ASR REPORT

100053181

Data Preparation

Speech recordings
To train and test our speech recognizer we recorded from scratch a total of 41 speech sentences that comprised of 18 predefined names. They were recorded on a high quality microphone and then processed in Adobe Premiere Pro to minimize any noise for a clean control test, and subsequently noise can be easily introduced to test its effects on recognition performance. The audio was stored as 16 bit/44.1kHz wave files.

Training recordings
20 training recordings were made by speaking the 18 names in identical order 20 times, with the subject speaking naturally with slight variations of the voice. This is the speech we want to replicate for the highest word accuracy during demonstration.

Testing recordings
An additional 21 testing recordings were made in a similar fashion with variations of the number of names, order of names, articulation speed and tone. In this way we can be sure that our speech recognizer works with certain amount of flexibility.

Live recordings
Our speech for the live demonstration were recorded on the same microphone using Adobe Audition with no post processing. One thing to note is that our recognition system was unable to pick out any vocabulary when we recorded on a different microphone with similar high quality playback (more in the **Results Evaluation** part of the report).

Task grammar and word network
For HTK to understand how the names in each sentence are spoken, a word network needed to be defined using HTK's Standard Lattice Format (SLF) in which each word to word transition is stated. To make things easier, a grammar file is made with the following notation ***([silent] < $name > [silent])*** that represents a random alternate name between optional silences at both start and end of the word. We finally produce the word network by running the grammar file through *HPARSE.*

The Dictionary
The dictionary defines how names in each sentence are pronounced as a sequence of phones, but since our dataset is very manageable, we just specify the training model name for each word entry:

silent silent
michael michael
jack jack
ben ben
…

Label files
To train the HMM for each vocabulary, we created label files for each recording that associated each name entity with the waveform at specified time segment in the recording. Praat was used to annotate

the name entities with ease and then a C# program written by us is used to batch convert Praat's TextGrid format to HTK's Master Label File (MLF) format.

Feature Extraction

Introduction
Our implementation of Mel Frequency Cepstral Coefficients (MFCC) feature extraction is achieved through calling a custom MATLAB function ***featureExtraction(sample, fs, filename, channelsNum)*** and providing four parameters:

- ***sample*** - The speech signal vector to be processed.
- ***fs*** - Sampling frequency of the speech signal.
- ***filename*** - Name of MFCC matrix with feature vectors as columns to output.
- ***channelsNum*** - Number of filterbank channels.

Resampling the audio file
The sample frequency of the original recordings was 44.1 kHz but the test noise files we were given were sampled at 16 kHz therefore resampling was necessary. Apart from the fact that resampling was required in order to add noise to the original recordings we have also discovered that the results we obtain with our speech recogniser are improved when the clean audio files are downsampled from 44.1 kHz to 16 kHz (more in the **Results Evaluation** part of the report)

Splitting the audio file into frames
The feature extraction uses 20ms frames with a 10ms overlap. The functions loops through the whole audio file and extracts a frame every overlap period. There is a possibility of further testing and evaluation with different frame sizes that could lead to potentially interesting results but due to time restrictions we were not able to conduct these tests.

Pre-emphasis
Initially, we set out to design a simple high-pass filter by following the example set in the lecture slides. Our implementation, however, did not result in an expected increase in the accuracy of the speech recogniser. Therefore, we used the built-in MATLAB filter() function ***filter([1,-0.97],1,frame)*** with alpha of 0.97 as specified in the lecture slides.

Hamming window
Using the built-in hamming() function we create a hamming window which in turn is applied to the current frame. The use of a hamming window smooths out the amplitude of the discontinuities at the boundaries of each frame reducing the spectral leakage that occurs when applying a Fourier Transform on a finite record of the signal (in our case a 20ms frame).

Spectral analysis
For speech recognition and feature extraction we are interested in the frequency domain of the signal. To obtain that data we apply a Fast Fourier Transform (using the built-in fft() function) to the current frame. The resulting complex spectrum is then converted into a frequency magnitude spectrum (using the built-in mag() function). Since the values of the magnitude spectrum are mirrored we only use the first half of the array.

Linear filterbank
The next step in feature extraction is obtaining the magnitude coefficients. Ideally, our feature extraction system would use the mel-scale filterbank which imitates the frequency response of a human

ear but due to time limitations we obtain the value of energy in each band through a rectangular, linearly-spaced filterbank. The function takes number of channels as a parameter and by testing different values we found that our speech recogniser produces the best results when the linear filterbank uses 100 channels to generate the coefficients (more in the **Results Evaluation** part of the report). The dynamic range of the coefficients is then reduced by applying a log function to the result.

Estimating the quefrency
In order to isolate the vocal tract information from the excitation information the feature extraction system converts the coefficients from the spectral representation to the cepstral representation. This is achieved through a Discrete Cosine Transform (using the built-in dct() function). The resulting quefrency estimate is truncated in half to only retain the useful vocal tract data.

Result
The result of looping through the audio file is a 2-D matrix containing a feature vector for each analysed frame. The number of feature vectors corresponds to half of the number of channels used in the linear filter bank. The matrix is then written to a file format recognised by HTK.

Acoustic Modelling

Prototype definition
Each HMM is built with a general topology that defines characteristics of our HMM such as the feature vector size, number of states and state transition probabilities. Since we are speaking random names for sentences, we decide our transition probability is 0.5 to advance to the next state and 0.5 to repeat the current one, left to right only. Different transition probabilities were tested with the same left to right structure and they were the same, therefore we decide to stick with a 0.6 advance probability and 0.4 repeat probability, with the exception of the last set, which is 0.9 advance probability and 0.1 repeat probability.

We manually created and tested 3 to 28 left to right states with no skips and chosen prototype 24 (states) with the best sentence accuracy for 50 dimensional feature vectors. We also tested these prototypes with 10 dimensional feature vectors.

Acoustic model training
We call *HINIT* to generate the acoustic model for each vocabulary in all our training recordings. It basically takes in all specified training data and their label files and estimate the initial means and variances perimeters. Next we call *HREST* to further re-estimate the parameters computed by *HINIT*.

Spectral Subtraction

Introduction
The largest impact on the accuracy of our speech recognition system is the presence of background noise, which causes deletion errors as the system is unable to make out actual speech with background noise mixed in. Hence we have to perform noise cancellation on the recordings to reduce any background noise as much as possible.

Spectral subtraction
We choose to use spectral subtraction as it is simple to implement. We call the method in MATLAB ***specsub(audio, fs)*** with the parameters:

- ***audio*** - The signal vector to be processed.
- ***fs*** - Sampling frequency of the signal.

Preprocessing checks
We made sure that we are only processing mono audio by performing a channel size check before running the algorithm. If the input audio is stereo, we simply take the average of both channels.

Add overlap windowing
Before we can perform frequency domain processing, we have to split the time domain signal into half overlapping windows called frames. A Hann window is applied to each of these frames before performing Fast Fourier Transform (FFT) to avoid spectral artifacts. Having multiple frames is also advantageous as it is faster to compute FFT on small amount of samples at once.

Complex spectrum subtraction
Next we apply FFT on the frames to get the magnitude and phase component. We assume that there are no active speech for the first 3 frames so we take their mean value and subtract them off the magnitude for the rest of the frames for the clean magnitude.

Maximum attenuation threshold
Since it is possible for the values of the clean magnitude to be negative, we set a minimum value threshold so that for each magnitude value lower than the threshold, it will be adjusted accordingly to its distance from threshold and not lower than zero. It performs similarly to a spectral floor.

Complex spectrum reconstruction
Finally we can combine the clean magnitude with the phase component to build the clean complex spectrum, and transforming it back to time domain using inverse FFT. We simply add the frames onto a new vector at the half window index to get the clean audio.

Further testing
We suspect that if we trained HMMs for the 3 noise samples (babble, factory and machine gun) in place of speech silences we may be able to see fewer insertion errors**,** however we have had insufficient time to test it out.

Results Evaluation

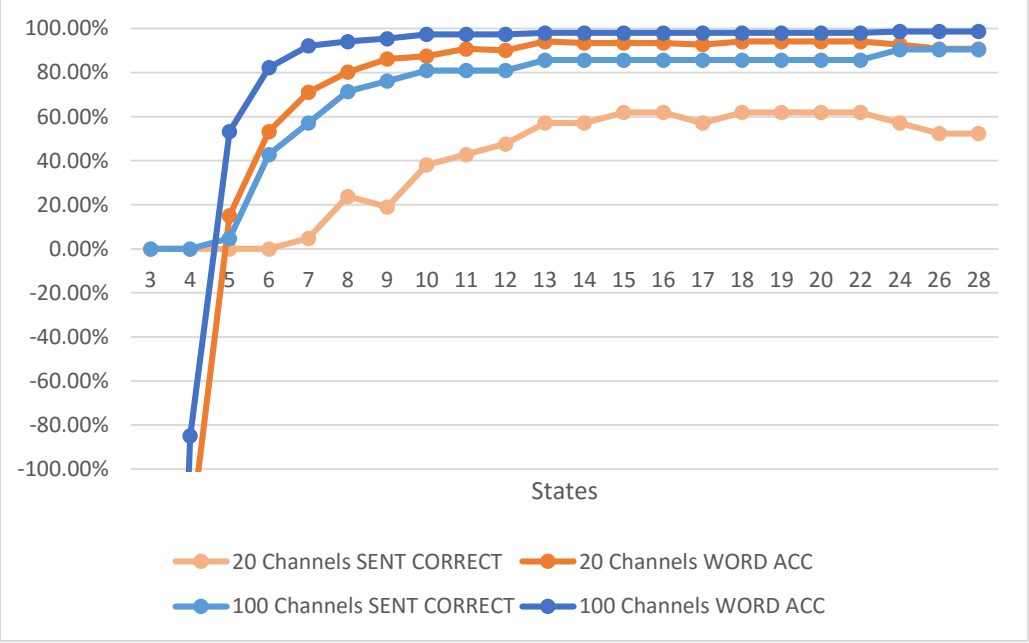
Introduction
Over the course of our project we have conducted close to a hundred tests in order to fine-tune our feature extraction function, pick the best HMM prototype and achieve as high recognition accuracy as possible. In the end our test results helped us built a speech recognition system that achieves 100% accuracy.

Recording with different microphones
We found that recording test speech data on a different microphone than the one used to train the system heavily impacted the recognition performance, despite both recordings being audibly clear and free from background noise. The reference microphone which we used for all our tests was a professional Rode directional microphone and the other test microphone was a cheap omnidirectional condenser microphone. The test were conducted in two types of environments – small quiet bedrooms and the busy Lewin Laboratory. In both cases, using the reference microphone to record the test data yielded the best results. The word accuracy never dropped below 100%, even in the recordings from the Lewin Laboratory where there was background chatter noise present. The test data recorded with the

other microphone, independently of the environment noise, yielded poor results (0-50%). There are two important factors at play that resulted in such drastic differences in accuracy. The reference microphone, because of its directional nature, tends to pick up mostly the speaker's voice and minimizes the amount of background noise and lessens the impact of room reverberation on the recording. This is not the case with the recordings from the omnidirectional test microphone. The reverberation from hard walls in the bedrooms and overall higher level of environmental sound pollution contributed to the lower recognition accuracy. Another important factor could be the differences in frequency response between the two microphones. The Rode microphone has better clarity and response in the lower frequencies and this is the band we are primarily analysing in our feature extraction system. This means that training an ASR in our case was not only speaker but also microphone dependent.

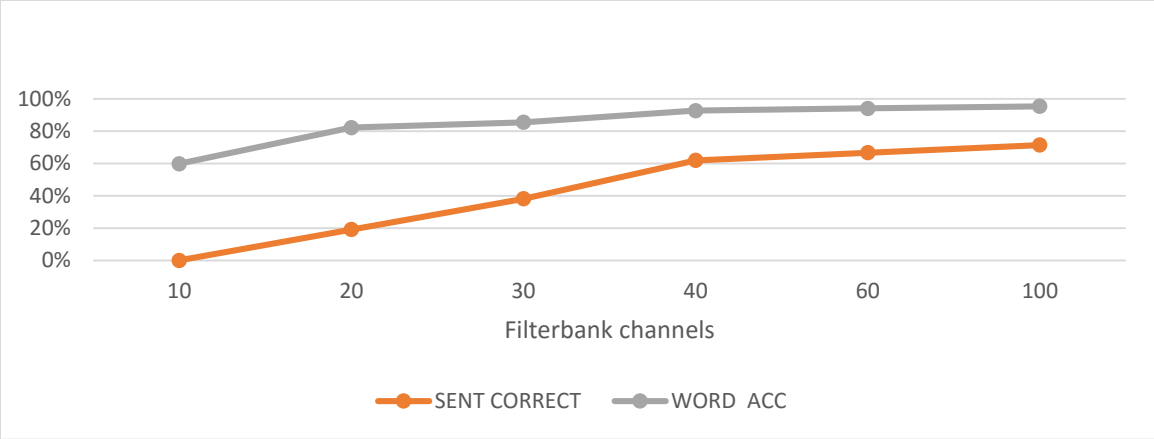
Prototype states

According to Gunter and Bunke (2003) the number of states for the HMM prototype must be chosen through empirical testing. In order to determine the optimal number of states for our ASR we conducted tests for 22 different numbers of states and for two different values for the number of filterbank channels. We found that the accuracy of our final 100 channel model peaks at 24 HMM states (WORD ACC=98.68%, SENT CORRECT = 90.48%) whereas the initial results obtained through the initial 20 channel model peak at 18 states. Adding more states after these points seems to either not improve the results or even, in the case of the 20 channel model, make them worse.



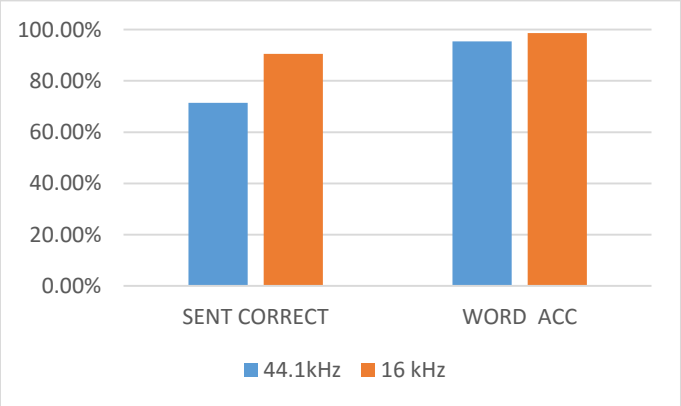
Filterbank channels

Human ears can detect differences between signals of lower frequencies better than the change of pitch in the high frequency spectrum (Mel Frequency Cepstral Coefficient (MFCC) tutorial), and, additionally, the lower frequencies contain the vocal tract data which we use for speech recognition. Therefore we needed more resolution in the lower frequency spectrum. Increasing the number of channels provided more granular resolution and helped in detecting the changes in the lower frequencies which led to more accurate results. Since our filter was linearly spaces as opposed to the Mel scale filterbank, that approach introduced overhead in the higher frequencies but we did not notice any significant performance impact. Based on the test results we chose a 100 channel filterbank for all our subsequent testing.



Downsampling audio files

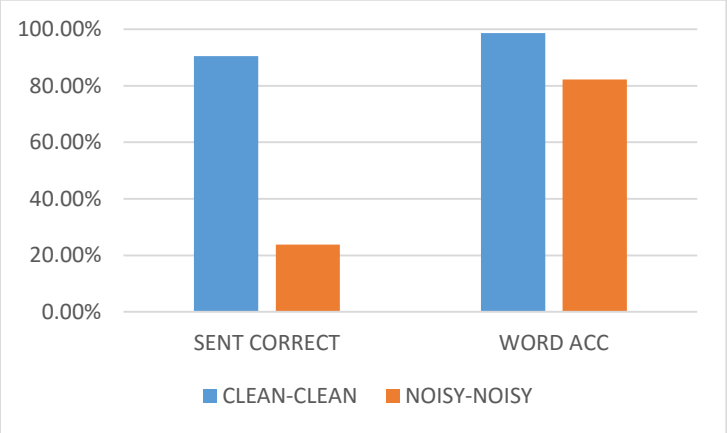
Our original audio test and training files were recorded in 44.1 kHz but then had to be downsampled so that they could be mixed with the 16 kHz noise files. To our surprise we found that the accuracy of the results had increased with the biggest change observed in SENTENCE CORRECT (an increase of 19%) which at first seemed counterintuitive because we were effectively decreasing the resolution and quality of our data. Some conclusions can be drawn as to why this is the case. The files were downsampled using built-in MATLAB function resample(). This function applies an antialiasing FIR lowpass filter to the output which could affect the results. Another interesting finding is that, although HTK takes in files of various sampling frequencies, 'The HTK Book' (Young et al., 2006) always refers to the 16 kHz as the default sampling frequency. Additionally, the HTK Audio format specification also lists 16 kHz as the default frequency for the generated files (README-HTK-AUDIO). There is a possibility that HTK was developed with that frequency in mind and thus the results of training and testing on a 16 kHz speech recordings would be the most accurate.



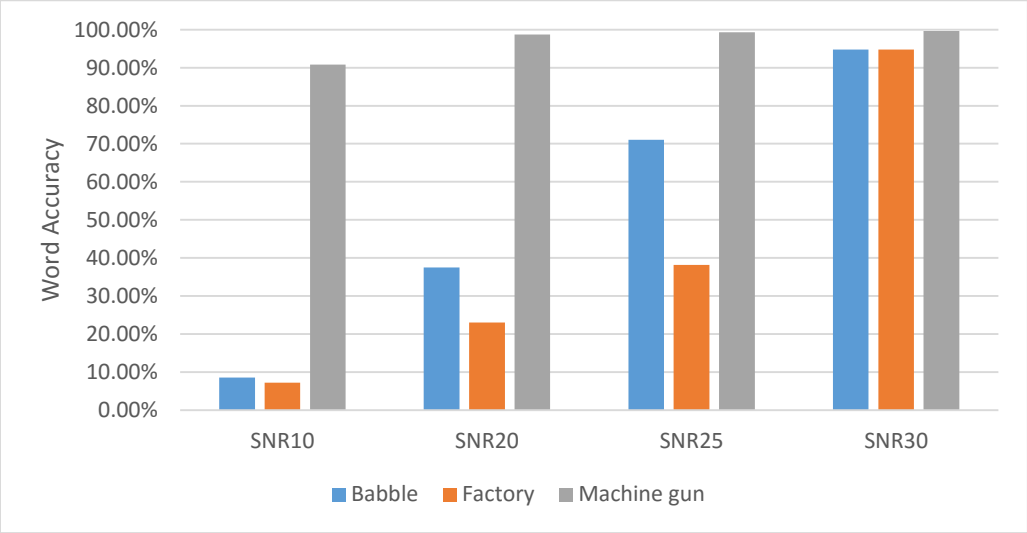
Adding noise

Introducing artificial noise to both the training and testing audio reduceed the speech recognition performance, especially in detecting the whole sequences of words but the word accuracy was still fairly decent at 82%. The graph shows the difference in speech recognition accuracy between testing clean speech data with a model that was also trained on clean data, and testing noisy data (babble noise at

10SNR) on a noisy model with the same characteristics. No spectral subtraction was performed at this point.



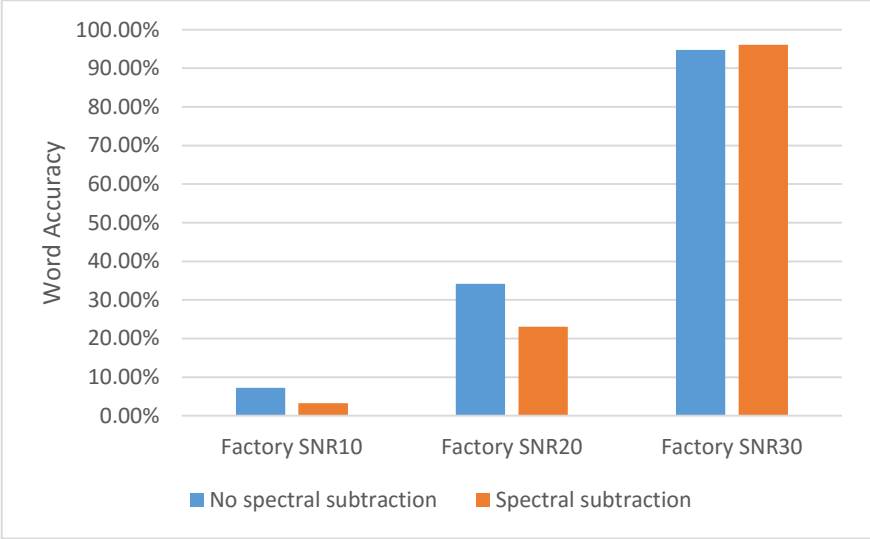
The next step was mixing the training and test data to determine how accurate the speech recogniser is when tested in a simulated noisy environment. For the purposes of this test we applied three types of noise (babble, factory and machine gun) at different Signal-to-Noise ratios. For non-stationary noises like babble and factory noise the speech recognition performance decreases drastically to around 8% with SNR10. We believe that the very good recognition we achieved with the machine gun noise compared to the other types of noise is a result of several factors. The noise is fairly stationary and consistent throughout which makes it predictable and easier to pick up. Additionally, the frequency content of the machine gun noise does not overlap with the human voice frequencies to the same extent as it does with babble and factory, therefore the most important frequency spectrum of the speech files is not heavily polluted. Predictably, the accuracy of the speech recogniser increases as the noise decreases. There is barely any audible noise at SNR30 and the results are comparable to tests on clean speech data.



Spectral subtraction

Spectral subtraction increased the recognition performance when applied to both testing and training data provided they both are either clean or noisy. In the case of clean training data and clean test data

with spectral subtraction applied to both we achieved 100% word and sentence accuracy. However, when applied to a combination of clean training data and noisy test data we get mixed results. Spectral subtraction drastically decreases the performance when applied to low SNR files and slightly improves the results with high SNR files. The higher the SNR the more likely the spectral subtraction will actually improve the performance.



Another factor is the type of noise we apply the spectral subtraction to. Because we know that spectral subtraction works best on stationary noise (Noise reduction, 2006) when applied to babble noise it introduces additional musical noise and has little effect on reducing the original noise.

We also found that for speech recognition buzzing noise is worse than musical noise. This was tested by raising the maximum attenuation threshold from 0.00 to 0.02. The recognition accuracy dropped by 10% for the noisy test data on a noisy model (babble noise 10SNR).

References

Gunter, S., & Bunke, H. (2003). Optimizing the number of states, training iterations and Gaussians in an HMM-based handwritten word recognizer. *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* Retrieved November 9, 2015, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.6366&rep=rep1&type=pdf

Noise reduction. (2006). Retrieved November 10, 2015, from http://sound.eti.pg.gda.pl/denoise/noise.html

Mel Frequency Cepstral Coefficient (MFCC) tutorial. (n.d.). Retrieved November 9, 2015, from http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

README-HTK-AUDIO. (n.d.). Retrieved November 9, 2015, from http://mi.eng.cam.ac.uk/projects/sacti/corpora/SACTI-1/utterance-audio/README-HTKAUDIO.TXT

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., . . . Woodland, P. (2006). *The HTK book* (Version 3.4 ed.). Cambridge: Entropic Cambridge Research Laboratory.