this point, but anything more than that would possibly be of little use and more of an encumbrance upon users on the network.
With *Gatekeeper Routed Call Signalling*, the gatekeeper could perhaps become more involved because of its capacity to handle the call signalling messages. Information could definitely be gleaned as to not only the identification of the calling party but of calling habits and behaviour over a period of time.
The *Gatekeeper Routed H.245 Control Mode* would perhaps add slightly more extended opportunities for filtering at the gatekeeper dues to the gatekeeper's ability to monitor connection and media usage stats with this mode.
However, as has been said it would probably be more unhelpful to use a one-size fits all SPAM detection approach due to varying usage by registered users.
Therefore, probably the most effective positions in the network to have the SPAM detection mechanisms would be at the client.

A general overview of a H.323 system is shown below in **figure A.7**. As with the SIP example, one is only concerned with those components and singles withing

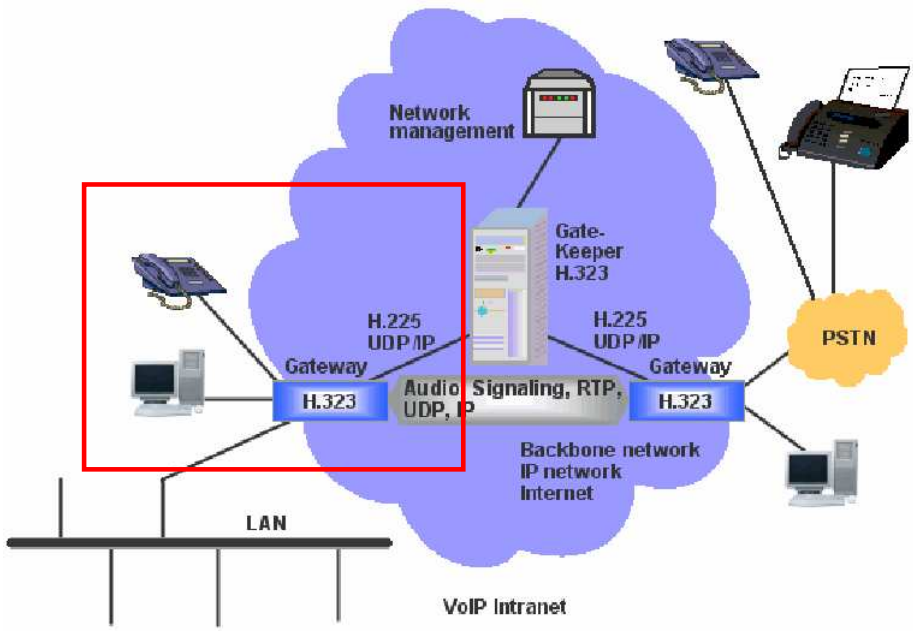

Figure A.7: H.323 Overview with selected region of interest (*solinet.com*)

the red box. Client side filtering is what is being developed here so background knowledge of the entire structure may be useful but not always relevant.
As can be seen, with the *Direct Signalling Mode*, the responsibility of dealing with SPAM is moved away from the Registering Authorities, since the setting up of the call and transferral of information happens directly between caller and callee. Therefore effective white-listing/black-listing would need to be performed on the client's own system, together with content filtering.

A.1.2.1 Implementation

An example of an H.323 call will now be introduced. Two users wish to establish a voice call from two separate locations, each with fixed and known IP addresses. The communication will require 2 TCP channels between the two terminal endpoints. One is for call-setup and the other for media control and capability exchange.
The initial call-setup messages are sent over the first TCP connection between the caller and a port at the callee endpoint (the standard port is 1720). Over this connection flows all the H.255.0 Q.931 call-signalling messages.
Once this phase is complete the second connection is opened carrying H.245 messages which deal with the multimedia capabilities of both sides. This phase also deals with the parties establishing separate logical channels through which the audio/video streams are to flow.

Part 1 - Establishing the connection: In **figure A.8**, Steve wishes to call Brian. He knows Brian's IP address and sends an H.225 SETUP message on port 1720 (the Call signalling channel port defined in H.225). It contains various fields and those of most interest are:

- The message type eg H.225: SETUP
- The call reference value. This is a locally unique number used to identify all further messages relating to each individual call. Eg. 10
- Call identifier. This is set by the calling party and will be a globally unique number. It is used to associate the call signalling messages with the RAS messages. Eg. 23849567
- An H.323 PDU which itself can contain
  - A source address field containing the aliases of the sender. Eg. H.323 ID of Y: Steve@Somewhere.com

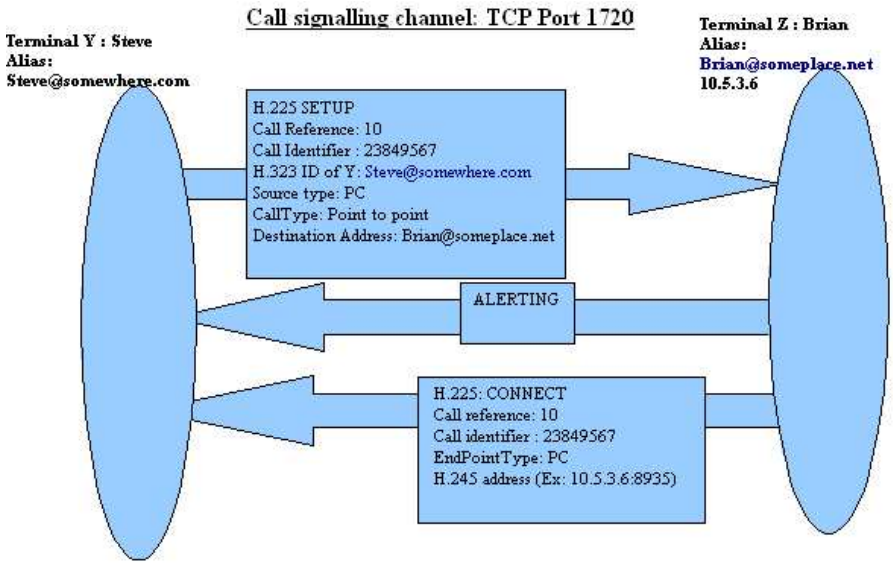

Figure A.8: H.323 Call Signalling 1

- A source information field showing the nature of the calling equipment. Eg. Source Type: PC
- A Destination address which is the called alias address - ¿ this can be a regular phone number, an H.323-ID which is a unicode string, a normal URL, a transport ID (eg. 10.5.3.6:1720) or an email ID. Eg. Destination Address: Brian@Someplace.net

Upon receipt of the SETUP message by Brian, he must return either a CALL PROCEEDING, ALERTING, CONNECT, or RELEASE COMPLETE message immediately back to Steve. Steve has to receive it before his setup timer expires (around 4 seconds).
In the example, ALERTING is sent to Steve to indicate that Brian's phone is ringing. Brian now has 3 minutes to either accept or reject the call. If he accepts it then a CONNECT message is immediately sent back to Steve. The CONNECT message contains the following information:

- The Message type (as before). Eg. H.225: CONNECT
- The Call reference value (as before) Eg. 10
- The Call identifier value (as before) Eg. 23849567

- Destination information allowing Y to know if Z is connected to a gateway or other hardware. Eg. EndPointType: PC
- The IP address/port for which Z would like to be used for the H.245 communication. Eg. H.245 Address (Ex: 10.5.3.6:8935)

Part 2 - Creating the control channel (Figure A.9): When the CONNECT message arrives back with the caller (in this case Steve), he can now open his H.245 control channel. This will stay open for the duration of the entire call and its purpose is to allow both parties to find out which codecs the other is capable of using.
Because Steve has received the callee IP number and port address from the CONNECT message (8935 in the example), he can now send a TERMINALCAPABILITYSET message over this control channel. This message contains the following information:

- A sequence number
- A Capability table, containing an ordered list of the codecs the terminal can support.
- A Capability Descriptor Structure. This is a more detailed structure which described which combinations of codecs the terminal can and can not support. It effectively details the configurations of codecs the terminal is capable of dealing with.

Both parties actually send out the TERMINALCAPABILITYSET message simultaneously. To Acknowledge Reception of the capabilities of the other, both send a TERMINALCAPABILITYSETACK message.

Part 3 - Opening of the media channels: By this stage, both caller and callee should have agreed that the conversation is to go ahead and will be aware of what each other is capable of making sense of, with respect to multimedia formats. Separate, unidirectional, logical channels now must be opened to allow transport of the actual streams.
For Y to open a logical media channel with Z, it must first send an OPENLOGICALCHANNEL message along to the same port as before (8935). This message will contain a number identifying the logical channel to be used, the encoding type of the media to be sent, the UDP address and port for RTCP reports to be received, a session number, the type of the RTP payload and the capacity to suppress silences in the communication. The type of codec to be

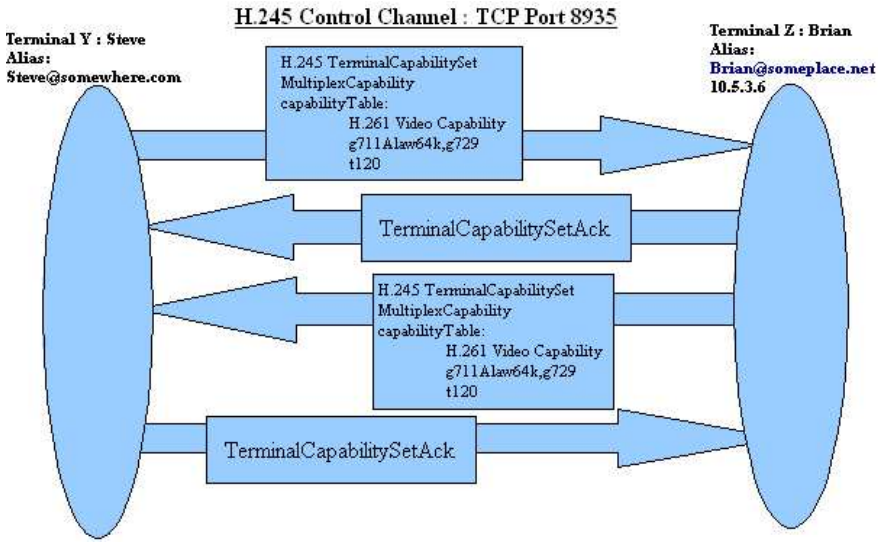

Figure A.9: H.323 Call Signalling 2

used will have been selected from the capability lists previously sent in Part 2. If there exist previous logical channels of communication then any new coders should be checked against the list of capabilities.
When the OPENLOGICALCHANNEL message has arrived at Z, and Z is ready to receive communication along this channel, it will reply with an OPENLOGICALCHANNELACK which contains the IP and port where Y should send the RTP data as well as the port where RTCP reports should be sent.
At the same time Z tries to open a logical channel to Y using the same procedure. **Figure A.10** explains the overall situation.

Part 4 - Dialogue: Both parties are now both content that

a) the conversation may take place
b) the codecs have been agreed on
c) the appropriate logical channels (ports and addresses) have been set up to facilitate communication.

The media/voice conversation is now sent from Y to Z and back again via RTP. Dual RTCP channels (one for the sender and the other for the receiver) report on network conditions, jitter, packet arrival and sequence number progress.
Figure A.11 continues on from **figure A.10**, showing the establishment of the

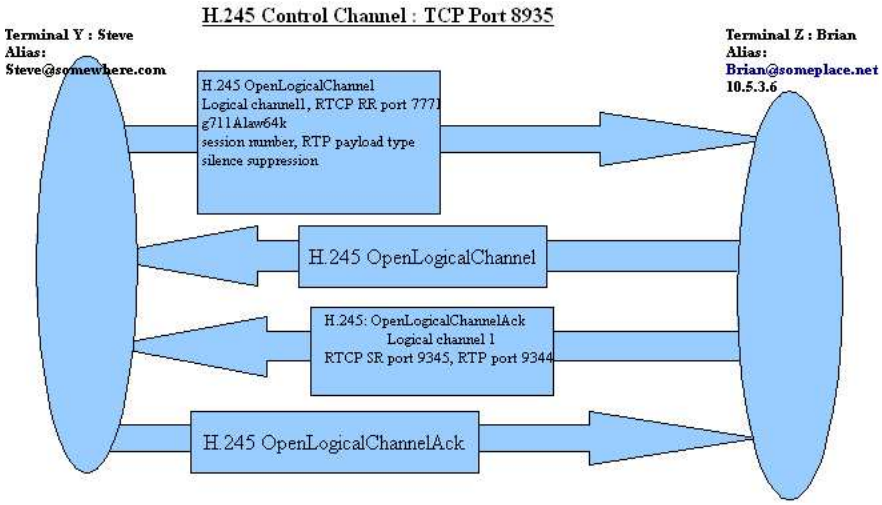

Figure A.10: H.323 Call Signalling 3

logical channels through which communication occurs. Note how the communication is taking place across several ports this time. In actual fact the RTP flow can be replicated in the opposite direction, as Z seeks to respond to Y.

Part 5 - Ending the call: For one of the parties to end the call their associated terminal must send an H.245 CLOSELOGICALCHANNEL message for each logical channel which has been opened. The other party should then acknowledge this with a CLOSELOGICALCHANNELACK acknowledgement. The channel is then officially closed and when all the logical channels have been closed, both parties send an H.245 ENDSESSIONCOMMAND to each other. This closes the H.245 control channel, but the H.225.0 call signalling channel may still be open. In which case, each party sends an H.225 RELEASECOMPLETE message to each other and the H.225.0 channel is closed on reception.

A.1.2.2 Concluding Remarks

It is necessary to consider how the system works to be able to establish any junctures during the chain of events at which some kind of filtering could be achieved. In other words, what information is gleaned from the caller at what stage and how may this be used to identify the credibility of the caller?
From the previous simple overview of how the system works the callee is pro-

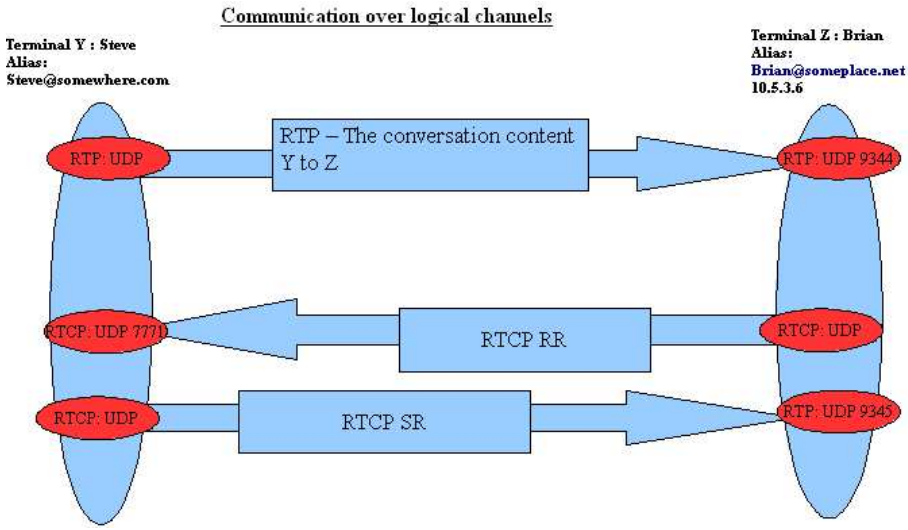

Figure A.11: H.323 Call Signalling 4

vided with information about the caller from several kinds of messages:

1)**H.225 SETUP message** (on port 1720)
The caller can be identified uniquely by either the calling party number and sub-address (if included) or the Call Identifier field. There is also the possibility of gleaning IP information from the TCP fields. The included source information field which lets the callee know about the calling equipment may be useful in determining patterns of credible or SPAM callers.

2)**H.245 TERMINALCAPABILITYSET message** (on call-signalling port decided by callee) It may be possible of build up capability set profiles of known SPAM callers. Whilst this should not automatically disqualify a caller, it may give clues as to their credibility.
The source address of this should be verified in relation to the previous SETUP message. Differences or inconsistency would arouse suspicion.

3)**H.245 TERMINALCAPABILITYSETACK message** (on call-signalling port decided by callee) Again, the source should be compared against previous messages.

4)**H.245 OPENLOGICALCHANNEL message** (on call-signalling port decided by callee) It may be possible to build up profiles of what known voice SPAMMERs tend to use with regards to the RTCP Receiver Report Port, se-

lected codec and payload type. Again, whilst not leading to automatic disqualification, it may hint at the caller'ts credibility.

5)**H.245 OPENLOGICALCHANNELACK message** (on call-signalling port decided by callee) In the same way as before, SPAMMERs may tend to choose familiar ports for their RTCP and RTP communication.

6)**RTCP Sender Report message** (on logical channel port decided by callee)

7)**RTP flow packets** (the conversation content) (on logical channel port decided by callee) This is really the last opportunity for the callee to discover if the packets sent are Voice SPAM. Up to this point, the callee will have been satisfied that the caller is genuine. The only remaining checks therefore are content-analysis checks of the RTP flow packets. In other words these are the packets which need to be collected in order to perform such checks.

If any of the above opportunities results in the successful detection of SPAM, then the callee should not be burdened with having to take the call. Instead, the call finishing sequence (shown previously in Part 5) should be immediately invoked. It would also be useful if some kind of statistical log be kept of the calling party so as to aid future detection of their species.

A.1.3 Review

The purpose of this short document was to familiarise the hardware and architecture of popular VOIP systems, namely SIP and H.323. This was so as identify points in the chain of events during a call setup where checks may take place in relation to SPAM filtering.
This has all been addressed very much from the point of view of the client, since it is there that any filtering will need to be taken care of. Therefore, complicated implementations of the two systems can be effectively ignored, since they are not especially relevant - in any event the client must deal with the same mandatory steps during which filtering may happen.

It has been described where about filtering based on headers and text keywords can occur. This is of course akin to the previously discussed methods used with email. This project is more interested in the actual content-based filtering options. Therefore, because both of the main VOIP systems are using RTP/RTCP for the transfer of voice data back and forth between the two endpoints involved, it is surely now necessary to understand this protocol and how packets of data sent back and forth are reconstructed into data forms which can be read via a given codec. Particular attention should be given to the sequenc-

ing of packets and any issues in the temporal realm.

Appendix B

Implementation Details

B.1 General Code structure

All of the code is basically divided into 3 main classes -

- *Secretary.py* - This is the main class which contains all of the graphics and interface code, together with the main method.
- *ExtractFeatures.py* - This contains all the methods related to extracting features and other information from the captured Sound() object.
- *DecisionMaker.py* - This contains everything which contributes to the "intelligent" decision making process, subsequent to feature extraction.

An overall, general view of the system implementation model is shown in **figure B.1**.

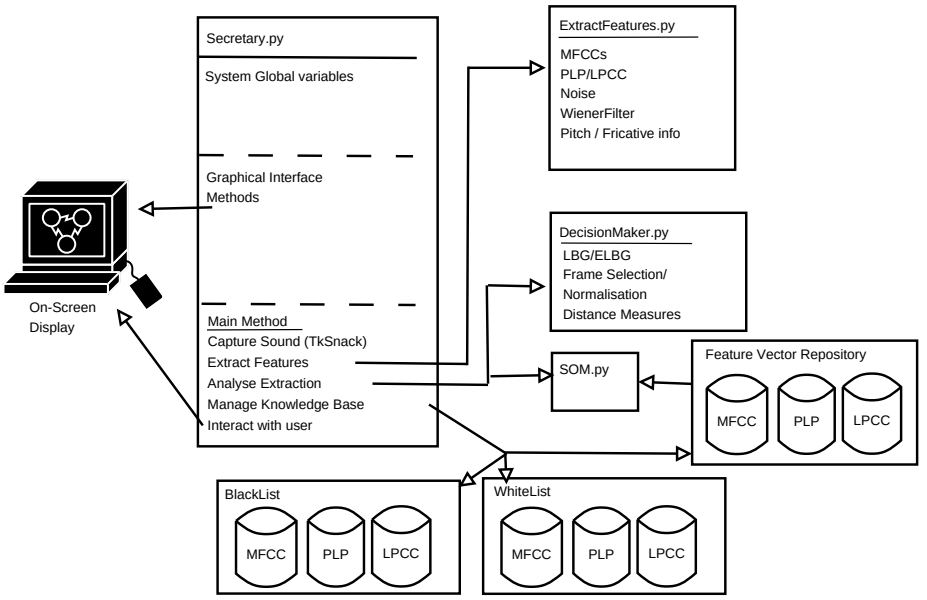

Figure B.1: System Overview

B.2 Programming the parts

Every single "cog in the wheel" will not be discussed because once wishes to stay within the bounds of relevance and appropriateness. Most of the technical calculations have been covered in the design section and a large amount of the implementation of the prototype system was merely a case of converting this to code.

B.2.1 Basic Sound recording/playback/handling/timing

B.2.1.1 Introduction

The **TkSnack** Sound() object is at the heart of all audio handling. One of the primary advantages of using it is that many of its real-time methods (recording, playback) are threaded procedures. In this way it is possible to record and playback at the same time, which was crucial in the detection of whether caller voice activity was happening simultaneously with the calling party greeting.
The Sound object can be created as an empty container and then filled subse-

quently.

$mySound = TkSnack.Sound()$

Once the Sound object has been created, stored data may then be read into it. Alternatively a Sound object may be created directly with the filename of the audio file.

$mySound = TkSnack.Sound(file = "myfile.wav")$
$mySound2 = TkSnack.Sound()$
$mySound2.read("myfile2.wav")$

A Sound() object is essentially **TkSnack's** own representation of audio data held in memory. In order to write the audio contents of a sound object to a permanent file, a write function is included.

$mySound.write(file = "myfile.wav")$

Normally the audio from a source may be recorded directly into an empty Sound() object by using the record method:

$mySound = TkSnack.Sound()$
$mySound.record()$

The recording continues until the stop() function is called.

$mySound.stop()$

The possibility of playing back some recorded content is also desirable given that the Secretary system plans to issue a challenge or greeting to the calling party. The Sound() object provides for this by including a play() function. Of course, the Sound() object must contain some audio data prior to playback.

$mySound.play()$

B.2.1.2 Timing

Combining these ideas allows the creation of the kind of playback/recording mechanism which will emulate the desired timing behaviour. A high level description of this in terms of the **TkSnack** techniques is as follows.

//Time allowed for the calling party to give a response

$timeForResponse = 3$

//read in the greeting file and get its length in seconds
$greeting = tkSnack.Sound(file = "greeting1.wav")$
$timeForGreeting = greeting.length(unit =' seconds')$

//Set up the recording parameters
$inputSource =' Mic'$
$capChannels =' mono'$ //no of channels
$capRate = 8000$ //frequency rate
$encType =$ "$Lin16$" //Encoding type

//Sound objects to catch the stream
//capture while greeting
$newStream1 = tkSnack.Sound(rate = capRate, channels = capChannels, encoding = encType)$
//capture after greeting
$newStream2 = tkSnack.Sound(rate = capRate, channels = capChannels, encoding = encType)$

//begin recording
$newStream1.record(input = inputSource, fileformat =' RAW')$
//simultaneously playback greeting
$greeting.play()$

//Timer function - calls "delayEnding1" after timeForGreeting seconds
$timer1 = threading.Timer(timeForGreeting, delayEnding1)$
$timer1.start()$

.....

The final pair of commands in the above sequence begin a timer object which has the effect of calling the function $delayEnding1()$ after the number of seconds indicated by the first enclosed parameter. Just prior to the introduction of the timer, the greeting sound is left to play and this needs no monitoring - the sound audio from the greeting Sound object just plays back until its finished. However, the simultaneous recording which occurs will begin recording and only stop when the Sound() object's *stop* function is called. Therefore, the $stop()$ function is placed at the beginning of the called procedure from the timer object and is consequently brought into play after the appropriate time gap.
But this only ensures the procurement of the audio data from the caller during the greeting. The actual response to the greeting must also be captured. This concept of starting a Sound() object recording and then stopping after a set

time period using a timer device is replicated once more in the $delayEnding1()$ method:

$def\ delayEnding1() :$
$newStream1.stop()$

//start recording caller response
$newStream2.record()$

//thread delays then stops the record
$t2 = threading.Timer(timeForResponse, delayEnding2)$
$t2.start()$

$defdelayEnding2() :$
$newStream2.stop()$

//...Feature extraction begins on newStream2....

Similarly to before, the timer object in $delayEnding1()$ calls the function $delayEnding2()$ after the determined interval, which in turn calls the appropriate $stop()$ function. By this stage, the two Sound() objects, newStream1 and newStream2 will hold all the sufficient audio data required for the rest of the system's calculations to be made.
Further information regarding the **TkSnack** sound object and its full range of methods and functions may be found at.

B.2.2 Mel Frequency Cepstral Coefficients

The Mel Frequency Cepstral Coefficient computation is set in motion through the calling of the $getMFCCs()$ method.

$features = getMFCCs(mySound, winLength, overLap, WINTYPE, PREEMP,$
$NOOFCOEFFS, ALPHA, B, capRate, DELTA =' false',$
$DELTADELTA =' false')$

the parameters included in the function call are explained as follows:

- mySound : the **TkSnack** Sound() object to be scrutinised.
- winLength : window length specification in samples

- overLap : window overLap length specification in samples (used with "winLength" to designate window boundaries for short term power spectrum determination).
- WINTYPE : the windowing function used (eg "hamming")
- PREEMP : the pre-emphasis value
- NOOFCOEFFS : The number of coefficients to be calculated.
- ALPHA : Noise calculation parameter (see section B.2.4)
- B : Noise calculation parameter (see section B.2.4)
- capRate : the largest frequency to be represented in the power spectrum (8000hz)
- DELTA : flag designating calculation of delta coefficients
- DELTADELTA : flag designating calculation of delta delta coefficients

B.2.2.1 Power Spectrum calculation

According to the Design of this procedure (section 3.2) the first three steps of the process are pre-emphasis, windowing and Fourier transform. Then from this, the squared magnitudes constitute the power spectrum of the recorded sound. Conveniently, the **TkSnack** Sound() object provides a method, *powerSpectrum*() which will return the very same directly whilst performing the pre-emphasis and windowing in the process.

$mySound.powerSpectrum(start = st, end = en, windowtype = WINTYPE,$
$preemphasisfactor = PREEMP)$

In this implementation the windowtype parameter is set as "hamming" and the preemphasisfactor parameter set to 0.95. The only other used parameters are the start and end points ("start" and "end") in the sound object delimiting the scope of the operation. The precision of the powerSpectrum, or in other words, the number of Fourier Transform points is 256 by default and this is left as an acceptable value.
The number of times this power spectrum calculation is to be calculated in a short term fashion depends on the number of windows, which itself is in accordance with the window length and overlap periods in relation to the overall length of the captured sample. It becomes necessary to compute the "start" and "end" parameters for the *powerSpectrum*() method, prior to operation and these are calculated from the particular window number via the method

$computeWindowParams()$. For example,

$startpoint = computeWindowParams(i,'start', NO\_OF\_WINDOWS, NO\_OF\_SAMPLES, winLength, overLap)$

The above example returns a single point in the Sound object as a sample number. This is calculated from the parameters:

- i : the window number
- 'start' OR 'end' : flags if the point to calculate is a start or end point of the window.
- NO_OF_WINDOWS : The number of windows
- NO_OF_SAMPLES : The number of samples
- winLength : the length of the window in samples
- overLap : the length of the overlap in samples

NO_OF_WINDOWS is calculated in advance from winLength, overLap and NO_OF_SAMPLES. The window length and overlap length in samples are also calculated from an initial designation in ms by multiplying by the frequency range (the top frequency in this case) in khz.
It should be noted that this procedure of looping through the entire Sound object to obtain the windowed power spectrum for each designated internal is not unique to just the Mel Frequency Cepstrum calculations. It is used throughout the code to obtain the same desired result for Noise Examination, LPCC computations and PLP operations.

B.2.2.2 Mel Filterbank incorporation

The Mel Filter banks must first be created and then applied. Section 3.2.4 gave a detailed account of how they are structured. The method $makeMelFilters()$ constructs the shape of the filterbank with a y axis scaled/normalised between 0 and 1 and an x axis representing the spectral partition (from 1 to 256)

This method is called in the following way:

$melFilters = makeMelFilters(NoSpectralPts, Nofilters, topF, wideningFactor)$

where

$NoSpectralPts$: The Number of pts in the Fourier Transform
$Nofilters$: The number of Mell filters to construct (24 is the chosen value)
$topF$: The top Frequency represented (in this case 8000hz)
$wideningFactor$: an approximation used to construct the ever widening triangular filters after the 1000hz mark. This approximation is used as a factor to multiply the current triangular width to get the width of the next one. It was found that 1.098 produced a pleasing result for the desired filterbank creation.

The filter bank created according to the above scale is then applied to each short term power spectrum window. Because the same set of filter banks is applied and then re-applied it need only be created once, at the outset, and then subsequently used to do the filtering. Each time a new power spectrum window is considered, the mel frequency banks are actually applied using the $applyMelBanks()$ method. For example:

$$melBanks = applyMelBanks(melFilters, FOURIERED)$$

where $melFilters$ is the filterbank window created by $makeMelFilters()$ (above) and FOURIERED is the current short term power spectrum array. The return value is a 2 Column array, the first column consisting of the total energy for the filtered window whilst the second holds a list of the log magnitude sums for each filter bank (24).

B.2.2.3 Coefficient Creation

The cepstral coefficients are created from the logged filterbank outputs, and python itself provides the Inverse Discrete Fourier Transform as a means of doing this via its FFT library:

$$Coefficients = FFT.inverse_real_ft(LOGMAGS, NOOFCOEFFS)$$

where LOGMAGS are the logged magnitude outputs of the filter banks, and NOOFCOEFFS is the number of coefficients one wishes to obtain (12). This also corresponds to the length of the return array.

B.2.3 Linear Prediction Cepstral Coefficients/Perceptual Linear Prediction

The calculation of the coefficients from the PLP method essentially shares many of the same techniques as the LPC method, therefore, both procedures are contained within the same $getLPCCs()$ method. One of the input parameters then acts as a switch so that the appropriate program flow can be selected according to the chosen method. The $getLPCCs()$ method is called as follows:

$features = getLPCCs(PLP, mySound, winLength, overLap, WINTYPE, PREEMP, NOOFCOEFFS, ALPHA, B, capRate, DELTA =' false', DELTADELTA =' false')$

the parameters included in the function call are explained as follows:

- PLP : boolean flag determining PLP or LPC as the selected mode of operation.
- *mySound* : the **TkSnack** Sound() object to be scrutinised.
- *winLength* : window length specification in samples
- *overLap* : window overLap length specification in samples (used with "winLength" to designate window boundaries for short term power spectrum determination).
- WINTYPE : the windowing function used (eg "hamming")
- PREEMP : the pre-emphasis value
- NOOFCOEFFS : The number of coefficients to be calculated.
- ALPHA : Noise calculation parameter (see section B.2.4)
- B : Noise calculation parameter (see section B.2.4)
- *capRate* : the largest frequency to be represented in the power spectrum (8000hz)
- DELTA : flag designating calculation of delta coefficients
- DELTADELTA : flag designating calculation of delta delta coefficients

B.2.3.1 Creating the PLP filter bank

On the designation of the PLP method the PLP filter bank described in section 3.4.3 is created at the outset via the *constructPLPFilter*() method, as follows:

$plpFilter = constructPLPFilter(NoFFTpts, topF, NoFilters)$

$NoFFTpts$ refers to the number of points to be contained in the short term power spectrum computation. $topF$ is the maximum frequency represented in the power spectrum and $NoFilters$ is the number of filters one wishes to have, in this case 20 is sufficient.
The calling of the *constructPLPFilter* has the effect of combining the procedures outlined in section 3.4.
The following subprocesses are called:

1. *makeBarkScale*(): This returns an array of the points represented in the standard frequency power spectrum as the alternative bark scale.

2. *equalLoudness*(): This returns the approximated equal loudness curve according to section 3.4.3.

3. *eqToBark*(): This is used to convert the equal loudness curve (created in accordance with the frequency scale) into the bark scale realm.

By normalising the product of 1 and 3 (above) between 0 and 1, it is then possible to obtain the filterbank suggested by Hermansky (see section 3.4.3).

B.2.3.2 Applying the Filterbank

The iterative part of the over-arching process of course includes the application of this created filterbank to each short term power spectrum window. These short term spectrum frames are created in the same way as with the *getMFCCs*() method (section B.2.2). The *applyPLPFilters*() method then applies the filter to the frame, returning the summed magnitudes of each filterbank output. After these undergo cubic compression, the cepstral coefficients are obtained by using the same inverse Fourier transform as before.

B.2.3.3 Extracting the Cepstral Coefficients - LPC

Under normal LPC conditions where the PLP flag is negative, the $getLPCCs()$ process will not calculate or apply any filterbanks. Instead the $autoLevDurb()$ function is called, returning the linear predictive parameters of the base power-spectrum:

$params = autoLevDurb(FOURIERED, NOOFCOEFFS, "a")$

where the parameters are :

- FOURIERED : the short term power spectrum window in question
- NOOFCOEFFS : the number of parameters to be calculated
- "a" or "k" : "a" returns the prediction parameters, "k" returns the reflection parameters.

The returned array contains whichever group of parameters (predictive or reflection) is specified, and then the cepstrum is calculated from these using the $getLPCepstrum()$ function.

B.2.4 Noise Calculations

The reason that both $getMFCCs()$ and $getLPCCs()$ include noise relevant parameters in their procedure calls is because it was thought it would be efficient if noise calculation was done in-line with the actual feature extraction. The reasoning here is that both noise calculation and feature extraction require the breaking up of the sound data into windows. So, why the need to perform the same division into frames more than once?
Therefore, once the extraction of the cepstral coefficients is complete, the calculated power spectrum windows are retained and used for the frame-wise noise estimation procedure. Regardless of which method is used to extract the cepstrals from the sound data, the calculation of noisy and non-noisy frames will be necessary anyway because subsequent frame selection and normalisation require the input of a noise mask array in their calculations.
Using the ALPHA and B parameters supplied then, a noise mask array is calculated using the method in section 3.5, where a noisy frame is denoted as "1" and "0" otherwise. This noise mask array forms one of the return values from the $getLPCCs()$ and $getMFCCs()$ methods.

B.2.4.1 AQBNE and QBNE

These enhanced methods for estimating the total noise and the noise spectrum itself are calculated using the enhancedNoiseEst() method. A typical call of this method is as follows:

$noise = enhancedNoiseEst(FRAMES, NOISE, selective, q, qmin, r)$

The parameters used here are:

- FRAMES : the array of all the frame power spectrums divided out from the Sound() object
- NOISE : a mask array of 1s and 0s defining a frame-wise noise estimation.
- *selective* : a boolean flag determining if the advanced noise estimation should apply to all frames or only those previous determined as noise by the frame-wise technique.
- q : the q-line parameter for the QBNE formula in section 3.5.3 (set to 0.5).
- *qmin* : the *qmin* parameter for the AQBNE formula in section 3.5.4(set to 0.45).
- r : the r parameter for the AQBNE formula in section 3.5.4(set to 15).

B.2.5 Pitch Calculations

Whilst not mentioned in the Design part of this thesis, pitch information may also be extracted as a feature of the analysed speech sample. The actual calculations rely on the AMDF estimation of pitches in the Sound() object. However, very little explanation is required for this because the **TkSnack** toolbox provides this calculation (much like the power spectrum) as a built in operation. So, in practise, to obtain the pitch data for an entire Sound() object "mySound" the command would be:

$pitches = mySound.pitch()$

This returns an array of pitch values where the interval length included in the AMDF calculation is 10ms by default.
Much like the enhanced noise level ratios there are no plans as yet to pass this

information through to any decision making logic, however there is no reason to suggest that collected pitch values could not be used in the future to perhaps influence decisions where any overall analysis proves inconclusive.
It was still considered useful to use the extracted pitch values in conjunction with the other gleaned information as part of the visual verification of the captured samples through the system's graphics interface (section 5.3.2)

B.2.6 Frame Selection/Normalisation

The process of normalising the extracted coefficients prior to clustering/SOM entry is performed by the $NormaliseVector()$ function in the $DecisionMaker$ class. It accepts as input a list of feature vectors. If there are N feature vectors then this list will take the form of N rows of M elements (where M is the number of feature values in each vector). The normalisation to be performed needs to be a column-wise normalisation so that corresponding feature values in each vector are normalised in respect to each other.
Python's standard list processing does not handle such a situation very well at all - it is much better suited to row-wise normalisation. However, it is possible to convert the entire 2D Feature Vector list into a matrix using Python's **MatPy** library. Once this is completed row-wise normalisation can be performed on the transpose of the matrix, effecting the initial goal. For example:

$inputVectorMatrix = Matrix.to_Matrix(inputVector)$
$inputVectorMatrixTranspose = inputVectorMatrix.T$

for each row in inputVectorMatrixTranspose:
$normalised[row] = normalise(inputVectorMatrixTranspose[row])$

return $normalised.T$

The normalisation itself is performed by the *normalise* sub-procedure and is merely a case of mapping the range of original values to the range 0...1.

B.2.7 Self Organising Maps

A separate class is used to represent the Self Organising Map since it is effectively an object in itself. A closer examination of this class will now be given since it was not provided as a standard resource.
The Self Organising Map is made up of two subclasses - $SOMLattice$ and $SOMNode$. $SOMLattice$ is concerned with the basic, high-level node-related

view of the map. It is made up of *SOMNodes* and its methods are concerned with calculating distances between nodes and finding the Best Matching Unit node from a given input vector. *SOMNode* is concerned with the lower level node operations such as setting and adjusting individual node weights, and setting the actual grid position of a node.

B.2.7.1 SOMLattice Methods

- $SOMLattice(noWeights, height, weight)$: This is the constructor. The height and weight parameters set the dimensions of the grid which is populated by SOMNodes appropriately. The number of weights per node is set by noWeights.
- $getNode(x, y)$: returns the Node object at coordinate point (x,y) on the SOMLattice.
- $getWidth()$: returns the width of the SOMLattice.
- $getHeight()$: returns the height of the SOMLattice.
- $euclideanDist(v1, v2)$: returns the euclidean distance between 2 vectors, v1 and v2. This is used by the $getBMU()$ method (below) to calculate the distance between a certain node and any input vector.
- $getBMU(inputVector)$: returns the point on the grid whose weights are closest (according to the euclidean distance measure above) to the inputVector parameter.

B.2.7.2 SOMNode Methods

- $SOMNode(noWeights)$: Constructor. The number of weights is set by the $noWeights$ parameter and is passed from the $SOMLattice$ constructor at initial setup. The weights themselves are initialised to random numbers between 0 and 1 at the beginning.
- $setX(x)$: sets the x-coordinate of the node on the grid.
- $setY(y)$: sets the y-coordinate of the node on the grid.
- $getX()$: returns the x-coordinate of the node on the grid.
- $getY()$: returns the y-coordinate of the node on the grid.
- $distanceTo(otherNode)$: returns the distance to $otherNode$ on the grid - used in the calculation of node neighbourhoods.

- $setWeight(wNo, value)$: sets the nodes weight at index wNo to value.
- $getWeight(wNo)$: returns the node weight at index wNo.
- $getVector()$: returns the node's entire weight array.
- $adjustWeights(inp, learningRate, distanceFalloff)$: Adjusts all of the weights of a node according to an input vector inp. The extent of which is dependant on the other parameters $learningRate$ and $distanceFalloff$ which define the learning rate of the SOM and the compensation based on how far the node in question lies from the best matching unit. Of course this method will only be called if the node in question has been found to rest within the neighbourhood of the Best Matching Unit.

B.2.8 LBG Clustering/Enhanced LBG

Like the SOM in the previous section, the LBG and Enhanced LBG clustering algorithms had to be implemented directly, and so will now be discussed in more detail.
As could be seen by the design of the algorithm in section 4.3.2.1 the general method for LBG is essentially combined with some notable extensions for the ELBG implementation. The programmed system uses 2 separate methods for the two procedures though much common ground is shared in each. To begin with the LBG algorithm will be considered. It is contained within the $LBG()$ function in the $DecisionMaker$ class.

B.2.8.1 LBG

The overall method is an iterative one, which continues until an efficiency value calculated from the distortions of the current and previous iterations is is less than an efficiency threshold ϵ. This is then made the exit condition to the main loop. The first step in the iteration is to calculate the Voronoi Partition for the given suggested codebook. The following pseudo-code explains its implementation in more detail:

```
while (maximumEfficiencyReached == "false")
     {
     // Step 1: Calculate the Voronoi partition
     for each i in inputVectors:
         {
         //Find the closest codeword and subsequent distortion
```

```
        thisMin=euclideanDistance(inputVectors[i],codewords[0])
        for each codeword in range(1,K):
            //find the distance between the current input vector
            //and the current codeword
            thisdist=euclideanDistance(inputVectors[i],codewords[j])
            //update the minimum if necessary
            if (thisdist is the minimum thus far):
                thisMin=thisdist
                index=j

        //record index of closest codeword
        Voronoi[index].append(i)
        }
    }
```

.....

On the first iteration of the method, there will have been no previous measures of the distortion of the previous codebook in relation to the input vectors. Therefore, the current distortion measurement is merely recorded and the next codebook is calculated from the first Voronoi partition. Of course, although unlikely, the distortion of the codebook on the initial iteration may be sufficiently small so as to warrant no further iterations. In this case, an exit condition is inserted appropriately:

```
if (first iteration)
    {
    efficiency=Distortions[0]
    if abs(efficiency) < ϵ
        {
        maximumEfficiencyReached = true
        return codebook
        }

    //Otherwise calculate new codebook

    for each codeword j in codebook
        {
        if Voronoi[j] != emptyset
            {
            //codeword[j] is equal to the centroid of the input vectors
            //referenced by the index values contained within Voronoi[j]
            total=0
```

```
                for each i in Voronoi[j]
                    {
                    total = add(inputVector[i],total)
                    }
                codeword[j] = total / length(Voronoi[j])
                }

            }
        }
```

.....

On subsequent iterations then, the previous iterations distortion measure from its Voronoi partition is available. The method then calculates whether to exit or not based on how the current distortion measurement relates to the previous. If the method does not exit at this point, then the new codebook is calculated in the same way as before.

```
    else if iteration>0
        {
        efficiency =
        (Distortions[iterationNo]-Distortions[iterationNo-1])/ Distortions[iterat
        if abs(efficiency) < ϵ
            {
            maximumEfficiencyReached = true
            return codebook
            }

        //Otherwise calculate new codebook

        for each codeword j in codebook
            {
            if Voronoi[j] != emptyset
                {
                //codeword[j] is equal to the centroid of the input vectors
                //referenced by the index values contained within Voronoi[j]
                total=0
                for each i in Voronoi[j]
                    {
                    total = add(inputVector[i],total)
                    }
                codeword[j] = total / length(Voronoi[j])
```

```
                }
            }
        }
    increment iterationNo
}
//end of outer While
```

The ϵ value used to denote the threshold against which the current distortion is measured in relation to the previous is originally set to 0.01.

B.2.8.2 ELBG

As detailed in section 4.3.2.1 the implementation of the ELBG Algorithm is just the same as the standard LBG but with the inclusion of the ELBG block just prior to the calculation of the new codebook for the subsequent iteration. The method for calling the ELBG function is merely called $ELBG()$ and it resides in the $DecisionMaker$ class. Two parameters are included in the function call:

- $inputVectors$: the list of normalised feature vectors extracted from the captured sound
- K : the number of codewords/centroids to produce.

ELBG requires the grouping of more information than the standard LBG algorithm. When the Voronoi partition is calculated, the following is also kept for use later in the ELBG block:

- The sum of all the distortions.
- The individual distortions for each codeword
- The Voronoi partition information itself.
- The actual feature vectors referenced from the Voronoi partition - $VoronoiVectors$.

This appears to be similar to the standard LBG implementation, however the final entity in the above list must be retained since the feature vectors referenced by the Voronoi Partition are going to be used in the re-calculation of centroids. The $getDistortions$ method in $DecisionMaker$ was created to calculate each of the above.

Skipping directly to the ELBG block implementation, the block itself inherits the current codebook and its calculated Voronoi partition, and then attempts to reposition some of the codewords. Integral to the procedure is the calculation of the Utility values which are calculated as follows (example in pseudo-code):

```
highUtilitySum = 0

for each codeword i in codebook
     {
     //Get the Utility Values and p and i areas
     if Voronoi[i]==[] //Voronoi set empty - isolated codeword, needs moved
         {
         temp=0.0
         }
     else
         {
         temp= sum(distortions[i])/float(len(distortions[i]))
         temp= temp/float(Dmean)
         }
     // record the utility value
     Utilities.append(temp)

     //sort out candidates from non-candidates
     if  temp<1
         {
         //store index of a low utility value
         lowUtilities.append(i)
         }
     else
         {
         //store index of a high utility value
         highUtilities.append(i)
         //sum up high utility values for use later
         highUtilitySum = highUtilitySum+temp
         }
     }
```

The Utility values themselves are recorded and their indexes stored in 2 arrays, one for utility values less than 1 (*lowUtilities*) and the other for utility values greater than 1 (*highUtilities*). *lowUtilities* is sorted in ascending order. The first index value of *lowUtilities* is then chosen as the candidate region for moving. The candidate region for a destination is then chosen from *highUtilities* as follows:

```
//Sum up the corresponding utility
//values for each index in the highUtility array
Sum=0
for each k in highUtilities
     {
     Sum = Sum + Utilities [highUtilities[k]]
     }

//Find the index of the chosen high utility
//region using the probability formula

maxProbIndex=0
for each i in highUtilities
     {
     tmp=Utilities[highUtilities[i]]/Sum
     if tmp>maxProb
         {
         maxProbIndex=i
         }
     }
chosenIndex=highUtilities[maxProbIndex]

//remove the chosen high utility index
//from the array in preparation for subsequent iterations
highUtilities.remove(highUtilities[maxProbIndex])
```

The final element returned by the *getDistortions*() function is the array of feature vectors associated with each codeword according to the Voronoi partition - $VoronoiVectors$. Now that the index of the high utility region has been calculated, access to the associated feature vector constituents is now available. This "cluster" of points is now split by calculating the two new points from the cluster's principal diagonal. The aforementioned array of cluster points is passed as a parameter to the *getDiagPts*() method which returns the two new candidate centroids. Thus:

```
newCWs = getDiagPts(VoronoiVectors[chosenIndex])
newCW1 = newCWs[0]
newCW2 = newCWs[1]
```

The old, low utility cluster centroid is now removed and the nearest cluster to it re-calculated to cater for its now en-widened constituent base:

```
oldClusterIndex = lowUtilities[0]
nearestCluster = nearest(oldClusterIndex,lowUtilities,codebook)
lowUtilities.remove(nearestCluster)
if length(lowUtilities)>1
     {
     //get the new Cluster from the VoronoiVectors
     newCluster= midWay(c[nearestCluster],c[oldClusterIndex])
     lowUtilities.remove(oldClusterIndex)
else
     {
     //No more low utility clusters left to try
     searching="false"
     }
```

In the above, $midWay()$ is an auxiliary method which will find the mid point between two vectors. It is used in this case as an approximation, to recalculate the new centroid of the previously vacated low utility cluster region in relation to its closest neighbour which would have shared the same proximity. The method $nearest()$ is used to find this closest neighbour with respect to the codebook as a whole.
With the positions of the new cluster set decided the overall distortion is then recalculated to see if the proposed changes are going to make a positive difference to the overall scenario. The proposed new codebook is retained if they do and rejected otherwise.

Appendix C

Test Results

Model name	No Trained	Designated as	Score
Alan01	8	Alan04	1.4
Alan07		Alan01	1.67
Anne02	10	LisaLotte10	1.33
Anne06		Esther10	1.3
AnneMarie07	11	AnneMarie09	1.09
AnneMarie10		AnneMarie07	1.07
Carol03	10	Carol02	1.15
Carol08		Carol03	1.09
Christian06	11	Christian04	1.2
Christian09		Christian05	1.13
Colin01	8	Colin07	1.29
Colin09		Colin05	1.43
Crystal03	11	Crystal10	1.56
Crystal04		Esther*	1.49
Edith07	11	Mareta08	1.37
Edith08		AnneMarie03	1.21
Esther02	10	Esther10	1.06
Esther06		Esther03	1.17
Henrietta03	11	Henrieta02	1.27
Henrietta11		Christian04	1.37
Karen01	8	Henrieta*	1.28
Karen07		Karen06	1.37
Lisa02	11	AnneMarie03	1.3
Lisa09		Carol02	1.42
LisaLotte03	10	Mareta08	1.47
LisaLotte10		LisaLotte09	1.29
Luke04	10	Luke09	1.39
Luke06		Ruben*	1.4
Mareta08	11	Mareta07	1.25
Mareta10		Mareta03	1.15
Michael02	10	Michael09	1.14
Michael10		Michael03	1.16
Monty04	6	Anne08	1.52
Monty06		Michael09	1.52
Ruben03	10	Ruben08	1.31
Ruben07		Ruben06	1.18
Stuart03	6	Christian*	1.49
Stuart07		Stuart04	1.28
Trina10	11	Trina04	1.24
Trina11		AnneMarie01	1.25
Wendy02	10	Mareta05	1.48
Wendy09		Christian03	1.42

Figure C.1: Test Results - MFCC with 10 codewords

Model name	No Trained	Designated as	Score
Alan01	6	Alan	1.35
Alan05		Alan	1.44
Anne10	11	Christian	1.28
Anne11		Christian	1.21
AnneMarie04	11	AnneMarie	1.19
AnneMarie05		AnneMarie	1.07
Carol01	10	AnneMarie	1.19
Carol03		Carol	1.12
Christian05	11	Christian	1.09
Christian09		Christian	1.11
Colin04	7	Colin	1.09
Colin08		Colin	1.4
Crystal08	8	Crystal	1.34
Crystal04		Anne	1.44
Edith01	11	Henrieta	1.27
Edith10		Edith	1.15
Esther06	10	Esther	1.17
Esther07		Esther	1.1
Henrietta07	11	AnneMarie	1.22
Henrietta09		Henrietta	1.18
Karen06	8	Karen	1.13
Karen10		AnneMarie	1.21
Lisa02	11	Esther	1.28
Lisa05		Michael	1.31
LisaLotte08	9	LisaLotte	1.27
LisaLotte05		Christian	1.29
Luke04	9	Michael	1.4
Luke07		Michael	1.46
Mareta07	11	Mareta	1.22
Mareta02		Mareta	1.2
Michael02	10	Michael	1.17
Michael10		Michael	1.13
Monty08	3	Ruben	1.51
Monty09		Carol	1.47
Ruben06	10	Ruben	1.1
Ruben09		Ruben	1.2
Stuart05	5	Stuart	1.31
Stuart08		Stuart	1.4
Trina04	11	Trina	1.18
Trina08		Trina	1.3
Wendy04	9	Wendy	1.42
Wendy08		AnneMarie	1.41

Figure C.2: Test Results - MFCC with 15 codewords

Model name	No Trained	Designated as	Score
Alan08	4	AnneMarie10	1.38
Alan03		Michael*	1.27
Anne03	11	Anne11	1.26
Anne09		Wendy03	1.32
AnneMarie01	11	AnneMarie03	1.05
AnneMarie07		AnneMarie09	1
Carol02	10	Carol03	1.14
Carol08		Carol03	1.08
Christian03	11	Christian02	1.08
Christian01		Christian02	1.16
Colin01	6	Colin03	1.26
Colin04		Colin07	1.04
Crystal02	6	Wendy11	1.42
Crystal06		Esther01	1.3
Edith05	10	Edith01	1.3
Edith09		Edith08	1.12
Esther02	10	Esther10	1.04
Esther07		Esther05	1.07
Henrietta05	11	Henrieta04	1.23
Henrietta08		Henrieta*	1.27
Karen03	7	Karen06	1.23
Karen10		Karen04	1.19
Lisa08	11	Lisa07	1.23
Lisa07		Lisa08	1.29
LisaLotte02	7	LisaLotte10	1.14
LisaLotte09		LisaLotte10	1.2
Luke04	5	Luke06	1.38
Luke09		Luke04	1.38
Mareta07	11	Mareta08	1.22
Mareta09		Mareta06	1.19
Michael10	10	Michael03	1.12
Michael07		Michael05	1.13
Monty	0		
Ruben03	10	Ruben05	1.07
Ruben10		Ruben06	1.13
Stuart01	4	Christian02	1.36
Stuart07		Stuart04	1.26
Trina07	11	Trina10	1.29
Trina02		Trina09	1.24
Wendy01	6	Henrieta06	1.34
Wendy03		Wendy06	1.29

Figure C.3: Test Results - MFCC with 20 codewords

Model name	No Trained	Designated as	Score
Alan02	8	Alan	0.85
Alan04		Alan	0.87
Anne03	11	Trina	0.69
Anne09		Anne	0.88
AnneMarie01	11	AnneMarie	0.6
AnneMarie04		AnneMarie	1.19
Carol02	10	Christian	0.54
Carol06		AnneMarie	0.5
Christian01	11	Colin	0.71
Christian04		Wendy	0.84
Colin05	8	Alan	0.71
Colin09		Alan	0.75
Crystal04	10	Colin	1.19
Crystal06		Mareta	0.8
Edith02	11	Crystal	0.95
Edith10		Christian	0.68
Esther06	10	Esther	0.54
Esther08		AnneMarie	0.7
Henrietta05	11	Michael	0.71
Henrietta09		Edith	0.54
Karen03	8	Karen	1.21
Karen07		Anne	0.88
Lisa03	11	Michael	0.75
Lisa01		Henrietta	0.68
LisaLotte06	10	Christian	0.77
LisaLotte02		LisaLotte	1.01
Luke04	10	Michael	0.74
Luke10		Monty	0.77
Mareta01	11	Mareta	0.68
Mareta08		Christian	1.42
Michael03	10	Michael	0.51
Michael05		Michael	0.5
Monty08	6	Mareta	0.74
Monty09		Monty	0.92
Ruben01	10	Carol	0.51
Ruben06		Christian	0.63
Stuart03	6	Stuart	0.65
Stuart08		Stuart	0.78
Trina04	11	Trina	0.93
Trina11		Michael	0.66

Model name	No Trained	Designated as	Score
Alan03	6	Colin	0.59
Alan08		Wendy	0.82
Anne01	11	Crystal	0.86
Anne05		Mareta	0.71
AnneMarie02	11	AnneMarie	0.54
AnneMarie09		AnneMarie	0.47
Carol06	10	AnneMarie	0.46
Carol07		AnneMarie	0.54
Christian04	10	Wendy	0.93
Christian08		Henrietta	0.49
Colin01	7	Colin	0.39
Colin05		Alan	0.67
Crystal05	8	Michael	0.8
Crystal02		Michael	0.99
Edith04	11	Anne	0.68
Edith09		Karen	0.68
Esther03	10	Edith	0.53
Esther07		Esther	0.49
Henrietta01	11	Wendy	0.38
Henrietta03		Henrieta	0.45
Karen02	8	Colin	0.88
Karen06		Colin	0.92
Lisa01	11	Henrieta	0.67
Lisa09		Ruben	0.9
LisaLotte01	9	Colin	0.92
LisaLotte11		Crystal	0.87
Luke02	9	Monty	0.84
Luke10		Monty	0.73
Mareta03	11	Henrieta	0.67
Mareta05		Luke	0.36
Michael09	10	Michael	0.55
Michael01		Michael	0.57
Monty10	3	Monty	0.85
Monty08		Monty	0.7
Ruben06	10	LisaLotte	0.57
Ruben03		Ruben	0.45
Stuart07	5	Michael	0.6
Stuart04		Stuart	0.8
Trina01	11	Trina	0.63
Trina05		Crystal	0.7

Model name	No Trained	Designated as	Score
Alan02	4	Alan	0.84
Alan08		Colin	0.9
Anne02	11	Henrieta	1.04
Anne07		Stuart	0.71
AnneMarie06	11	Karen	0.8
AnneMarie04		Trina	1.18
Carol01	10	Ruben	0.39
Carol09		Stuart	0.35
Christian05	11	AnneMarie	0.55
Christian09		Ruben	0.48
Colin02	6	Christian	0.69
Colin07		Colin	0.34
Crystal02	6	Michael	0.98
Crystal11		AnneMarie	1.34
Edith03	10	LisaLotte	0.62
Edith09		Henrieta	0.65
Esther05	10	Ruben	0.59
Esther09		Michael	0.8
Henrietta04	11	Mareta	0.77
Henrieta08		Henrieta	0.26
Karen03	7	Karen	1.2
Karen09		Karen	1.02
Lisa01	11	Henrieta	0.64
Lisa08		Christian	0.73
LisaLotte05	7	Alan	0.83
LisaLotte07		Ruben	0.59
Luke06	5	Henrieta	0.24
Luke10		Ruben	0.73
Mareta05	11	Henrieta	0.33
Mareta08		Lisa	1.36
Michael04	10	Michael	0.5
Michael10		Stuart	0.52
Monty	0		
Ruben02	10	Ruben	0.52
Ruben05		Michael	0.57
Stuart04	4	Luke	0.77
Stuart08		Stuart	0.71
Trina05	11	Trina	0.69
Trina09		Wendy	0.88

Model name	No Trained	Designated as	Score
Alan08	8	Christian	1.01
Alan04		Alan	1.03
Anne02	11	Anne	1.18
Anne10		Anne	1.03
AnneMarie04	11	AnneMarie	1.37
AnneMarie09		AnneMarie	0.98
Carol01	10	Christian	0.9
Carol05		Carol	1.11
Christian07	11	Christian	0.85
Christian08		Christian	0.92
Colin04	8	Colin	1.07
Colin09		Colin	1.15
Crystal03	10	Stuart	1.11
Crystal09		Crystal	1.15
Edith05	11	Henrieta	1.21
Edith06		Edith	1.08
Esther01	10	Esther	1.09
Esther02		Esther	0.98
Henrietta04	11	Henrieta	1.08
Henrieta09		Trina	1.12
Karen02	8	Esther	1.01
Karen07		Anne	1
Lisa08	11	Lisa	1.11
Lisa10		Lisa	1.2
LisaLotte05	10	LisaLotte	1.1
LisaLotte09		LisaLotte	1.13
Luke10	10	Christian	0.8
Luke02		Luke	1.04
Mareta05	11	Mareta	0.94
Mareta03		Mareta	1.13
Michael05	10	Michael	0.79
Michael02		Michael	0.8
Monty06	6	Crystal	1.01
Monty03		Monty	0.94
Ruben04	10	Ruben	0.98
Ruben09		Ruben	0.83
Stuart04	6	Christian	1.01
Stuart07		Stuart	0.73
Trina02	11	Trina	1.08
Trina04		Trina	1.13

Model name	No Trained	Designated as	Score
Alan02	6	Alan	1.01
Alan05		Alan	1.08
Anne03	11	Christian	1.12
Anne05		Anne	1.17
AnneMarie06	11	AnneMarie	1.04
AnneMarie11		AnneMarie	1.11
Carol03	10	Carol	0.9
Carol06		Christian	0.99
Christian02	11	Christian	0.83
Christian01		Christian	0.82
Colin04	7	Colin	1.01
Colin08		Colin	1.2
Crystal04	8	Anne	1.07
Crystal10		Christian	1.2
Edith09	11	Edith	0.99
Edith05		Henrieta	1.19
Esther06	10	Esther	0.94
Esther04		Esther	1.09
Henrietta04	11	Carol	1.04
Henrietta08		Christian	1.07
Karen03	8	Karen	1.27
Karen04		Esther	1.02
Lisa10	11	Lisa	1.16
Lisa04		Lisa	1.14
LisaLotte07	9	LisaLotte	1.04
LisaLotte05		LisaLotte	1.09
Luke03	9	Luke	1
Luke08		Luke	0.99
Mareta01	11	Mareta	1.09
Mareta06		AnneMarie	1.05
Michael03	10	Michael	0.75
Michael10		Michael	0.77
Monty09	3	Monty	1.11
Monty10		Crystal	1.2
Ruben03	10	Ruben	0.81
Ruben01		Ruben	1.01
Stuart04	5	Christian	0.98
Stuart01		Christian	0.99
Trina09	11	Trina	1.09
Trina10		Trina	1.1

Model name	No Trained	Designated as	Score
Alan03	4	Christian	1
Alan02		Alan	1.02
Anne07	11	Anne	1
Anne08		Anne	1.09
AnneMarie02	11	AnneMarie	0.81
AnneMarie09		AnneMarie	0.95
Carol03	10	Carol	0.89
Carol05		Carol	1.06
Christian05	11	Christian	0.9
Christian10		Christian	0.88
Colin04	6	Colin	0.97
Colin07		Colin	0.96
Crystal06	6	Christian	1.15
Crystal07		Christian	1.14
Edith07	10	Mareta	1.14
Edith02		Edith	1.23
Esther05	10	Esther	1.06
Esther08		Esther	0.94
Henrietta05	11	AnneMarie	1.1
Henrietta01		AnneMarie	1.11
Karen01	7	Karen	1.08
Karen09		Carol	1.05
Lisa02	11	Lisa	1.08
Lisa01		AnneMarie	1.19
LisaLotte06	7	LisaLotte	1.05
LisaLotte02		LisaLotte	1.09
Luke09	4	Christian	0.91
Luke10		Christian	0.7
Mareta02	11	Mareta	1.12
Mareta06		Mareta	1.16
Michael10	10	Michael	0.78
Michael02		Michael	0.77
Monty	0		
Ruben04	10	Ruben	0.87
Ruben10		Ruben	0.88
Stuart04	4	Christian	0.95
Stuart07		Christian	0.68
Trina04	11	Trina	1.05
Trina05		Trina	1.06

Test Model	BLScore	WLScore	Designation	Verdict
Whitelisted Models				
Alan02	1.42	1.41	Alan	White
Alan04	1.44	1.4	Alan	Grey
Anne02	1.31	1.32	Carol	Grey
Anne08	1.3	1.31	Anne	White
AnneMarie07	1.22	1	AnneMarie	White
AnneMarie02	1.3	1.02	AnneMarie	White
Carol02	1.23	1.14	Carol	White
Carol05	1.26	1.16	Carol	White
Christian01	1.23	1.16	Christian	White
Christian09	1.28	1.11	Christian	White
Colin02	1.3	1.19	Colin	White
Colin07	1.27	1.15	Colin	White
Crystal06	1.39	1.3	Esther	White
Crystal10	1.3	1.39	Christian	Grey
Edith09	1.25	1.12	Edith	White
Edith01	1.29	1.24	Edith	White
Esther02	1.18	1.04	Esther	White
Esther08	1.19	1.08	Esther	White
Henrieta03	1.31	1.21	Henrieta	White
Henrieta01	1.27	1.19	Henrieta	White
Karen03	1.3	1.23	Karen	White
Karen07	1.27	1.17	Karen	White
Blacklisted Models				
Lisa07	1.3	1.33	Christian	White
Lisa01	1.29	1.22	Esther	White
LisaLotte03	1.15	1.3	Carol	Black
LisaLotte10	1.2	1.22	Carol	Black
Luke06	1.39	1.43	Christian	White
Luke04	1.39	1.43	Carol	Grey
Mareta08	1.24	1.3	Esther	White
Mareta07	1.22	1.3	Anne	Black
Michael02	1.13	1.3	Colin	Black
Michael06	1.1	1.27	Christian	Black
Ruben03	1.07	1.18	Christian	Black
Ruben09	1.2	1.31	Christian	Black
Stuart04	1.34	1.32	Christian	White
Stuart07	1.27	1.39	Christian	White
Trina08	1.31	1.34	Christian	White
Trina01	1.24	1.27	Edith	White
Wendy03	1.3	1.33	Anne	Grey
Wendy09	1.46	1.4	Christian	Grey

Figure C.10: Test Results - Blacklist/Whitelist tests - MFCC/20

Test Model	BLScore	WLScore	WL Designation	Verdict
WhiteListed Models				
Alan04	1.05	1.03	Alan	White
Alan08	1.08	1.01	Christian	White
Anne02	1.23	1.17	Anne	White
Anne08	1.16	1.15	Anne	White
AnneMarie07	1.1	0.98	AnneMarie	White
AnneMarie02	1.1	0.88	AnneMarie	White
Carol02	1.01	0.93	Carol	White
Carol05	1.15	1.11	Carol	White
Christian01	0.91	0.84	Christian	Black
Christian09	0.98	0.83	Christian	White
Colin02	1.1	1	Colin	White
Colin07	1.22	1.07	Colin	White
Crystal04	1.23	1.11	Anne	Grey
Crystal09	1.23	1.15	Crystal	White
Edith09	1.14	1	Edith	Grey
Edith01	1.28	1.24	Edith	White
Esther02	1.14	0.98	Esther	White
Esther08	1.13	0.95	Esther	White
Henrieta03	1.16	1.13	Henrieta	White
Henrieta01	1.19	1.12	AnneMarie	White
Karen03	1.27	1.01	Esther	White
Karen07	1.17	1	Anne	White
Blacklisted Models				
Lisa07	1.14	1.22	Christian	Grey
Lisa01	1.27	1.27	Henrieta	Grey
LisaLotte03	1.39	1.36	Anne	White
LisaLotte10	1.1	1.09	Karen	White
Luke03	1.1	1.17	Christian	White
Luke06	0.86	0.8	Christian	Black
Mareta08	0.91	0.94	Anne	Black
Mareta07	0.92	0.98	Christian	Black
Michael02	0.8	1	Christian	Black
Michael06	0.84	0.88	Christian	Black
Monty06	1.1	1.01	Crystal	White
Monty10	1.31	1.28	Crystal	Grey
Ruben03	0.8	1.02	Christian	Black
Ruben09	0.84	1.08	Christian	Black
Stuart03	0.77	0.86	Christian	Black
Stuart07	0.73	0.76	Christian	Black
Trina08	1.12	1.2	Henrieta01	White
Trina01	1.17	1.18	AnneMarie	White
Wendy02	1.2	1.26	AnneMarie	White
Wendy03	1.26	1.23	AnneMarie	White

Figure C.11: Test Results - Blacklist/Whitelist tests - PLP/10

Spoofed identity (WL)	Michael BLScore	PLP/10 BLModel	WLScore	WLModel
Alan	1.09	Luke	0.96	Christian
	1.1	Luke	1.09	Alan
Anne	1.01	Michael	1.05	Christian
	0.95	Michael	0.92	Christian
AnneMarie	0.98	Ruben	0.91	Christian
	0.91	Michael	0.95	Christian
Carol	0.86	Luke	0.76	Christian
	0.86	Luke	0.81	Christian
Christian	0.97	Ruben	0.95	Christian
	0.92	Michael	0.91	Christian
Colin	0.95	Michael	1.01	Christian
	0.89	Michael	0.92	Christian
Crystal	0.99	Michael	1.04	Christian
	1.01	Michael	0.99	Christian
Edith	1.03	Ruben	0.99	Christian
	1.06	Michael	1.09	Christian
Henrieta	0.98	Michael	0.96	Christian
	0.94	Michael	0.98	Christian
Karen	0.9	Michael	0.93	Christian
	0.91	Michael	0.94	Christian

- caught by blacklist checks

Figure C.12: Test Results - Spoof test - Identity 1 with plp

Spoofed identity (WL)	Michael BLScore	MFCC/20 BLModel	WLScore	WLModel
Alan	1.21	Ruben	1.25	Christian
	1.32	Ruben	1.3	Christian
Anne	1.24	Michael	1.24	Christian
	1.05	Michael	1.11	Christian
AnneMarie	1.17	Michael	1.19	Christian
	1.07	Michael	1.09	Christian
Carol	1.24	Michael	1.26	Christian
	1.24	Michael	1.28	Christian
Christian	1.19	Ruben	1.16	Christian
	1.17	Ruben	1.18	Colin
Colin	1.28	Michael	1.29	Colin
	1.23	Michael	1.26	Christian
Crystal	1.33	Michael	1.39	Christian
	1.29	Michael	1.35	AnneMarie
Edith	1.24	Ruben	1.28	Colin
	1.28	Michael	1.24	Colin
Henrieta	1.27	Michael	1.31	Christian
	1.19	Ruben	1.24	Colin
Karen	1.21	Michael	1.3	Colin
	1.22	Michael	1.3	Colin

- caught by blacklist checks

Figure C.13: Test Results - Spoof test - Identity 1 with mfcc

Spoofed identities (WL)	Esther BLScore	PLP/10 BLModel	WLScore	WLModel
Alan	1.19	Esther	1.11	AnneMarie
	1.18	LisaLotte	1.12	AnneMarie
Anne	1.04	Esther	1.07	AnneMarie
	1.08	Esther	1.12	AnneMarie
AnneMarie	1.07	Mareta	1.08	AnneMarie
	1	Esther	0.9	AnneMarie
Carol	0.85	Luke	0.6	Karen
	0.86	Luke	0.56	Karen
Christian	1.01	Esther	1.06	AnneMarie
	1.04	Esther	1.12	AnneMarie
Colin	1.15	Mareta	1.11	Carol
	1.14	Esther	1.16	Karen
Crystal	1	Esther	1.17	Carol
	1.05	Esther	1.16	AnneMarie
Edith	1.12	Esther	1.11	Edith
	1.09	Esther	1.11	AnneMarie
Henrieta	1.11	Esther	1.14	Henrieta
	1.08	Esther	1.11	AnneMarie
Karen	1.11	Esther	1.06	Karen
	1.12	Esther	1.2	Carol

- caught by blacklist checks

Figure C.14: Test Results - Spoof test - Identity 2 with plp

Spoofed identities (WL)	Esther BLScore	MFCC/20 BLModel	WLScore	WLModel
Alan	1.24 Esther		1.2 AnneMarie	
	1.22 Esther		1.16	
Anne	1.1 Esther		1.08	
	1.15 Esther		1.12	
AnneMarie	1.04 Esther		1.01	
	1.02 Esther		1.08	
Carol	1.26 Esther		1.28 AnneMarie	
	1.17 Esther		1.14 Karen	
Christian	1.15 Esther		1.1	
	1.15 Esther		1.16	
Colin	1.16 Esther		1.2	
	1.22 Esther		1.22	
Crystal	1.14 Esther		1.15	
	1.13 Esther		1.2	
Edith	1.24 Esther		1.26 AnneMarie	
	1.08 Esther		1.13	
Henrieta	1.14 Esther		1.12	
	1.14 Esther		1.12	
Karen	1.17 Esther		1.14 Carol	
	1.2 Esther		1.23 Karen	

- caught by blacklist checks

Figure C.15: Test Results - Spoof test - Identity 2 with mfcc

Ruben BLScore	PLP/10 BLModel	WLScore	WLModel
Spoofed identities (WL)			
Alan	1.04 Michael	1.08 Colin	
	0.94 Michael	0.97 Colin	
Anne	0.94 Ruben	0.98 Christian	
	0.93 Michael	0.95 Christian	
AnneMarie	0.9 Stuart	0.88 Christian	
	0.86 Stuart	0.85 Christian	
Carol	0.77 Stuart	0.87 Christian	
	0.89 Michael	0.87 Christian	
Christian	0.92 Michael	0.87 Christian	
	0.9 Luke	0.88 Christian	
Colin	0.89 Michael	0.91 Colin	
	0.82 Michael	0.9 Christian	
Crystal	0.95 Michael	0.97 Christian	
	0.96 Michael	1 Christian	
Edith	0.99 Ruben	1 Christian	
	0.98 Michael	0.96 Christian	
Henrieta	0.95 Michael	0.94 Christian	
	0.92 Michael	0.97 Christian	
Karen	0.87 Michael	0.88 Christian	
	0.94 Michael	1.01 Christian	

- caught by blacklist checks

Figure C.16: Test Results - Spoof test - Identity 3 with plp

	Ruben BLScore	MFCC/20 BLModel	WLScore	WLModel
Spoofed identities (WL)				
Alan	1.14 Ruben		1.21 Christian	
	1.12 Ruben		1.23 Christian	
Anne	1.23 Ruben		1.25 Christian	
	1.17 Ruben		1.23 Colin	
AnneMarie	1.12 Michael		1.11 Christian	
	1.18 Ruben		1.19 Christian	
Carol	1.18 Ruben		1.22 Christian	
	1.18 Ruben		1.22 Christian	
Christian	1.15 Michael		1.15 Christian	
	1.13 Michael		1.22 Christian	
Colin	1.16 Ruben		1.23 Christian	
	1.14 Ruben		1.24 Christian	
Crystal	1.26 Ruben		1.24 Christian	
	1.22 Ruben		1.25 Christian	
Edith	1.27 Ruben		1.3 Colin	
	1.22 Ruben		1.25 Christian	
Henrieta	1.09 Ruben		1.17 Christian	
	1.17 Ruben		1.2 Christian	
Karen	1.22 Michael		1.25 Christian	
	1.2 Michael		1.33 Anne	

- caught by blacklist checks

Figure C.17: Test Results - Spoof test - Identity 3 with mfcc

Unknown Speakers	MFCC / 20				
	BL Person	BL Score	WL Person	WL Score	Identified as?
Sevan02	Ruben	1.2	Christian	1.24	Black
Sevan03	Michael	1.2	Carol	1.36	Black
Sevan04	Trina	1.19	Christian	1.18	Black
Sevan05	Ruben	1.18	Christian	1.15	Black
Martin01	Lisa	1.2	Carol	1.14	Black
Martin02	Lisa	1.18	Carol	1.08	Black
Martin03	Lisa	1.19	Carol	1.18	Black
Martin04	Michael	1.13	Carol	1.06	Black
Navar01	Ruben	1.2	Christian	1.19	Black
Navar02	Ruben	1.32	Anne	1.35	White
Navar03	Ruben	1.24	Christian	1.29	White
Navar04	Michael	1.13	Carol	1.2	Black

Unknown Speakers	PLP / 10				
	BL Person	BL Score	WL Person	WL Score	Identified as?
Sevan02	Michael	0.93	Christian	1	Black
Sevan03	Michael	1.01	Christian	1.09	White
Sevan04	Michael	1.01	Christian	1	White
Sevan05	Ruben	1.02	Christian	1.04	White
Martin01	Mareta	1.05	Anne	1.03	White
Martin02	Michael	1	Anne	1	Grey
Martin03	Mareta	1.02	Christian	0.94	White
Martin04	Michael	0.92	Christian	0.93	White
Navar01	Michael	1.09	Christian	1.04	White
Navar02	Michael	1.06	Christian	1.07	White
Navar03	Michael	1.04	Colin	1.08	White
Navar04	Michael	0.86	Christian	0.91	Black

Figure C.18: Test Results - Tests with unknown identities

Bibliography

[1] C. Becchetti and L.P. Ricotti. *Speech Recognition, Theory and C++ Implementation*. John Wiley and Sons, Bordoni, Rome, Italy, 1999.

[2] Graversen C. et al. Bonde C.S. Noise robust automatic speech recognition with adaptive quantile based noise estimation and speech band emphasising filter bank.

[3] Mike Brooks. Voicebox: Speech processing toolbox for matlab. `http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html`.

[4] Atal B.S. Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification. pages 1304–1311, Jan 1974.

[5] Reynolds D.A. Automatic speaker recognition : Current approaches and future trends. *Speaker Verification : From Research to Reality*, 2001.

[6] Atukorale D.A.S. *Self-Organizing Maps for Pattern Classification*. PhD thesis, University of Queensland, Australia, March 1999.

[7] S. Davis and P. Mermelstein. Comparison of parmetric representations for mono-syllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-28, No. 4:357–366, 1980.

[8] Hansen J.H.L. Deller J.R., Proakis J.G. *Discrete Time Processing of Speech Signals*. Prentice Hall, 1993.

[9] GFI Mail Essentials. *GFI Mail Essentials Mail Software Manual*, Feb 2006. `http://support.gfi.com/manuals/en/me11/me11manual-1-18.html`.

[10] Obermayer K. Graepel T., Burger M. Self-organizing maps: Generalisations and new optimisation techniques. April 1998.

[11] Paul Graham. A plan for spam. August 2002. `http://www.paulgraham.com/spam.html`.

[12] Paul Graham. Better bayesian filtering, Jan 2003. `http://paulgraham.com/better.html`.

[13] Mark Hasegawa-Johnson. Speech recognition acoustic and auditory model features. Lecture Slides, 2004.

[14] Michael Schmidt Herbert Gish. Text independant speaker identification. *IEEE Signal Processing Magazine*, pages 18–32, Oct 1994.

[15] Hynek Hermansky. Perceptual linear predictive analysis of speech. *Acoustical Society of America*, pages 1738–1748, April 1990.

[16] Gurle D. Hersent O., Petit J. *IP Telephony - Deploying Voice Over IP Protocols*. John Wiley and Sons, 2005.

[17] John-Paul Hosom. Features of the speech signal. Lecture Slides, April 2006.

[18] GAO Research Inc. Gao voice activity detector (vad). `http://www.gaoresearch.com`.

[19] Gunther Palm Jialong He, Li Liu. A discriminative training algorithm for vq-based speaker identification. *IEEE Transactions on Speech and Audio Processing*, 7, No. 4:353–356, 1999.

[20] Tomi Kinnunen. Spectral features for automatic text-independent speaker recognition. Master's thesis, University of Joensuu, Finland, December 2003.

[21] T. Kohonen. The self organising map (som). May 1998. `http://www.cis.hut.fi/projects/somtoolbox/theory/somalgorithm.shtml`.

[22] K. Ang Kwok and Alex C. Kot. Speaker verification for home security system. *IEEE*, pages 27–30, 1997.

[23] Yih Chuan Lin and Shen Chuan Tai. A fast linde buzo gray algorithm in image vector quantization. *IEEE Transactions on Circuits and Systems - 2: Analogue and Digital Signal Processing*, 45, No. 3:432–435, 1998.

[24] A. B. Y. Linde and R. M. Gray. An algorithm for vector quantization design. *IEEE Transactions on Communications*, pages 84–95, 1980.

[25] Ashraf W. Luthra A. Security in voip systems. Master's thesis, Technical University of Denmark, Kgs Lyngby, Denmark, Aug 2005.

[26] Sambur M. Selection of acoustic features for speaker identification. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-23, No. 2:176–182, Apr 1975.

[27] John Makhoul. Linear prediction - a tutorial review. *Proceedings of the IEEE*, 63, No 4:561–581, 1975.

[28] David Mertz. Spam filtering techniques. Sept 2002. `http://www.128.ibm.com/developerworks/linux/library/l-spamf.html`.

[29] K.K. Paliwal and N.P. Koestoer. Robust linear prediction analysis for low bitrate speech coding.

[30] Russo M. Patane G. The enhanced lbg algorithm. 2001.

[31] Prof. Bryan Pellom. Automatic speech recognition: From theory to practise. Lecture Slides, Sept 2004.

[32] Ramachandran R.P.; Farrell K.R.; Ramachandran R.; Mammone R.J. Speaker recognition - general classifier approaches and data fusion methods. *Elsevier Pattern Communication 35*, pages 2801–2821, Feb 2000.

[33] Magnus Rosell. An introduction to front-end processing and acoustic features for automatic speech recognition, January 2006.

[34] Kaare Sjolander. The snack sound toolkit. Website, 1997-2004. `http://www.speech.kth.se/snack/`.

[35] Kohonen T. Somervuo P. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letter*, 10, 1999.

[36] Bippus R. Stahl V., Fischer A. Quantile based noise estimation for spectral subtraction and wiener filtering. *IEEE*, pages 1875–1878.

[37] W. Tetschner. *Voice Processing.* Artech House Inc., Norwood, MA, 02062, 1993.

[38] Jo Twist. Dna analysis spots e-mail spam, Aug 2004. `http://news.bbc.co.uk/1/hi/technology/3584534.stm`.

[39] S. van Vuuren. Comparison of text-independent speaker recognition methods on telephone speech with acoustic mismatch. *Spoken Language, 1996. ICSLP 96. Proceedings*, pages 1788 –1791, 1996.

[40] Mark Ward. How to spot and stop spam, May 2003. `http://news.bbc.co.uk/1/hi/technology/3014029.stm`.

[41] Mark Ward. How to make spam unstoppable, Feb 2004. `http://news.bbc.co.uk/1/hi/technology/3458457.stm`.

[42] wikipedia.org. Stopping e-mail abuse, Aug 2006. http://en.wikipedia.org/wiki/Stopping_e-mail_abuse.

[43] Press W.H.; Flannery B.P.; Teukolsky S.A.; Vetterling W.T. *Numerical Recipes in C: The art of scientific computing*, pages 564–572. Cambridge University Press, 1988-1992.

[44] www.ai junkie.com. Kohonen's self organizing feature maps tutorial. Website. http://www.ai-junkie.com/ann/som/som1.html.

[45] E.J. Yannakoudakis and P.J. Hutton. *Speech Synthesis and Recognition Systems*. John Wiley and Sons, 1987.