Implementation of Speech Recognition System for Mobile Phones

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Abhinav Singh
(140101002)

under the guidance of

Prof. Pradip K. Das

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis titled "**Implementation of Speech Recognition System for Mobile Phones**" is a bonafide work of **Abhinav Singh (Roll No. 140101002)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Prof. Pradip K. Das**

Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam

November, 2018

Guwahati

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Pradip K. Das, for his continuous support, enthusiasm and immense knowledge. He has always guided and motivated me towards the completion of my work despite his busy schedule.

I also take this opportunity to thank my mentor Parabattina Bhagath P for his consistent assistance and the Department of Computer Science and Engineering, IIT Guwahati for providing me the facilities that were essential to carry out my project.

Abstract

Voice centric interfaces have become widely available with the evolution of mobile phones. Progress in speech recognition algorithms, increased network capabilities and computational powers have promoted this growth. Applications of speech recognition include, but are not limited to chat-bots, dialing, web search, and dictation of text messages. In this work, we implement a portable word based embedded speech recognition library, a graphical user interface for training, and a demonstrative android application.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Speech recognition - the process of recognizing human speech and converting it to text, has emerged as a new trending technology in the IT industry. Mobile users desire devices that they can control without much physical interaction. Applications like Cortana are steadily gaining popularity with their hands free features like taking reminders and notes, opening other applications, and their witty replies. With the advancement in network capabilities, these applications can send data over to the servers where sophisticated algorithms provide exceptionally accurate recognition results. However, in countries like India where over 75 percent of the population has no internet, these applications are not apt. Moreover, privacy is a big issue as these systems share the voice data with third party servers.

In this work, we aim to implement a portable word based embedded speech recognition system for mobile phones, along with a graphics user interface for training on desktop, and a demonstrative voice controlled assistant for android. In chapter 2, we discuss a few related works and technologies of the proposed system. In chapter 3 we present the design and implementation of the proposed systems. In chapter 4 we discuss the expermients performed and their observations. Finally, we draw out conclusions and discuss the future work in chapter 5. In the appendix, we provide details of the assumed knowledge.

Chapter 2

Related Works and Technologies

2.1 Speech Feature Extraction

[LLNB04] investigate various feature extraction techniques - MFCC, LPCC, PLP for speech recognition on mobile phones. Their study shows that MFCC features give the most accurate result. In [SN15] a review of the latest feature extraction techniques is done. In this work, various advantages and disadvantages of techniques like RASTA and PLDA are pointed out.

2.2 Speech Modeling Techniques

[MAA17] presents a comparative analysis of Hidden Markov Model, Multi-Layer Perceptron, and Dynamic Multi-Layer Perceptron. They conclude that DMLPs are infeasible on mobile phones, while HMMs are the most accurate, and MLPs are the fastest. With the increase in size of the vocabulary, the execution performance of HMM suffers. [TS18] talks about implementing a lightweight speech recognition system using HMMs for Gujarati language and achieves 87% accuracy.

2.3 Distance Measures

Cepstral coefficients are widely used in speech recognition. Various distortion methods have been applied to find the distance between two coefficients. [Toh87] brings out a novel distance measure, the Tohkura distance - a weighted distance that outperforms Itakura distance and Euclidean distance. It also mentions a more accurate Mahalanobis distance and the main difficulties for applying it to speech recognition.

2.4 Native Development on Android

[FWSC10] discusses a new component development approach by using Java Native Interface technology. The Native Development Kit (NDK) is a set of tools that allows writing native code for Android in C++. Speech processing and recognition requires fine grain control over memory management and speed, hence NDK is preferred over Dalvik Virtual Machine. C and C++ libraries act as the glue code while the frontends can be written with Java or Swift.

2.5 Network Speech Recognition

In this mode of speech recognition, the feature extraction and recognition are both delegated to the network server. The server is capable of doing complex computation and returns extremely accurate results. One of the major constraints of this mode is the dependence on telephone network and the performance degradation caused by low bit rate codecs, which becomes severe in the presence of transmission errors [KTH$^+$11].

Chapter 3

Design and Implementation

3.1 Speech Recognition Library

The library is written in C++11 for its performance, familiarity and wide spread adoption including on android, iOS, and desktops. C++11 and later versions provide smart pointers and a standard threading library which is important for portability. To aid building the library across various platforms, the CMake build system is used.

The speech recognition system works as given in the following flowchart:

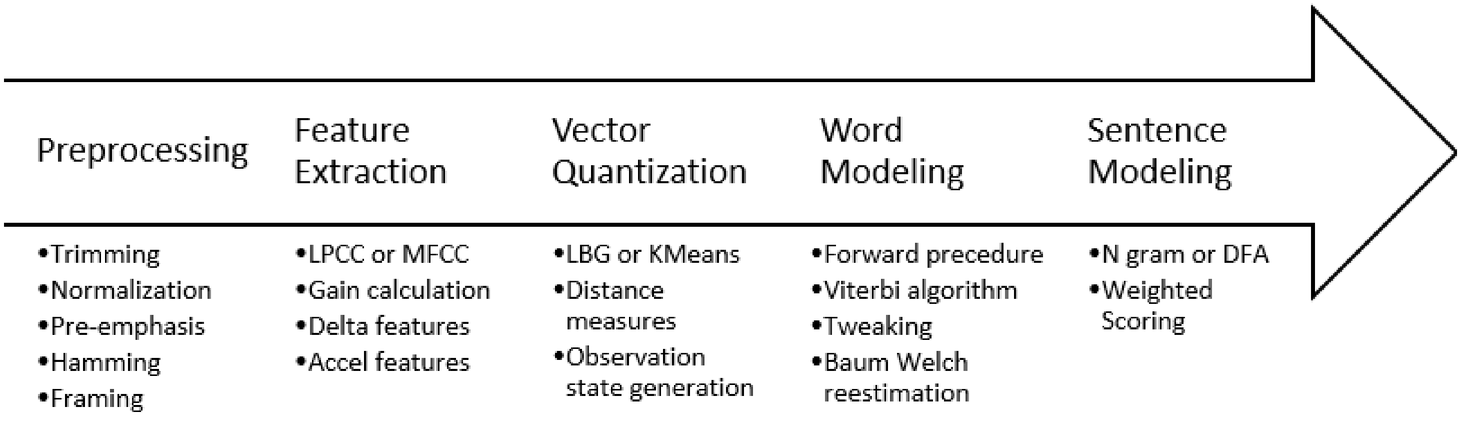


Fig. 3.1 Process flowchart

The appendix describes each of the stages of this process in good detail along with code implementation examples.

The library is written such that its clients do not have to dive deep into the technicalities of speech recognition. It has been divided into two abstraction layers:

1. Base

   Provides implementation of various algorithms and techniques - Preprocessing, LPCC, MFCC, Derivative features, KMeans, LBG, HMM, NGram. Data objects - Codebook, Feature, Model, Gram are also described in this layer.

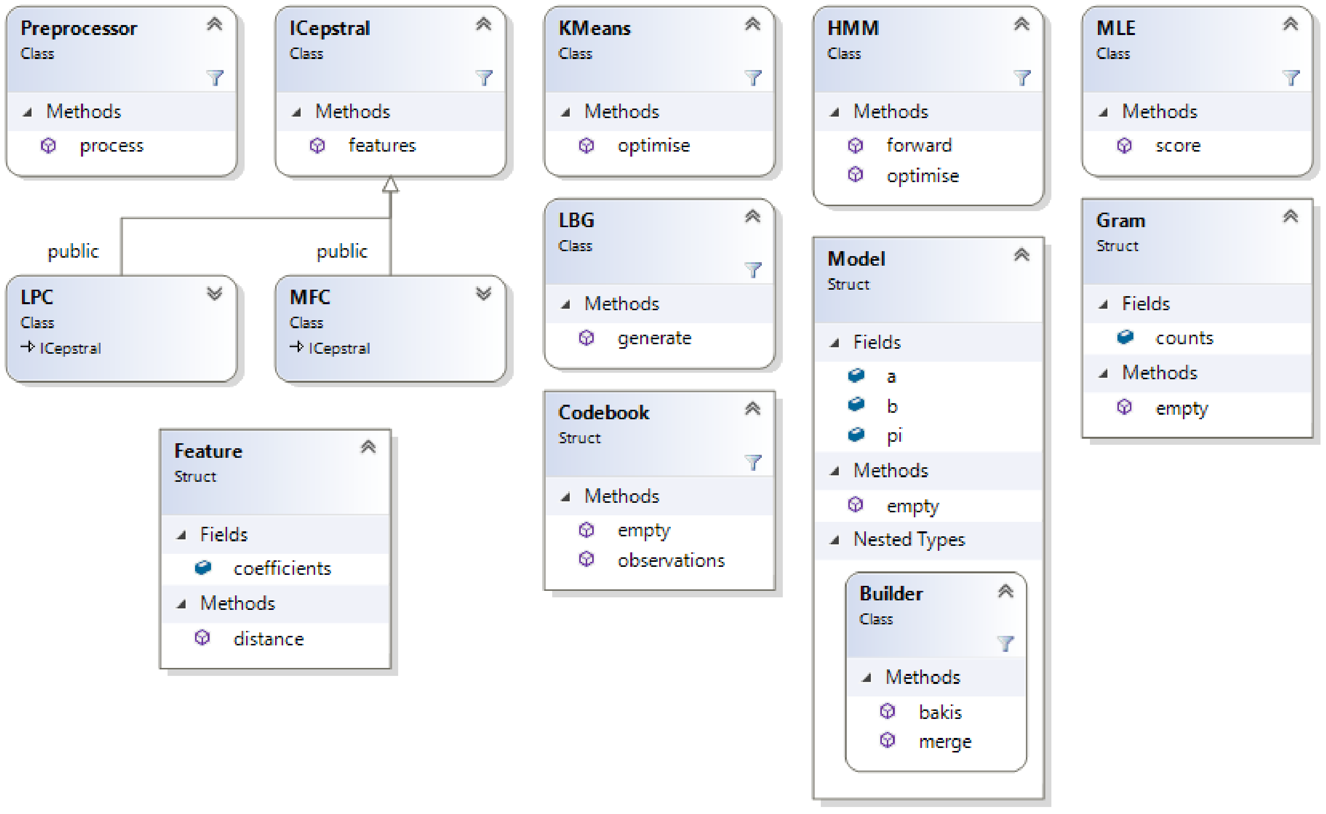


Fig. 3.2 Architecture of Base

Scaling has been done wherever necessary to avoid the floating point errors, and various optimizations have been made whenever possible.

This layer is not parallelized and strictly does not interact with the input/output e.g. WAV audio or saving/loading the data objects to the file system.

2. Word

Provides training and testing interfaces. The clients of the library are only allowed to interact with this layer. This layer foresees construction and testing of HMM and NGram models. It implements handling inputs - WAV audio files, word list, sentence list, etc. It also implements functions to save and read generated data objects to and from the file system.

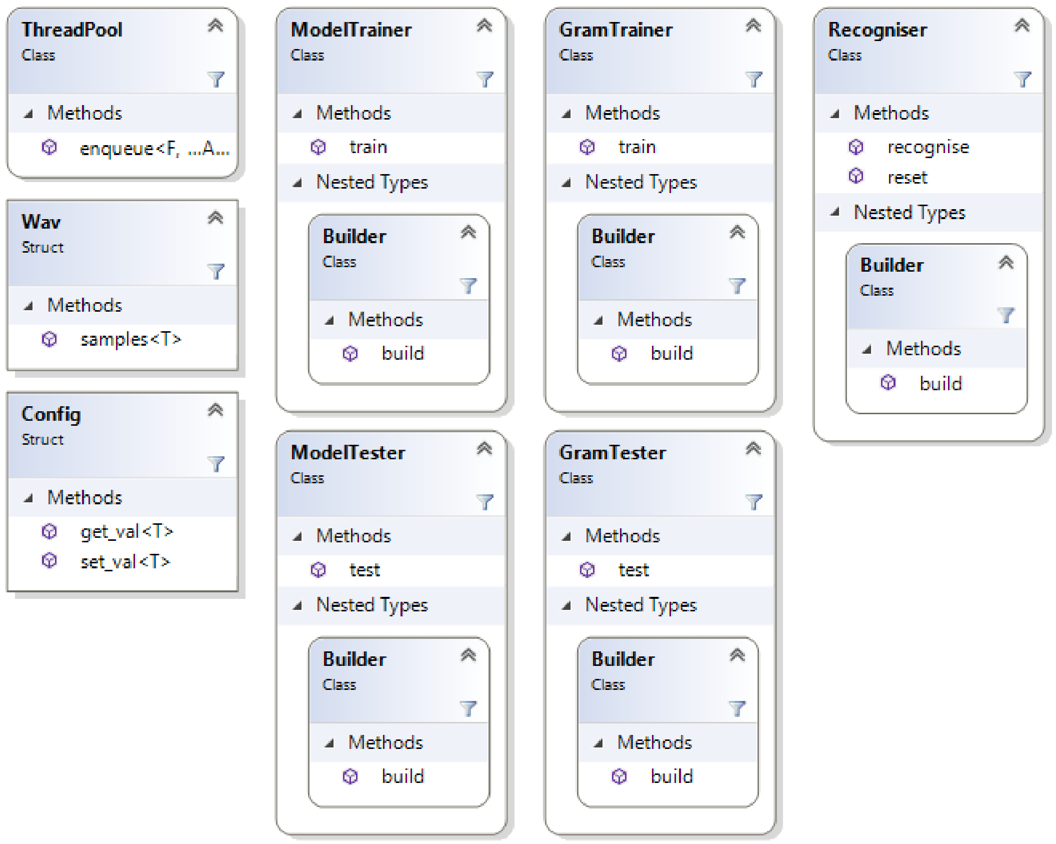


Fig. 3.3 Architecture of Word

This layer also handles the parallelization and configuration of the library.

The following tasks are parallelized:

- Feature generation of multiple utterances
- Generation of multiple word models using HMM
- Matching of the input utterance with multiple word models

Key	Type	Description
q_cache	bool	whether cached training files should be used
n_thread	int	number of threads used for parallel execution
q_trim	bool	whether the samples should be trimmed for background noise
x_frame	int	number of samples in a frame
x_overlap	int	number of samples to be overlapped while framing
cepstra	string	mfc or lpc variants of feature generation
n_cepstra	int	number of features
n_predict	int	P of autocorrelation, only for lpc
q_gain	bool	whether gain term should be added to features
q_delta	bool	whether delta terms should be added to features
q_accel	bool	whether accel terms should be added to features
x_codebook	int	size of codebook
n_state	int	number of states in HMM
n_bakis	int	connectedness of initial bakis model for HMM
n_retrain	int	number of times each model should be trained
n_gram	int	number of previous words to be considered for prediction
q_dfa	bool	command based word prediction or probability based
gram_weight	double	linear weight for the final scoring with recognition result
cutoff_score	double	cutoff for final score

Table 3.1 Keys for the configuration

As this is a word based recognition system, the basic unit of recognition in the library is a word. During the training, HMM Models are generated for each word, and the probabilities of occurrence of the words are calculated by parsing the sentence list. These probabilities are saved as NGram or DFA models depending upon the configuration.

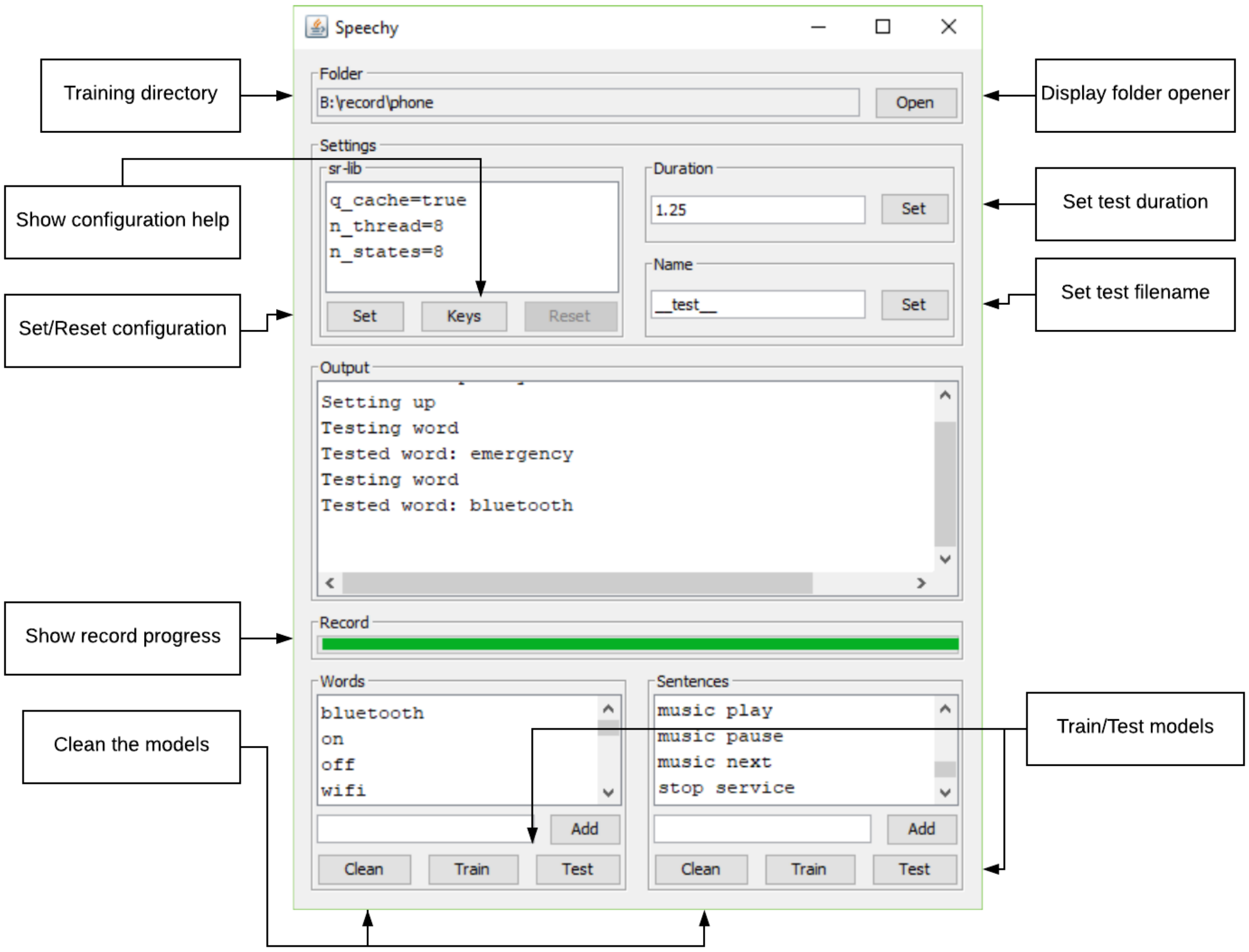
During testing, a sentence is input as a combination of words. For each word A in the sentence, and for each word B in the training data, following steps are taken:

- Using HMM model of B find probability of occurrence of A.
- Using NGram models, find probability of occurrence of B given A's position in the input sentence.
- Final score is a linear combination of these two scores.

The word B with the maximum final score is best possible recognition. If its score is lesser than cutoff score, discard it, else output it.

3.2 Graphical User Interface for Training

The desktop graphical user interface is written in Swing Java framework. The interface can be run on Windows or Linux based OSes. It uses the Java Native Interface (JNI) for communicating with the speech recognition library.



Folder: B:\record\phone

sr-lib:
```
q_cache=true
n_thread=8
n_states=8
```

Duration: 1.25

Name: __test__

Output:
```
Setting up
Testing word
Tested word: emergency
Testing word
Tested word: bluetooth
```

Words:
```
bluetooth
on
off
wifi
```

Sentences:
```
music play
music pause
music next
stop service
```

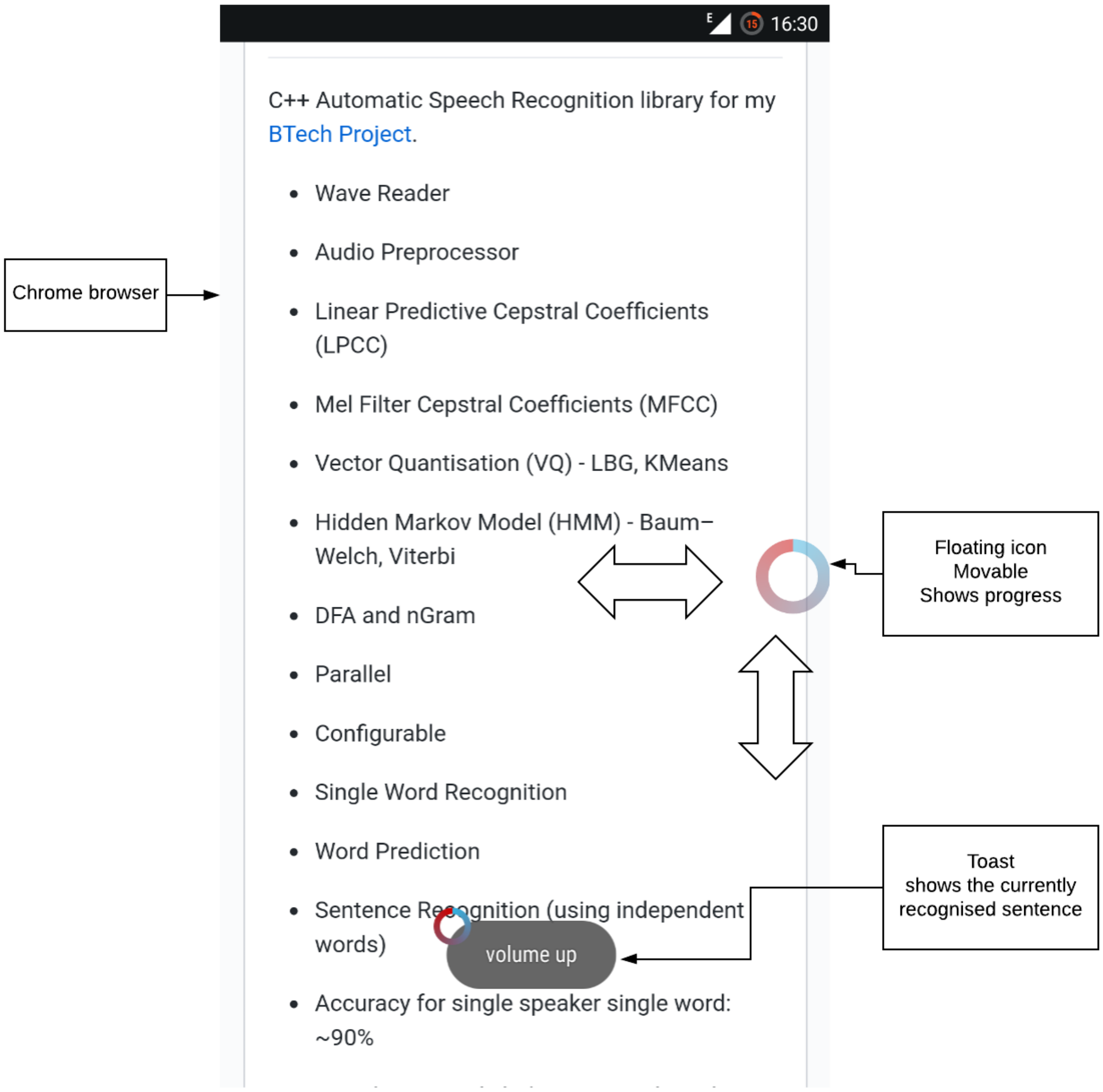
Fig. 3.4 Graphics User Interface for Training

The interface has been made intuitive. Buttons are enabled only when required. For example, Test and Clean button will not be available until training is finished. At any moment only one task can be performed from the interface, and data race conditions between GUI and IO are avoided. The system is completely asynchronous and does not block the user interface.

3.3 Voice Controlled Assistant for Android

The android application is written in Kotlin and uses the Android NDK to run the speech recognition library. The design of this application is inspired by Facebook Messenger chat heads. The main button is controlled by a service class that runs in the background. The main button is completely movable over the screen. It displays a round progress bar of the recording being done. A single click on the button starts recording for a sentence and user is expected to speak. A message toast is shown with every recognized word. If the recognition results match the list of actions, the corresponding action is performed. Currently, the application supports the following actions:

- Open applications
- Enable/disable Bluetooth
- Control music player
- Enable/disable Wi-Fi
- Make phone calls
- Control brightness
- Enable/disable torch
- Control volume



C++ Automatic Speech Recognition library for my BTech Project.

- Wave Reader
- Audio Preprocessor
- Linear Predictive Cepstral Coefficients (LPCC)
- Mel Filter Cepstral Coefficients (MFCC)
- Vector Quantisation (VQ) - LBG, KMeans
- Hidden Markov Model (HMM) - Baum–Welch, Viterbi
- DFA and nGram
- Parallel
- Configurable
- Single Word Recognition
- Word Prediction
- Sentence Recognition (using independent words)
- Accuracy for single speaker single word: ~90%

Fig. 3.5 Android Voice Controlled Assistant

The application makes extensive use of Kotlins co-routines to provide responsive GUI feedback. With the architecture of the application, more actions can be added with ease. To improve accuracy, the application also saves the recordings that are recognized so they can be trained by connecting to the desktop.

Chapter 4

Experiment and Observation

We obtained two datasets for the purpose of experimentation - digits dataset of 15 utterances per digit, and commands (for the android project) dataset with 25 utterances per word. The recordings were done in moderately quiet environments using a general-purpose microphone.

For the obtained datasets, the trimming algorithm presented very accurate results. The results were close to 88% for the error of speech signal's endpoints missing by 100ms.

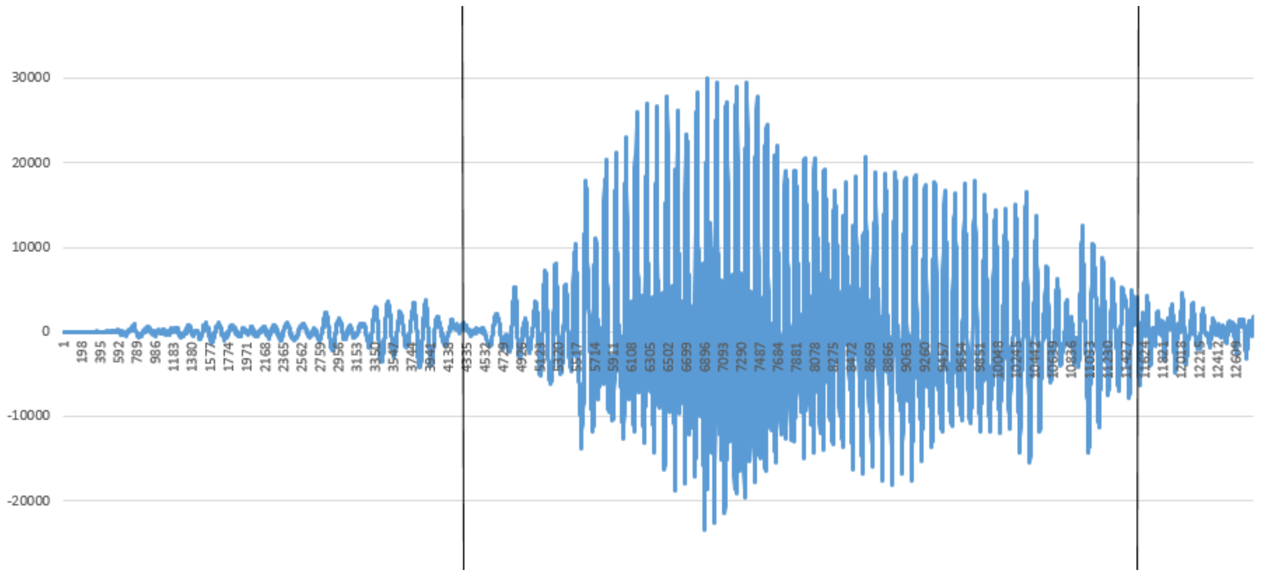


Fig. 4.1 Automatic trimming of nine

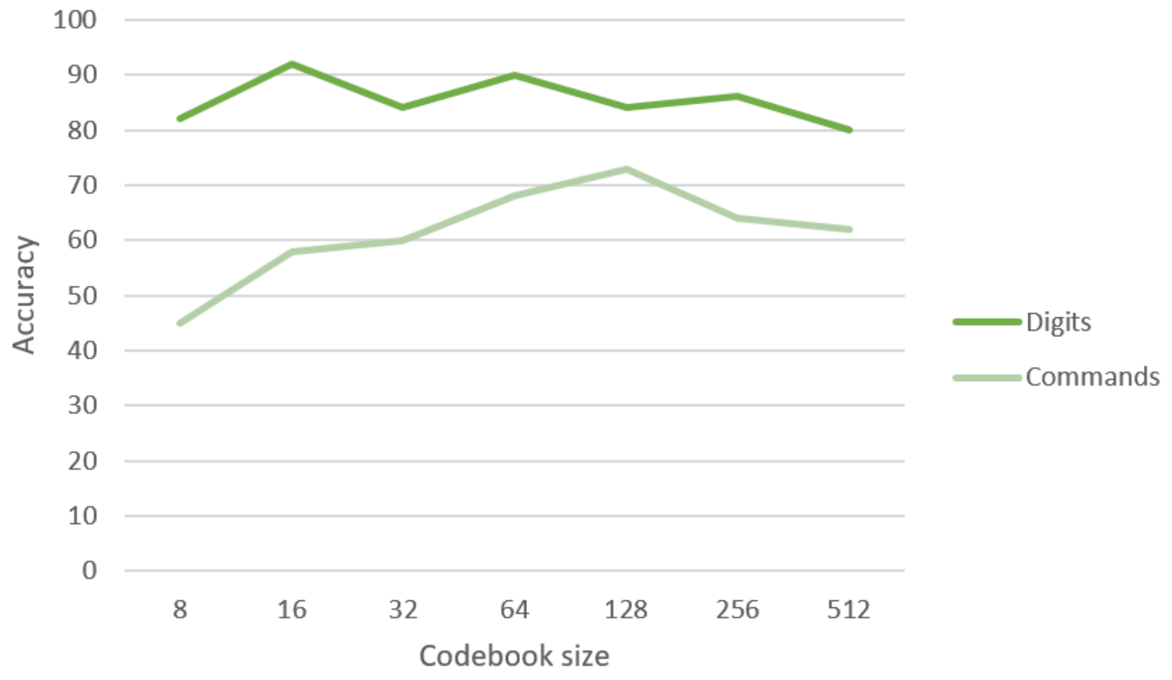


Fig. 4.2 Codebook size vs Accuracy

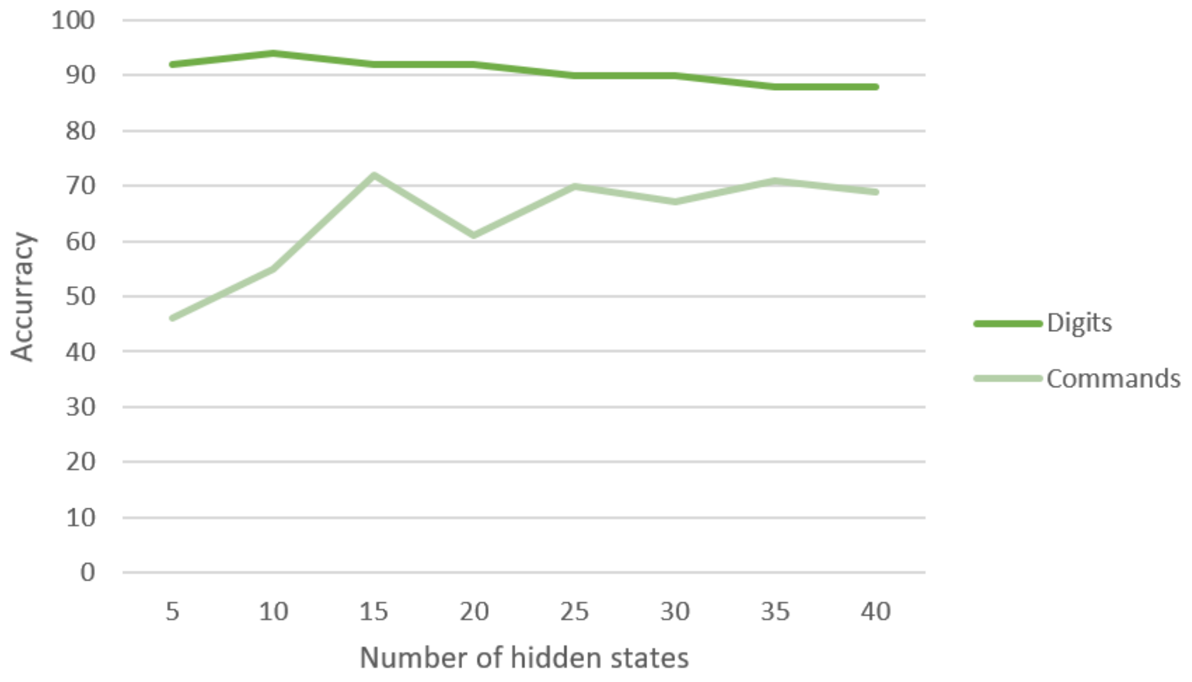


Fig. 4.3 Number of hidden states vs Accuracy

It was observed that for the two datasets, different configurations were necessary. The accuracy of digits dataset decreased if delta or delta delta features were added; on the contrary, it increased for commands dataset. The optimal codebook size and the number of HMM states were higher for commands dataset compared to digits dataset. Moreover, bakis model with more jumps improved the performance for commands dataset.

Description	Digits dataset	Commands dataset
Number of words	10	21
Number of utterances	150	525
Delta features	No	Yes
Delta delta feautres	No	Yes
Codebook size	16	128
Number of hidden states	10	15
Initial feed forward connectedness	1	3
Training time (s)	3.3	7.4
Recognition time (ms)	8	9
Accuracy (%)	94	73

Table 4.1 Results of speech recognition

For profiling the library on a desktop, Visual Studio Profiler was used and a Intel Corei7 3.0GHz + 8GB RAM laptop was selected. During training, the maximum CPU usage increased up to 80% while the maximum RAM usage was below 20MB for our datasets.

For profiling the android application, Android Studio Profiler was used and an octa-core 1.7GHz + 2GB RAM mobile phone was selected. During testing, the maximum CPU usage increased up to 50% while the RAM stayed constant at 12MB. The average recognition time for input utterances was 15ms.

Chapter 5

Conclusion and Future Work

Our speech recognition system gave satisfactory accuracy while also being real-time. It works well even with less amounts of training data. It can be used to make simple recognition systems for individual persons and command based devices like wheelchairs or home automation systems. With time, our system can be trained on the previously recognized recordings to improve the performance.

Word recognition systems are easier to train and perform exceptionally well for simple tasks with less number of words. With increase in number of words though, the execution becomes slow as there are more models to match. Hence, word based speech recognition systems are not suitable for large vocabulary systems. For future work, a phoneme based recognition system like tri-phone system can be added to the library.

The graphical user interface was very helpful for training. Feature for automatically dividing the dataset into train and test sets, and then finding accuracy can be added to the interface. Moreover, a brute-force approach can be used to select parameters like codebook size and number of HMM states from a set of pre-chosen values. The android application can be improved significantly by adding more actions and an artificial voice feedback.

References

[E.10] Blahut Richard E. *Fast Algorithms for Digital Signal Processing.* Cambridge University Press, 2010.

[FWSC10] X. Fu, X. Wu, M. Song, and M. Chen. Research on audio/video codec based on android. In *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*, pages 1–4, Sept 2010.

[KTH$^+$11] Anuj Kumar, Anuj Tewari, Seth Horrigan, Matthew Kam, Florian Metze, and John Canny. Rethinking speech recognition on mobile devices, 2011.

[LLNB04] C. Levy, G. Linares, P. Nocera, and J. F. Bonastre. Reducing computational and memory cost for cellular phone embedded speech recognition system. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages V–309–12 vol.5, May 2004.

[LRR78] Ronald W. Schafer Lawrence R. Rabiner. *Digital Processing of Speech Signals.* Prentice-Hall, 1978.

[MAA17] Mohammed Kyari Mustafa, Tony Allen, and Kofi Appiah. A comparative review of dynamic neural networks and hidden markov model methods for mobile on-device speech recognition. *Neural Computing and Applications*, Jun 2017.

[Rab89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[SCS] G. Saha, Sandipan Chakroborty, and Suman Senapati. A new silence removal and endpoint detection algorithm for speech and speaker recognition applications.

[SN15] Divya Gupta Shreya Narang. Speech feature extraction techniques: A review. *International Journal of Computer Science and Mobile Computing*, pages 107–114, March 2015.

[Toh87] Y. Tohkura. A weighted cepstral distance measure for speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(10):1414–1422, Oct 1987.

[TS18] Jinal H. Tailor and Dipti B. Shah. Hmm-based lightweight speech recognition system for gujarati language. In Durgesh Kumar Mishra, Malaya Kumar Nayak, and Amit Joshi, editors, *Information and Communication Technology for Sustainable Development*, pages 451–461, Singapore, 2018. Springer Singapore.

Appendix A

Preprocessing

Normalization

This step is done to avoid the variations in amplitude between sound recordings from different devices.

$$x_i = x_i - x_{mean}$$

Trimming

This step is necessary to save computational resources from being spent on unvoiced parts of the signal. Hence, silent parts of the signal should be removed. The trimming is done using the algorithm in [SCS]. This algorithm uses the background noise features to set thresholds, hence no need to assume any ad-hoc values. If d (Mahalanobis distance) of a sample comes out to be lesser than 3, it is rejected. b_{mean}, b_{sd} are the Gaussian parameters of the background noise.

$$d = x - b_{mean}/b_{sd}$$

Pre-emphasis

Due to the nature of the glottal impulse, the human speech has more energy in the lower frequencies. Hence, the signal is applied with pre emphasis filter (a first order high pass filter) that boosts the higher frequencies.

$$x_i = x_i - \alpha x_{i-1}$$

This enables the acoustic models to gain more information from the higher formants. $\alpha = 0.95$ has been taken for our system.

Framing and Hamming Window

We want to extract features from a small window of speech that represents a sub phoneme. To convert the signal to a stationary signal frame wise, a hamming window is applied on each frame. It disallows discontinuity by suppressing the boundaries of the signal. One of the most commonly used hamming window is:

$$w(i) = 0.54 - 0.46\frac{2\pi(i-1)}{N-1}, \quad 1 \leq i \leq N$$

Appendix B

Feature Extraction

B.1 LPCC

Linear Predictive Coding is a very useful technique for encoding high quality speech at a low bit rates. It provides very accurate estimates of the speech parameters. Its main advantage comes from the fact that it is based on a simplified vocal tract model. LPC includes a speech production function mimicking the air from our lungs, and the error filter mimicking our larynx.

We find out the linear predictive cepstral coefficients (LPCC) using the autocorrelation method from [LRR78].

Autocorrelation

The first step is to find p autocorrelation values for each frame of windowed signal using

$$\sum_{n=0}^{n-1-m} \tilde{x}_l(n)\tilde{x}_l(n+m), \quad m = 0, 1, ..., p.$$

p is the linear prediction order.

Durbin solve

A pXp Toeplitz matrix is created by using the above p autocorrelation values. The Toeplitz matrix is solved using Durbin method to give the LPC coefficients:

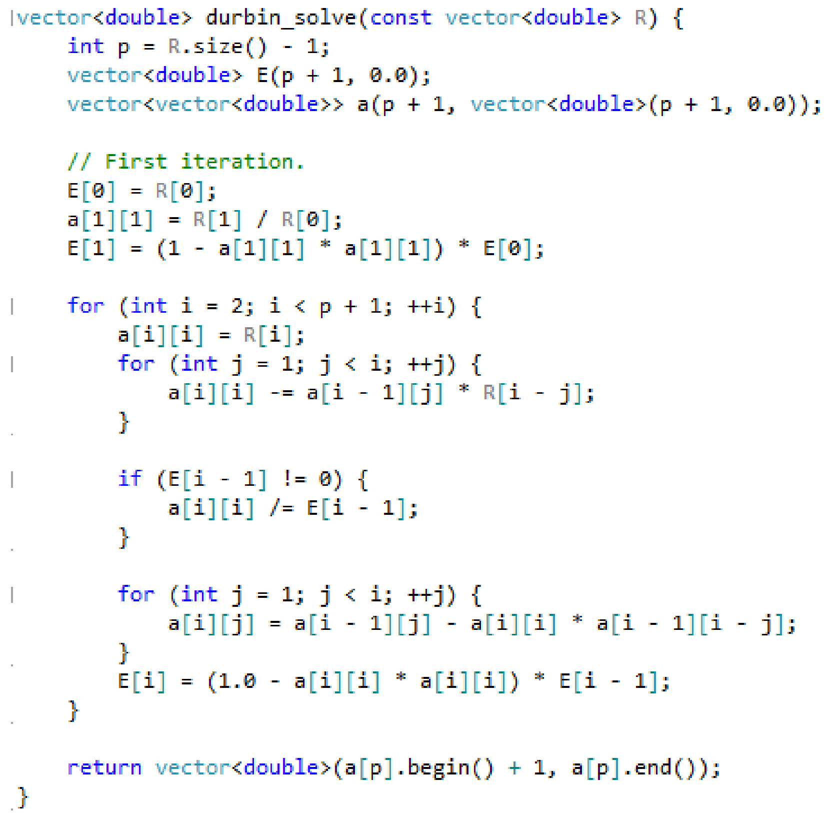
$$E^{(0)} = R(0)$$

$$k_i = \frac{R(i) - \sum_{j=1}^{i-1} \alpha_j^{i-1} R(|i-j|)}{E^{i-1}}, \quad 1 \leq i \leq p.$$

$$\alpha_i^{(i)} = k_i$$

$$\alpha_j^{(i)} = \alpha_i^{(i-1)} - k_i \alpha_{i-1}^{(i-1)}, \quad 1 \leq j \leq i-1.$$

$$E^i = (1 - k_i^2) E^{i-1}$$



```
vector<double> durbin_solve(const vector<double> R) {
    int p = R.size() - 1;
    vector<double> E(p + 1, 0.0);
    vector<vector<double>> a(p + 1, vector<double>(p + 1, 0.0));

    // First iteration.
    E[0] = R[0];
    a[1][1] = R[1] / R[0];
    E[1] = (1 - a[1][1] * a[1][1]) * E[0];

    for (int i = 2; i < p + 1; ++i) {
        a[i][i] = R[i];
        for (int j = 1; j < i; ++j) {
            a[i][i] -= a[i - 1][j] * R[i - j];
        }

        if (E[i - 1] != 0) {
            a[i][i] /= E[i - 1];
        }

        for (int j = 1; j < i; ++j) {
            a[i][j] = a[i - 1][j] - a[i][i] * a[i - 1][i - j];
        }
        E[i] = (1.0 - a[i][i] * a[i][i]) * E[i - 1];
    }

    return vector<double>(a[p].begin() + 1, a[p].end());
}
```

Fig. B.1 Implementation of Durbin's solve

Cepstral Coefficients

In the last step, the LPC cepstral coefficients are derived from the LPC coefficients. σ^2 is the gain term of the LPC model.

$$c_0 = ln\sigma^2$$

$$c_m = a_m + \sum_{k=1}^{m-1} (\frac{k}{m}) c_k a_{m-k}, \quad 1 \leq m \leq p.$$

$$c_m = \sum_{k=1}^{m-1} (\frac{k}{m}) c_k a_{m-k}, \quad m > p.$$

LPCC are proved to be more robust than LPC. A sine window is applied over these cepstral coefficients to desensitize them towards noise and noise like sounds.

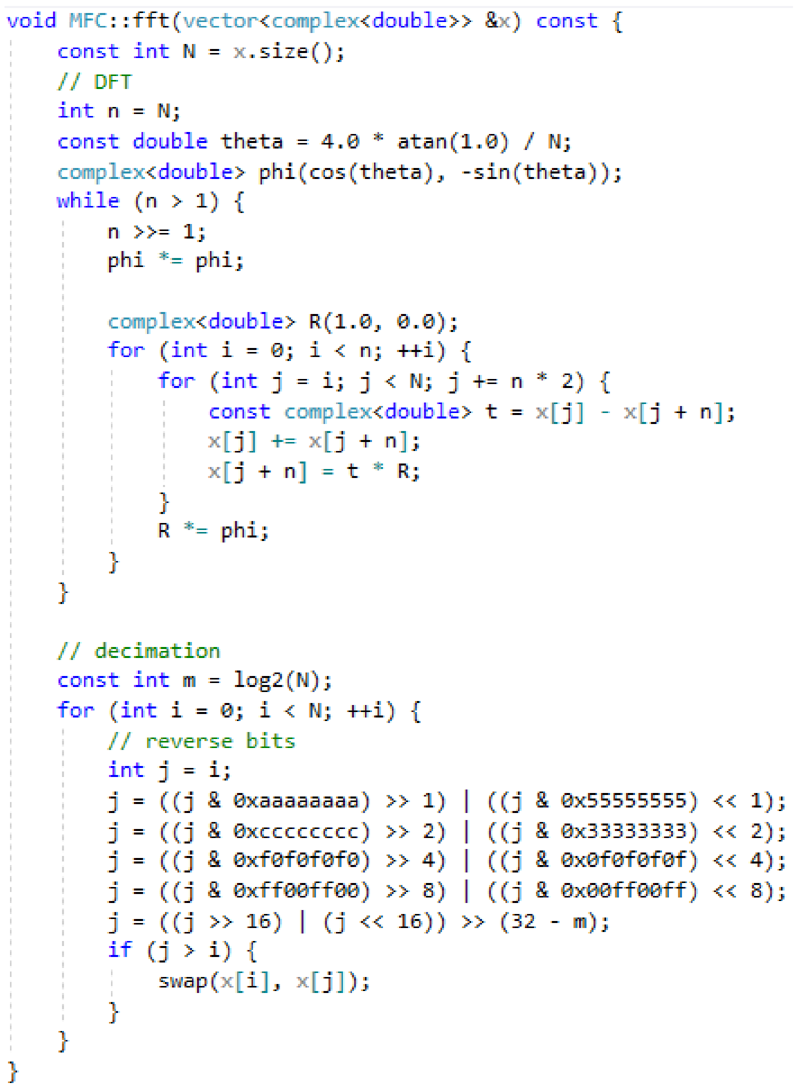
B.2 MFCC

MFCC is one of the most popular feature extraction technique. MFCC aims to mimic the human hearing by penalizing higher frequencies and boosting lower frequencies. The mapping between Hertz and Mel is logarithmic above 1000Hz and linear below 1000 Hz. The Mel scale is based on an empirical study conducted by Stevens and Volkmann.

$$m = 2595 log_1 0(1 + \frac{f}{700})$$

DFT

The frequency spectrum is extracted by finding the Discrete Fourier Transform (using FFT) of each frame. We implement the FFT using Cooley-Tukey, in-place, breadth-first, decimation-in-frequency algorithm described in [E.10].



```
void MFC::fft(vector<complex<double>> &x) const {
    const int N = x.size();
    // DFT
    int n = N;
    const double theta = 4.0 * atan(1.0) / N;
    complex<double> phi(cos(theta), -sin(theta));
    while (n > 1) {
        n >>= 1;
        phi *= phi;

        complex<double> R(1.0, 0.0);
        for (int i = 0; i < n; ++i) {
            for (int j = i; j < N; j += n * 2) {
                const complex<double> t = x[j] - x[j + n];
                x[j] += x[j + n];
                x[j + n] = t * R;
            }
            R *= phi;
        }
    }

    // decimation
    const int m = log2(N);
    for (int i = 0; i < N; ++i) {
        // reverse bits
        int j = i;
        j = ((j & 0xaaaaaaaa) >> 1) | ((j & 0x55555555) << 1);
        j = ((j & 0xcccccccc) >> 2) | ((j & 0x33333333) << 2);
        j = ((j & 0xf0f0f0f0) >> 4) | ((j & 0x0f0f0f0f) << 4);
        j = ((j & 0xff00ff00) >> 8) | ((j & 0x00ff00ff) << 8);
        j = ((j >> 16) | (j << 16)) >> (32 - m);
        if (j > i) {
            swap(x[i], x[j]);
        }
    }
}
```

Fig. B.2 Implementation of FFT

LMFB

This step is performed by using filter banks with filters centered according to the Mel frequencies. We have used 40 filter banks in our implementation.

DCT

Inverse Fourier Transform is done using Discrete Cosine Transform in this step. The result of this step are the Mel frequency cepstral coefficients.

$$c[m] = \sum_{i=1}^{M} logY(i)cos[\frac{m\pi}{M}(i-0.5)]$$

M is the number of filters, $c[m]$ are the cepstral coefficients, $Y(i)$ are the filter outputs.

Normalization

To reduce the channel effect of the recordings, cepstral mean subtraction is done. When people record from different channels, different channel effect is created. Q is the number of cepstral coefficients.

$$c_i = c_i - c_{mean}, \quad 1 \leq i \leq Q.$$

B.3 Derivative coefficients

Co-articulation and irregularities in speech cause the speech signal to not be stationary frame to frame. To acknowledge this delta and delta delta (acceleration) features can be calculated from the cepstral coefficients. We use the linear regression method to calculate the derivative features.

$$\Delta c_j[i] = \frac{\sum_{m=-M}^{M} mc_{j+m}[i]}{\sum_{m=-M}^{M} m_2}$$

Now the complete feature set is:

- Q cepstral coefficients (using LPCC or MFCC)
- Q delta cepstral coefficients
- Q acceleration cepstral coefficients

Appendix C

Vector Quantization

Vector Quantization was originally being used for data compression, but it performs well even for speech recognition. A large set of vectors are divided into groups such that they have almost equal number of neighbors closest to them. VQ vastly decreases the data that has to be processed. Instead of having full utterances and words as training data, we will be able to have only a small sized codebook.

C.1 KMeans

KMeans is one of the most basic and yet useful algorithm or clustering of data. The algorithm is described as:

1. Choose N random points as centroids
2. Assign all other points to one of the N centroids based on distance
3. Recalculate the centroids by using the above created buckets
4. Repeat from step 2 till the centroids converge

C.2 LBG

Linde Buzo Gray algorithm solves one of the very important shortcomings of KMeans. It solves the initialisation problem. KMeans chooses N random points and can reach local optimal states. To overcome this LBG algorithm completely removing initial randomness by using the following steps:

1. Start with number of centroids as 1 (the mean of all vectors)

2. Split the centroids into two

3. Optimize the centroids using KMeans

4. Repeat from step 2 till the desired size of centroids is reached

Appendix D

Word Modeling

Now the feature extraction is done on input signals and then using the codebook, the closest bucket is found for each frame. This greatly reduces the amount of data that we have to process. These bucket values are referred to as the observations. HMM has a number of hidden states. The probability of being in a state at the beginning is given by the initial state distribution. Each state can output one observation with probability given by the output probability matrix. Reaching one state from another has the probability given in transition probability matrix.

Hidden Markov Models defines three basic problem:

1. Evaluation problem

   Given a model and a sequence of observations, find the probability that the observations were generated by this model. This corresponds to recognizing the given input signal. Models for all words will be created and the model that gives the highest probability is the recognized word.

2. Decoding problem

   Given a model and a sequence of observations, find the most optimal state sequence that produced given observations.

3. Learning problem

   Given a model and a sequence of observations, how to adjust the model so it maximizes the probability from the Evaluation Problem. This corresponds to training the models on the given dataset. Each model for a word will be trained on all the utterances of that word in the training data. The initial model taken is a feed forward model. In feed forward model, every state can jump to the states next to itself. The length of jumps can be limited.

All implementations regarding HMM were implemented by referring to [Rab89]. A model is defined by initial state distribution π, transition probabilities A, and output probabilities B.

Given states $S = S_1, S_2 ... S_N$, observations $O = O_1 O_2 ... O_T$, and $\lambda = (A, B, \pi)$, and let the state sequence be $q = q_1 q_2 ... q_T$.

D.1 Forward Procedure

Forward procedure solves the Evaluation problem. It uses a dynamic programming approach. Let $\alpha_t(i)$ be the forward probability variable at instant t state i.

Formula:

$$\alpha_t(i) = P(O_1 ... O_t, q_t = S_i | \lambda)$$

Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1) \quad 1 \leq i \leq N$$

Induction:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(O_{t+1}) \quad 1 \leq t \leq T-1, 1 \leq i \leq N$$

Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

$P(O|\lambda)$ is the probability that the observations were generated by this model.

D.2 Viterbi Algorithm

Viterbi algorithm solves the Decoding problem. It also uses a dynamic programming approach. Let $\delta_t(i)$ be the delta probability variable at instant t state i.

Formula:

$$\delta_t(i) = \max_{q_1,q_2...1_{t-1}} P(q_1, q_2, ...q_{t-1}, q_t = S_i, O_1O_2...O_t|\lambda)$$

Initialization:

$$\delta_1(i) = \pi_i b_i(O_1) \quad 1 \leq i \leq N$$

$$\psi_1(i) = 0$$

Induction:

$$\delta_t(j) = \max_{i \leq i \leq N} [\delta_{t-1}(i)a_{ij}]b - J(O_t) \quad 2 \leq t \leq T$$

$$\psi_t(j) = arg \max_{i \leq i \leq N} [\delta_{t-1}(i)a_{ij}]b - J(O_t) \quad 2 \leq t \leq T$$

Termination:

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)] q_T^* = arg \max_{1 \leq i \leq N} [\delta_T(i)]$$

Backtracking the path:

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \quad T-1 \geq t \geq 1$$

q^* is the optimal state sequence.

D.3 Baum Welch Re-estimation

Baum Welch re-estimation solves the Learning problem.

Backward procedure is a necessity for this algorithm. Backward procedure is similar to the forward procedure but the flow is in backward direction. Let $\beta_t(i)$ be the probability variable at instant t state i.

Formula:

$$\beta_t(i) = P(O_{i+1}, O_{i+2}...O_T, q_t = S_i|\lambda)$$

Initialization:

$$\beta_T(i) = 1 \quad 1 \leq i \leq N$$

Induction:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1})\beta_{t+1}(j) \quad T > t \geq 1, 1 \leq i \leq N$$

Now let us define $\gamma_t(i)$ as the probability of being in state i at instant t.

Formula:

$$\gamma_t(i) = P(q_t = S_t|O, \lambda)\gamma_t(i) = P(O, q_t = S_i|\lambda)\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

Let $\xi_t(i,j)$ as the probability of being in state i at time t and going to state j at time $t+1$.

Formula:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda)$$

Using forward and backward variables, we find:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1}\beta_{t+1}(j))}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

Calculation of new π:

$$\hat{\pi_i} = \text{expected number of times in state } S_i \text{ at time } t = 1$$

$$\hat{\pi_i} = \gamma_1(i)$$

Calculation of new A:

$$\hat{a_{ij}} = \frac{\text{expected number of transition from state } S_i \text{ to } S_j}{\text{expected number of transition from state } S_i}$$

$$\hat{a_{ij}} = \frac{\sum_{t=1} T - 1\xi_t(i,j)}{\sum_{t=1} T - 1\gamma_t(i)}$$

Calculation of new B:

$$\hat{b_j(k)} = \frac{\text{exptected number of times in state } S_j \text{ and observing output } V_k}{\text{expected number of times in state } S_j}$$

$$\hat{b_j(k)} = \frac{\sum_{t=1,O_t=V_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

D.4 Scaling and Tweaking

As the number of observations increases, calculations of δ, α, β head to very small numbers. These numbers cannot be represented even in double float precision. Hence scaling has to be performed.

For Viterbi algorithm, scaling can be avoided if all the calculations are done in log space.

On the contrary, both forward and backward procedures involve summation of large number of terms. Therefore, log space cannot be used. To scale these procedures, scaling coefficients are used [Rab89].

To make up for the lack of infinite precision on the processors, tweaking has to be done of the model at each step of Baum Welch re-estimation. In the transition matrix and the output matrix, the probabilities that turn to zero are given a minimum probability value. This helps the model to learn well during training.

Appendix E

Sentence Modeling

Sentence modeling in our system is estimating probability of each word given its prior context. Example: $P(\text{off}|\text{turn torch})$

E.1 NGram

NGram is a (N-1) order Markov model. The Markov assumption is the presumption that future behavior of a dynamic system depends only on its recent history. NGram posterior probabilities can be found using the relative frequencies of words. This method is called Maximum Likelihood Estimation. Assume word sequence $w_1 ... w_n$ and C is the count of the NGram.

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{w_{n-N+1}^{n-1}}$$

This estimates the NGram probability by dividing the observed frequency of a particular sentence by the observed frequency of the prefix.

E.2 DFA

Deterministic Finite Automata for sentence modeling is created by adding a necessary initial point condition to the NGram generator. In the implementation, only the sentences that start from the initial word are counted. Hence, the formula for MLE remains the same.

Any sentence that do not follow the complete structure of sentences in the training is assigned zero probability. Hence, it can never occur as one of the recognized sentence.