A Dynamic Vocabulary Speech Recognizer Using Real-Time, Associative-Based Learning

By

Trevor Purdy

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2006

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Conventional speech recognizers employ a training phase during which many of their parameters are configured - including vocabulary selection, feature selection, and decision mechanism tailoring to these selections. After this stage during normal operation, these traditional recognizers do not significantly alter any of these parameters. Conversely this work draws heavily on high level human thought patterns and speech perception to outline a set of precepts to eliminate this training phase and instead opt to perform all its tasks during the normal operation. A feature space model is discussed to establish a set of necessary and sufficient conditions to guide real-time feature selection. Detailed implementation and preliminary results are also discussed. These results indicate that benefits of this approach can be seen in increased speech recognizer adaptability while still retaining competitive recognition rates in controlled environments. Thus this can accommodate such changes as varying vocabularies, class migration, and new speakers.

Acknowledgements

I would like to thank Professor George Freeman for his consultations on this research that helped shape it right from the beginning to the form it is today. Also this research was supported by both the Natural Sciences and Engineering Research Council (NSERC) and an Ontario Graduate Scholarship (OGS).

Table of Contents

List of Tables

List of Figures

1 Introduction

Humans do not always think logically. Why should we expect computers to? A question echoed, in spirit, from a recent seminar by Minsky[1] and also the underlying basis for the proposed automated speech recognizer (ASR) presented in this thesis. The scientific method requires a more definitive hypothesis that can be controlled and analyzed in an algorithmic manor. Thus logical thinking (or illogical) must be defined within the context of this thesis.

The Oxford Dictionary[2] defines the adjective logical as:

- Of or according to logic
- Correctly reasoned
- Defensible or explicable on the grounds of consistency
- Capable of correct reasoning

Within this context, the second and third definitions are most valuable. To alleviate interpretation ambiguity, it is necessary to further define logical thinking / processes as:

- Correct reasoning to provide an optimal solution
- Defensible or explicable on the grounds of consistently providing an optimal solution

It follows that illogical thinking is that which is incorrectly reasoned or is inconsistent in providing an optimal solution. This thesis uses the terms logical and illogical frequently and the reader is encouraged to remember their meaning is in context to avoid confusion. Within this context, it is clear that often humans do not think logically. A famous example of such behaviour is the travelling salesman problem.

Details may be found in numerous texts but briefly this graph problem requires a salesman, given a starting location, to visit every city / node in the connected graph. Each connection has a cost and each node has a reward associated with it. The question then is to find the optimal path through the graph to maximize the reward. Both brute force and more elegant algorithms exist for a computer to find this optimal solution consistently and therefore can be considered to employ logical thinking. However humans asked to solve this problem in a limited time and without the aid of a machine will very rarely find the optimal solution on the first try. On the average, their solutions are inconsistent and

sub-optimal implying they are illogical. However they are also very close to optimal and therefore can be used with little loss.

This illogical thought phenomenon is not unique to the travelling salesman problem but instead arguably permeates the vast majority of human thought. Call it instinct, unconscious thought, a hunch, or an educated guess, humans routinely use sub-optimal, illogical solutions / data to make decisions in day-to-day life. Still humans are able to solve countless problems effectively, if not optimally, and even perform complex tasks with ease that to date elude machines. It is possible for humans to use the optimal algorithms to generate solutions for their decision making processes. Given the choice however, most will favour using their original methods as the optimal solution is often unnecessary or, given time constraints, detrimental. As an example consider course corrections while driving a vehicle on a road. When making a turn, humans do not calculate the exact degree to turn the wheel to navigate but instead turn the wheel what they 'feel' is enough to remain on the road. If that correction was not accurate enough it can be adapted by feedback during the time the turn is in progress. If, however, an exact degree were to be calculated there would be two immediate problems. First the correction may need to be made quickly and such a calculation is difficult to perform and second the calculation would require exact information about the environment and vehicle (e.g., kinetic friction with the road) that would be unavailable.

In every scientific or engineering discipline it is easy to find areas where an estimate, prediction, sub-optimal, or otherwise illogical solution can be used to solve a problem and the results are still quite valuable. Yet when algorithms are written for a computer to solve a problem the general consensus is to attempt to provide the optimal solution using a logical consistent process under all conditions. That is a natural desirable goal. The closer we can get to an optimal solution the better. If the optimal can be achieved then why not achieve it? It is also a natural desire to implement a logically consistent algorithm. One unifying algorithm that can optimally solve a problem in any conditions is certainly of great value. Computers have been called logical machines, implemented via repeated use of simple logic functions. They lend themselves to consistent optimal logic processes. All these natural tendencies lead us to implement this type of aforementioned logical algorithms even though we ourselves generally use ***illogical*** thought processes.

Consider specifically the topic of this thesis, an automated speech recognizer, where humans outperform their computer algorithm counterparts. The human thought processes in this area are relatively poorly understood again adding another tendency away from modelling them directly when the current algorithms fail. Instead we are lead to look for

a better algorithm with more logical steps (e.g., LPC, Cepstrum) or to a better logical model (e.g., HMM, Neural Network). Though some success has been experienced in these pursuits, clearly humans still surpass computers at this task. The basis for the system described herein is the argument that these tendencies lead us from exploring an alternate solution based on the illogical human thought processes. Though it may sound to be an oxymoronic statement: By setting the goal of achieving an illogical ASR algorithm, it is proposed that this system will achieve better results than those based on achieving a logical solution.

Specifically, the ultimate goal of the proposed system is to create a speech recognizer that adapts to changes in its environment that include variations to its recognized vocabulary, changes to specific members of its vocabulary (class migration) and new speakers. It will also determine, and make decisions based, on its current environment rather than a general one. The solutions to the problems related to achieving these goals will be inspired by human behaviour as much as possible. The proposed system emphasises maximum customization. It is designed specifically to accommodate new words, new users, and make decisions based on the current environment as opposed to conventional ASRs that generally assume a general environment and treat new words and users as a problem. This allows not only tailoring of the ASR to a specific user and his/her needs but also allows greater recognition rates versus conventional counterparts in environments that greatly differ from an assumed general environment.

Section 2 contains a summary of the necessary background for the discussions in the remainder of the thesis. Any information in it that is directly pertinent to the proposed system will be repeated when needed in the remainder of the thesis. Thus those familiar with conventional automated speech recognition techniques may bypass section 2 and proceed directly to section 3 that discusses the precepts on which the system was defined and their motivation.

2 Background

Automated speech recognizers are a particular form of automated pattern recognition (henceforth called pattern recognition) and therefore share similar background, problems, and solutions. To develop a successful pattern recognizer, the following ***core recognition questions*** (referred to as core questions) must be answered:

1. What is the overall goal of the recognition system?
2. What are the individual patterns / objects that need to be differentiated to achieve that goal?
3. What features of the objects uniquely differentiates them from each other?
4. What automated mechanism can be employed to determine those features?
5. What automated mechanism can be employed to uniquely identify the objects based on the features?

The following subsections will provide the common answers research has provided to these questions for the task of automated speech recognition. This will also provide the necessary background to discuss the system described in this thesis.

2.1 The Dream and the Reality

When major research on ASRs began in the 1950s, it was believed by researchers that, given their present knowledge of acoustics and speech production, the phonetic transcription process could be mechanized and that conversion to conventionally written and spelt language would follow from this (i.e., to create a "vocal typewriter") [3]. This was then, and still is, the dream of ASR researchers as it has yet to become a reality. The reason for this is largely due to practical realities in speech recognition such as those described below.

Speaker-to-Speaker Variation – In general, no two people sound alike. The physical vocal tract as well as accents and styles vary between speakers. Therefore the acoustical data contains speaker dependent variables.

Individual Speaker Variation – An individual's voice does not remain consistent over a large portion of time. Even with careful pronunciation the following factors significantly influence the resulting acoustical output:

- ***Careless Speech*** - People tend not to speak any more precisely than necessary for comprehension. This can result in slurred words, adjacent words being connected acoustically together, and missed phonemes.

- ***Temporal Changes*** – Though often still a result of careless speech, temporal changes play a major role and deserve a category of their own. The duration of a word, its individual syllables, and even phonemes may change from pronunciation to pronunciation. Even with no slurring or acoustical connections as present in careless speech resulting in near perfect pronunciation, temporal changes still exist.

- ***Coarticulation*** - The context of an element of speech (i.e., adjacent phonemes and words) alters the pronunciation of that element. This can be explained by the physics that require smooth articulator (physical components of speech production along the vocal tract) changes. Also when a language is not strict in its pronunciation of sounds, it allows, for example, the speaker to choose to use nasalization or not. By allowing such freedom, the variations introduced by coarticulation are also is increased.

- ***Long Term Variation*** – Over time, an individual's voice changes due to such things as picking up a new accent or physical vocal track changes.

Phonetic Ambiguity – Many acoustical variables cannot be mapped one to one onto phonemes. Thus additional knowledge beyond acoustical data is required to uniquely transcribe phonemes from this data.

Noise and Other Interference Data – Only in extremely controlled environments can all external sounds be eliminated from acoustical speech data. This interference (other speakers and background noise) further obfuscates the desired acoustical data.

Though the vocabulary and environments of ASRs vary, the overall goal still remains as it was in the 1950's, to create a "vocal typewriter" that compares in functionality to that of humans. That goal now also requires the ASR to handle all these about ***core recognition problems*** (referred to as core problems). The solutions to these problems are very ASR specific and difficult to discuss in general. However when a particular technique is introduced that specifically deals well with one or more of these core problems it will be discussed.

2.2 Types of Automated Speech Recognizers

Although the second core question may seem simple, what are the individual objects to be differentiated, it still, to date, has no single accepted answer. One of the reasons for this is that its answer influences the degree that the core problems influence the overall design. Though others exist, conventionally two answers have been given to this question: words or phonemes. That is the individual objects are either all the words that make up an ASR's vocabulary or they are the all the phonemes that make up all the words of an ASR's vocabulary.

In order to mitigate many the practical problems presented, the individual objects are often chosen to be words. This choice is often associated with the first type of conventional ASR, the ***isolated word recognizer***. An isolated word recognizer recognizes words that are separated by distinct pauses in the acoustical data. This is accomplished by requiring the users themselves to insert distinct pauses between their words. By choosing individual objects to be words, phonetic ambiguity is eliminated and individual speaker variation is lessened by mitigating the effects of both coarticulation and careless speech. Also separating those objects with distinct pauses eliminates word-to-word coarticulation. Combining these choices makes locating the words in an acoustical data stream an easy task. This type of recognizer is often chosen for new research proposals as it eliminates many potential problems just by design.

For large vocabularies, the number of words, and hence the number of objects to be differentiated, can become prohibitive to process and store necessary data for in isolated word recognizers. This motivated many researchers to choose phonemes as the individual objects as their number is significantly smaller than the number of words for large vocabularies. The processing power and memory capacity of today's computers allows for much larger vocabularies making this motivation less relevant than in the early days of research. However it still remains a popular choice today due to its ease in representing large vocabularies.

The second type of conventional ASR, the ***continuous speech recognizer***, is often associated with choosing phonemes as the individual objects. A continuous speech recognizer operates on normal, unaltered speech acoustical data. That is there are no distinct pauses between words or any other form of manipulation before processing by the ASR. Thus this type of ASR processes standard human speech patterns and as a result, unlike the isolated word recognizer, does not mitigate any of the core problems by

these choices and must handle them all. This is essentially the true form of the "vocal-typewriter" dream.

Although these are the main two types of ASRs, it is important to note that many variations on these types exist. For example isolated word recognizers may use phonemes as the individual objects (or another choice all together). It is also possible to create a continuous speech recognizer with word level objects. In fact the system described in this thesis is one such variation. It is also possible to divide these types further into talker dependent and independent varieties. As the name implies, talker dependent varieties are customized to a single user whereas talker independent ones make no such assumptions. Note however that despite these variations that do exist, when referred to henceforth in this thesis, it is assumed that they will be as described above for convenience.

2.3 Feature Selection

Probably the most key question, and unfortunately also the most difficult, is core question three, what features of the objects uniquely differentiates them from each other. If this question can be answered conclusively for any recognizer, it is fair to consider a major portion of it is complete. In the ASR area, the process of determining distinguishing features for a particular recognizer is called feature selection. It has been the focus of much study for since the advent of ASR research. As such it is a rich area and far too large to describe in detail within this thesis. Instead, a high level overview of the common features employed in conventional ASRs will be provided.

Though the selection of features used may depend on the type of ASR to be developed, conventional ASRs tend to draw mainly from those below. The development of most of these features has been motivated by human physiology. These motivations will require knowledge of human speech production and comprehension. The interested reader is encouraged to consult a reference[5] on that subject, as only brief background of the specific motivation will be provided for each feature. A working knowledge of signal processing and frequency domain concepts[6] would be valuable and is assumed. Also note that when referring to frequency domain responses and transfer functions, it is intended to mean that an appropriate Fourier transform has been applied.

Amplitude over time (or ***power over time***) – For anyone familiar with the appearance of graphed sound wave data, the use of amplitude or power as a feature comes as little surprise. Even with common knowledge of human physiology, a person can be trained to

identify words merely based on the graphical appearance of their sound waves[4]. Thus as amplitude makes up a key component of that graphical appearance, it naturally became an early feature used in ASRs. Studying human physiology lends more support to its use as a feature. Human speech can be largely divided into two major categories: voiced and unvoiced. Technically speaking, ***voiced*** speech is characterized by deterministic waveforms whereas ***unvoiced*** speech corresponds to stochastic waveforms. Without delving into extensive details on the meanings of such definitions, in general voiced speech is generated by passing air though vocal cords, that are under sufficient tension to vibrate, and having a relatively obstruction free vocal tract. Unvoiced speech, however, is characterized by a major obstruction in the vocal tract and sound in generated by forcing air though that obstruction. The word 'see' contains both voiced and unvoiced regions, the 's' being unvoiced whereas the 'ee' being voiced. As intuition would suggest from this, voiced speech is louder and contains more power than unvoiced thus making amplitude a good distinguishing feature. Though exceptions exist, most vowels are voiced and most consonants are unvoiced further emphasizing the use of amplitude as a feature.

Low & high frequency power – This feature is again motivated by human physiology, though this time both the ear and vocal tract. The ear contains a sensitive construct able to extract frequency content from incoming sounds waves thus implying that, on some level, humans use frequency content of sound to classify it. Further if we investigate the physics of sound generation, one can find that voiced speech contains more low frequency energy than unvoiced speech (due to the vibration of the vocal cords) and unvoiced speech contains more high frequency energy than voiced (due to a major constriction combined with an almost white-noise source). Thus by using low and high frequency power as features it is often possible again to distinguish between voiced and unvoiced speech (especially helpful in the determination of fricatives from other types of speech[3]). The specific division between high and low frequency is arbitrary however a typical division might be low frequency is everything below ~1300Hz and high frequency is everything above.

Zero crossing rate – As its name implies, the zero crossing rate is a count of the number of times a sound waveform crosses the zero amplitude level over a specific region of interest. This is another measure of frequency content of the data but is not as precise or detailed as a specific measure such as a Fourier transform would provide. The zero crossing rate is more often used in word spotting (to be discussed in the next sub-section) but can also be used to identify types of speech. For example, low energy, high frequency

dominated speech will have a high zero crossing rate compared to high energy, low frequency dominated speech (again corresponding to the above categories).

Formant frequencies – One of the most key developments in identify phonemes and speech types in general, are the format frequencies (referred to as formants). Basically, consider the vocal tract as a sound propagation structure that contains resonance (or, if preferred, harmonic) frequencies. Resonance frequencies are those that are local maxima in the structure's frequency domain transfer function (i.e., frequencies that are attenuated the least by the structure compared to their adjacent frequencies). Those frequencies, for the vocal tract, are considered formant frequencies (technically the formant frequencies are usually defined as only those that appear in generated sound waves and not all the resonance frequencies of the structure but that requires source modelling and is insignificant for the discussion at hand). By altering the shape of the vocal tract to produce different phonemes, the formants are also altered. Thus if a mapping could be made between resulting formats and intended phonemes, and that mapping is not ambiguous, the formant frequencies could be very valuable in distinguishing phonemes and other parts of speech. Indeed such a mapping can be formed for voiced phonemes making formants a favoured feature for ASRs. A similar mapping exists for unvoiced phonemes but is rather ambiguous and therefore much less valuable. However it is still employed in unvoiced regions because, although complete classification is not possible, valuable information is still present and can be combined with additional features for classification.

Fourier spectrum – Often employed as a mechanism to extract formant frequencies and frequency power measures, the Fourier, or frequency, spectrum is a powerful tool for an ASR designer. Some of its functionality has been replaced by linear predictive codes, but still it is unsurpassed for direct frequency measurements. In addition to aiding in calculating the earlier frequency measures, other detailed frequency information that is specific to identifying certain phonemes or parts of speech can be extracted from a frequency spectrum. Since it is largely believed that human ear's process speech primarily based on frequency content, using similar frequency content gained from a frequency spectrum makes for good feature candidates. However even using the Fast Fourier Transform, computing the frequency spectrum can be processor intensive if repeated too often or on too large a block of data. Thus although a good source of features, care must be taken in the use of the Fourier transform in ASRs.

Linear predictive code parameters – The formants are valuable because they provide information on the location of articulators along the vocal tract and that provides

information to identify the phoneme being uttered. However it often proved challenging to extract formants directly from a frequency spectrum (in some cases they are not even present) and in the case of some parts of speech the formants were insufficient for identification. For these reasons the formants alone were often insufficient to uniquely (or even relatively) identify the location of articulators along the vocal tract. Thus more information about how the vocal tract affects the resulting acoustic wave was needed. To provide this information, a mathematical model of the vocal tract was developed and this model lead to the linear predictive code (LPC) parameters becoming features. The model's development leading to LPC parameters is too complex to describe in detail but the general steps are beneficial to understand and are described follows:

1. The vocal tract itself is a curved, complex chamber that is difficult to analyze. For simplification, begin by modelling it as a series of connected cylindrical tubes of varying diameters, as the physics of sound propagation along such structures is well known. The diameters represent the size of the opening at their respective points along the vocal tract. Note that these diameters vary based on articulator positions and hence if the output can be determined as a function of the diameters, and if the mapping is sufficiently unambiguous, the articulator positions can be inferred from them.

2. By applying boundary conditions between each of the connected cylinders and making assumptions about the beginning and ending boundaries of the overall structure, create a system of equations to solve for the reflection and transmission coefficients at each boundary.

3. Using the reflection and transmission coefficients, create an overall transfer function for the structure. This function can be treated as a linear filter. Note that this transfer function is an all pole model.

4. To proceed several assumptions are needed. First assume the nasal cavity is decoupled from the primary vocal tract. That is no sound is generated through the nose as otherwise the cylindrical model is inaccurate. Second assume the source excitation of the vocal tract can be approximately modelled by a weighted delta function. This is required so that the generated output contains no zeros and that all the poles in the acoustic output are from the transfer function. Finally assume that the acoustic output is relatively ergodic over the region of interest. This assumption is relatively equivalent to assuming that the articulators move relatively slowly (ideally not at all) over the region of interest. This is necessary

so that a data-average over several sample of acoustic output with the same articulator configuration is equivalent to a time average over a single acoustic output with that configuration, as the former is hard to obtain.

5. With all the assumptions, incorporate the source delta function and finally note that the all pole model is in a format that may be approximated by a pole estimation technique in digital signal processing known as linear predictive coding. Thus the LPC parameters from that method can be related to the transmission and reflection coefficients and they, in turn, can infer information about the diameters or articulator positions, as desired.

The astute reader familiar with human speech production and LPC methods will realize several problems with the development presented. First the assumption that the nasal cavity is decoupled from the primary vocal tract is not necessarily true and often varies from dialect to dialect. By introducing the nasal cavity, zeros and additional poles are present in the transfer function and the pole estimates based on an all pole model will be inaccurate and further the number of poles themselves underestimated. Second assuming the source excitation can be modelled by a weighted delta function is only generally true in unvoiced regions of speech. However the later assumption that the articulators are slow moving over the region of interest is only generally true in voiced regions of speech. Thus the assumptions are conflicting. The LPC method is very poor when handling noise, which is a core problem of ASR's discussed earlier. To function optimally, LPC methods require accurate modelling, pole estimates, and a high signal-to-noise ratio, all of which has been brought into question.

These above issues plague the accuracy of the LPC method in its articulator position estimation task and yet it remains a powerful tool that has proven repeatedly to increase the accuracy of any ASR that uses their parameters as features[4]. Thus despite theoretical problems, the LPC parameters are valuable features and are still commonly used today. It is important to note also that since the poles of the output function represent maxima points on its frequency spectrum, the pole estimates are also estimates of formant frequencies, and the LPC method is also often used to estimate formant frequencies.

Cepstrum (or mel-cepstrum) parameters – The cepstrum parameters are similar to the LPC parameters in many ways and in fact can be derived from them. However their merits become clearer when viewed from a frequency domain approach. Before proceeding, recall that when developing the LPC parameters it was necessary to assume that the source was approximately a delta function so that it did not influence the output.

That is, it was required so that the output reflected the vocal tract transfer function, as that was the desired information.

As with the LPC parameters, consider the vocal tract as a linear filter being excited by some source. Let the source frequency response be ***S(w)*** and the filter transfer function be ***H(w)***. Thus the output frequency response, ***O(w)***, is

$$O(w) = S(w)H(w) \tag{2.1}$$

The desired information on the vocal tract is present in ***O(w)*** but has been mangled by multiplication by the unknown frequency response of ***S(w)***. Instead of assuming a transfer function, as was done with the LPC parameters, calculate the following:

$$\text{Log } |O(w)| = \text{Log } |S(w)| + \text{Log } |H(w)| \tag{2.2}$$

Using the properties of logarithms, the multiplication that was difficult to work with has been changed into an addition, which now allows the use of linear operators. During voiced speech, the vocal cords open and close at a rate known as the ***pitch frequency***. The pitch frequency can be measured and / or theoretically modelled to gain some information about the nature of ***S(w)***. Of import to this discussion, during voiced speech the source time domain function is approximately periodic with a rate at the pitch frequency (it is approximately periodic as naturally voiced speech does not continue indefinitely). Thus ***S(w)*** has its major contributions at multiples of the pitch frequency. Theoretical models (like the connected cylindrical model) show that the vocal tract transfer function, ***H(w)***, is relatively flat and changes much slower than the pitch frequency. Thus computing the "frequency response", ***C(s)***, via a Fourier transform of the frequency response Log $|O(w)|$ will separate the slow moving Log $|H(w)|$ from the fast moving Log $|S(w)|$. If the non-zero regions of ***C(s)*** do not overlap, it will be trivial to separate the information resulting from the vocal tract from the information resulting from the source. Thus the desire to obtain information directly about the vocal tract that in turn provides information about the articulator positions has been achieved. The values of the function, ***C(s)***, are defined as the cepstrum parameters. On an interesting note, cepstrum is merely the word spectrum with the first 4 letters flipped. A wide variety of such interesting terms exist in the area of cepstrum parameter concepts.

Though the discussion was motivated around obtaining information about the vocal tract transfer function from the acoustical output, the cepstrum parameters are also used to obtain information about the source as well as many other elements of interest. Their

independence from assumptions on the source model has proven effective, as many ASR's performance has been improved when utilizing them over their LPC counterparts. They are therefore valuable features.

The mel-cepstrum parameters are a modification of the cepstrum parameters based on studies in an area called psychoacoustics. In a superficial overview, the human auditory system does not perceive frequency in a linear fashion and thus the perceived frequency of a sound is not linearly related to its actual frequency. The mel is the unit of measurement of this perceived frequency. The mel-cepstrum coefficients, then, are simply cepstrum parameters altered by an appropriate filter to correspond more closely to how humans perceive frequency. That is they undergo a form of hertz to mel conversion. This process apparently improves recognition performance [4].

It is important to note that although the definition of ***C(s)*** was given to be the actual definition of the cepstrum parameters, in reality the definition involves an inverse Fourier transform and not a direct one. Although mathematically this is equivalent for ASR work as the source and vocal tract functions are always real, it should be noted for completeness. Also there exist an alternate set of cepstrum parameters, called complex cepstrum parameters that are very similar to the ones presented here but offer different benefits largely in computation efficiency under certain circumstances. They will not be presented here.

2.4 Feature Extraction

Given sufficient knowledge of the features from section 2.3 and the data to evaluate them on, computing their values is straightforward. However core question four, what automated mechanism can be employed to determine those features, is still not necessarily routine.

Consider the following general problem to motivate further discussion: Given a sample of raw acoustical data, how would you identify the word(s) present in it? Initially you would have to determine if any words were present in the data at all and, if so, where they were located. In order to determine the value of a feature, the ASR has the same problem. It needs to locate the appropriate data to calculate it on. The process of locating the appropriate sound data within an acoustic data stream, performing any necessary filtering, and evaluating features on it is called feature extraction, and is the answer ASR's provide to the fourth core question.

The first task is to separate the regions of data that contain only background noise from those that contain speech. The algorithms to do this are called word-spotters. They are both quite numerous and often tailored to the specific ASR. For example the process of word spotting in an isolated word recognizer is trivial due to the introduction of pauses between the words. However for a continuous speech recognizer, the arrival of words is random with no assumptions of pauses allowed. Combining that with the core problems of coarticulation and careless speech that can merge adjacent words in a data stream make word spotting much more challenging in this mode. The reader may wonder if a similar task of phoneme spotting exists for ASRs. Indeed it does but is much rarer as, under many methods, spotting phonemes requires identifying them, which is equivalent to automated speech recognition itself.

Rabiner and Sambur developed a popular algorithm for word spotting[7] often employed in many types of ASRs. This method combines the use of two of the features discussed earlier: power over time and zero crossing rate. The algorithm follows:

1. Compute the zero crossing rate and power of time measures every 10 msec on frames (blocks of data) of length 10 msec.

2. Search the power data, starting from the earliest, to find the first crossing of some predefined threshold, ***PUpper***.

3. From the crossing of ***PUpper***, "back down" to the nearest crossing of another predefined threshold, ***PLower***. Call this crossing ***S***.

4. From ***PUpper***, proceed forward to the nearest crossing of ***PLower***. Call this crossing ***E***. ***S*** and ***E*** are the tentative end points of the word.

5. Examine the zero crossing rate before ***S***. If, within 25 frames, the zero crossing rate exceeds a predefined threshold, ***ZC***, at least 3 times, adjust ***S*** to be the beginning of the earliest frame that the zero crossing rate exceeded ***ZC***.

6. Repeat step 5 for ***E*** moving ahead instead of back.

7. ***S*** and ***E*** are now the predicted endpoints of the word.

The above algorithm has proven effective. It still however fails in instances when words are merged so closely together that appropriate crossings of ***PLower*** cannot be located. There is no single perfect algorithm to handle all cases. In some instances, however, this particular problem of word merging can be handled at a later point via a decision-mechanism correction.

Establishing the location of the words in many instances is insufficient to begin computing features. Recall the core problem of temporal variation. Using features such as amplitude over time or zero crossing rates, for example, require fairly accurate timing measurements to be of use. For example consider two different utterances of the word "seven." The word "seven" is notorious in digit recognizers for being a word that can be one syllable or two when pronounced. Obviously the two-syllable version will be quite longer and therefore have quite different power over time measures than the shorter version. Thus power over time, in this instance, has become useless as it varies from utterance to utterance. This temporal variation problem was investigated for quite some time before a very popular algorithm to mitigate it was introduced: dynamic time warping.

The ***dynamic time warping*** algorithm (also known as simply time-warping) is a method of temporally aligning two signals in a least energy fashion within a certain degree of allowed warp, or range of allowed variation. The details of the algorithm can become complex and what follows is merely an overview example of its use in temporally aligning a word in an ASR for clarification. Note that time warping can also be used on the phoneme or other levels.

1. Isolate a single word via a word spotting algorithm in an acoustic signal. Call that isolated word the candidate signal.

2. Using a pre-recorded set of template signals, at least one for each word the ASR recognizes, the time warping algorithm compares the candidate signal to each of the templates in the following manner:

    a. Each value of the template is compared against an allowed range of values of the candidate centred on the same time instance as the value of the candidate signal being compared. That is if, say, the candidate value being compared was sample 5 and the allowed range was 2, all values of the template between sample 3 and 7 would be compared.

b. The best match, under some distance measure, within the range is chosen as the new value of the candidate at that time instant. The distance measure is also retained. This proceeds for all time instances.

3. A new set of signals, one for each template, are now available that represent the best match possible between the candidate and the template within the allowed range, point-by-point. Based on the retained distance measures, the best of these new signals (i.e., the one with the smallest total distance measure to the candidate) is chosen. This chosen signal is the temporally aligned version of the candidate.

The candidate signal that originally represented the word can now be replaced with the new temporally aligned, signal chosen by the time-warping algorithm again making time-dependent features relevant. On an interesting note, since time warping compares a signal to a series of templates the ASR recognizes, it can be used as a recognition process by itself. In fact it often has been and has proven effective under certain circumstances in this capacity in addition to just temporal alignment.

Another common filter used to handle individual speaker variation is a normalizer. Separate utterances of the same word often vary significantly in overall power due to how loudly the talker uttered the word. Clearly this should not affect recognition. A simple normalizer that scales the data to a standard range before evaluating features solves this form of variation. Also utilized is a sample-by-sample differencer. This has the effect of eliminating any DC bias due to windowing or sampling and also provides a high frequency emphasis that can prove numerically beneficial, as high frequency powers can be quite small compared to low frequency counterparts during certain types of speech.

For completeness of the feature extraction discussion, several features discussed earlier only function properly in either voiced or unvoiced regions of speech. The reader can verify, by using discussions of human physiology, that the word-spotting algorithm presented could easily be modified to differentiate voiced from unvoiced regions. Therefore such detection is possible and may prove beneficial depending on the features employed. Despite that fact however it is important to note that conventionally ASRs do not attempt to identify the voicing type before evaluating said features.

2.5 Decision Mechanisms

The final core question, what automated mechanism can be employed to uniquely identify the objects based on the features, is essentially a question of discovering a mapping between feature values and the individual objects recognized by the ASR. Ideally the mapping between feature values and individual objects should be one to one. However the mapping is usually more complex. The implementation of such a mapping in an ASR is called a decision mechanism.

Before discussing specific decision mechanisms, a special mode of ASR operation used to develop decision mechanisms should be introduced: ***training mode***. Unlike regular operation where it is the intention of the ASR to classify acoustical data, during training mode the ASR already knows the correct classifications of the acoustical data and instead its goal is to develop the decision mechanism (i.e., feature mapping) to allow regular operation to occur as accurately as possible. How this decision mechanism is developed naturally depends heavily on the mechanism itself and will be discussed within that context. Conventionally, ASRs have completely separated training and regular modes. That is, training does not occur simultaneously with regular operation and usually only occurs once during the initial design. The system proposed in this thesis, however, differs from convention by attempting to combine these modes.

It is not possible to create a simple linear representation of a mapping that is ambiguous. However it is possible to model it using probabilities. That is it is not possible to state, for example, that feature set A implies the object is B but it is possible to say that feature set A implies the object is B 80% of the time. ASR's that have decision mechanisms based on this principle are called ***Bayesian recognizers***. Assume a Bayesian recognizer uses feature set, $\boldsymbol{F}$, and recognizes objects from the set $\boldsymbol{O}$. During training this recognizer develops a set of conditional probability mass functions, one for each possible feature $\boldsymbol{F}_{\mathbf{A}}$ in $\boldsymbol{F}$ of the form $P(\boldsymbol{F}_{\mathbf{A}} \mid \boldsymbol{O})$. After training, using the well known Bayesian property of conditional probability, the recognizer develops either a joint probability mass function between all the features and the object, $P(\boldsymbol{F}, \boldsymbol{O})$, or it develops a set of probability mass functions, one for each object $\boldsymbol{O}_{\mathbf{A}}$ in $\boldsymbol{O}$ of the form $P(\boldsymbol{O}_{\mathbf{A}} \mid \boldsymbol{F})$. During regular operation the Bayesian recognizer uses these developed probability mass functions to perform its decision making task. It uses the probability mass functions to classify an acoustical sample as the object with the highest probability, given the feature values for that sample (i.e., the highest $P(\boldsymbol{O}_{\mathbf{A}} \mid \boldsymbol{F})$).

There are many types of Bayesian recognizer in addition to the one described above, which is known as the maximum a posteriori type. The others, however, in general behave in an identical manner and have similar features and drawbacks. Bayesian recognizers are very effective if and only if the underlying features are chosen well. If sufficient features are present to perform the recognition task within a desired degree of accuracy, Bayesian recognizers are fast, simple and will perform well. If however the features are insufficient, the Bayesian recognizers will not accommodate that insufficiency and the overall performance will be poor. Note that there are also a class of recognizers known as quasi-Bayesian. These again are usually based on probabilities but, as their name implies, their implementations vary.

Feature selection is probably the hardest task to perform when designing a successful ASR and the Bayesian recognizers have no leniency towards poor features. Since the problem of automated speech recognition is not solved (save for very under constrained conditions) and therefore no 'perfect' set of features have been identified, it is natural that to some degree all feature selections are insufficient. Thus if the decision mechanism could help accommodate for such insufficiency, it would help alleviate some of the difficulty of feature selection. Two such decision mechanisms are commonly used in modern ASRs: Artificial neural networks and hidden Markov models. Like many other areas in speech recognition, both artificial neural networks and hidden Markov models are very rich and far too complex to describe accurately within this thesis. Further since neither of these will be utilized by the proposed system, they are not within its context either. An overview of each will be presented, which is by no means complete, that will provide the reader with an understanding of their functionality and intended use.

A human neural network is an interconnected network of small elements called neurons. Communication is achieved through both electrical and chemical interactions along the connections between neurons. Adjusting the strength of these connections can alter the interaction. ***Artificial neural network*** (referred to as neural networks) are mathematical models that attempts to mimic human functionality. They use artificial neurons (called nodes) and use connections that can be altered by positive and negative weights attached to them further mimicking the excitatory and inhibitory actions of our mind. Figure 2.1 shows an example of a neural network.

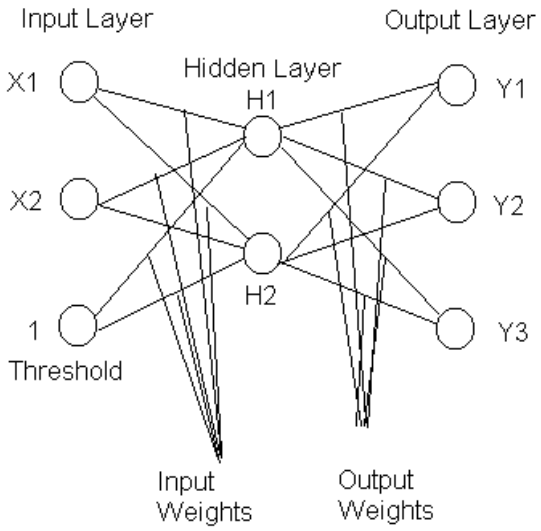

Figure 2.1 - Three layer feed-forward neural network

Of important note from the diagram is that $\boldsymbol{X}_{\mathbf{A}}$, $\boldsymbol{H}_{\mathbf{A}}$, and $\boldsymbol{Y}_{\mathbf{A}}$ refer to input, hidden, and output nodes respectively. The meaning and number of the input and output nodes is determined by the designer and the system being modeled. The number of hidden nodes is arbitrary and related to the accuracy of the model. The lines connecting the nodes are weights and represented by positive and negative real numbers. The value of each hidden node is calculated by a weighted sum of the input nodes using the input weights to the hidden node in question. Often a non-linear function is applied to the sum before assigning it to the hidden node. Similarly output values are calculated as a weighted summation of the hidden nodes using output weights. Again a non-linear function is often applied to the output value. The weights are the only adjustable parameters in the network and they determine what function the network models. It is known that this type of network (i.e., three-layer, feed-forward network with a threshold input node always of value 1 and containing non-linear adjustment functions), can model any deterministic mathematical function.

As with ASR's, neural networks have a training mode. In this case training refers to the process of making the network model the desired system. The conventional algorithm used to train neural networks is back propagation. To train a network using this algorithm, sample input and valid output must be provided. The input is fed into the network and its output compared with valid output. The error in the output is simply the

magnitude of the vector difference of these values. The weights are adjusted to minimize this. Typical operation of a neural network consists of a training phase to emulate model a system and then using the network model that system[18].

When used as decision mechanisms for ASRs, a conventional approach is to create a neural network model of each of the individual objects the ASR recognizes. The inputs to these networks are the feature values and the output is typically some measure of accuracy to indicate whether the input features values correspond to the object the network is modelling. The intention is that by incorporating all the features together to generate a model during training, the model itself contains additional information not readily apparent in the features themselves. It "fills in the gaps" between the features and their values. Thus in some instances even with insufficient features, the model can accommodate the missing information and thereby increase overall recognition rates. Neural networks have been used successfully as described above in many ASR environments.

Hidden Markov models (HMM) are in many ways similar to neural networks. In fact it can be shown (with some effort) that for any HMM, a neural network exists that is completely equivalent to it. Unlike neural networks however, HMMs are not motivated by human physiology but instead by Markov models themselves from probability theory. Consider a state machine. If the next state of the machine is completely statistically determined by knowledge of the present state and a probability function of the form P(next state | current state), then the state machine is said to be Markovian. That is, knowledge of previous states before the current state does not impact the statistical likelihoods of the next state.

Now consider the same state machine but further assume that there is no method of determining the current state. Instead only an output function that is based on the present state is measurable. Thus the output function itself is not Markovian but is a function of the underlying state of a state machine that is Markovian. Thus the output function is considered to have a hidden Markov model. This is a loose definition in a mathematical sense but does convey the key point: The output itself is not Markovian but is based on a hidden underlying state machine that is Markovian.

Again similar to neural networks, when using HMMs in ASRs, a typical approach is to create an HMM for each object the ASR recognizes. The HMM is intended to model that object and, as such, requires a training phase. Once trained, the HMM is used as a decision mechanism by taking extracted features and attempting to determine how likely

it is that each HMM generated said features. A process known as the Viterbi algorithm is a commonly used for this task. Thus again the model is intended to “fill in the gaps” between the features to increase recognition rates.

It is extremely difficult to demonstrate with decent support that the speech production system contains an underlying Markov state machine and thus the assumption that individual speech objects can be modelled by HMMs is largely unsupported. However despite this the HMM is extremely common in modern ASR research to the point that it is rare to see other models used. It has both speed and convenience advantages over the more general neural networks (though it does lose modelling capability). Also unlike neural networks where oft times the connecting weights are meaningless, if one designs an HMM well the equivalent “weights” can have some physical significance. It has even been credited with the ability to incorporate the effects of dynamic time warping thereby making that time consuming algorithm unnecessary when using HMMs. Simply put HMM models are effective and will not disappear easily from ASR research.

This concludes the discussion on the background ASR material. Although this background is interesting and used in many ASRs, only a small subset of it is actually utilized in the proposed ASR as it is based on a very different set of precepts than conventional ASRs. To continue the discussion, the precepts themselves will be introduced next in section 3.

3 Defining Precepts

The precepts underlying the speech recognition system presented were conceptualized based on high-level human thought patterns. However it is of value to examine a consequence first before proceeding.

Note that the reader interested specifically in the novel elements of the proposed method should pay particular attention to this section as the theoretical basis on which the system was constructed is given by the 11 precepts defined in this system. Each one individually varies to different degrees to their conventional ASR counterparts and on the whole result in defining a fairly significantly different system. Specifically of interest should be the precepts based on feature space concepts (1 – 3) and those used to guide real-time feature selection (4 – 9).

3.1 Feature Space Limitations

To facilitate this discussion, consider the following definitions relevant to an ASR:

A ***class*, C_i,** is defined as a unique, atomic unit that can be recognized by the ASR. Atomic in this instance refers to the fact that it is the smallest unit of speech used to construct all words in the ASR vocabulary (e.g., word, phoneme).

A ***vocabulary***, ***V*,** is defined as the set of all classes that the ASR recognizes, represented as $V = \{C_1, C_2, \ldots C_N\}$

A ***feature***, F_i, is defined as some form of processing that can be evaluated on any member of the ASR's vocabulary ***V***.

Modern ASRs distinguish between members in their vocabulary, ***V***, by evaluating some number of features on sample data and, based on the feature values and a decision mechanism, choose a member of ***V*** as the most likely candidate. This can be viewed as a mapping between the ASR's feature space and the vocabulary space. It can also be viewed as a ***feature constellation***, which has a parallel definition to a ***signal constellation*** in communication systems[8], the hyper-dimensional axis in this case being feature values rather than transmitted bit values. Generally the attempt is to make this feature-to-vocabulary mapping *many* : 1. It is most valuable in this format as it implies that given

feature values for a sample a unique class can be accurately predicted from them by a proper decision mechanism.

Figure 3.1 shows a representation of a 2-dimensional feature constellation. It represents two features, feature 1 and feature 2, being used to distinguish 2 classes, class 1 and class 2. Based on the values of feature 1, plotted along the x-axis, and feature 2, plotted along the y-axis, the class corresponding to those feature values can be determined if they fall within one of the circular class regions. Note however that those class regions overlap and, hence, the mapping between feature values and classes is not *many* : 1.

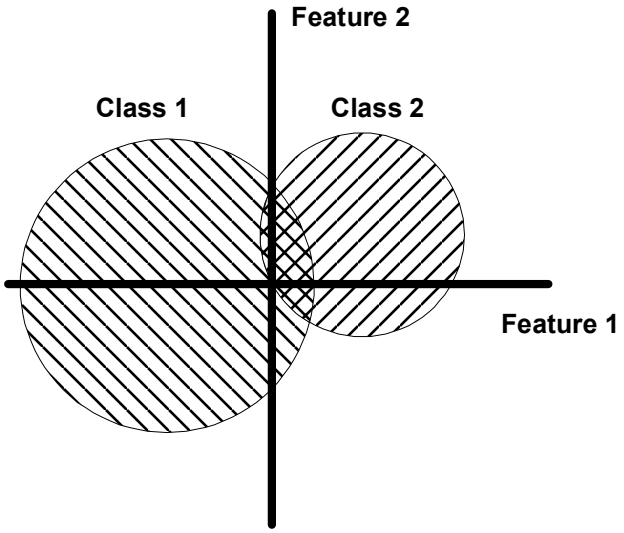

Figure 3.1 - Feature constellation of two classes differentiated by two features

Given a vocabulary, $\boldsymbol{V}$, (and a decision mechanism is some instances) it is theoretically possible to determine a set of features that can distinguish uniquely between the classes of $\boldsymbol{V}$ to any desired degree of accuracy. Determining those features is the heart of the ASR design problem[3] and, although theoretically possible, is in practice very challenging meeting with many practical issues[4]

To represent this, the following notation is useful:

$$\{\boldsymbol{C}_{\mathbf{i}} \mid \boldsymbol{V}\} = \{\boldsymbol{F}_{\mathbf{a}}, \boldsymbol{F}_{\mathbf{b}}, \ldots, \boldsymbol{F}_{\mathbf{n}}\} \tag{3.1}$$

This is a symbolic way of representing that class, $\boldsymbol{C}_{\mathbf{i}}$, may be distinguished from all other members of $\boldsymbol{V}$ by the features $\boldsymbol{F}_{\mathbf{a}}$, $\boldsymbol{F}_{\mathbf{b}}$, …, $\boldsymbol{F}_{\mathbf{n}}$. In other words, $\boldsymbol{F}_{\mathbf{a}}$, $\boldsymbol{F}_{\mathbf{b}}$, …, $\boldsymbol{F}_{\mathbf{n}}$, represent the

feature constellation axis for $\boldsymbol{C_i}$ – the theoretically determinable features that can uniquely distinguish $\boldsymbol{C_i}$ given $\boldsymbol{V}$.

When and how the various $\{\boldsymbol{C_i} \mid \boldsymbol{V}\}$'s of an ASR are determined constitutes the major differences between traditional ASRs and the one presented in this thesis. Conventional ASRs determine their distinguishing features during the process generally referred to as training and do not subsequently alter those choices during normal operation despite potential changes to its operating environment[3]. This thesis' ASR, conversely, determines these features as part of its normal operation and further allows those choices to be refined or completely altered based on its environment. This small paragraph hides the widespread implications these changes imply on the ASR design. Thus this section is entirely devoted to establishing those precepts that form the basis of this thesis' ASR.

It should be noted that there are instances in which some conventional ASR do change their feature selection during normal operation. For example some employ methods of eliminating corrupt features (or portions of them) and some do not use all their features for every classification if them deem them unnecessary. These approaches, while effective, are geared to the task of eliminating one of more features and are not intended to completely select new features for a particular ASR environment, independent of the selections for all other environments, as this thesis' ASR is.

Consider a simple ASR vocabulary with word level atomic units,

$\boldsymbol{V} = \{\boldsymbol{C_1}, \boldsymbol{C_2}, \boldsymbol{C_3}\}$,
$\boldsymbol{C_1}$ = "one", $\boldsymbol{C_2}$ = "two", $\boldsymbol{C_3}$ = "throughout"

and with the following features defined:

$\boldsymbol{F_1}$ evaluates the sample utterance time duration
$\boldsymbol{F_2}$ extracts the first phoneme and evaluates its average amplitude

By inspection one can write the desired *many* : 1 mapping feature constellation axis as:

$\{\boldsymbol{C_1} \mid \boldsymbol{V}\} = \boldsymbol{F_2}$
$\{\boldsymbol{C_2} \mid \boldsymbol{V}\} = \boldsymbol{F_1} + \boldsymbol{F_2}$
$\{\boldsymbol{C_3} \mid \boldsymbol{V}\} = \boldsymbol{F_1}$

C_1 can be distinguished by F_2 as it is the only word in V that begins with a vowel implying its first phoneme has larger amplitude than the other two classes.

C_3 can be distinguished by F_1 as it has the longest duration of the classes in V.

C_2 requires both F_1 and F_2 as it is neither unique in first phoneme amplitude nor sample duration but is uniquely determined when the features are combined.

Now further consider adding the new class, C_4 = "tree", to V. $\{C_1 \mid V\}$ and $\{C_3 \mid V\}$ can remain unchanged. However both F_1 and F_2 take on similar values for C_2 and C_4 thus requiring a new feature be added to distinguish them.

This example is quite simple but yet demonstrates very interesting results:

1 The features required to uniquely identify a class are dependent upon the vocabulary of the ASR

2 The features required to uniquely identify a class are not necessarily the same for all classes belonging to the same vocabulary

3 Larger vocabularies tend to require more features to uniquely identify their classes. The number of features required never decreases (unless the original features are changed) hence the number of features required for increasing vocabulary size forms a non-decreasing sequence. This is similar to requiring more bits to transmit a larger symbol set over a communication channel (i.e., more information is needed to uniquely identify a symbol). The use of the word require here is key. It is possible for a small vocabulary to employ more features than necessary and thus a superset of that vocabulary may not ***require*** more features than were used in the original vocabulary. In fact it may require less features than were originally employed. The number of ***required*** features, however, never decreases without changing the features themselves.

4 Though not directly demonstrated here, the feature set that forms a vocabulary's feature constellation is not necessarily unique. In this example the original V was distinguished with F_1 and F_2 however other features such as dynamic time warping[9] or the cepstrum coefficients[10] could be employed to great accuracy.

These results are generally true and can be proven under certain assumptions. However they are instinctively correct and a bit of thought will verify that they are correct under conditions of interest.

To re-emphasize, determining features is the key problem in ASR design. Uniquely distinguishing feature sets theoretically exist but are practically very challenging to identify. If we now consider the infinite set of possible candidate features that are potentially members of one of these distinguishing feature sets, these above results can establish limitations on the search for these features. Conversely conventional ASR systems (commercial and research) tend to employ far more restrictive limitations.

1 Result 1 implies that different vocabularies can employ different distinguishing features. Unless tailored for a very specific task, conventional ASRs largely draw from the same set of features and are used regardless of vocabulary. The most common of these recently are the mel-cepstrum coefficients. However even if a non-standard set is employed, the following remains true - During operation of the ASR, changes to the vocabulary (e.g., adding words, removing words, temporary reduction due to a pre-emptive "fast-match") do not change the features that are used for distinguishing among the modified vocabulary members. Requiring the use of the same features for all vocabularies the ASR operates on is a more restrictive limitation than result 1 requires.

2 Result 2 implies that an ASR may employ different features to identify different classes. In a similar argument to result 1, conventional ASRs do not do this again resulting in a more restrictive limitation than result 2 requires.

3 Result 3 implies that larger vocabularies require larger feature sets. This is more of a guide than a limitation. If a feature set is good for a certain vocabulary, a superset of that vocabulary wanting to employ the same features will likely require additional features to achieve the same accuracy. Successful small vocabulary ASRs have been documented in many papers and are in commercial use. Large vocabularies ASRs often are still based on the same features but do not show as great a performance. This intuitive result explains this behaviour suggesting more features are needed for a larger vocabulary.

4 Perhaps the most key result, result 4 states that distinguishing features for a particular vocabulary are not necessarily unique. Therefore the infinite candidate feature set almost certainly will contain more than one subset that can uniquely distinguish a

given vocabulary. Though no attempt will be made to prove this statement, it is reasonable to assume a large number of such sets exist given an infinite feature space and a finite vocabulary. If the distinguishing set was unique, the further restricted the search through this candidate space, the better. However since a large number of distinguishing sets exist, only necessary restrictions should be employed as further restrictions may eliminate viable candidates.

Another mathematical approach to this analysis is possible. Consider a vocabulary, $\boldsymbol{V}$, and assume the theoretical $\{\boldsymbol{C}_i \mid \boldsymbol{V}\}$'s exist but are unknown. Define the notation:

$$\|\{\boldsymbol{F}_a \mid \boldsymbol{C}_i, \boldsymbol{V}\}\| = \begin{cases} 1, \text{if } \boldsymbol{Fa} \text{ is a member of } \{\boldsymbol{Ci} \mid \boldsymbol{V}\} \\ 0, \text{Otherwise} \end{cases} \tag{3.2}$$

$$\|\{\boldsymbol{F}_a \mid \boldsymbol{V}\}\| = \frac{\sum_{Ci \in V} \|\{\boldsymbol{Fa} \mid \boldsymbol{Ci}, \boldsymbol{V}\}\|}{\|\boldsymbol{V}\|} \tag{3.3}$$

Equation 3.2 is an indication function of the presence of a feature, $\boldsymbol{F}_a$, in a member of a distinguishing feature set for the class, $\boldsymbol{C}_i$. It follows then that equation 3.3 represents the relative number of occurrences of $\boldsymbol{F}_a$ in the distinguishing sets for the entire vocabulary, $\boldsymbol{V}$, as a percentage of total number of elements in the vocabulary (i.e., the relative prevalence of a particular feature in the distinguishing features of the vocabulary).

Conventional ASRs attempt to find a single set of distinguishing features for the entire vocabulary. Due to a large vocabulary size and a large number of potential features, manual consideration of each feature is prohibitive requiring potentially millions of individual evaluations. Therefore automated selection mechanisms were created such as the add-in and knockout algorithms[11]. The details of these algorithm, and others of their kind, are beyond the scope of this thesis. Essentially these algorithms successively add (or remove) features to a base feature set one at a time and, depending upon the results of using the ASR with the modified feature set, determine if that feature that was added (or removed) is valuable.

Define T, the desired accuracy threshold of the ASR, as a number between 0 and 1. Assume the ASR in question has achieved this desired accuracy by employing features that are a subset of a theoretically optimal distinguishing feature set. In other words all

features that have been selected for use are members of the same optimal feature set and, along with additional features, will form a distinguishing set for this vocabulary. It seems reasonable to assume that at least T% of the optimal features have been selected. It also seems reasonable to assume that at least T% of the vocabulary can be uniquely distinguished. Both these statements are incorrect however.

Assume $\boldsymbol{F}_{\mathbf{a}}$ is a member of the theoretical optimal feature set. There are now two cases to consider:

Case 1 - $\|\{\boldsymbol{F}_{\mathbf{a}} \mid \boldsymbol{V}\}\| << (1 - \mathrm{T})$: Since $\boldsymbol{F}_{\mathbf{a}}$ is only required to distinguish a very small percentage of the classes in the vocabulary, it is not necessary for this feature to have been selected to achieve the desired threshold and is unlikely to be selected by the add-in or knockout algorithm families.

Case 2 - $\|\{\boldsymbol{F}_{\mathbf{a}} \mid \boldsymbol{V}\}\| \leq (1 - \mathrm{T})$ or $\|\{\boldsymbol{F}_{\mathbf{a}} \mid \boldsymbol{V}\}\| > (1 - \mathrm{T})$: Since $\boldsymbol{F}_{\mathbf{a}}$ is at least on the order of $(1 - \mathrm{T})$ it would seem that in order to achieve T, $\boldsymbol{F}_{\mathbf{a}}$ must be selected. However consider the optimal hyper-dimensional feature constellation for $\boldsymbol{V}$. Each class in this feature constellation has a region that is uniquely mapped to itself to achieve the *many* : 1 mapping. Now consider the achieved hyper-dimensional feature constellation for the selected features and assume $\boldsymbol{F}_{\mathbf{a}}$ has not been selected. It is now required that classes that included $\boldsymbol{F}_{\mathbf{a}}$ have overlapping regions in the reduced dimension feature constellation. If, as a percentage of total occurrences, the feature values that lie in these overlapping regions represent less than $(1 - \mathrm{T})$% of the sample data, it is not necessary for this feature to have been selected to achieve the desired threshold. If the features have been pre-whitened[12], then this can also be viewed as a direct volume ratio between overlapping region volume to total volume.

In both cases, the reduced dimension feature constellation must have overlapping regions. The actual achieved dimensions could be significantly reduced from the optimal dimensions so long as $\|\{\boldsymbol{F}_{\mathbf{a}} \mid \boldsymbol{V}\}\| << (1 - \mathrm{T})$ for all removed features or the resulting overlapped region is ‘small’ enough. Therefore many features could be missing and the first assumption made that at least T% of the optimal features have been selected is not necessarily true. Also it is possible that all classes contain at least some of their region in the overlapped and again this negates the second assumption, that the selected features can uniquely distinguish at least T% of the vocabulary.

Without these features, the resulting mapping has become *many* : *many* and no matter the decision mechanism employed, resulting members of the vocabulary are unable to be

distinguished uniquely, or even distinguished at all potentially, for all samples. Thought will show problem may be exasperated by skewing the training set towards relative prevalence of the vocabulary in the language versus a uniform distribution and also by a potential shift during actual use to increased prevalence of samples in the overlapped region. Also further remember that the assumption was made that the selected features were a part of some optimal set. If this is not true then potentially very few, or even none, of the optimal features have been selected. This implies these non-optimal features need to be removed and other features are needed and yet the desired threshold could still be achieved with the selected features.

Ultimately achieving a threshold is a double-edged result. Achieving the threshold indicates a certain level of performance has been met. This is naturally a good result. It also implies an optimality and sufficiency of the features that have been selected that is not necessarily true. Unfortunately other evaluation mechanisms are unavailable. It is not possible to determine by other means whether or not the features selected are theoretically optimal. However care must be taken to avoid assumptions on the optimality of the selected features.

Eliminating these problems reduces to the problem of finding the theoretical optimal features again. However by attempting to find a single set of optimal features for the entire vocabulary, these problems are made more challenging. If features were selected and evaluated on a class-by-class basis, it would no longer be possible that sections of the vocabulary be completely missed and still achieve a threshold. Each individual class would instead be required to reach that threshold avoiding the amalgamation phenomenon that tends to filter out poor performance in individual classes. Further by not limiting the search space, theoretically optimal features for each class will be easier to find and it will be easier to evaluate their effectiveness. Any well-behaved signal can be represented by a Taylor series, including a pure sinusoidal signal. The sinusoidal signal, however, requires an infinite number of terms to represent. If, however, a Fourier series is allowed for that sinusoid, only a signal term is required. Similarly, in theory, the same set of features can represent all classes, but by allowing features to vary from class to class, the individual representations become easier to find and evaluate. The same results presented in the simple example earlier follow from these arguments.

To summarize, the results from the simple example outline certain ***necessary and sufficient*** conditions for choosing feature sets given a vocabulary. Conventional ASRs generally employ a set of ***sufficient*** conditions that are more restrictive than are required.

The analysis indicates that less restrictive sets are better. This leads to the first defining precepts of this ASR:

Precept 1: Features may vary from vocabulary to vocabulary.

Precept 2: Features may vary from class to class within the same vocabulary.

Precept 3: A mechanism to evaluate features on a class-by-class basis should exist.

3.2 Feature Selection

Training of an ASR is responsible for its ultimate success or failure. It is a very time consuming and processing intensive task. The precepts just defined increase the magnitude of this task by not just requiring features to be selected once for an ASR but instead for each vocabulary the ASR must deal with and further each class within each vocabulary. Further complicating this task is an issue briefly mentioned earlier: By allowing the ASR user to add and remove words from the vocabulary, the vocabulary will change. By employing potential fast-match algorithms or other pre-processing filters to eliminate certain vocabulary possibilities during the processing of each sample, the vocabulary will also change for that sample. This implies that feature selection, something normally done during the training process, must be performed during run-time operation and potentially even within the processing of a sample. For guidance on this task we look to the only widely successful implementation of such a process: the human mind.

3.2.1 Associations

Quickly, what is the letter before 'u' in the alphabet? A surprising number of people need to run through the alphabet in their minds to answer that question. If you, the reader, are one of them have no fear. There is nothing wrong with you. You did employ however a powerful mechanism the human mind uses to find information – the ***association***. Suppose instead the question was, what follows after 'hijklmnopqrs'? That question has the same answer but is much easier for most people. Having learned the alphabet starting with 'a', each letter has become associated with the letter before it. So the sequence 'hijklmnopqrs' is associated with 't'. The details of the process in the mind are a mystery (The interested reader should refer to some relevant background information by Myers[17]

and a discussion of those concepts as they pertain to human cognition by Nearey[5]), but in a similar way to how a hyperlink functions on the Internet, associations provide references from some source information (sequence, object, phrase, or many other forms) to somehow related (sometimes strangely) destination information. Many forms of associations exist and many types of information can be associated. Think of the words of a song, the colour blue, or even reading this thesis. With thought, all these objects remind us of many things. One could say that the human mind can incorporate associations for everything and would be arguably correct. Within the thesis' context, that statement is not particularly useful and therefore a formal definition of an association is in order.

An a***ssociation*** is a reference between one object**,** called the ***associated source***, and another other object, called the ***associated destination***. Any information object may be associated with any other information object. Therefore the associated source and associated destination can be any information object of interest. The following notation is defined to express this relationship:

associated source → associated destination (3.4)

It is important to note that a → b does not mean b → a. That is object 'a' can be associated with object 'b' without object 'b' being associated with object 'a'. The relationship is one way. Also associated source object 'a' can be associated with many different associated destination objects simultaneously (i.e., a → b, a → c, etc) but each association only expresses one single one-way relationship. Multiple individual associations are needed to establish one to many relationships. Finally the associated source and associated destination are often referred to as simply the source and destination though their meaning should be clear from context. These associations are loosely related on a high level to their data-mining counterparts[13].

Humans use associations for everything. How are they used? The answer comes from this question: When presented with a problem, how do you solve it? There are two cases: a familiar problem and a new problem. If it is a familiar problem the question of how associations are used has already been answered: The problem has already been ***associated*** with a similar problem in the past and therefore you can solve it with the same method as before. If it is a new problem, then invariably (directly consciously or not) you ask yourself: What does this problem remind me of? What parts of it seem similar to something I've done before? If and when those questions can be answered, you have determined a potential method of how to solve this new problem and, again, the question of how associations are used has been answered. In both cases associations are used to

choose actions, make decisions, or determine how to process the given problem. ***Associations are used to guide cognitive processes***.

As an example, consider a human classification task of recognizing a tree. Although much of this exact procedure is unknown in detail, what is presented here is a high-level overview of the procedure and a sample of how associations could play a role in its analysis as follows:

- Raw scene data is provided by the eyes to the image processing areas of the brain

- After some unknown pre-processing, borders of shapes are identified based on associations from past experience indicating that, for example, sharp changes in texture or colour indicate a border.

- Based on the border information, distinct shapes are classified based on associations to past experience and knowledge of the shapes. The tree shape may be identified directly or in a multi-step manor such as identifying a rough cylindrical shape below a spherical shape. At this stage other classifications are possible based on alternate associations to the object shape(s).

- Using these possible classifications, checks may now be performed to verify if the classification is correct. For example, trees are associated with the colours brown and green, the texture of bark, the leaves, the rustling sounds in the wind, etc, etc. These associations can guide the processing to check for the existence of those conditions and if found will be able to identify the object as a tree and if not another classification can be investigated by its relevant associations.

This example shows the potential power of the concept of associations guiding cognitive processing. The image recognition task of classifying an object can be guided to the correct answer purely with associative reasoning. The human mind does this so quickly and without conscious effort on our part making it is impossible to verify if this is the actual procedure. Earlier arguments suggest that the mind can use a form of association for everything. Conscious thought often shows the use of associations in this manor and examples of this were given. Therefore it is plausible that the low-level subtasks that make up a higher-level task such as pattern classification are also association based. ***Human pattern classification is association based***. Combining this with the earlier statement, humans use associations to guide cognitive processes, yields: ***Associations are used to guide the cognitive processing involved in human pattern classification.***

This section began in an attempt to discover how humans evaluate incoming acoustic data to perform speech recognition in an attempt to determine a method of ASR feature selection at run-time. For an ASR, the 'cognitive processes' are the decision mechanism and the features themselves. The next defining precepts follow:

Precept 4: Associations should guide feature selection.

Precept 5: Associations should guide the decision mechanism.

3.2.2 Association Rules

Before leaving the topic associations completely, a few guidelines should be established on their use. These guidelines will be developed by a series of exploratory examples based on human behaviour.

Example 1: Often during our lives we encounter new experience. In order for later analysis / problems to make use of this humans must be able to form associations to and from these new objects to existing objects. This implies the following:

Precept 6: Associations must be able to be created at run-time.

Example 2: Consider a student learning a mathematical task such as how to apply vector cross products (i.e., A x B, A = (a, b, c), B = (d, e, f)). Initially using an association of the multiplication symbol and another association to previous knowledge about vector addition may lead the student to the conclusion that this operation may be performed via term-by-term multiplication of the vectors. His / her mind would create an association between this problem (source) and the term-by-term multiplication solution (destination). This, of course, is incorrect however and eventually the student, by some form of feedback, would realize this and remove this association and replace it with a correct one. This leads to the following:

Precept 7: A feedback mechanism must exist to verify the accuracy of associations.

Precept 8: Associations must be able to be destroyed at run-time.

Example 3: Consider a person standing in front of a conventional oven with a red-hot burner. Due to past experiences, the following associations are defined in this person's mind:

Red → Hot
Red → Stop Signs
Red → Tomatoes
Conventional Oven → Hot
Food → Tomatoes
Car → Stop Signs

There are three conflicting associations that have red as a source and yet this person will chose the hot association correctly. To resolve this conflict, recall associations guide cognitive processes. Each conflicting association would have initiated a cognitive process for verification. In this example the other source conditions would not exist the local environment to support either the stop sign destination or the tomato destination exist thus resulting in choosing hot as the proper destination. Therefore locally relevant associations are chosen to guide the ultimate response.

Precept 9: A mechanism must exist to resolve conflicting associations based on how relevant an association is to current ASR conditions.

3.2.3 Elimination versus Classification

A subtle, yet important, distinction exists between elimination and classification in pattern recognition. Ultimately in order to classify a class as the proper classification all other classes must have been eliminated making the procedure of elimination versus classification identical. However humans when presented with an unfamiliar object are often unable to classify the object immediately but are still able to eliminate many possibilities. Further investigation of the unknown object may or may not ultimately yield a classification. However this shows that elimination often requires less information and is therefore easier than classification. ***Elimination tends to be easier than classification.***

The earlier precepts defined for the ASR requires feature selection to occur at run-time. They also require feature selection to be dependent upon the vocabulary that is allowed to change at run-time during the processing of a sample. Smaller vocabulary ASRs, as discussed earlier, require fewer features and tend to be more accurate. Finally elimination

tends to be easier than classification. Combining these statements implies that the ASR can choose features to eliminate classes in a vocabulary for a particular sample resulting in a smaller vocabulary. The smaller vocabulary will subsequently have new features chosen for it and evaluated. If the processing of the sample during the smaller vocabulary is independent of its previous processing in the larger vocabulary, then the processing can occur as if the smaller vocabulary was the original vocabulary with the effective result of the sample now being processed in a smaller vocabulary ASR. This argument may be applied repeatedly until the vocabulary cannot be reduced by elimination any further. In other words it has one remaining member in which case classification is complete.

Since elimination tends to be easier than classification and smaller vocabulary ASRs tend to be more accurate than large vocabulary ASRs, this method will tend to provide more accuracy than direct classification at the initial large vocabulary level would. This leads to the final precepts:

Precept 10: Feature selection and evaluation of a sample for a particular vocabulary must be independent of any previous evaluation of that sample for another vocabulary.

Precept 11: Features may be selected to eliminate, rather than classify, a sample for a particular vocabulary.

Having now developed the defining precepts, section 4 will discuss the actual implementation of these precepts and how they are combined to create the overall proposed ASR.

4 System Description

In order to meet the precepts of the preceding section an ASR must be able to adapt to its environment, which, in terms of an algorithm, often means many control statements and highly abstract processing until the present environment can be determined. The system described herein meets all those precepts but unfortunately is rather complex due to said adaptation requirements. To aid the reader in understanding this system, the description will begin by introducing an example that will be followed throughout the remainder of the system description. Next an overview of the system's structures will be introduced followed by pseudocode to represent its algorithms. Elaboration on any particularly complex pseudocode will be given and clarified with the running example. Further, some details of the implementation that are not unique or directly relevant to ASR tasks will be merely referenced so as not to detract from more important aspects of the system.

Note that up until this point, the discussion of class atomicity has been in general. This system's vocabularies are made up of word-level atomic classes. Although much of the discussion remains valid for other levels of class atomicity, for this system it is assumed that a class represents a single, unique word.

Also note that this section is written with the intention that a reader interested in duplicating the obtained results or using this system as part of their work / research has sufficient information to do so. The sections that elaborate on the algorithm details can be fairly challenging reads to the first time reader who does not have the details of the structures memorized. It has been attempted to be as clear and concise as possible without sacrificing completeness in these sections by repeating important information and providing examples. However for the reader not interested in the nuances of the actual implementation used for testing but instead interested in the concepts and the understanding of the system may wish to skim the algorithm elaboration sections (i.e., those actually labelled **pseudocode elaboration**) as they are generally not needed to understand the proposal.

Finally although many of the details for this system are necessarily different then their conventional ASR counterparts (or do not exist in conventional ASRs), the reader interested in the novel aspects of the design should focus specifically on the use of specific processing elements customized to particular environments (class set processors), how environments are determined during run-time, how associations to guide their real-time processes, and how those associations are created and destroyed based on feedback.

4.1 System Example

The example that will be used to clarify the system is its use as a simple digit recognizer (the vocabulary is {'***1***', '***2***', '***3***'}). Henceforth it will be referred to as the system example. The system example will follow the processing of a single data sample after the ASR has been in use for some time so that some structures and data are already present in the various pools. Figure 4.1 represents the current state of the system at the beginning of processing the new sample. In the descriptions, the data from this figure will be used to demonstrate the system objects and how the algorithms would operate on them to process a new sample. Note that while at least one of each system object is present in the figure, not all of the objects present are given detailed views (for example Class Set Processor ***CS('1','2')***). Those missing details will not prove critical to the discussions.

4.2 System Objects

To draw parallels to high-level human recognition processes discussed earlier, the system has been conceptualized into four pools of information (processes or data): The s***ystem***, ***cognitive, associative,*** and ***memory pools.*** Together these pools implement all the functionality of the ASR. Every object within each of the pools will be described and details given about their roles in the system. Further an overview of the algorithms contained within the pools will be given as well. Thus the reader is encouraged to use this section as a glossary when reading the algorithms. Also for each of the pool descriptions, a connection to a human recognition process presented in section 3 will be given. They may be interpreted as corresponding to their human counterparts. This interpretation may be helpful while reading the pool's algorithms by recalling what the corresponding human counterpart's functionality is and comparing it to what the pool's algorithm is doing. However, they may also be interpreted as mere container objects from an object-oriented programming point of view. Figure 4.1 shows a representation of all the objects in the system and should be referred to when reading about the individual structures.

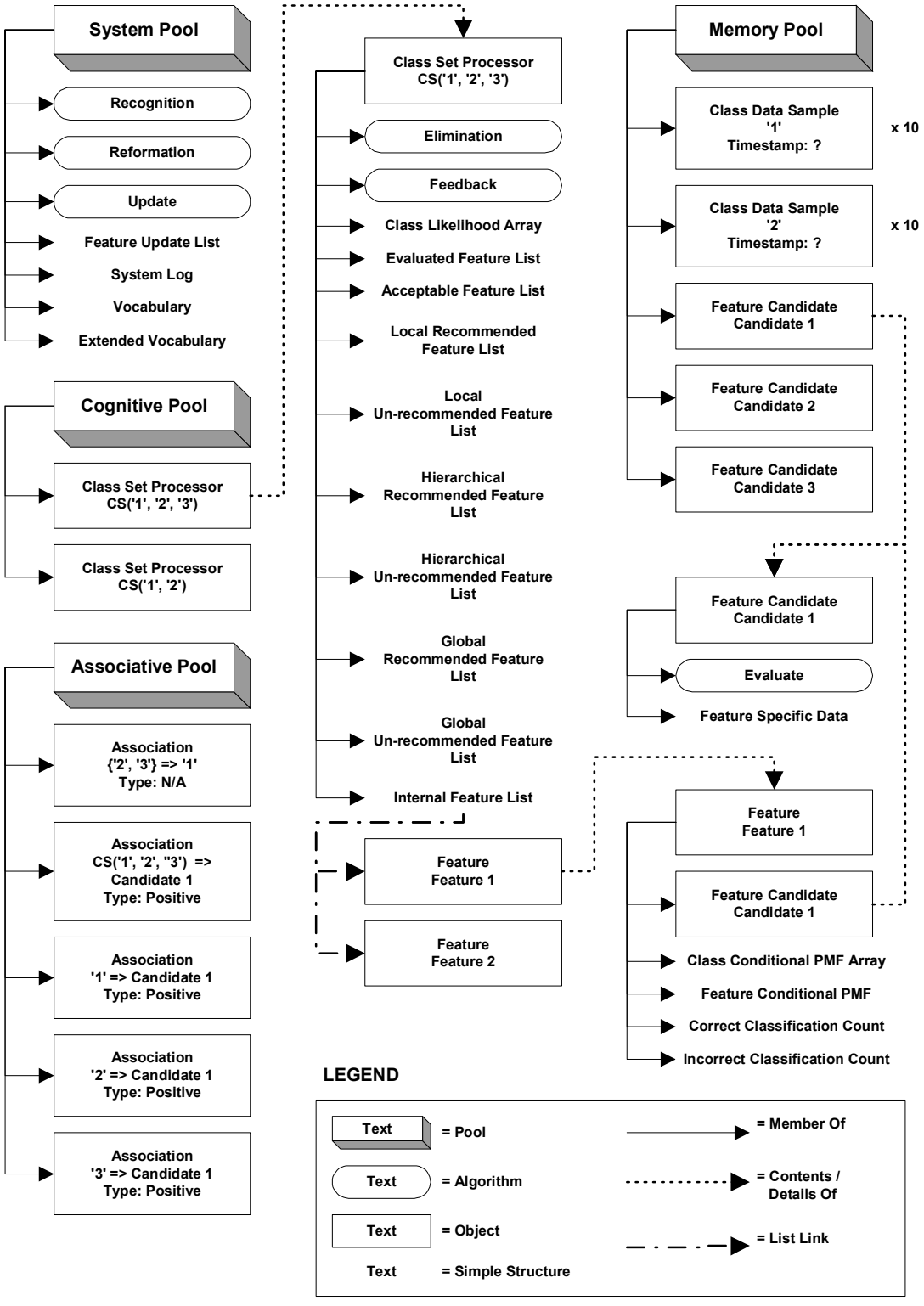

Figure 4.1 - Example of all the speech recognizer objects and their relationships

4.2.1 Memory Pool Objects

The ***memory pool*** stores raw data and processing elements. It stores raw data in the form of ***class data samples*** and it stores processing elements in the form of ***feature candidates***. Though conceptually the memory pool stores all members of the theoretically infinite candidate feature set discussed in section 3, in practice this is impossible and only a subset are present. When more data or features are needed within the other pools the memory pool is accessed.

The memory pool acts as a raw, general purpose, storage area. It has no specific knowledge about anything it is storing beyond an identifier for retrieval purposes and a time stamp. For class data storage, its human counterpart would be the area of the human mind that stores raw, past experience data (that is assuming of course that the human mind actually stores raw data without some pre-processing which is unknown). Recall the example of human image recognition of a tree given in section 3. An example of such data would be the raw tree data or raw leaf data. By storing a theoretically infinite set of processing elements, the pool also represents the mind's ability to create new methods of analysis at any time. Again from the human tree recognition example, such a new, potential processing element would be the creation of a processor that differentiates between bark and leaves by analyzing colour content. Whereas the human mind does not contain an infinite set of potential processors but instead constructs them as needed, the ASR's memory pool cannot create processors and thus simulates that effect by storing a large set of potential processors.

Each ***class data sample*** is a single sample of acoustic data previously processed by the ASR. It contains a unique class identifier obtained during processing and a time stamp of when it was processed. The class data samples are used to train features, as needed, to evaluate new data samples during real time operation. In the system example, 20 class data samples exist, 10 for each of the words '***1***' and '***2***'. Although not indicated on the figure for brevity (via the x 10 label), each class data sample would have a different timestamp. Recall the system example recognizes the vocabulary {'***1***', '***2***', '***3***'}. However it does not contain any class data samples for '***3***'. This indicates that it has not yet processed any data sample from the word '***3***' and thus could not store anything for it.

A ***feature candidate*** represents a single feature, as defined in section 3, and contains all the information and processes necessary to its evaluation. That includes an ***evaluate algorithm***, any ***feature specific data*** necessary to that evaluation (e.g., the discretization

level), and a unique identifier. The feature candidates are drawn upon during processing of a data sample when the currently employed features are insufficient and additional ones are needed for accurate decision making. In the system example, three such feature candidates are members of the memory pool with unique identifiers ***candidate 1***, ***candidate 2*** and ***candidate 3***. Of these, ***candidate 1*** is detailed to show the evaluate algorithm and the feature specific data.

4.2.2 Associative Pool Objects

The associative pool contains all associations relevant to the ASR's operation. As per human association behaviour described earlier, the members of the associative pool are used to guide the processing of the other pools.

An ***association*** stored in the associative pool is identical to those defined in section 3. In addition, however, some of these associations also have a corresponding type, either positive or negative, to indicate the nature of the source – destination relationship. The meaning of this type varies depending upon what objects are associated. Associations of the positive and negative types are referred to as ***positive associations*** and ***negative associations*** respectively.

Five associations exist in the system example. Four total kinds of associations exist. Three of those four are represented in the system example. The kinds will not be described here but instead detailed within the context of the cognitive and system pools where they are utilized.

4.2.3 Cognitive Pool Objects

The cognitive pool is made up of a set of processing elements called ***class set processors.*** Its human counterpart would be the individual processors or analysis methods used to ultimately perform pattern classification. Again returning to the human tree recognition example, the ability to identify a particular shape or analyze a texture would be an example of such a processor. Of interest is that while the ability to recognize a particular shape, say a rough cylinder, is a processor, the actual connection between recognizing the tree and the rough cylinder shape would be considered an association. In a similar manor it can viewed that the processors stored in the cognitive pool perform analysis on acoustic

data but their relevancy or connection to actual speech recognition only become useful through the associations in the associative pool.

A ***class set processor*** contains the functionality to evaluate sample data for a particular vocabulary and eliminate one or more of its potential classes. This functionality includes feature selection, feature evaluation, and decision-making algorithms. To implement this it contains two algorithms, ***elimination*** and ***feedback***, as well a large set of structures. The class set processor that operates on a vocabulary, ***V***, is represented as ***CS(V)***.

Each ***CS(V)*** can be viewed loosely as an ASR itself - the main differences from conventional ASRs being that feature selection is performed at real time and only a single class need be eliminated as opposed to complete classification. Also for a given vocabulary, ***V***, only one unique ***CS(V)*** exists within the cognitive pool. Being comparable to ASRs themselves, the ***CS(V)***'s are used in a similar manor as an ASR would be by the system. That means that when a class data sample has a potential vocabulary that matches a ***CS(V)***'s vocabulary, it is called upon to operate on that sample and make a decision on it. Again the difference being that decision need only be one of elimination – not classification. Two class set processors exist in the system example, ***CS('1', '2', '3')*** and ***CS('1', '2')***, and the first is detailed further to show all the structural contents of a ***CS(V)***.

A ***CS(V)***'s ***elimination algorithm*** performs its namesake task: class elimination. It implements the majority of a ***CS(V)***'s functionality providing the ability to eliminate at least one potential class from the vocabulary, ***V***, for a data sample. It uses several structures shown in the detailed view of ***CS('1', '2', '3')*** in the system example that will be described below.

The ***feedback algorithm*** is used to update the class set processor after it has been used to process a data sample and the correct class has ultimately been determined for it from the user. It uses the ***evaluated feature list*** and the correct class to update the evaluated features ***correct*** and ***incorrect classification counts***. After updating the counts, it determines if the creation or destruction of associations based on these counts.

The ***class likelihood array*** is used by the elimination algorithm to store its current predictions for the individual classes that make up the ***CS(V)***'s vocabulary. It has the same dimension therefore as the number of members of that vocabulary. When the likelihoods meet a certain criteria (i.e., that one or more is low enough and at least one is high enough) the ***CS(V)*** is ready to eliminate a class and elimination algorithm is complete.

The elimination algorithm maintains a list of feature candidates used on the current data sample both to log them for later use by the feedback algorithm and so as not to use the same feature twice. This list is the ***evaluated feature list***.

A set of recommended and un-recommended feature lists are constructed from associations by the elimination algorithm for each data sample it processes. Being constructed from associations, these lists are used to guide the feature selection and thus decision making process of the elimination algorithm. Before describing the lists individually however it is of benefit to discuss the meaning of the three relevancy levels a ***CS(V)*** uses: ***local***, ***hierarchical***, and ***global relevancy***.

Recall precept 9, a mechanism must exist to resolve conflicting associations based on how relevant an association is to current ASR conditions, and its discussion. These levels and the elimination algorithm that chooses features from them implement that precept mechanism. The degree of relevancy to the current ***CS(V)***, and hence current ASR conditions, is ranked according to these levels in the order from most to least relevant as local, hierarchical, and global. The algorithm therefore chooses features first based on locally relevant associations, then hierarchical, and finally global. Relevancy levels are determined based on the kind of associations and, in some cases, what source object they associate. The three kinds of associations used in determining relevancy level are ***class set processor***, ***class***, and ***feature associations***.
Class set processor associations take the form: class set processor → feature candidate. ***Class associations*** take the form: class → feature candidate. ***Feature associations*** take the form: feature candidate → feature candidate. They all have an attached type, either positive or negative, to indicate whether the source object (i.e., the class set processor, the class, or the feature candidate) recommends or un-recommends, respectively, the use of the feature candidate. The system example contains one class set processor association, ***CS('1', '2', '3')*** → ***Candidate 1***, and three class associations, '***1***' → ***Candidate 1***, '***2***' → ***Candidate 1***, and '***3***' → ***Candidate 1***. All four of these associations are of the positive type, or recommendations in other words. The elimination algorithm does not use the fifth association in the system example.

Thus returning to the lists in the class set processor, the ***local recommended feature list*** is a list of the all the locally relevant, recommended feature candidates. The ***local un-recommended feature list*** is a list of the all the locally relevant, un-recommended feature candidates. Assume the current class set processor has vocabulary, ***V***. A locally relevant association is then of the form ***CS(V)*** → feature candidate. In other words the

association's source is the current class set processor, ***CS(V)***. The elimination algorithm constructs these lists by adding the feature candidates from all such associations to the recommended list and un-recommended lists based on the association's type, positive or negative, respectively.

The ***hierarchical recommended feature list*** is a list of all the hierarchically relevant, recommended feature candidates. The ***hierarchical un-recommended feature list*** is a list of all the hierarchically relevant, un-recommended feature candidates. Again assume the current class set processor has vocabulary, ***V***. A hierarchically relevant association is one of the form ***CS(V2)*** → feature candidate, where ***V2*** is a vocabulary that is 'close' to ***V***. Loosely, a vocabulary ***V2*** is considered 'close' to ***V*** if it shares many of the same members but is not identical. For example, the vocabulary {'1', '2', '3'}, is close to {'1', '2'}, {'1', '2', '7'}, and {'1', '2', '3', '6'}, amongst others depending upon what classes are members of the overall ASR vocabulary. Mathematically, ***V2*** is defined as 'close' to ***V*** if:

1. $-1 \leq ||V2|| - ||V|| \leq 1$
2. $||V \cap V2|| \geq ||V|| - 1$
3. $V \neq V2$

As mentioned earlier, the locally relevant (un)recommendations are ranked higher than these (un)recommendations as the local ones are based on associations with the current class set processor whereas these are based on associations with class set processor's that are only close to the current environment. The elimination algorithm constructs these lists by adding the feature candidates from all such 'close' vocabulary associations to the hierarchical recommended list and un-recommended lists based on the association's type, positive or negative, respectively.

The ***global recommended feature list*** is a list of the all the globally relevant, recommended feature candidates. The ***global un-recommended feature list*** is a list of the all the globally relevant, un-recommended feature candidates. Again assume the current class set processor has vocabulary, ***V***. Further assume that ***C*** represents any member class of ***V*** and ***F*** represents any feature candidate that is on the ***CS(V)***'s evaluated feature list, defined earlier. A globally relevant association is then of the form ***C*** → feature candidate or ***F*** → feature candidate. This means that either a class within the current vocabulary or a feature candidate already evaluated on the current data sample (hence is on the evaluated feature list) is the source of the association. These kinds of associations indicate that the destination feature candidate works well (or does not) with the source

class or source feature candidate. However these associations are not attached to any particular class set processor in the system but instead to any class set processor that contains the source class or uses the source feature candidate. Thus these (un)recommendations are of the lowest rank as they are less environmentally relevant than hierarchical (un)recommendations, which are 'close', or local ones which come from the current class set processor itself. The elimination algorithm constructs these lists by adding the feature candidates from all such associations to the recommended list and un-recommended lists based on the association's type, positive or negative, respectively.

After constructing all the recommendation and un-recommendation lists, the elimination algorithm combines the features on them (details given in the algorithm section) resulting in a new list, the ***acceptable feature list***. This list represents the features that are acceptable choices for the next feature to evaluate on the current data sample by the elimination algorithm.

The final list, the ***internal feature list***, is a list of ***features*** (i.e., not feature candidates) that the ***CS(V)*** has evaluated on a data sample in the past. A ***feature*** stored within a class set processor is linked to a single feature candidate from the memory pool. Thus using its linked feature candidate functionality, it can evaluate said feature on acoustic data. In addition, it contains class set processor specific data (i.e., vocabulary specific data) necessary to its use within the said class set processor's algorithms: a ***conditional probability mass function array***, a ***correct classification count***, and an **in*correct classification count***. The internal feature list is used to maintain a list of features that have been used before and hence have at least some specific data already developed for them. As this data is necessary before any candidate feature can be used within a class set processor, the elimination algorithm attempts to first select features from the internal feature list that are also on the acceptable feature list for evaluation to save time since these algorithms execute in real-time. If, however, no acceptable features can be found on the internal feature list, the feature candidates in the memory pool must be used to create a new feature and add it to the internal feature list. The system example has a detailed view of ***CS('1','2','3')*** and that class set processor contains two internal features, ***Feature 1*** and ***Feature 2***, of which ***Feature 1*** is detailed further.

The ***class conditional probability mass function array*** contains conditional PMFs in the form of the probability of a feature value given a particular class. There is one conditional PMF for each class in the class set processor's vocabulary. The probability mass functions are updated when the feature is used. It is important to note that they are stored in the form of a set of ratios of occurrence count to total evaluation count, one for each

possible value the feature can take on. This method of storage is needed for a particular function of the feedback algorithm, to be discussed later.

The ***feature conditional probability mass function*** is a PMF of the form of the probability of any class in the class set processor's vocabulary given a particular feature value. After the feature is evaluated on a data sample and hence its value is known, the elimination algorithm constructs the feature conditional probability mass function from the class conditional PMF array for that particular feature value. It is then used to update the class likelihood array.

Finally each feature contains a ***correct classification count*** and an **in*correct classification count*** to indicate the count of times the feature has been used correctly or in error, respectively. The feedback algorithm uses them to determine when the creation of the three kinds of associations already discussed (i.e., class set processor, class, and feature associations) is necessary.

Before leaving the discussion of features completely, the reader may be wondering why both features and feature candidates must exist. The question comes to mind: would it not be more efficient for both data storage and execution times to store only a single copy of the data for a feature rather than independent copies that must be developed for each class set processor that uses said feature? Recall however the desired independence precept (precept 10) and the desire to allow decision mechanisms, feature selection, and feature data itself to be vocabulary dependent (precepts 1 and 2). To achieve those precepts, the data for a feature used with one class set processor (and hence associated with one vocabulary) must be separate and independent of the data for the same feature used in any other class set processor. Also of interest, note that it is quite possible for several class set processor's to employ the same features and hence link to the same feature candidates. Thus this is the reason why associations have feature candidates as their destinations, and not features themselves. The feature candidate is a universal data structure accessible to all class set processors whereas a feature is class set processor specific. If an association referenced a feature instead of a feature candidate, that association would only be meaningful to the class set processor that contains that feature, as other class set processors have no knowledge of that feature. Essentially this would eliminate the ability to use both the hierarchical associations and global associations, key elements of this ASR design.

4.2.4 System Pool Objects

The system pool contains the administrative functions of the ASR. It provides the user interface, obtains whole-word sample data, and co-ordinates the activities of the other pools. Of these three functions, only the last is relevant to this thesis (see this summary book[3] for the other functions). The system pool ultimately implements the three primary ASR algorithms: ***Recognition***, ***reformation***, and ***update algorithms***. It does this even though directly it does not perform either feature selection or classification relying instead on the other pools for these tasks. The system pool also contains various system wide data to track the progress of a recognition attempt or to update the system while recognition is not in progress.

The system pool acts as a central processor. Like the CPU of a computer that assigns tasks and co-ordinates the activities of the other devices in the computer, the system pool ultimately performs all tasks but often just delegates sub-assignments rather than perform them itself. Again returning to the human tree recognition example, the system pool would behave like the human counterpart that, for example, identifies the task at hand (i.e., to classify an object), what analysis methods to use to solve it (i.e., texture changes, colour changes, shapes), and when it is solved. It other words it ultimately chooses the actual processing to be done. The human mind uses associations to guide these selections and, similarly, so does the system pool by using associations to choose the correct class set processor(s) from the cognitive pool and, in turn, those class set processors use associations to choose features and make their decisions.
The ***recognition algorithm***, as its name implies, attempts to recognize data samples presented to the ASR system. To do this it includes the functionality to determine the initial environment for a data sample (e.g., initial vocabulary) and execute appropriate class set processors based on the changing environment during the recognition of that sample. It uses several structures shown as members of the system pool in the system example that will be described below.

Executed immediately following the recognition algorithm for each data sample, the ***reformation algorithm*** is responsible for maintaining and improving the ASR's accuracy during its operation. It is similar to the training algorithms for conventional ASRs in that it has the same goal but differs significantly in its approach and scale. Being executed in real time also allows for the advantage of being able to track changes in the ASR's environment and accommodate them. It includes the functionality to obtain the correct classification for the current data sample from the user and, using the ***system log***, give

feedback to the class set processors used during recognition of the data sample based on the correct classification.

The ***update algorithm***, unlike the other two system pool algorithms, is not necessarily executed during the processing of every sample. In fact it generally only executes when the other two system pool algorithms are idle, meaning there are no pending data samples to be recognized. However it is required for either of the other algorithms, and hence the system, to function effectively. That is because the update algorithm is responsible for integrating new data into the conditional PMF arrays of all class set processors features, as needed. As stated, it generally operates only when the system is idle but may be called upon when a particular feature is lacking data. For example when a new feature candidate is integrated into a class set processor it is executed to provide initial data for the new feature. The update algorithm uses the ***feature update list*** to determine what features need updating. Each time it is executed, the update algorithm updates the top feature on the feature update list.

As just stated, the ***feature update list*** indicates which features from the various class set processors need updating by the update algorithm. When a class set processor evaluates a feature, it is added to the bottom of the feature update list. If, however, the class set processor determines the update is urgently needed, due to a severe lack of data, it adds the feature to the top of the feature update list and executes the update algorithm immediately.
The ***system log*** is used by the recognition algorithm to record it class set processor selections during a single data sample evaluation. The reformation algorithm later uses the system log to determine which class set processors were used to give them feedback.

The ***vocabulary*** variable is used to store the initial vocabulary for a data sample as determined by the recognition algorithm. The method for determining this initial vocabulary is dependent upon the mode of operation of the ASR. The three intended modes of operation are: ***stand-alone***, ***tied-in***, and ***cascade mode.***

In ***stand-alone mode*** the ASR operates as conventional large vocabulary ASRs. The initial vocabulary determined by the recognition algorithm for each sample is simply the entire vocabulary the ASR recognizes.

In both ***tied-in*** and ***cascade mode*** a conventional large vocabulary ASR is used on every whole-word sample before this thesis' ASR is used to evaluate the sample. In tied-in mode, this thesis' ASR has access to the likelihood predictions of the conventional ASR

(i.e., it has knowledge of the conventional ASR and hence has been integrated or tied-in) and the recognition algorithm forms the initial vocabulary for a data sample as the top 10 most likely classes as predicted by the conventional ASR. In cascade mode, however, this thesis' ASR has no knowledge of the conventional ASR and thus the initial vocabulary determined by the recognition algorithm is simply the single class predicted by the conventional ASR.

The ***extended vocabulary*** variable is used by the recognition algorithm to store the current vocabulary for a data sample. In other words, at any given point during execution of the recognition algorithm, it stores the classes the ASR believes are possible classifications of the data sample. Again, the initialization of the extended vocabulary depends on the mode of operation of the ASR. If the ASR is in stand-alone mode, the extended vocabulary is simply initialized to be the same as the vocabulary. However in cascade and tied-in modes, the fourth and final kind of association, ***vocabulary associations***, are used by the recognition algorithm to potentially expand the initial vocabulary. Thus the algorithm could potentially initialize the extended vocabulary to a superset of the initial vocabulary.

Vocabulary associations take the form: vocabulary → class. Unlike the other kinds, this association has no attached type. These associations indicate that if the initial vocabulary for a data sample match the source object vocabulary, that the destination class should be included as a possible class for that data sample. For example if the initial vocabulary is {'1', '2'} and associations exist of the form {'1', '2'} → '5' and {'1', '2'} → '3', the resulting vocabulary would become {'1', '2', '3', '5'}. Thus after determining the initial vocabulary in cascade or tied-in mode, the recognition algorithm constructs the extended vocabulary by adding any classes from vocabulary associations that match the initial vocabulary. The system example contains one such vocabulary association, {'***2***', '***3***'} → '***1***'.

4.3 System Algorithms

The following five sections detail the five algorithms used by system. Each one will contain a summary of the algorithm, algorithm pseudocode in a figure, followed by any necessary elaboration of the pseudocode, and finally an example of its execution of the system example.

Note that within the pseudocode all system objects will be represented in bold. Also several set theory symbols are used in the following sections to represent common operators. They are:

- The binary **&** operator that represents the intersection of two sets.
- The binary | operator that represents the union of two sets.
- The unary **!** operator that represents the not of a set. In other words anything that is not on the set.
- The unary ||***S***|| operator that represents the size of the set, ***S***. As used before in this thesis, it represents simply the count of members of the set ***S***.

4.3.1 Recognition Algorithm

Algorithm Summary – The system pool's recognition algorithm represents the first step in the recognition process of a single, whole-word data sample. It begins by obtaining the new data sample and determining its initial vocabulary. It then searches for any matching vocabulary associations to expand the initial vocabulary and store the results in the extended vocabulary. It now searches for a class set processor in the cognitive pool that match this extended vocabulary. If found it executes its elimination algorithm. If not, it creates a new class set processor for the extended vocabulary, initializes it, places it in the cognitive pool, and executes its elimination algorithm. Upon regaining control from that algorithm, the elimination algorithm has updated its extended vocabulary. If a single class remains in that vocabulary, recognition is complete and the algorithm proceeds to the reformation algorithm. Otherwise it repeats the search for an additional class set processor. Figure 4.2 represents the pseudocode of this process.

01: Clear the ***system log***
02: Obtain the new ***data sample***
03: Initialize the ***vocabulary*** for the ***data sample*** based on mode of operation

04: E***xtended vocabulary*** = v***ocabulary***
05: For each ***association*** in the ***associative pool*** of the form ***vocabulary → ?***
06: {
07: Incorporate ***?*** in the **extended vocabulary**
08: }
09: Let ***V*** be the ***extended vocabulary***

10: If there exists a ***CS(V)*** in the ***cognitive pool***
11: {
12: Append ***CS(V)*** to ***system log***
13: Execute the ***CS(V)***'s ***elimination algorithm***
14: }
15: Else
16: {
17: Create the ***CS(V)***
18: Initialize the ***CS(V)'s internal feature list*** to null
19: Add the ***CS(V)*** to the ***cognitive pool***
20: Append ***CS(V)*** to ***system log***
21: Execute the ***CS(V)'s elimination algorithm***
22: }

23: If ||***V***|| != 1
24: {
25: Goto line 10
26: }

27: Proceed to ***reformation algorithm***

Figure 4.2 - Recognition Algorithm

Pseudocode Elaboration – The lines in this code are straightforward and the structures already discussed in the system objects section. The main point of potential confusion would be in lines 13 and 21, where the selected ***CS(V)***'s elimination algorithm is executed. As mentioned in the summary, the elimination algorithm updates the system pool's extended vocabulary, or ***V*** as it is referred to in the code. Thus the reader not understanding how line 23 will ever function be false, remember that the code modifies ***V*** is contained in the elimination algorithm. Note also that the algorithm does not demonstrate the handling of the trivial case where the extended vocabulary in line 9 still has only one member.

Algorithm Example – The system example does not indicate what mode of operation the ASR is in. Thus is will be assumed that it is operating in tied-in mode and further that the conventional ASR operating before this thesis' ASR has predicted for the current data sample the two most likely classes to be '***2***' and '***3***'. Thus after clearing the ***system log*** and obtaining the new ***data sample***, the algorithm will set ***vocabulary*** to {'***2***', '***3***'} in line

3. In lines 4 – 9, since the association {'***2***', '***3***'} → '***1***' exists, the algorithm will incorporate the class '***1***' in the ***extended vocabulary*** with the overall resulting extended vocabulary in line 9 becoming: {'***1***', '***2***', '***3***'}.

The existence of that association indicates that at some point the initial vocabulary for a data sample was determined to be {'***2***', '***3***'} but, ultimately, after recognition the user reported that the correct class was '***1***', which was not even present in the initial vocabulary making it impossible for the system to correctly classify the data sample. Thus the association was created so that '***1***' would be added to any subsequent initial vocabulary of the form {'***2***', '***3***'}. This ensures that the correct classification is at least possible if the same situation occurs again.

Continuing on, since the ***class set processor***, ***CS('1', '2', '3')***, exists in the cognitive pool, line 10 will find a matching ***CS(V)*** and, after updating the ***system log***, its ***elimination algorithm*** will be executed. Assume that the ***elimination algorithm*** eliminated the class '***1***' resulting in the ***extended vocabulary*** becoming: {'***2***', '***3***'}. Since the ***extended vocabulary*** still has two elements, line 23 will be true and control will return to line 10. This time however no matching ***CS(V)*** will be found as the ***class set processor***, ***CS('2', '3'}***, does not exist in the ***cognitive pool***. Thus ***CS{'2', '3'}*** will be created and its ***elimination algorithm*** executed in lines 17 – 21. Assume that ***elimination algorithm*** eliminated class '***2'*** leaving the ***extended vocabulary*** as {'***3***'}. This time line 23 will be false and control will transfer to the ***reformation algorithm*** in line 27. Thus the recognition of the ***data sample*** is complete.

4.3.2 Elimination Algorithm

Algorithm Summary – Called upon by the system pool's recognition algorithm to eliminate a class from the extended vocabulary for a data sample, a class set processor's elimination algorithm performs the most challenging tasks for the ASR as a whole as it is required to perform feature selection, feature evaluation, and decision making. It begins by obtaining the data sample from the system pool, clearing its evaluated feature list, and initializing its class likelihood array. It then must construct its various recommended and un-recommended feature lists to select a feature to evaluate. If no acceptable features can be found on the internal feature list, it must use the same recommended and un-recommended feature lists to select a new feature candidate from the memory pool, construct a feature from it, and add it to its internal feature list.

Once a feature is selected, it adds it to the evaluated feature list then proceeds to evaluate it on the data sample to obtain a feature value. It then uses that feature value in conjunction with the feature's class conditional PMF array to compute the feature conditional PMF. That feature conditional PMF can be used to update class likelihood array, which, in turn, is used to determine if a class can be eliminated. If a class can be eliminated, the algorithm is complete and control returns to the system pool's recognition algorithm. If not, the process of feature selection and evaluation is repeated until a class can be eliminated. Figure 4.3 represents the pseudocode of this process.

01: Obtain the ***data sample*** from the ***system pool***
02: Clear the ***evaluated feature list***
03: Initialize the ***class likelihood array*** to the maximum entropy state

04: Construct the ***recommended*** and ***un-recommended feature lists***

05: Construct the ***acceptable feature list*** to choose a ***feature*** from the ***internal feature list***
06: If ***acceptable feature list*** != null
07: {
08: Select a ***feature***, ***F***, from the ***acceptable feature list*** based on mutual information

09: Goto line 16
10: }

11: Construct the ***acceptable feature list*** to choose a ***feature candidate*** from the ***memory pool***
12: Create a new ***feature***, ***F***, and set its data to null
13: Select a ***feature candidate*** from the ***acceptable feature list*** at random and link it to ***F***
14: Add ***F*** to the top of the ***system pool's feature update list***
15: Execute the ***system pool's update algorithm***

16: Use ***F's evaluate algorithm*** on the ***data sample*** to obtain a ***feature value***
17: Add ***F*** to the ***evaluated feature list*** and to the bottom of ***system pool's feature update list***
18: Construct ***F's feature conditional PMF***
19: Use ***F's feature conditional PMF*** to update the ***class likelihood array***

20: If a class can not be eliminated based on the ***class likelihood array***
21: {
22: Goto line 04
23: }

24: Remove the eliminated classes from the ***system pool's extended vocabulary***
25: Return control to the ***system pool's recognition algorithm***

Figure 4.3 - Elimination Algorithm

Pseudocode Elaboration – Unlike the recognition algorithm, this algorithm needs elaboration for several of its lines in order for the reader to be able to reproduce it in their own system. The casual reader not interested in reproducing the code may wish to simply

skim this section. While the actual algorithm is simple, it uses so many structures and has so many little nuances that that the interested reader is strongly encouraged to refer back to the system objects section whenever the purpose of a structure, or how it is being used, is unclear.

In line 3, recall that the class likelihood array has the same dimension as vocabulary of the class set processor it is contained in. Since the currently executing class set processor must have a vocabulary equal to the system pool's extended vocabulary, each of the ||extended vocabulary|| members of the class likelihood array will be initialized to 1 / ||extended vocabulary||.

The six recommended and un-recommended feature lists are constructed in line 4 according to the methods already discussed in their individual descriptions in the system objects section.

Constructing the acceptable feature list in line 5 is actually quite involved. Recall that the system must choose features first based on local recommendations, then hierarchical, and then global. Further to this, in the absence of recommendations (which occurs frequently when the system has only been operating a short time) it must be able to choose those features that are simply not un-recommended.

In summary the procedure begins by constructing the acceptable feature list based on local recommendations and attempts to choose a feature from it. If it cannot, it constructs the acceptable feature list based on hierarchical recommendations and again attempts to choose a feature from it. Again if it cannot it repeats this procedure for the global recommendations, and finally for those features that are not un-recommended. Line 5 represents all that functionality. It was not elaborated in the pseudocode, as it is very repetitive and would obfuscate the remainder of the procedure. The details follow.

- Construct the acceptable feature list as:

  Acceptable feature list = internal feature list **&**
  local recommended feature list **&**
  ! evaluated feature list

- If the acceptable feature list is null, construct it again this time as:

  Acceptable feature list = internal feature list **&**
  hierarchical recommended feature list **&**
  ! hierarchical un-recommended feature list **&**
  ! local un-recommended feature list **&**
  ! evaluated feature list

- If the acceptable feature list is null, construct it again this time as:

  Acceptable feature list = internal feature list **&**
  global recommended feature list **&**
  ! global un-recommended feature list **&**
  ! hierarchical un-recommended feature list **&**
  ! local un-recommended feature list **&**
  ! evaluated feature list

- If the acceptable feature list is null, construct it again this time as:

  Acceptable feature list = internal feature list **&**
  ! global un-recommended feature list **&**
  ! hierarchical un-recommended feature list **&**
  ! local un-recommended feature list **&**
  ! evaluated feature list

Of particular interest, notice in the algorithm how more relevant un-recommendations override less relevant recommendations (e.g., a global recommended feature is only added if a hierarchical or local un-recommendation does not exist).

Now recall precept 8, associations must be able to be destroyed at run-time. In order for a feature that has a un-recommendation associated with to get that association destroyed, it must be used successfully a number of times (i.e., it must increase its correct classification count). However such a feature would never appear on an acceptable feature list in any of the steps described above. Thus one final step is necessary to meet precept 8:

- If the acceptable feature list is null, with a fixed probability, called the ***choose un-recommended feature probability***, construct it again this time as:

    Acceptable feature list = internal feature list **&**
    ! evaluated feature list

The choose un-recommended feature probability is used to limit the percentage of times the algorithm actually chooses an un-recommended feature so that overall system recognition rates are not significantly impacted by choosing poor features routinely while still allowing those features that are falsely labelled as poor to be chosen.

If one of the attempts in line 5 was the acceptable feature list is not null and a feature will be selected on line 8. If more than one feature exists on the acceptable feature list, some mechanism must be employed to choose between them. As the pseudocode indicates, that mechanism is based on mutual information. Using a feature's class conditional PMF array, the mutual information between the classes and the feature can be computed. This is done for each feature and then the feature with the highest mutual information measure is selected. Discussing mutual information measures in detail is beyond the scope of this thesis. The interested reader should see the paper by Basir[14] for a very similar approach.

If all the attempts to form a non-null acceptable feature list in line 5 fail, a new feature must be created from a feature candidate in the memory pool. Lines 11 – 15 represent the selection of the feature candidate and preparation of the new feature for immediate use.

Since the selection of feature candidates from the memory pool is also association guided, the recommended and un-recommended feature lists must again be used. Thus line 11 employs a remarkably similar approach to that of line 5. The details follow.

- Construct the acceptable feature list as:

    Acceptable feature list = hierarchical recommended feature list **&**
    ! hierarchical un-recommended feature list **&**
    ! local un-recommended feature list **&**
    ! internal feature list

- If the acceptable feature list is null, construct it again this time as:

Acceptable feature list = global recommended feature list **&**
! global un-recommended feature list **&**
! hierarchical un-recommended feature list **&**
! local un-recommended feature list **&**
! internal feature list

- If the acceptable feature list is null, with the choose un-recommended feature probability, construct it again this time as:

  Acceptable feature list = **!** internal feature list
  (i.e., all feature candidates not on the internal feature list)

- If the acceptable feature list is null, construct it again this time as:

  Acceptable feature list = **!** global un-recommended feature list **&**
  ! hierarchical un-recommended feature list **&**
  ! local un-recommended feature list **&**
  ! internal feature list
  (i.e., all feature candidates not on any of these lists)

Note that the omission of a construction based on the local recommended feature list is not an accident. If order for a feature to be locally recommended it must have been used successfully within the current class set processor. Thus it must necessarily already be an internal feature. Also note that in theory at least one of these construction attempts must produce a non-null acceptable feature list due to the fact that there are an infinite number of feature candidates in the memory pool. In practice, however, this is not guaranteed but the designer must ensure that a large enough number of feature candidates are present to avoid problems.

Line 12 indicates that a feature candidate is chosen from the acceptable feature list at random, as opposed to using mutual information as in line 8. Recall that to use mutual information the algorithm must use PMF data. Since no PMF data exists within a feature candidate, using mutual information is impossible. Thus, lacking information about the feature candidates, one is simply chosen from the acceptable feature list at random.

In lines 14 and 15, recall that the system pool's update algorithm always updates the feature at the top of feature update list. Thus adding the newly created feature, ***F***, to that

list and calling the update algorithm immediately will cause its class conditional PMF array to be developed with whatever relevant class data is available in the memory pool.

On line 18, ***F's*** feature conditional PMF is constructed. Recall that the feature conditional PMF is of the form of the probability of any class in the class set processor vocabulary given a particular feature value. The feature value has already been determined on line 16. Let ***C*** represent any class in the extended vocabulary. The desired PMF is then of the form P(***C*** | ***F*** = feature value). The elimination algorithm can construct this PMF from the class conditional PMF array by the well-known Bayes formula as follows:

$$P(\boldsymbol{C} \mid \boldsymbol{F} = \text{feature value}) = \frac{P(\boldsymbol{F} = \text{feature value} \mid \boldsymbol{C})\, P(\boldsymbol{C})}{\sum_{\boldsymbol{C}\, i} P(\boldsymbol{F} = \text{feature value} \mid \boldsymbol{C}\, i)} \qquad (4.1)$$

The only missing piece of information is P(***C***). However recall that when the elimination algorithm began, the class likelihood array was initialized to the maximum entropy state. Thus it was assumed that, without any information, each class in the extended vocabulary was equally likely. Hence P(***C***) in the above formula can be replaced with 1 / ||extended vocabulary||.

Having computed feature conditional PMF, the class likelihood array can be updated on line 19. Let the entry for class ***C*** in the class likelihood array be represented by: class likelihood array [***C***]. The elimination algorithm updates the entry for each class ***C*** in the class likelihood array according to the following:

$$\text{Class likelihood array } [\boldsymbol{C}] \mathrel{+}= P(\boldsymbol{C} \mid \boldsymbol{F} = \text{feature value}) - \frac{1}{\|\text{extended vocabulary}\|} \qquad (4.2)$$

Since features are evaluated until a class can be eliminated, the contributions of multiple feature conditional PMF's may be combined in the class likelihood array. Combining multiple features in this fashion could present false negatives or false positive results due to ignoring the interactions amongst the features. This method of combination essentially assumes the features are independent. This problem, however, is addressed by using associations to select features. For example, if one or more features consistently produce false results within a class set processor, an association will be created by the feedback algorithm to indicate that and they will subsequently not be used together.

The final elaboration needed is of line 20 and how the elimination algorithm determines if a class can be eliminated based on the class likelihood array. The algorithm does this by checking for two conditions in the class likelihood array. First there must exist a class, ***C***, such that:

Class likelihood array [***C***] $\geq$ ***Class acceptance threshold***

Second, there must exist a class, ***C2***, such that:

Class likelihood array [***C2***] $\leq$ ***Class elimination threshold***

If those two conditions are met, the elimination algorithm determines that a class can be eliminated. The ***class acceptance threshold*** is used to ensure that the class set processor believes there is at least one class for the data sample that is sufficiently probable. The ***class elimination threshold*** is used to ensure that the class set processor believes there is at least one class for the data sample that is sufficiently improbable. Together these thresholds help to ensure that the elimination algorithm is operating normally. Both thresholds are simply numbers that designer of the system chooses to tailor its behaviour. For example, the more extreme the thresholds, the more features are used before a class can be eliminated.

The second condition is used again in line 24 to remove classes from the extended vocabulary. That is, the elimination algorithm removes any class, ***C***, such that:

Class likelihood array [***C***] $\leq$ ***Class elimination threshold***

Algorithm Example – The processing of the system example during the ***reformation algorithm*** resulting in using the ***elimination algorithm*** from two ***class set processors***. However since data is present in the system example for ***CS('1', '2', '3')***, the processing of its ***elimination algorithm*** will be used as the example here.

After clearing the ***evaluated feature list*** and obtaining the ***data sample***, the algorithm will initialize each of the three members of the ***class likelihood array*** to 1/3, indicating that each of the three classes, '***1***', '***2***', and '***3***', are initially equally likely.

For this example, the algorithm will construct the ***recommended*** and ***un-recommended feature lists*** from the associations shown in the system example to be:

Local recommended feature list = ***candidate 1***
Local un-recommended feature list = null
Hierarchical recommended feature list = null
Hierarchical un- recommended feature list = null
Global recommended feature list = ***candidate 1***
Global un-recommended feature list = null

Using these lists based on the details of line 5 given in the pseudocode elaboration section, the algorithm will determine the ***acceptable feature list*** to be ***candidate 1***, based on the local recommendation. Since the acceptable feature list is not null, the algorithm will proceed to line 8 where ***feature 1***, the ***feature*** linked to ***candidate 1***, as ***candidate 1*** is the only member of the ***acceptable feature list***.

Feature 1 will now be evaluated on the ***data sample*** to obtain the ***feature value***. It will also be added to the ***evaluated feature list*** and to the bottom of the ***system pool's feature update list***. Via equation 4.1, the algorithm can now compute ***feature 1's feature conditional PMF*** for class, ***C,*** that can take on values '***1***', '***2***', or '***3***', as:

$$P(\boldsymbol{C} \mid \textbf{\textit{Feature 1}} = \textbf{\textit{feature value}}) = \frac{P(\textbf{\textit{Feature 1}} = \text{feature value} \mid \boldsymbol{C}) * 1/3}{\sum_{Ci='1','2','3'} P(\textbf{\textit{Feature 1}} = \text{feature value} \mid Ci)}$$

Using this it will proceed to update the ***class likelihood array*** according to equation 4.2 as:

$$\textbf{\textit{Class likelihood array}}\ [\boldsymbol{C}] \mathrel{+}= P(\boldsymbol{C} \mid \textbf{\textit{Feature 1}} = \textbf{\textit{feature value}}) - 1/3$$

Assume that the evaluation of ***feature 1*** was sufficient to allow the ***elimination algorithm*** to determine it can eliminate a class in line 20. Recall it was assumed that class '***1***' was eliminated by this ***class set processor*** during the evaluation of the system example in the ***system pool's recognition algorithm***. To make these assumptions true, the following conditions must in the class likelihood array at line 20:

Class likelihood array ['***1***'] $\leq$ ***Class elimination threshold***
and either ***class likelihood array*** ['***2***'] $\geq$ ***Class acceptance threshold***
or ***class likelihood array*** ['***3***'] $\geq$ ***Class acceptance threshold***

Thus, under those assumed conditions, line 24 will remove '***1***' from the ***system pool's extended vocabulary*** and return control to its ***recognition algorithm***.

4.3.3 Reformation Algorithm

Algorithm Summary –The system pool's reformation algorithm is not concerned with the recognition of data samples but instead on the adaptation of the system based on those recognition attempts. It is executed immediately after the recognition algorithm. It begins by reporting the results of the completed recognition attempt on the data sample to the user. It then obtains the correct classification from the user. Having the correct classification, a new class data sample can be created for the data sample that was processed. Such a class data sample is created, labelled as the correct class, time stamped and added to the memory pool.

Using the correct classification, the algorithm checks if the initial vocabulary determined by the recognition algorithm contained the correct class. If it did not, that means the system had no chance of correctly classifying the data sample as the correct class was not ever a member of its vocabulary. In this case, the algorithm creates an association between the initial vocabulary and the correct class. This association will be used by the recognition algorithm to ensure that whenever the same initial vocabulary is determined, it will expanded to include the correct class.

The algorithm then examines the system log that contains a list of class set processors used in the last execution of the recognition algorithm. For each ***CS(V)***, it determines if the correct class was a member of its vocabulary, ***V***, and, if it was, executes that ***CS(V)'s*** feedback algorithm. It ignores the other ***CS(V)s***. The feedback algorithm is used to report to a ***CS(V)*** whether it made a correct decision as part of the classification of the data sample. The ***CS(V)***s used that did not contain the correct class in the vocabulary could not make an incorrect decision as it was not even possible for them to eliminate the correct class. The absence of the possibility to make a mistake negates the ability to be right or wrong, thus making feedback to those ***CS(V)s*** irrelevant. The pseudocode for this process is presented in figure 4.4.

01: Report the member of the ***extended vocabulary*** as the predicted classification of the ***data sample*** to the user
02: Obtain the ***correct class*** from the user

03: Create a new ***class data sample*** from the ***data sample***
04: Label the new ***class data sample*** as the ***correct class*** and ***timestamp*** it
05: Add the new ***class data sample*** to the ***memory pool***

06: If ***correct class & vocabulary*** == null and
no ***association*** exists in the ***associative pool*** of the form ***vocabulary → correct class***
07: {
08: Create an ***association*** of the form ***vocabulary → correct class***

09: Add that ***association*** to the ***associative pool***
10: }

11: For each ***CS(V)*** in the ***system log***
12: {
13: If ***correct class & V*** != null
14: {
15: Execute the ***CS(V)'s feedback algorithm***
16: }
17: }

Figure 4.4 - Reformation Algorithm

Pseudocode Elaboration – The reformation algorithm is relatively straightforward having but two points of clarification.

For line 1, recall that at the end of the recognition algorithm the extended vocabulary had but one member, the single class that had not been eliminated. This is why the reformation algorithm uses the extended vocabulary to report the predicted classification.

On line 6 the algorithm checks both whether the correct class was a member of the initial vocabulary and whether an association of the form vocabulary → correct class exists already. Recall that in the recognition algorithm, the initial vocabulary is potentially expanded to an extended vocabulary. Thus checking the initial vocabulary is insufficient to determine whether the correct class was a member of the initial extended vocabulary, the one the recognition algorithm actually uses to perform its classification. To determine, then, whether the correct class was a member of the initial extended vocabulary, the algorithm must check both the initial vocabulary and for the presence of an association that would expand that initial vocabulary. This is reason for both checks.

Algorithm Example – The ***reformation algorithm's*** processing of the system example will begin by reporting the predicted classification of class '***3***' to the user, the last

remaining class in the ***extended vocabulary*** at the end of the ***recognition algorithm***. It will now obtain the ***correct class*** from the user. It is now assumed that the ***correct class*** was indeed '***3***'. Recall that the initial ***vocabulary*** was determined to be {'***2***', '***3***'} and thus the ***correct class*** was one of its member. Therefore no new ***association*** need be created in line 6. Two ***class set processors***, ***CS('1', '2', '3')*** and ***CS('2', '3')***, were used in the ***recognition algorithm*** and hence are on the ***system log***. Both of the ***class set processor's*** vocabularies contain the ***correct class***, a necessity for a correct classification, and therefore both will have their ***feedback algorithm*** executed on line 15.

4.3.4 Feedback Algorithm

Algorithm Summary – The system pool's reformation algorithm is responsible for the adaptation of the entire system to its environment based on the processing of the current data sample. It uses the class set processor's feedback algorithms to do this. Thus a class set processor's feedback algorithm is responsible for the adaptation of its individual class set processor based on the processing of the current data sample. The algorithm visits each of the features on its evaluated feature list, which lists all the features used in the last execution of its elimination algorithm. For each feature it determines, based on the correct class obtained from the system pool and the feature's feature conditional PMF, if it made a correct decision. If it did it increments its correct classification count, and if not it increments the incorrect classification count. Based on these new classification counts, it then determines if the feature is sufficiently good or bad enough to create associations to recommend or un-recommend, respectively, the features use in the future, if they do not already exist. Figure 4.5 shows the pseudocode for this algorithm.

```
01: Obtain the correct class from the system pool

02: Let CS(V) be this class set processor, the one the feedback algorithm is executing within

03: For each feature, F, on the evaluated feature list
04: {
05:     If F's feature conditional PMF indicates that the feature made a correct decision
06:     {
07:         Increment F's correct classification count
08:     }
09:     Else
10:     {
11:         Increment F's incorrect classification count
12:     }

13:     Let ratio = F's correct classification count / F's incorrect classification count

14:     If ratio ≥ recommended feature threshold
15:     {
16:         If a set of negative type associations exist in the associative pool for F
17:         {
18:             Destroy that set of associations
19:         }
20:         If a set of positive type associations do not exist in the associative pool for F
21:         {
22:             Create that set of associations and add them to the associative pool
23:         }
24:     }
25:     Else if ratio ≤ un-recommended feature threshold
26:     {
27:         If a set of positive type associations exist in the associative pool for F
28:         {
29:             Update the occurrence counts of F's class conditional PMF array
30:             Destroy that set of associations
31:         }
32:         If a set of negative type associations do not exist in the associative pool for F
33:         {
34:             Create that set of associations and add them to the associative pool
35:         }
36:     }
37: }
```

Figure 4.5 - Feedback Algorithm

Pseudocode Elaboration – This algorithm needs some elaboration as many of the specific details of implementation have been omitted from the pseudocode figure.

Beginning with line 5, the method the algorithm uses to determine if the feature made a correct decision is directly related to the method used in the elimination algorithm to update the class likelihood array, via equation 4.2. That equation updates the individual entries in the likelihood array based on the difference between the individual entries in

the feature conditional PMF and the maximum entropy state. That is, it updates the class likelihood array based on whether a feature predicts a class is more or less likely after its evaluation than it was initially in its maximum entropy state. Recall that the elimination algorithms constructs each feature's feature conditional PMF for a particular feature value and can be represented as P(***C*** | feature = feature value), for any class, ***C,*** in the class set processor's vocabulary. Thus in determining if a feature has a made a correct decision, the following condition is checked:

$$P(\text{correct class} \mid \text{feature} = \text{feature value}) \geq \frac{1}{\|\text{extended vocabulary}\|} \qquad (4.3)$$

If that condition is true, the algorithm decides the feature made a correct decision.

On lines 14 and 25 two constants were introduced, the ***recommended feature threshold*** and the ***un-recommended feature threshold***. Like all other constants in the system, they are determined by the designer and influence the behaviour of the overall system. In particular these constants are used to determine when the ratio of correct classifications to incorrect classifications for a feature are sufficient enough to recommend or un-recommend, respectively, the continued use by the system, by creating associations. The more extreme these constants, the more accurate (or inaccurate) a feature must be before being recommended (or un-recommended) for use.

When a feature exceeds one of those thresholds a set of associations is created. The set is called the ***feature association set***. The set created is identical, save for the association types, for both thresholds. If the recommended feature threshold is exceeded, all associations in the set are created with positive types. Similarly if the un-recommended feature threshold is exceeded, all associations in the set are created with negative types. For convenience of discussion, the contents of the positive set are presented. However as stated the negative set is created identically by replacing all positive types with negative ones.

Let the feature for which the feature association set is being created be ***F***. Let the class set processor containing ***F*** be ***CS(V)***. Let the feature candidate linked to ***F*** be ***Candidate***. The first association in the set is then: ***CS(V)*** → ***Candidate***, with positive type. Recall this is a class set processor kind of association and is used in the construction the local and hierarchical recommended feature lists.

Let ***C*** be any class in ***V.*** Also included in the set are associations of the form: ***C*** → ***Candidate***, with positive type. One is included for each ***C*** in ***V***. Recall this is a class kind of association and is used in the construction the global recommended feature list.

Finally let ***Candidate 2*** be any feature candidate such that an association exists, with a positive type, in the associative pool of the form: ***CS(V)*** → ***Candidate 2***. The existence of this kind of association indicates that the ***CS(V)*** that contains ***F*** also recommended another feature, linked to ***Candidate 2***. Both features work well within the same environment. It is then assumed that they also work well together. Thus for each feature candidate, ***Candidate 2***, two associations are created of the forms: ***Candidate 1*** → ***Candidate 2***, with positive type, and ***Candidate 2*** → ***Candidate 1***, with a positive type. Recall this is a feature kind of association and is used in the construction the global recommended feature list.

All these associations together make up the feature association set. In the pseudocode the association set referred to by lines 16, 18, 20, 22, 27, 30, 32, and 34 is this feature association set.

For clarity, lines 14 – 24 are checking if ***F*** is sufficiently good enough to warrant the creation a positive type feature association set for it. If it is, line 16 first checks whether a negative type feature association set already exists for ***F*** in the associative pool. If it does that means that this class set processor has, at one time, determined ***F*** to be poor. This is no longer the case so it destroys them if they exist. Then it checks if a positive type feature association set already exists for ***F*** in the associative pool. If they do, they do not need to be recreated. If not, the set is created and added.

Lines 25 – 36 function in a similar manor to lines 14 – 24 with one exception, line 29. If the algorithm reaches line 29, this means that the feature ***F*** is considered poor and needs a negative type feature association set created for it. However in addition to this, it has also found the existence of a positive type feature association set for ***F*** in the associative pool. This implies that, at one time, ***F*** was considered to be a good feature. Thus it is assumed that something must have changed in the environment to make ***F*** a poor feature. That ‘something’ is considered to be class migration.

Class migration is, loosely, when the pronunciation of a particular class changes with time by a user / talker. Thus further it is assumed that new data samples for that class are more accurate than its older data samples. To allow these new data samples to have greater impact on the ***F’s*** class conditional PMF array, and hence a greater impact on ***F***

itself, that PMF array is modified. Recall in the system object section that it was important to store the class condition PMF array as a set of ratios of occurrence counts to total occurrence counts. It is for this modification to that array that this particular method of storage was needed.

The algorithm modifies each individual PMF in ***F's*** class conditional PMF array by dividing all the occurrence counts and total occurrence counts by 2. Thus the probabilities the PMF's themselves predict remain unchanged, but any new class data integrated into them a greater impact on those probabilities.

Note that when the system is operating with a relatively low level of class sample data present in the memory pool, the features, too, will be making decisions based on very few samples defining their PMFs. This will generally result in them making many mistakes. Even features that are generally good within a vocabulary will make mistakes without sufficient data thereby artificially boosting their incorrect classification count. Thus one modification to this algorithm used in that instance is to check before incrementing either of a feature's classicisation counts, how much data a feature's PMF's are based on. If that data level is determined as insufficient, the counts are left unchanged. This will prevent the artificial boosting of the incorrect classification count and the improper creation of negative type feature association sets.

Algorithm Example – This algorithm is not particularly conducive to demonstration by example as its effects only occur after it is executed numerous times until a sufficiently good or poor enough ***feature*** is located. Nonetheless, its execution on the system example will be presented. In particular, the ***feedback algorithm's*** execution for the ***class set processor CS('1', '2', '3')*** will be presented.

Recall from the ***elimination algorithm's*** execution for ***class set processor CS('1', '2', '3')*** a single feature, ***Feature 1***, was used. Thus the only member of ***CS('1', '2', '3')'s evaluated feature list***, is ***Feature 1*** (actually technically it is ***Candidate 1***, the ***feature candidate*** linked to ***Feature 1*** but in effect this is just an implementation detail and un-important). Thus after obtaining the ***correct class*** from the ***system pool***, the analysis of ***Feature 1*** will begin.

Looking at the ***associative pool*** in the ***system example***, it is clear that the ***feature association set*** for ***Feature 1*** already exists in the form of the four associations: ***CS('1', '2', '3') → Candidate 1***, ***'1' → Candidate 1***, ***'2' → Candidate 1***, ***'3' → Candidate 1***. For those to exist, the current ratio of ***correct classification count*** to ***incorrect classification***

count must exceed the ***recommended feature threshold***. Note also that since ***CS('1', '2', '3')*** did not eliminate the ***correct class***, it is reasonable to assume that ***Feature 1's feature conditional PMF*** is such that line 5 will determine that ***Feature 1*** made a correct decision thus incrementing its ***correct classification count***. Note that it is actually possible to have not eliminated the ***correct class*** but still have the feature make an incorrect decision according to the determination method used in line 5. This is a pedantic case though and, as such, would not be a good assumption to make in demonstrating the algorithm.

Proceeding to line 14, as discussed, the ***ratio*** was already sufficient before incrementing the ***correct classification count*** and therefore line 14 should be true. No such ***negative type associations*** exist in the ***associative pool*** so execution will proceed to line 20. As already discussed, such a ***positive type association*** set already exists, and therefore, will not be created again. This concludes the algorithm.

4.3.5 Update Algorithm

Algorithm Summary – A very simple algorithm that generally operates only when the system is idle, the system pool's update algorithm keeps all the feature's class conditional PMFs up to date, in terms of class sample data. It begins by retrieving the top feature on the feature update list. It examines that feature's class conditional PMF to determine which classes the feature operates on. It then searches the memory pool for any new class sample data from any of those classes and, if found, uses the feature to evaluate that data. Based on that feature evaluation, it updates the feature's class conditional PMF. When no more data can be located, the feature is removed from the feature update list and the algorithm repeats, if the system is idle, or waits, if it is not. Figure 4.6 shows the pseudocode of this process.

01: Let the top entry on the ***feature update list*** be ***F***

02: Let ***C*** be any class represented in ***F's class conditional PMF array***

03: For each ***class data sample***, ***S***, in the ***memory pool*** having data from any ***C***
04: {
05: If ***S's timestamp*** indicates that ***F*** has never evaluated it before
06: {
07: Use ***F's evaluate algorithm*** on ***S*** to obtain a ***feature value***

08: Integrate that ***feature value*** into the appropriate PMF in ***F's class conditional PMF array***
09: }
10: }

11: Remove ***F*** from the ***feature update list***

Figure 4.6 - Update Algorithm

Pseudocode Elaboration – An implementation detail is presented in the algorithm that has not been mentioned previously. For completeness it will be described here. That is the purpose of the timestamp. Note in line 5, the class data sample, ***S's,*** time stamp is used to determine whether the feature, ***F***, has seen ***S*** before. The method in which this is achieved is that each PMF in a feature's class conditional PMF array also has a timestamp.

Recall that each PMF in class conditional PMF array corresponds to a single class. Therefore the PMF timestamps are used to indicate the latest class data sample from its class that it has processed. Thus any class data samples from that same class in the memory pool with newer timestamps have not been seen by the feature, and thus, this is the comparison used on line 5.

On line 8, the obtained feature value is integrated into the class conditional PMF array. This integration simply involves locating the PMF that matches the class that the class data sample, ***S***, came from and incrementing the occurrence count corresponding to the feature value, as well as the total occurrence count.

Algorithm Example – Recall in the processing of the system example during the recognition algorithm, a new ***class set processor***, ***CS('2', '3'),*** was created. Being a new ***class set processor***, it would have to create at least one new ***feature*** from a ***feature candidate*** as well. Assume that created ***feature*** is ***F***. When ***F*** was created in the ***class set processor's elimination algorithm***, it would have been added to the top of the ***feature***

update list and the ***update algorithm*** called immediately. This example will demonstrate the processing of the ***update algorithm*** in that instance.

Line 1 will retrieve ***F*** from the top of the ***feature update list***. Since ***F*** is new, it will not have seen any ***class data samples*** from the ***memory pool***. However only the 10 ***class data samples*** from class ***'2***' are relevant. The other ***class data samples*** are from class '***1'*** and ***F*** only operates on classes ***'2'***, and ***'3'***. Thus the 10 samples from class '***2'*** will be separately evaluated on line 7 and integrated into ***F's class conditional PMF array***. Note that array contains 2 PMF's, one for class '***2***' and one for class '***3***'. In this case, lacking data for class '***3***', only the PMF corresponding to class ***'2'*** will have data integrated into it. After this, ***F*** will be removed from the top of the ***feature update list***.

This concludes the discussion of the major algorithms and structures that compose the proposed ASR. Section 5 will now outline and describe the results of testing performed on this system.

5 Preliminary Results and Discussion

ASR proposals are typically tested against standard databases to derive statistics for equivalent comparison and results analysis to previous designs. However it is believed that those derived statistics would be inconclusive in this instance, as they do not directly test the altered algorithms. Consider again the defining precepts that lead to the inclusion of a run-time, vocabulary dependent feature selection algorithm. This can be considered the quintessential ASR element proposed in this thesis and is not present in conventional ASRs. Thus any test conducted to demonstrate the effectiveness of this ASR must test that feature selection algorithm.

Consider evaluating this ASR against a standard database to derive the overall recognition rate. This recognition rate is dependent upon what feature candidates are present in the memory pool and hence what features ultimately can be used for evaluation. Suppose the same features used in effective, conventional ASRs are used for this ASR. Given that past research has proven those features effective, this ASR's overall recognition rate should be similar to any conventional ASR that uses these same features (within some factor allowed for alternate decision making algorithms and other environmental variables). Yet a high recognition rate here does not directly imply anything about the feature selection algorithm, as that algorithm was only given effective features to choose from. Now suppose an alternate set of unproven features are utilized. Again the overall recognition rate directly implies nothing about the feature selection algorithm. If the rate is high it may mean that the algorithm was successful but it may also mean that the features were optimal. If it is low it may mean the algorithm was unsuccessful but may also mean that the features were poor. Thus the overall recognition rate, without further information about the features themselves, does not indicate anything about the effectiveness of the feature selection algorithm.

This thesis is not presented to prove the effectiveness of a set of features and therefore requires a feature independent mechanism for evaluating the new algorithms. One such method is to include both proven effective and ineffective features as candidates for this ASR. This means that it can be manually determined whether or not any given feature is effective at distinguishing between the classes of any subset of the ASR's overall vocabulary. The ASR example in section 3 demonstrates manual determination. The larger the vocabulary the more prohibitive this manual determination is. Consequently a standard database's large vocabulary is not feasible. The tests conducted on the feature

selection algorithm use carefully chosen vocabulary and features such that this manual determination is possible.

As mentioned, a large vocabulary test is prohibitive since manual determination of features for such a large vocabulary is practically impossible. In addition, however, two other problems currently prohibit a large vocabulary test. First bootstrapping (i.e., the time between the initial operation of the system and the time it becomes more "stable") is presently very slow and random for large vocabularies. For the interested reader, the randomness largely occurs because of line 13 of the elimination algorithm. Secondly large vocabularies result in large numbers of associations. Those associations often conflict with each other (i.e., (un)recommend many different features, or one recommends a feature while another un-recommends it). The three level system (ranking on the local, hierarchical, global levels) is able to accommodate these conflicts in small vocabularies (hence small numbers of associations) but a more complex resolution mechanism is needed for larger vocabularies. This is unacceptable for proper large vocabulary ASR operation.

Also large vocabulary ASRs can gain benefits by using additional sources of information such as grammar and / or semantics[16] (the actual boundary between grammar and semantics is fuzzy when used in this fashion). To make use of this information, information on the context of the data sample is a sentence (or simply in its surrounding speech) is necessary. Thus data samples can no longer be processed one-by-one, as both future and past context is necessary. A more parallel operation of the system is therefore needed that processes multiple samples at a time resulting in the ultimate recognition of each individual data sample not simply being determined by the features evaluated on it but also by information provided by parallel operating recognition attempts on other data samples.

The resolutions to these issues are presently being implemented and, once complete, the results of recognition tests on a standard database will be conducted. Instead a preliminary study of the ASR implemented to date will be more performed to both act as a controlled base for future modifications and to demonstrate its current functionality. It is only preliminary, however, in the sense that it does not operate on a large vocabulary. It does completely test all the changes detailed in this thesis and, specifically, the quintessential real-time, vocabulary dependent feature selection algorithm. The recognition parameters discussed in the thesis will be provided. However not provided are details deemed irrelevant within the scope of the thesis. Specifically the details of the actual features used are omitted. As discussed, the thesis does not test the effectiveness of

the features, but instead the effectiveness of feature selection. More information however is available upon request for the interested reader.

5.1 Stand Alone Mode

The overall vocabulary used for this test is a common small vocabulary: the digits. The initial vocabulary was always determined to be current overall vocabulary for the ASR. The feature candidates in the memory pool meet the manual determination criteria discussed above. As close as possible the features were selected such that when averaged over all potential class set processors, approximately 50% of them are effective for any individual class set processor. Other recognition parameters are contained within Table 5.1.

Table 5.1 - Recognition parameters

Recognition Parameter	Value
Class Acceptance Threshold	65%
Class Elimination Threshold	8%
Choose Un-recommended Feature Probability	5%
Recommended Feature Threshold	80%
Un-recommended Feature Threshold	60%

Thirty samples of each digit from 1 – 10 were recorded. Initially 25 batches of data containing one sample of each digit 1 – 9 were formed. The ASR formed its original overall vocabulary to be just the digits 1 – 9. It then processed these batches one-by-one while monitoring their recognition logs. The ASR now expanded its original vocabulary to include the word 10. It then formed 5 additional batches of data each one containing one remaining sample of the digits 1 – 9 and 5 samples of the word 10. It again processes these batches one-by-one while monitoring the logs. This procedure was repeated 50 times using the same data to average out random effects. The results of this processing are in Table 5.2 and Table 5.3.

Table 5.2 - Results from processing initial 25 batches

Batch Number	Recognition Rate	Percentage Of Feature Selections Guided By Associations	Percentage Of Effective Features Used In Recognition
5	19%	0%	47%
10	41%	0%	51%
15	75%	88%	73%
20	92%	97%	82%
25	99%	~100%	96%

Table 5.3 - Results from processing last 5 batches

Batch Number (Cumulative Data Sample Totals)	Recognition Rate For Digits 1 - 9	Recognition Rate For Digit 10
26 (26 samples of 1-9, 5 of 10)	98%	44%
27 (27 samples of 1-9, 10 of 10)	99%	92%
28 (28 samples of 1-9, 15 of 10)	99%	97%
29 (29 samples of 1-9, 20 of 10)	~100%	99%
30 (30 samples of 1-9, 25 of 10)	~100%	~100%

The recognition rate is not in and of itself impressive. It has been demonstrated that single speaker, small vocabulary, isolated word ASRs have proven[15] to be quite effective. The other two columns of Table 5.2 are of interest.

When a class set processor attempts to select a new feature for processing, it may or may not have its selection guided by associations. The percentage of feature selections guided by associations is a measure of how often this guidance occurs. Features cannot be (un)recommended by associations until the meet the (un)recommended feature threshold. This did not occur by batch number 10 and thus no associations guide feature selection until after that. However by batch 15 many features must have achieved one of those thresholds as a rapid increase in associative feature selection guidance can be seen reaching 88% on average. The remaining 12% of selections are unguided, as, due to random occurrence, some features used have not yet met either threshold. More data should force those presently indeterminate features above or below one of those appropriate thresholds. This can also be seen as by batch 25 practically 100% of the feature selections are guided by associations.

Simply because feature selection is guided by associations does not necessarily imply it will be guided to choose effective features. The third column in Table 5.2, the percentage of effective features used in recognition, represents how often, on average, the feature selection of each class set processor chooses a feature that is effective to evaluate. When no associations exist, up to and including batch 10, feature selection is random. Therefore, since approximately 50% of the features are effective on average for any class set processor, it should be expected that approximately 50% of the selected features should be effective as demonstrated. Correspondingly the recognition rate should be low as both ineffective features and severe lack of data deter proper ASR functionality. By batch 10 the recognition rate has improved as the amount of data has improved as well. By batch 15 a great deal of associations have been formed and thus should deter the use of ineffective features and encourage the use of effective ones. This too is demonstrated as the use of effective features, on average, has risen to 73%. It is expected that, if, on average, 73% of a class set processors selected features are good discriminators that a correspondingly similar recognition rate should be achieved. Further since now there are more effective features being used on average than not, it is expected that in some instances the effective ones will 'drown out' the ineffective thus theoretically pushing the recognition rate above the percentage of effective features. Again these effects are demonstrated by the recognition rate of 75%.

As more data becomes available and more associations are formed, both the use of effective features and corresponding recognition rates should rise as demonstrated in batch 20 and 25. Thus the data in Table 5.2 demonstrates the functionality to form associations to select effective features of the ASR.

The data in Table 5.3 is intended to demonstrate the ability of the ASR to expand its vocabulary quickly during real-time operation. After adding the word 10 to the overall vocabulary, a small initial recognition rate decrease is seen when attempting to recognize the original 1 – 9 classes in batch 26. The reduction is largely due to the original digits being incorrectly recognized as 10. However after data for 10 is obtained the ASR adapts and the recognition rate again rises to near perfect in batches 27 – 30.

After the first 5 samples of 10 in batch 26, the recognition rate is already 44%. Comparatively this is much higher than the recognition rate of the original digits after 5 samples in batch 5. This is largely due to the employment of hierarchical associations (those 'close' to the current environment) to select effective features for use with the newly formed class set processors that include 10 in their vocabulary. This allows them a much greater chance to choose effective features instantly only encountering difficulties

when a recommended feature actually turns out ineffective due to the inclusion of 10 in the vocabulary. Even with choosing effective features the recognition rate still remains only at 44% however as only very little data is present for the new class.

After 10 samples the recognition rate for 10 has increased to 92% and by 15 samples to 97%. This is much better than the original digits corresponding rates of 41% and 75% respectively with the same amount of data. This trend continues into batch 30. This demonstrates the ASR's ability to change its vocabulary without seriously affecting the ability to recognize the original classes and quickly effectively recognize the new class through use of associations.
Two small tests not shown in the tables were also conducted. The first was class migration. After batch 30 was processed, additional samples of a digit were recorded. The pronunciation of that digit was purposely altered to simulate class migration. The ASR then processed this new data. Similar to the incorporation of a new class into the vocabulary, the recognition rate initially dropped significantly for the migrated digit and much more insignificantly for the other digits. The features that were no longer effective for the class set processors containing the migrated digit underwent migration and new features were selected as required. Though it took more data than the incorporation of a new class, eventually the recognition rate rose to the level it started at before the class migrated. It took longer because associations had been formed to features for use with the migrated digit before migration. These features had to first migrate and then destroy these associations before new data, new features, and new associations could be incorporated. The ASR did successfully handle class migration as intended.

The second small test involved features. A ***complementary feature*** was created for a feature used by the ASR. A complementary feature is one that whenever the original feature predicts a class is more or less likely, the complementary feature predicts the opposite. Thus the two features when combined in a class set processor are ineffective. Both features were added to a class set processor that the original feature was effective for. That class set processor than processed data to determine if a global negative feature association pair would be formed between the original and complementary features. On average testing showed that between batch 17 and 18 such an association pair was created. This was to demonstrate that using the quasi-Bayesian decision mechanism with associations could still handle inter-feature conflicts without using a full Bayesian recognizer.

5.2 Cascade Mode

For this test, a conventional ASR was used as the front end for this thesis' ASR. As these are the cascade mode tests, the initial vocabulary for each sample is simply the single predicted class from the conventional ASR. That vocabulary may be expanded in step 2 via associations. This ASR only operates when such a vocabulary expansion occurs. The recognition parameters are the same as in the stand-alone test given in Table 5.1.

The sample data was obtained by selecting several paragraphs from this thesis and recording them word-by-word 25 times. The first 15 recordings were processed without analysis to allow the ASR to obtain class data and form associations (both feature and vocabulary expansion associations). The remaining 10 recordings were processed and the recognition logs monitored. This procedure was repeated 50 times on the same data and averaged to mitigate random effects.

The thesis ASR only operates on words that the conventional ASR has made mistakes on during previous processing. These instances in which it operators can be categorized into two separate modes based on why the conventional ASR made mistakes. If the conventional ASR failed because the correct class of the sample it attempted to recognize was not present in its vocabulary, the mode is called ***vocabulary expansion mode***. If however it made a mistake but the correct class was in its vocabulary, the mode is called ***reclassification mode***. The results of this processing in the various modes are shown in Table 5.4.

An example will clarify these modes and reported results. Consider the acronym 'ASR'. That acronym is not a part of the conventional ASR's vocabulary and is misclassified as, say, class ***C***. The thesis' ASR creates an association between ***C*** and the class 'ASR'. In future processing, whenever the conventional ASR predicts ***C***, the thesis ASR evaluates features to predict whether the sample is indeed ***C*** or whether it is 'ASR'. The recognition rate is therefore how often the correct class is chosen between these two classes. To further this example, it is possible that the conventional ASR also misclassifies another sample whose correct class is ***C2*** as ***C***. Thus ***C*** will also be associated with the correct class ***C2*** and the class 'ASR'. Thus in future processing whenever the conventional ASR predicts ***C***, the thesis ASR must evaluate features to predict between these there classes, ***C***, ***C2***, and 'ASR'. A corresponding recognition rate is thus how often the correct class is chosen between these three.

Table 5.4 - Results from the cascade mode test

Operating Mode	Recognition Rate
Vocabulary Expansion Mode	97%
Reclassification Mode	91%

The results from Table 5.4 show that the ASR is very effective at adding new words to the conventional ASR with a recognition rate of 97%. The reclassification mode, intended to demonstrate the viability of the thesis ASR to customize a conventional ASR to a particular user's pronunciations and environment, also demonstrates a high recognition rate. These results demonstrate that this may indeed be plausible. It is important to note that the results are only preliminary at this point. The results are sufficiently interesting however to pursue this test when the modifications to this ASR are complete to handle larger vocabularies.

These results indicate a general success within the tested environment of achieving the goals outlined in section 1. A re-cap of all the major points of the thesis will be presented in section 6 to follow.

6 Conclusions

This thesis presented a novel approach to the design of an ASR that includes association guided real-time feature selection and decision making. This approach was intended to remove the reliance of conventional ASRs on finding globally effective features and to enhance their presently weak adaptive capabilities.

Initially in section 1 it was argued that completely logical solutions to the ASR problem may not be needed and instead an ASR that includes sub-optimal, illogical steps may actually achieve better results than the present day, logical based ASRs.

The overall goal of the ASR to be environmentally adaptive by allowing variations in users, vocabularies, and pronunciations was introduced. Also the ASR would make decisions based on its current determined environment. Conventional ASRs instead assume a general operating environment. Thus by customizing the decisions made to the current environment, this ASR is able to have greater recognition rates versus conventional counterparts in environments that greatly differ from an assumed general environment.

Section 3 defined the precepts used to guide to implementation of the system. Precepts 1 thru 3 were based on feature space concepts and provide a set of necessary and sufficient conditions needed to select features for use in an ASR. The argument was also made that these conditions are less restrictive than those used in conventional ASRs and thereby using them would ease the difficult task of feature selection.

Allowing more freedom in feature selection introduced the problem of determining features at run-time. In order to facilitate this, the human mind was used to inspire precepts 4 and 5 that outlined the use of associations to guide decision mechanisms and feature selection. Using further examples from human behaviour, precepts 6 thru 9 were defined to give further rules guiding the use of associations in the ASR.

Finally precepts 10 and 11 were defined suggesting that a series of independent eliminations can be performed and this would be identical to the problem of recognition. Further to this it was argued that in some instances the series of independent eliminations would actually be easier than the corresponding recognition and thus the proposed system would use elimination rather than direct recognition.

Section 4 discussed the implementation of the system beginning with the objects and structures that it is composed of. The ASR was divided into four pools of objects and algorithms, the system, cognitive, associative, and memory pools, and their functionality was described. One particularly important object, the class set processor, was introduced. Each class set processor is customized to a particular environment and makes decisions tailored specifically to it based on associations formed from past processing. This section was concluded with the algorithm presentations in both words and pseudocode along with the specific implementation details used in testing that was described in section 5.

The preliminary study in section 5 showed interesting results in a small vocabulary environment. It showed successful adaptation to changes in vocabulary and class migration. In addition to successful adaptation, it also showed that, once established, new words and / or migrated classes could be incorporated successfully with fewer samples than the system would have required if it had never run before. This demonstrated the effectiveness of using (hierarchical) associations to suggest appropriate features. A cascade mode test was also conducted and showed that the environmental adaptation capabilities of the proposed system could indeed increase the overall performance of a conventional ASR when presented with varying environmental factors.

Finally some brief details of a follow-up large vocabulary study was described along with some issues that needed resolutions before such a study could be conduction. When those resolutions are complete the results of that study will be published.

References

[1] M. Minsky, IdeaCity 2003 Presentation, transcripts available from http://www.ideacityonline.com/2003.asp (accessed Oct. 15, 2005).

[2] *The Oxford Dictionary of Current English*, Oxford University Press, New York, NY, 1996.

[3] T. Parson, *Voice and Speech Processing,* McGraw-Hill Company, New York, NY, 1987.

[4] J. R. Deller, J. G. Proakis, and J. H. L. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillan Publishing Company, New York, NY, 1993.

[5] T. M. Nearey, "Perception: cognitive and automatic processes," Proc. XIIth Int. Conf. Of Phonet, Sci. 1, 40 –49

[6] B. P. Lathi, *Linear Systems and Signals*, Berkeley-Cambridge Press, Carmichael, CA., 1992.

[7] M. R. Sambur, and L. R. Rabiner, "An algorithm for determining the endpoints of isolated utterances," Bell System Technical Journal, vol. 54, Feb. 1975, 297-315.

[8] F. G. Stremler, *Introduction to Communication Systems – Third Edition*, Addison-Welsely Publishing Company, 1990.

[9] F. Itakura, "Minimum prediction residual principle applied to speech recognition," IEEE Trans., vol. ASSP-23, no. 1, Feb, 1975, 67 – 72.

[10] A. M. Noll, "Cepstrum pitch determination," J. Acoust. Soc. Am. vol. 41, Feb. 1967, 293 – 309.

[11] U. Goldstein, "Speaker-identifying features based on formant tracks," J. Acoust. Soc. Am. vol. 59, no. 1, 176 - 182, Jan. 1976.

[12] B. S. Atal, *Automatic speaker recognition based on pitch contours,* Ph.D. Thesis, Polytech. Inst. Of Brooklyn, 1968.

[13] R. Agrawal, and R. Srikant, **"**Fast Algorithms for Mining Association Rules," VLDDBB, 1994, 487-499.

[14] O. A. Basir, and H. C. Shen, "Interdependence and Information Loss in Multi-Sensor Systems," J. Robotic Systems, 16(11), 1999, 597-612.

[15] M. R. Sambur, and L. R. Rabiner, "A speaker-independent digit recognition system," BST J, vol. 54, Jan. 1975, 81 –102.

[16] D. S. Jurafsky, And J. H. Martin, "Speech and Language Processing: An Introduction To Natural Language Processing," Computational Linguistics and Speech Recognition, Upper Saddle River, NJ, Prentice-Hall, 2000.

[17] D. G. Myers, *Psychology – Fifth Edition*, Worth Publishers, New York, NY. 1998.

[18] S. T. Welstead, *Neural Network and Fuzzy Logic Applications in C/C++*, John Wiley & Sons, Inc. Professional Reference and Trade Group, New York, NY., 1994.