IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 14, Issue 4 (Sep. - Oct. 2013), PP 61-67
www.iosrjournals.org

Analytical Review of Feature Extraction Techniques for Automatic Speech Recognition

Rajesh Makhijani[1], Ravindra Gupta[2]
[1](M. Tech Scholar, IT Dept, Sri Satya Sai Institute of Science & Technology, Sehore (M.P.), India)
[2](Assoc. Professor, IT Dept, Sri Satya Sai Institute of Science & Technology, Sehore (M.P.), India)

Abstract: *Speech recognition is a multileveled pattern recognition task, in which acoustical signals are examined and structured into a hierarchy of sub word units (e.g., phonemes), words, phrases, and sentences. Each level may provide additional temporal constraints, e.g., known word pronunciations or legal word sequences, which can compensate for errors or uncertainties at lower levels. This hierarchy of constraints can best be exploited by combining decisions probabilistically at all lower levels, and making discrete decisions only at the highest level.*

Keywords: *ASR (Automatic Speech Recognition)1; Dynamic Time Warping2; FET (Feature Extraction Technique)3.*

I. Introduction

In speech recognition, the main goal of the feature extraction step is to compute a parsimonious sequence of feature vectors providing a compact representation of the given input signal. The feature extraction process is usually performed in three stages, in which the first stage is called the speech analysis or the acoustic front end. It performs some kind of spectro-temporal analysis of the signal and generates raw features describing the envelope of the power spectrum of short speech intervals. The second stage compiles an extended feature vector composed of static and dynamic features. Finally, the last stage (which is not always present) transforms these extended feature vectors into more compact and robust vectors that are then supplied to the recognizer. Although there is no real consensus as to what the optimal feature sets should look like, one usually would like them to have the following properties: they should allow an automatic system to discriminate between different through similar sounding speech sounds, they should allow for the automatic creation of acoustic models for these sounds without the need for an excessive amount of training data, and they should exhibit statistics which are largely invariant cross speakers and speaking environment.

Mel Spectral Coefficients

The human ear does not show a linear frequency resolution but builds several groups of frequencies and integrates the spectral energies within a given group. Furthermore, the mid-frequency and bandwidth of these groups are non–linearly distributed. The non–linear warping of the frequency axis can be modeled by the so-called mel-scale. The frequency groups are assumed to be linearly distributed along the mel-scale. The so–called mel–frequency f_{mel} can be computed from the frequency f as follows:

$$f_{mel}\ (f) = 2595 \cdot log\left(1 + \frac{f}{700\ Hz}\right) \quad (1)$$

The human ear has high frequency resolution in low–frequency parts of the spectrum and low frequency resolution in the high–frequency parts of the spectrum. The coefficients of the power spectrum $|V\ (n)\ |^2$ are now transformed to reflect the frequency resolution of the human ear.

Cepstral Transformation

Since the transmission function of the vocal tract $H(f)$ is multiplied with the spectrum of the excitation signal $X(f)$, we had those un-wanted "ripples" in the spectrum. For the speech recognition task, a smoothed spectrum is required which should represent $H(f)$ but not $X(f)$. To cope with this problem, cepstral analysis is used. We can separate the product of spectral functions into the interesting vocal tract spectrum and the part describing the excitation and emission properties:

$$S(f)\ =\ X(f) \cdot H(f) \cdot R(f) = H(f) \cdot U(f) \quad (2)$$

We can now transform the product of the spectral functions to a sum by taking the logarithm on both sides of the equation:

$$\begin{aligned} \log\big(S(f)\big) &= \log(H(f) \cdot U(f)) \\ &= \log\big(H(f)\big) + \log(U(f)) \end{aligned} \quad (3)$$

This holds also for the absolute values of the power spectrum and also for their squares:

$$\log(|S(f)|^2) = \log(|H(f)|^2 \cdot |U(f)|^2)$$

$$= \log(|H(f)|^2) + \log(|U(f)|^2) \quad (4)$$

In Fig. 1, we see an example of the log power spectrum, which contains unwanted ripples caused by the excitation signal $U(f) = X(f) \cdot R(f)$.

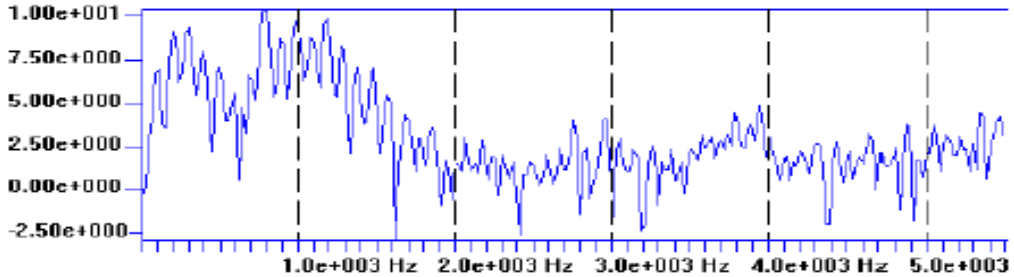

Fig. 1: Log power spectrum of the vowel /a: / (f_s = 11 kHz, N = 512). The ripples in the spectrum are caused by X (f)

In the log–spectral domain we could now subtract the unwanted portion of the signal, if we knew $|U(f)|^2$ exactly. But all we know is that $U(f)$ produces the "ripples", which now are an additive component in the log–spectral domain, and that if we would interpret this log–spectrum as a time signal, the "ripples" would have a "high frequency" compared to the spectral shape of $|H(f)|$. To get rid of the influence of $U(f)$, one would have to get rid of the "high-frequency" parts of the log–spectrum (remember, we are dealing with the spectral coefficients as if they would represent a time signal). This would be a kind of low–pass filtering. The filtering can be done by transforming the log–spectrum back into the time–domain (in the following, FT^{-1} denotes the inverse Fourier transform):

$$\hat{s}(d) = FT^{-1}\{\log(|S(f)|^2)\} = FT^{-1}\{\log(|H(f)|^2)\} + FT^{-1}\{\log(|U(f)|^2)\} \quad (5)$$

The inverse Fourier transform brings us back to the time–domain (d is also called the delay or frequency), giving the so–called *cepstrum* (a reversed "spectrum"). The resulting cepstrum is real–valued, since $|U(f)|^2$ and $|H(f)|^2$ are both real-valued and both are even: $|U(f)|^2 = |U(-f)|^2$ and $|H(f)|^2 = |H(-f)|^2$. Applying the inverse DFT to the log power spectrum coefficients $\log|V(n)|^2$ yields:

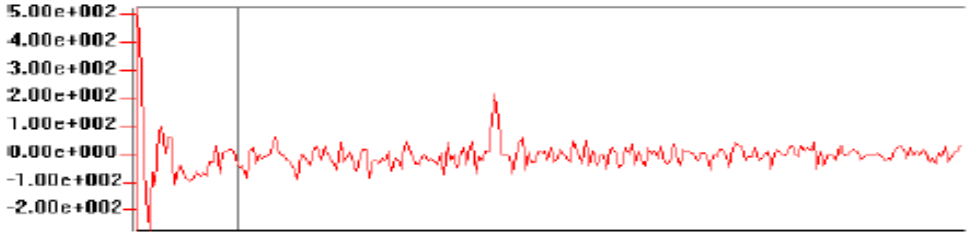

Fig. 2: Cepstrum of the vowel /a: / (f_s = 11 kHz, N = 512). The ripples in the spectrum result in a peak in the cepstrum

Mel Cepstrum

After being familiar with the cepstral transformation and cepstral smoothing, mel cepstrum is computed, which is commonly used in speech recognition. As stated above, for speech recognition, the mel spectrum is used to reflect the perception characteristics of the human ear. In analogy to computing the cepstrum, we now take the logarithm of the *mel* power spectrum (instead of the power spectrum itself) and transform it into the frequency domain to compute the so–called *mel cepstrum*. Only the first Q (less than 14) coefficients of the mel cepstrum are used in typical speech recognition systems. The restriction to the first Q coefficients reflects the low–pass liftering process as described above.

Since the mel power spectrum is symmetric due to "Eq. (5)", the Fourier-Transform can be replaced by a simple cosine transform:

$$c(q) = \sum_{k=0}^{\kappa-1} \log\big(G(k)\big) \cdot \cos\left(\frac{\pi q(2k+1)}{2K}\right); q = 0, 1, \dots, Q-1 \quad (6)$$

While successive coefficients $G(k)$ of the mel power spectrum are correlated, the *Mel Frequency Cepstral Coefficients (MFCC)* resulting from the cosine transform "Eq. (6)" are de-correlated. The MFCC are used directly for further processing in the speech recognition system instead of transforming them back to the frequency domain.

II. Feature And Vector Space

Until now, we have seen that the speech signal can be characterized by a set of parameters (features), which will be measured in short intervals of time during a preprocessing step. Before we start to look at the speech recognition task, we will first get familiar with the concept of feature vectors and vector space.

If we have a set of numbers representing certain features of an object we want to describe, it is useful for further processing to construct a vector out of these numbers by assigning each measured value to one component of the vector. As an example, think of an air conditioning system which will measure the temperature and relative humidity in the office. If we measure those parameters every second or so and we put the temperature into the first component and the humidity into the second component of a vector, we will get a series of two–dimensional vectors describing how the air in the office changes in time. Since these so–called feature vectors have two components, we can interpret the vectors as points in a two–dimensional vector space. Thus we can draw a two–dimensional map of our measurements as sketched below. Each point in our map represents the temperature and humidity in our office at a given time. As we know, there are certain values of temperature and humidity which we find more comfortable than other values. In the map the comfortable value– pairs are shown as points labeled "+" and the less comfortable ones are shown as "-". We can see that they form regions of convenience and inconvenience, respectively.

Let's assume we would want to know if a value–pair we measured in our office would be judged as comfortable or uncomfortable by us. One way to find out is to initially run a test series trying out many value–pairs and labeling each points either "+" or "-" in order to draw a map as the one shown below.

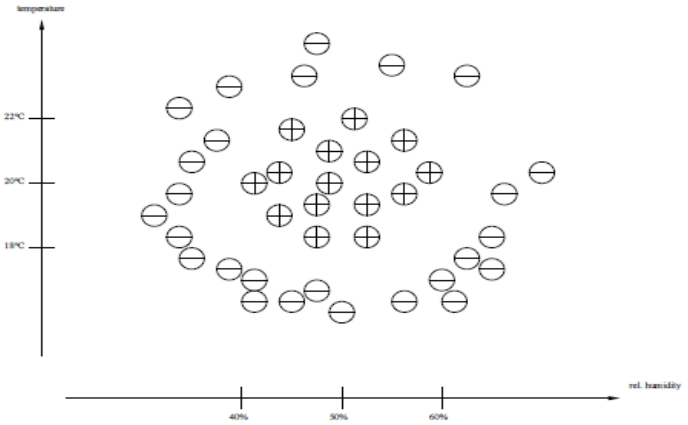

Fig. 3: A map of feature vectors

Now if we have measured a new value–pair and we are to judge if it will be convenient or not to a person, we would have to judge if it lies within those regions which are marked in your map as "+" or if it lies in those marked as "-".This is our first example of a classification task: We have two classes ("comfortable" and "uncomfortable") and a vector in feature space which has to be assigned to one of these classes. — But how do you describe the shape of the regions and how can you decide if a measured vector lies within or without a given region.

Classification of Vectors

A. Prototype Vectors

The problem of how to represent the regions of "comfortable" and "uncomfortable" feature vectors of our classification task can be solved by several approaches. One of the easiest is to select several of the feature vectors we measured in our experiments for each of our classes (in our example we have only two classes) and to declare the selected vectors as "prototypes" representing their class. We will later discuss how one can find a good selection of prototypes using the "k–means algorithm". For now, we simply assume that we were able to make a good choice of the prototypes, as shown in figure 4.

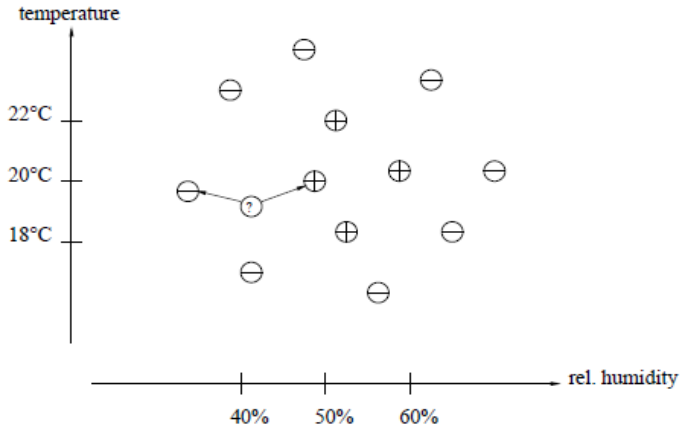

Fig. 4: Selected prototypes

B. Nearest Neighbor Classification

The classification of an unknown vector is now accomplished as follows: Measure the distance of the unknown vector to all classes. Then assign the unknown vector to the class with the smallest distance. The distance of the unknown vector to a given class is defined as the smallest distance between the unknown vector and all of the prototypes representing the given class. One could also verbalize the classification task as: Find the nearest prototype to the unknown vector and assign the unknown vector to the class this "nearest neighbor" represents (Hence the name). *Fig. 4* shows the unknown vector and the two "nearest neighbors" of prototypes of the two classes. The classification task we described can be formalized as follows: Let $\Omega = \{\omega_1, \omega_2 \ldots \omega_{(V-1)}\}$ be the set of classes, V being the total number of classes. Each class is represented by its prototype vectors $\vec{p}_{k,\omega_v}$, where $k = 0, 1, \ldots, (K_{\omega_v} - 1)$. Let $\vec{x}$ denote the unclassified vector. Let the distance measure between the vector and a prototype be denoted as $d(\vec{x}, \vec{p}_{k,\omega_v})$ (e.g. the Euclidean distance. We will discuss several distance measures later). Then the class distance between $\vec{x}$ and the class ω_v is defined as:

$$d_{\omega_v}(\vec{x}) = \min_k\{d(\vec{x}, \vec{p}_{k,\omega_v})\}; k = 0, 1, \ldots, (K_{\omega_v} - 1) \quad (7)$$

III. Distance Measurement

So far, we have found a way to classify an unknown vector by calculation of its class–distances to predefined classes, which in turn are defined by the distances to their individual prototype vectors. Now we will briefly look at some commonly used distance measures. Depending on the application at hand, each of the distance measures has its pros and cons, and we will discuss their most important properties.

Euclidian Distance

The Euclidean distance measure is the "standard" distance measure between two vectors in feature space (with dimension *DIM*) as we know it from school:

$$d^2_{Euclid}(\vec{x}, \vec{p}) = \sum_{i=0}^{DIM-1} (x_i - p_i)^2 \quad (8)$$

To calculate the Euclidean distance measure, we have to compute the sum of the squares of the differences between the individual components of $\vec{x}$ and $\vec{p}$. This can also be written as the following scalar product:

$$d^2_{Euclid}(\vec{x}, \vec{p}) = (\vec{x} - \vec{p})' \cdot (\vec{x} - \vec{p}) \quad (9)$$

Where $'$ denotes the vector transpose. Note that both equations "Eq. (8)" and "Eq. (9)" compute the square of the Euclidean distance, d^2 instead of d. The Euclidean distance is probably the most commonly used distance measure in pattern recognition.

City Block Distance

The computation of the Euclidean distance involves computing the squares of the individual differences thus involving many multiplications. To reduce the computational complexity, one can also use the absolute values of the differences instead of their squares. This is similar to measuring the distance between two points on a street map: We go three blocks to the East, then two blocks to the South (instead of straight trough the buildings as the Euclidean distance would assume). Then we sum up all the absolute values for all the dimensions of the vector space.

Weighted Euclidean Distance

Both the Euclidean distance and the City Block distance are treating the individual dimensions of the feature space equally, i.e., the distances in each dimension contributes in the same way to the overall distance. But if we remember our example from section 2.1, we see that for real–world applications, the individual dimensions will have different scales also. While in our office the temperature values will have a range of typically between 18 and 22 degrees Celsius, the humidity will have a range from 40 to 60 percent relative humidity. While a small difference in humidity of e.g., 4 percent relative humidity might not even be noticed by a person, a temperature difference of 4 degrees Celsius certainly will. In Fig. 5, we see a more abstract example involving two classes and two dimensions. The dimension x_1 has a wider range of values than dimension x_2, so all the measured values (or prototypes) are spread wider along the axis denoted as "x_1" as compared to axis "x_2". Obviously, a Euclidean or City Block distance measure would give the wrong result, classifying the unknown vector as "class A" instead of "class B" which would (probably) be the correct result.

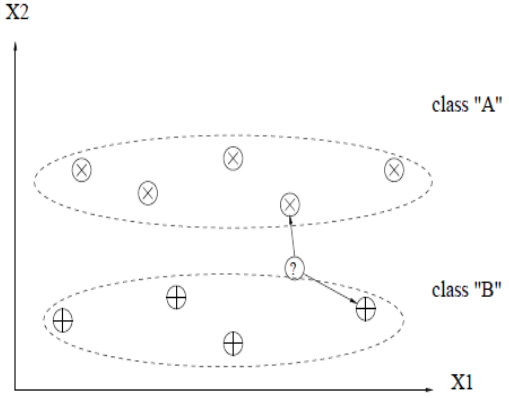

Fig. 5: Two dimensions with different scales

To cope with this problem, the different scales of the dimensions of our feature vectors have to be compensated when computing the distance. This can be done by multiplying each contributing term with a scaling factor specific for the respective dimension. This leads us to the so–called "*Weighted Euclidean Distance*"

Mahalanobis Distance

So far, we can deal with different scales of our features using the weighted Euclidean distance measure. This works very well if there is no correlation between the individual features as it would be the case if the features we selected for our vector space were statistically independent from each other. What if they are not? Fig. 6 shows an example in which the features x_1 and x_2 are correlated.

Obviously, for both classes A and B, a high value of x_1 correlates with a high value for x_2 (with respect to the mean vector (center) of the class), which is indicated by the orientation of the two ellipses. In this case, we would want the distance measure to regard both the correlation and scale properties of the features. Here a simple scale transformation will not be sufficient. Instead, the correlations between the individual components of the feature vector will have to be regarded when computing the distance between two vectors. This leads us to a new distance measure, the so–called Mahalanobis Distance.

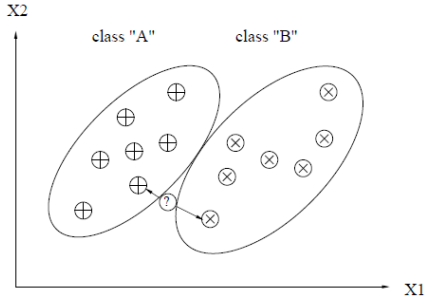

Fig. 6: Correlated Features

IV. Dynamic Time Warping

In the last section, we were dealing with the task of classifying single vectors to a given set of classes which were represented by prototype vectors computed from a set of training vectors. Several distance measures were presented, some of them using additional sets of parameters (e.g., the covariance matrices) which also had to be computed from training vectors.

How does this relate to speech recognition?

As we know that our speech signal is represented by a series of feature vectors which are computed every 10 ms. A whole word will comprise dozens of those vectors, and we know that the number of vectors (the duration) of a word will depend on how fast a person is speaking. Therefore, our classification task is different from what we have learned before. In speech recognition, we have to classify not only single vectors, but sequences of vectors. Let's assume we would want to recognize a few command words or digits. For an utterance of a word w which is T_X vectors long, we will get a sequence of vectors $\tilde{X} = \{\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{X-1}\}$ from the acoustic preprocessing stage. What we need here is a way to compute a "distance" between this unknown sequence of vectors $\tilde{X}$ and known sequences of vectors $\widetilde{W}_k = \left\{\vec{w}_{k0}, \vec{w}_{k1}, \ldots, \vec{w}_{kT_{W_k}}\right\}$ which are prototypes for the words we want to recognize. Let our vocabulary (here: the set of classes Ω) contain V different words $w_0, w_1, \ldots w_{V-1}$. In analogy to the Nearest Neighbor classification task from section 2.1, we will allow a word w_v (here: class $w_v \in \Omega$) to be represented by a set of prototypes $\widetilde{W}_{k,\omega_v}, k = 0, 1, \ldots, \left(K_{\omega_v} - 1\right)$ to reflect all the variations possible due to different pronunciation or even different speakers.

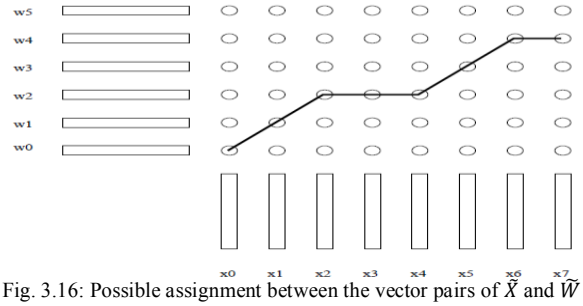

Fig. 3.16: Possible assignment between the vector pairs of $\tilde{X}$ and $\widetilde{W}$

The Dynamic Programming Algorithm

In the following formal framework we will iterate through the matrix column by column, starting with the leftmost column and beginning each column at the bottom and continuing to the top. For ease of notation, we define $\mathrm{d}(\mathrm{i},\mathrm{j})$ to be the distance $\mathrm{d}(\vec{w}_\mathrm{i}, \vec{x}_\mathrm{j})$ between the two vectors $\vec{w}_\mathrm{i}$ and $\vec{x}_\mathrm{j}$.

Let $\delta_j(i)$ be the accumulated distance $\delta(i,j)$ at grid point (i,j) and $\delta_{j-1}(i)$ the accumulated distance $\delta(i,j-1)$ at grid point $(i,j-1)$. It should be mentioned that it possible to use a single array for time indices j and $j-1$. One can overwrite the old values of the array with the new ones. However, for clarity, the algorithm using two arrays is described here and the formulation for a single–array algorithm is left to the reader.

To keep track of all the selections among the path hypotheses during the optimization, we have to store each path alternative chosen for every grid point. We could for every grid point (i,j) either store the indices k and l of the predecessor point (k,l) or we could only store a code number for one of the three path alternatives (horizontal, diagonal and vertical path) and compute the predecessor point (k,l) out of the code and the current point (i,j). While the description of the DTW classification algorithm might let us think that one would compute all the distances sequentially and then select the minimum distance, it is more useful in practical applications to compute all the distances between the unknown vector sequence and the class prototypes in parallel. This is possible since the DTW algorithm needs only the values for time index t and $(t-1)$ and therefore there is no need to wait until the utterance of the unknown vector sequence is completed. Instead, one can start with the recognition process immediately as soon as the utterance begins (we will not deal with the question of how to recognize the start and end of an utterance here).

To do so, we have to reorganize our search space a little bit. First, let's assume the total number of all prototypes over all classes is given by M. If we want to compute the distances to all M prototypes simultaneously, we have to keep track of the accumulated distances between the unknown vector sequence and the prototype sequences individually. Hence, instead of the column (or two columns, depending on the implementation) we used to hold the accumulated distance values for all grid points; we now have to provide M columns during the **DTW** procedure. Now we introduce an additional "virtual" grid point together with a specialized local path alternative for this point: The possible predecessors for this point are defined to be the upper–right grid points of the individual grid matrices of the prototypes. In other words, the virtual grid point can only be reached from the end of each prototype word, and among all the possible prototype words, the one with the smallest accumulated distance is chosen. By introducing this virtual grid point, the classification task itself (selecting the class with the smallest class distance) is integrated into the framework of finding the optimal path.

Now all we have to do is to run the DTW algorithm for each time index j and along all columns of all prototype sequences. At the last time slot $(T_W - 1)$ we perform the optimization step for the virtual grid point, i.e; the predecessor grid point to the virtual grid point is chosen to be the prototype word having the smallest accumulated distance. Note that the search space we have to consider is spanned by the length of the unknown

vector sequence on one hand and the sum of the length of all prototype sequences of all classes on the other hand. The backtracking procedure can of course be restricted to keeping track of the final optimization step when the best predecessor for the virtual grid point is chosen. The classification task is then performed by assigning the unknown vector sequence to the very class to which the prototype belongs to whose word end grid point was chosen.

Of course, this is just a different (and quite complicated) definition of how we can perform the DTW classification task. Therefore, only a verbal description was given and we did not bother with a formal description. However, by the reformulation of the DTW classification we learned a few things:

- The DTW algorithm can be used for real–time computation of the distances.
- The classification task has been integrated into the search for the optimal path.
- Instead of the accumulated distance, now the optimal path itself is important for the classification task.

V. Conclusion

Speech is the primary, and the most convenient means of communication between people. Whether due to technological curiosity to build machines that mimic humans or desire to automate work with machines, research in speech and speaker recognition, as a first step toward natural human-machine communication, has attracted much enthusiasm over the past five decades. We have also encountered a number of practical limitations which hinder a widespread deployment of application and services. In most speech recognition tasks, human subjects produce one to two orders of magnitude less errors than machines. There is now increasing interest in finding ways to bridge such a performance gap. What we know about human speech processing is very limited. Although these areas of investigations are important the significant advances will come from studies in acoustic-phonetics, speech perception, linguistics, and psychoacoustics. Future systems need to have an efficient way of representing, storing, and retrieving knowledge required for natural conversation. This paper attempts to provide a comprehensive survey of research on speech recognition and to provide some yearwise progress to this date. Although significant progress has been made in the last two decades, there is still work to be done, and we believe that a robust speech recognition system should be effective under full variation in: environmental conditions, speaker variability s etc. Speech Recognition is a challenging and interesting problem in and of itself. We have attempted in this paper to provide a comprehensive cursory, look and review of how much speech recognition technology progressed in the last 60 years. Speech recognition is one of the most integrating areas of machine intelligence, since; humans do a daily activity of speech recognition. Speech recognition has attracted scientists as an important discipline and has created a technological impact on society and is expected to flourish further in this area of human machine interaction. We hope this paper brings about understanding and inspiration amongst the research communities of ASR.

References

[1] Sadaoki Furui, 50 years of Progress in speech and Speaker Recognition Research, *ECTI Transactions on Computer and Information Technology,Vol.1. No.2, November 2005*.

[2] K. H. Davis, R. Biddulph, and S. Balashek, Automatic recognition of spoken Digits, *J. Acoust. Soc. Am., 24(6):* 637-642, 1952.

[3] H. F. Olson and H. Belar, Phonetic Typewriter, J. Acoust. Soc. Am., 28(6): 1072-1081, 1956.

[4] D. B. Fry, Theoritical Aspects of Mechanical speech Recognition, and P. Denes, The design and Operation of the Mechanical Speech Recognizer *at Universtiy College London, J. British Inst. Radio Engr., 19: 4*, 211 - 299, 1959.

[5] J.W. Forgie and C. D. Forgie, Results obtained from a vowel recognition computer program, *J.A.S.A., 31(11)*, pp.1480-1489. 1959.

[6] J. Suzuki and K. Nakata, Recognition of Japanese Vowels Preliminary to the Recognition of Speech, *J. Radio Res. Lab 37(8)*:193-212, 1961.

[7] T. Sakai and S. Doshita, The phonetic typewriter, *Information processing 1962, Proc .IFIP Congress, 1962*.

[8] K. Nagata, Y. Kato, and S. Chiba, Spoken Digit Recognizer for Japanese Language, *NEC Res. Develop, No. 6, 1963.*

[9] T. B. Martin, A. L. Nelson, and H. J. Zadell, Speech Recognition & Feature Abstraction Techniques, *Tech. Report AL-TDR-64-176, Air Force Avionics Lab, 1964.*

[10] T. K. Vintsyuk, Speech Discrimination by Dynamic Programming, *Kibernetika, 4(2):81-88, Jan.-Feb.1968*.

[11] H. Sakoe and S. Chiba, Dynamic programming: algorithm optimization for spoken word recognition, IEEE Tran. Acoustics, Speech, Signal Proc., ASSP-26(1). pp. 43- 49, 1978.

[12] D. R. Reddy, An Approach to Computer Speech Recognition by Direct Analysis of the Speech Wave, *Tech. Report No. C549, Computer Science Dept., Stanford Univ., September 1966*.

[13] V. M. Velichko and N. G. Zagoruyko, Automatic Recognition of 200 words , *Int. J. Man-Machine Studies, 2:223, June 1970*.

[14] H. Sakoe and S. Chiba, Dynamic Programming Algorithm Optimization for Spoken Word Recognition, *IEEE Trans. Acoustics, Speech, Signal Proc., ASSP-26(1):43-49, February 1978.*

[15] F. Itakura, Minimum Prediction Residual Applied to Speech Recognition, *IEEE Trans. Acoustics, Speech, Signal Proc., ASSP-23(1): 67-72, February 1975.*

[16] C.C. Tappert, N. R. Dixon, A.S. Rabinowitz, and W. D. Chapman, Automatic Recognition of Continuous Speech Utilizing Dynamic Segmentation, Dual Classification, Sequential Decoding and Error Recover, *Rome Air Dev. Cen, Rome, NY, Tech. Report TR-71-146,1971.*