B140607

Wordcount - 2877

Table of Contents

1. Introduction

In this report, we focus on creating our own digit recogniser using the HTK toolkit. "HTK is a portable software to solving automatic speech recognition (ASR) task based on Hidden Markov Model (HMM) method." (Woodland et al, 1995). This first part of this report provides details essential giving the theory as to what steps are involved in building a digit recogniser. The latter part of the report provides details of the experiments carried out to test the performance of the model by measuring the Word Error Rate (WER). It involves coming up with a hypothesis for the experiments, creating the training and test data from data provided by individuals as well as data created by myself and providing explanations of the results. Discussion and concluding remarks are given to round up the report.

2. Theory

2.1 Data collection and acoustic features

Supervised machine learning requires labelled data. Thus, we collect speech data which is used to prepare for feature engineering. In the speaker-dependent system stage for digit recognition, we recorded our speech of 10 digits starting from 0 to 9 and separated them into a training and testing set. This was done by recording my speech, to be more specific recording 7 repetitions of each digit which would be the training set and a further 3 repetitions of each digit which would be the test set. This was done using Praat software and the training set was labelled using wavesurfer software. The labelled data was used to extract acoustic features from frames of speech, transforming the waveform into an acoustic feature vector. Mel Frequency Cepstral Coefficients (MFCC) is used to represent the acoustic features and create MFCC feature vectors from the training and testing waveform files. This is done by using the HTK command HCopy. The first step of MFCC extraction involves uplifting amount of energy that exists in high frequencies which improves phone detection accuracy as it makes the information from the high formants available to the model, which is done by a filter (Jurafsky & Martin 2014, Chapter 9.3.1). The next step is windowing. This involves breaking up the audio signal (waveform) into different segments to detect phones in the speech. Hamming window is used for MFCC extraction. The next step involves applying the Discrete Fourier Transform (DFT) so that the signal from the time domain can be converted to the frequency domain as it is easier to analyse in the frequency domain and obtain spectral information for the signal. The next step involves using the mel scale to map the actual frequency to the frequency that is perceivable by humans as humans tend to perceive sound differently from machines thus improving the performance of the model. The log function is applied to the output of the mel-filter. The next step involves applying the inverse discrete fourier transform and the result of this computation is known as the cepstrum. This process is done so that the source and filter can be separated as we only require the filter as it contains important details about phones which the fundamental frequency doesn’t give so it’s discarded. This results in 12 cepstral coefficients for each frame, one more is added which is the energy of the signal sample thus 13 and 39 in total. MFCC files are generated for our waveform files.

2.2 Training HMMs

Training the HMM model involves using commands from the HTK tool. The HMM model

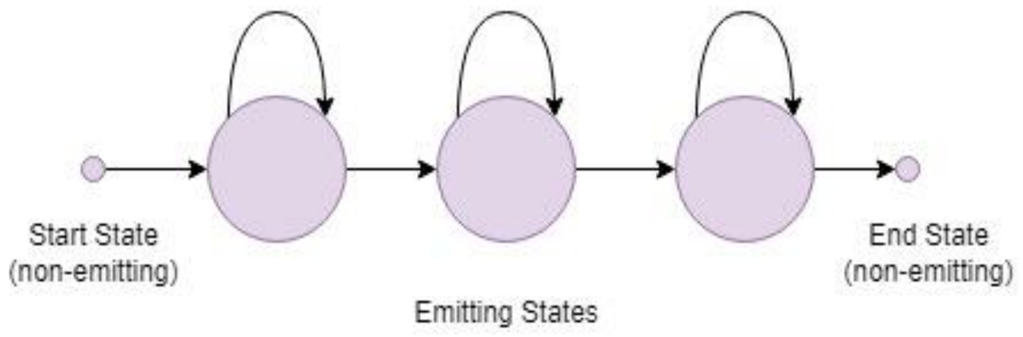


Figure 1: shows a 5-state HMM model with emitting and non-emitting states being labelled.

consists of states which are emitting and non-emitting (see figure 1), in each state there is a Gaussian probability density function whose job is emit observations. The model is trained so that it can learn from the training data and make good predictions.

State	1 (start state)	2	3	4	5 (end state)
1 (start state)	0.0	1.0	0.0	0.0	0.0
2	0.0	0.5	0.5	0.0	0.0
3	0.0	0.0	0.5	0.5	0.0
4	0.0	0.0	0.0	0.5	0.5
5 (end state)	0.0	0.0	0.0	0.0	0.0

Table 1: Initial transition probability between states

The HMM model is initialised with a prototype model as the training algorithms require a model to start with. The Gaussian Mixture Model (GMM) is initialised with a mean of 0 and variance of 1. The size of the MFCC vectors is 39 due to the 39 features. The probability of transitioning to another state in the model is also initialised with a transition matrix (see table 1). The HTK command HInit is used to provide the initial estimate of a single HMM using a set of observation sequences. This is done by simply aligning the observations and states done by uniform segmentation (done once as it gives same values for the parameters) (see figure 2) and then performing Viterbi training where Viterbi alignment is used to segment the training observations and then recomputing the parameters by pooling the vectors in each segment.

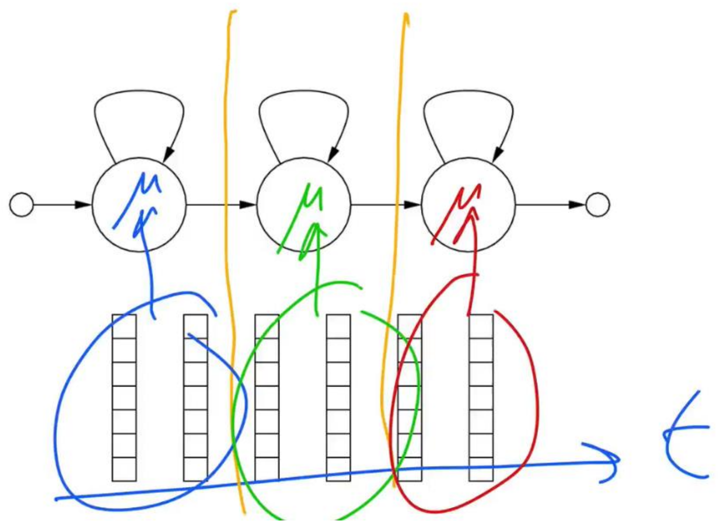


Figure 2: depicts an example of uniform segmentation where MFCC vectors are divided up into three equal bits, calculate the means and variances for each of those bits, and then assign those as the parameters for each state.

The Viterbi algorithm is a dynamic programming algorithm that gives the highest probability for the most likely sequence of the hidden states for the observation sequence. It is used to find an alignment between states and observations and use it to update the models' parameters (mean and variance) and we re-run the algorithm again iteratively to get a different alignment till the model stops improving and settles down to stable parameters so when the probability of observations stops improving the algorithm is terminated. The next step involves using the HTK command HRest to perform basic Baum-Welch re-estimation of the parameters of a single HMM using a set of observation sequences. This algorithm is a special case of Estimation Maximum (EM) algorithm which finds and tunes the unknown parameters of the HMM.

2.3 Language modelling

The language model is a generative finite-state model that, given a sentence, emits a word sequence by giving its probability. There are no HMMs, Gaussians or MFCCs involved in the language model. The main role of a language model role is to compute the prior probability of the word sequence, so if W is a word sequence, then it would compute P(W). This probability is required as the HMM computes for observation sequence O, P(O|W) which is the probability of any observation given a particular model (whole words). We don't care about P(O) as we don't care about how likely some MFCCs are, we are interested in how likely each word is given these MFCCs. Thus, we want to compute argmax P(W|O), which using and rearranging the bayes theorem gives argmax P(O|W) * P(W) which is the probability of words given the observations. The most common form of language model is an N-gram. This is a finite state model that emits a word sequence. The probability of a given word sequence is the product of the probability of each word in the sequence. Those word probabilities are computed given only the N-1 preceding words. The main reason as to why the n-gram model is used is because it can be written as a finite state network as we can only compile the language model and acoustic models (HMM) if they are finite state. For this digit recogniser assignment, we use another form of finite state language model which is the hand-crafted grammar assuming that there is an equal (uniform) probability of each digit. For the sequences part of the assignment, we also assume all sequences have equal probability. This is model is much simpler than an n-gram model as its non-probabilistic and is converted by the HTK command HParse into the equivalent finite state graph. For recognition of connected digits, we would need to connect the end state of our model to the start state by drawing a line from the end state to the start state in the graph to ensure that the model can recognise the digits iteratively (see figure 3).

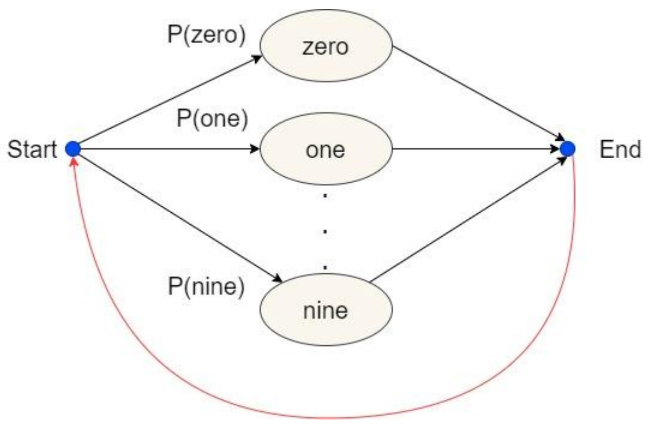


Figure 3: shows the language model of recognising digit sequences.

2.4 Recognition using HMMs

To perform recognition using HMMs we collect test data which is used to test the system on and is different from the data used to perform training and estimate the model's parameters on which hasn't been seen before. For the digit recognition task, we are testing the generalisation of models to unseen data. We label each test sample with the correct transcription to count how often the recogniser was right or wrong, but the labels are not used during the recognition process as it would be cheating but rather used for the evaluation at the end for computing results. The trained HMM model is then compiled with the language model into a single network (see figure 4).

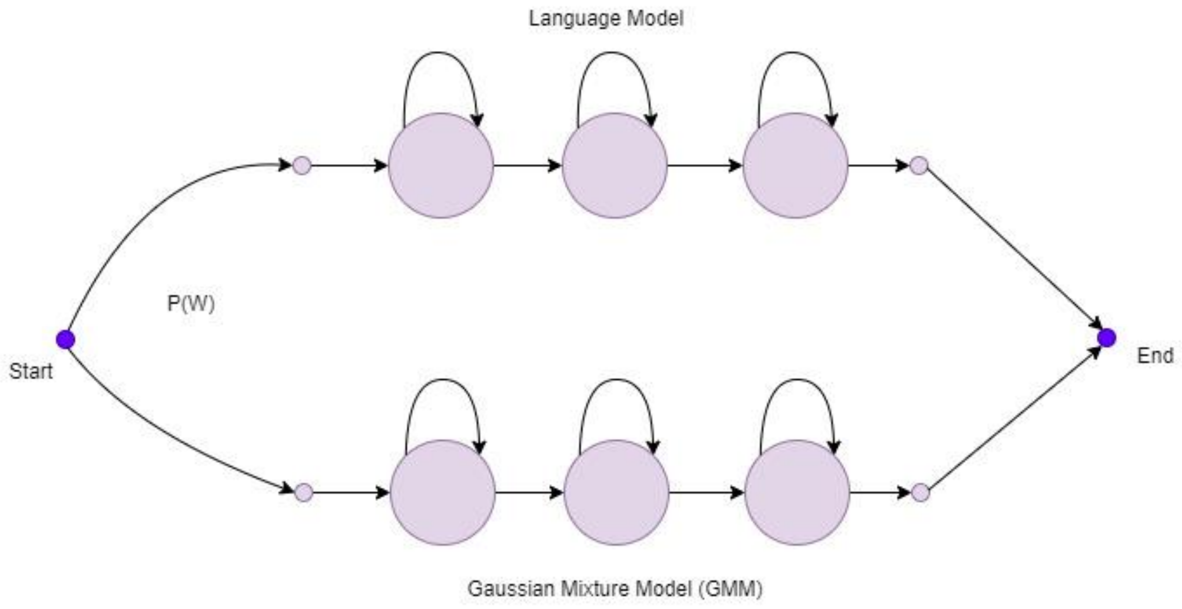


Figure 4: shows the single network consisting of the HMM model finite state machine being compiled with the language model finite state machine.

This recognition network is also an HMM, just with a more complicated topology than the individual word HMMs. The Viterbi algorithm is used to find the most likely path through it that generated the given observation sequence. The Viterbi algorithm can be implemented using either a lattice or token passing. A lattice is a data structure used to perform the computation of finding a single most likely state sequence (see figure 5). For this assignment, we use token passing as it uses a much smaller data structure: the HMM (a finite state model) itself, which is one "column" (all model states at a particular time) of the lattice. So, token passing is equivalent to working with the lattice whilst only ever needing one column of it in memory at any given time. Token passing is a time-synchronous algorithm meaning all paths (tokens) are extended forwards in time at once. The HTK command HVite does token passing and performs recognition on the MFCC test files and generate recognition label .rec files. The HResults command is used to measure the accuracy of the digit recogniser by comparing the ground truth text for each file with the predicted text in the .rec files.

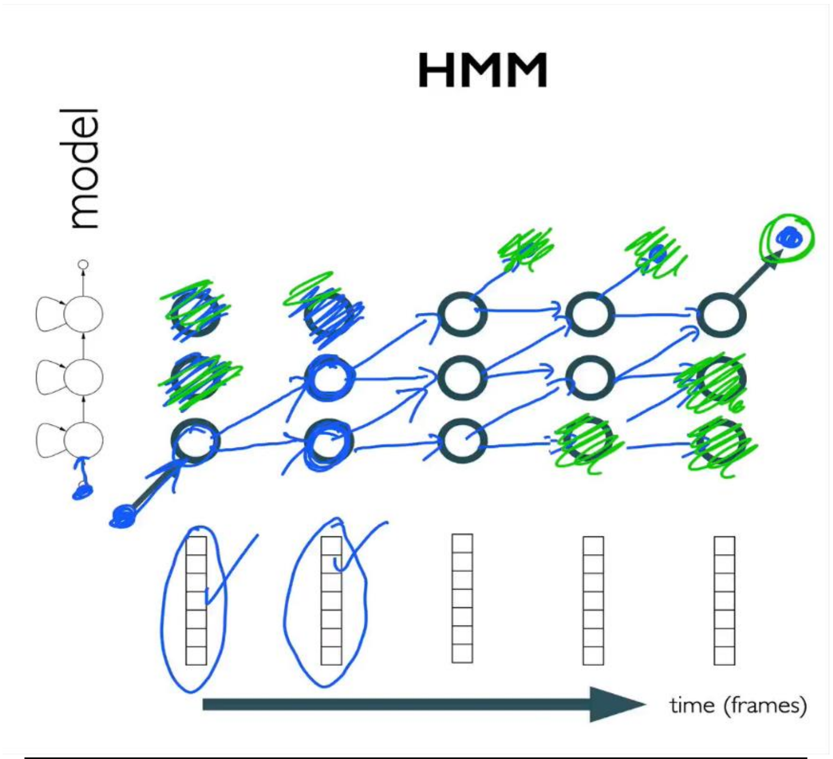


Figure 5: shows representation of the lattice, representing the correct available paths to take and arrive at the final non-emitting state (circled in green).

3. Experiments

3.1 Impact of gender on Automated Speech Recognition (ASR)

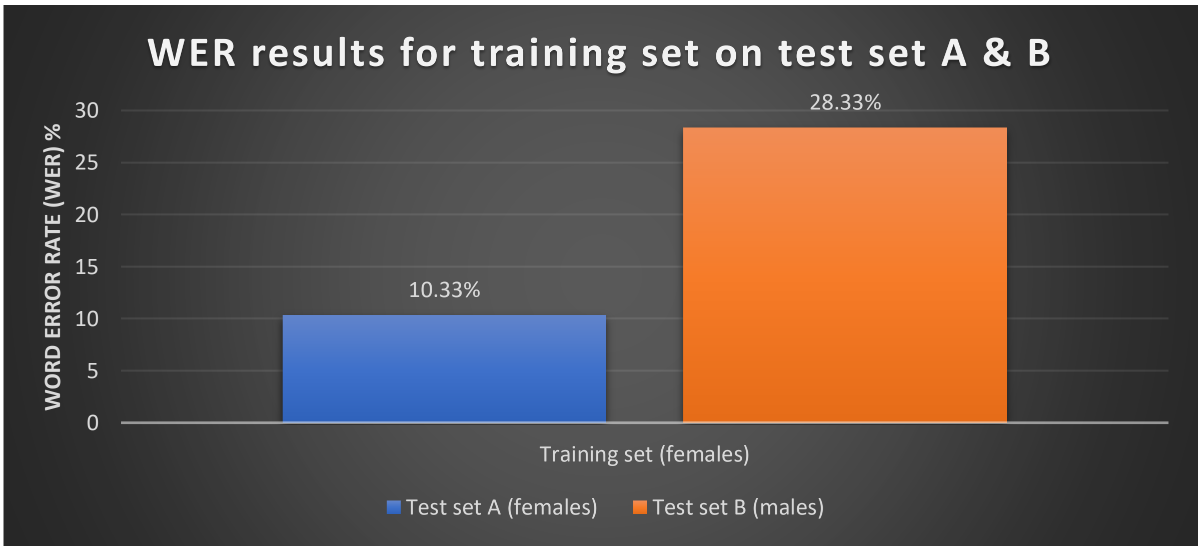
In this experiment, I intend to investigate the effect of gender with control over accent and microphone has on the model's performance.

Hypothesis: I hypothesised that the models trained and tested on data by the same gender will have better performance meaning lower Word Error Rate (WER) compared to testing on data by a different gender.

The experiment is set up as the following:

- Training set: the training data of 14 females UK English speakers using headset microphones.
- Test set A: the test data of 10 females UK English speakers **not** in the training set, also using headset microphones.
- Test set B: the test data of 10 males UK English speakers using headset microphones.

From Graph 1, we see that the performance of the model is better on test set A as the Word Error Rate (WER) is 10.33% which is lower compared to the WER on test set B which is 28.33% and higher in comparison, thus proving that the hypothesis given is correct. A reason to explain this result would be the fact that the difference in intonation between males and females as "males will usually speak in full pitch, with little rise and fall. Females are much more likely to vary their pitch when they talk. Male voices tend to be low pitched, while female voices are usually high pitched." (Saunders, 2019). Another reason could be the fact that the training sample size is too small which may mean that it doesn't cover the complete range of speech by females suggesting that this might be something to be explored further.



Graph 1: shows the Word Error Rate (WER) of test set on different genders with control over other factors like accents and microphone types.

3.2 Impact of different headphones on ASR

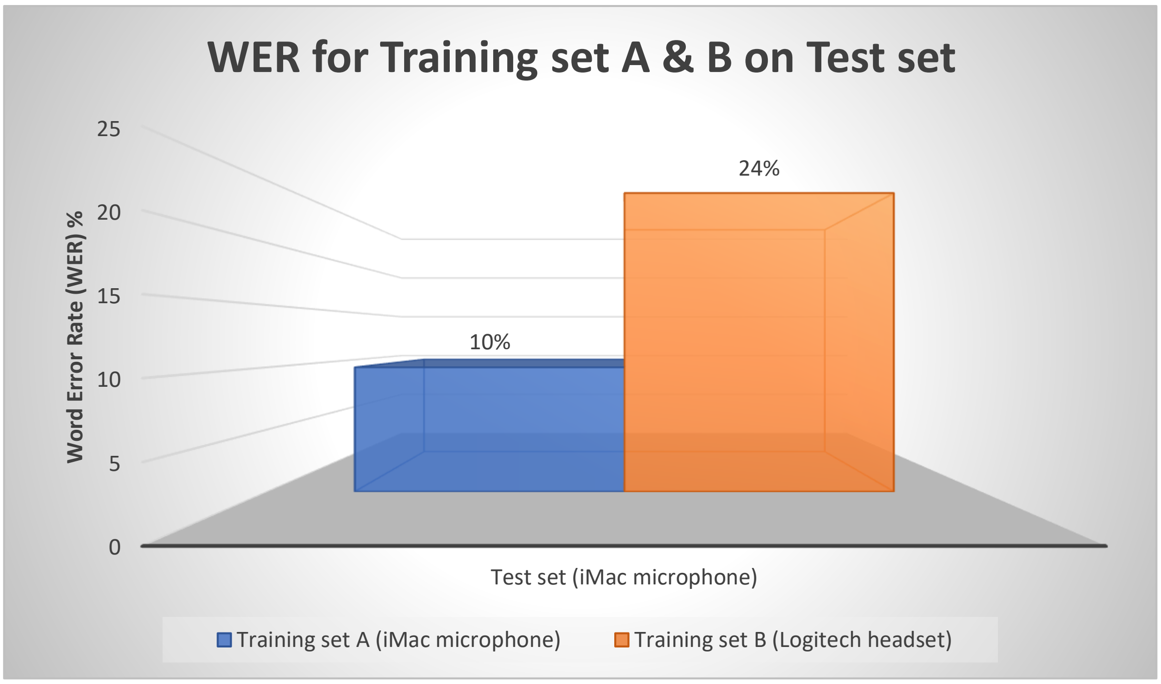
In this experiment, I intend to investigate the effect of varying headphones with control over gender and accents has on the model's performance.

Hypothesis: I hypothesised that models trained on Non-Native (NN) male speakers using microphones from Logitech headset will give a higher Word Error Rate (WER) when tested on Non-Native (NN) male speakers using microphones from iMacs.

The experiment is setup as the following:

- Training set A: the training data of 13 males Non-Native (NN) speakers using microphones from iMacs.
- Training set B: the training data of 13 males Non-Native (NN) speakers using microphones from Logitech headset.
- Test set: the test data of 10 males Non-Native (NN) speakers **not** in the training set, also using microphones from iMacs.

From Graph 2, we see that the performance of the model is better on training set A as the Word Error Rate (WER) is 10% which is lower compared to the WER on training set B which is 24% and higher in comparison, thus proving that the hypothesis given is correct. A reason to explain this result could be the different interactions between microphones and computer hardware which might mean that one microphone may not work that well when paired with some systems compared to others (Wahrenberger). Another reason could be the different techniques used from users when recording that includes microphone positioning, dictation technique, distance away from microphone when recording and varying levels of external noise contamination (Wahrenberger). This might be something which can be investigated in more detail to reduce likelihood of these reasons impacting the model's performance.



Graph 2: shows the Word Error Rate (WER) of using different microphones (iMac & Logitech headset) with control over other factors like gender and accent.

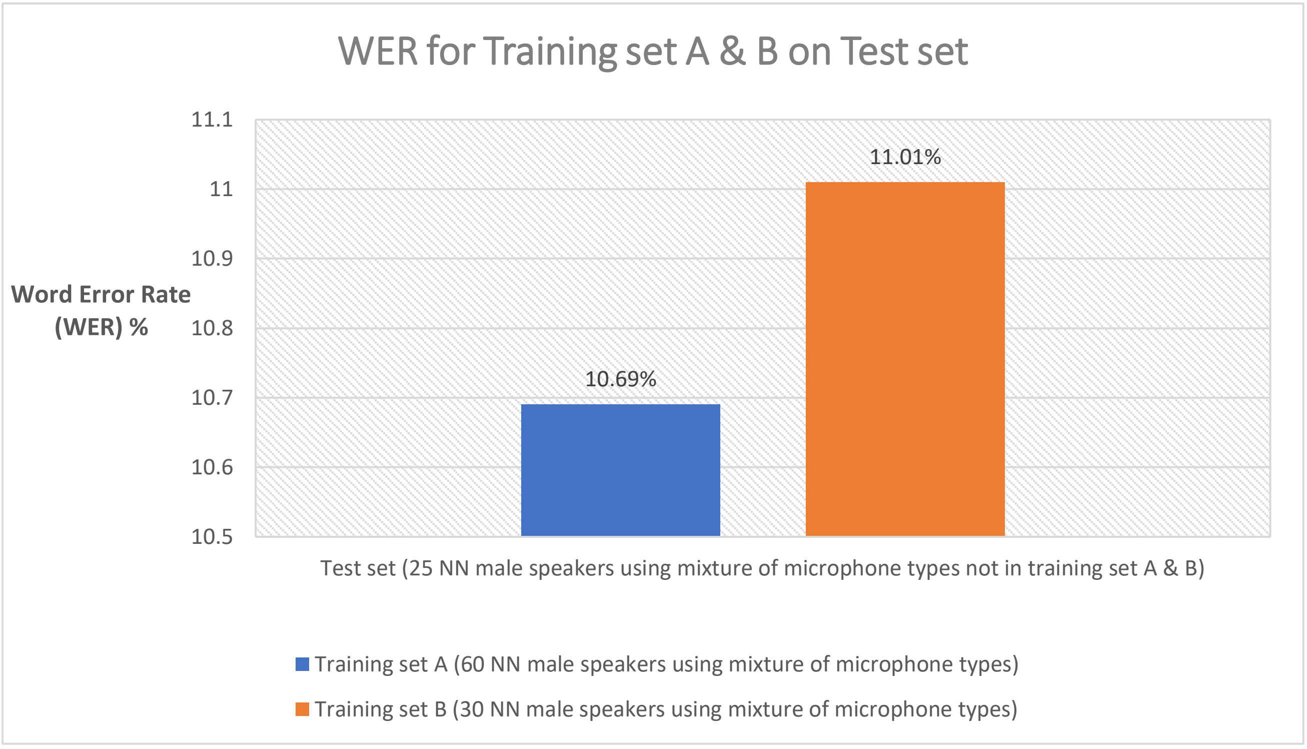
3.3 Impact of varying amount of training data on ASR

In this experiment, I intend to investigate how varying training data affects the performance of the model.

Hypothesis: I hypothesised that using a higher quantity of training data would improve the performance of the model meaning that the WER would be lower compared to using less quantity of training data.

The experiment is setup as the following:

- Training set A: the training data of 60 males Non-Native (NN) speakers using a mixture of microphone types.
- Training set B: the training data of 30 males Non-Native (NN) speakers using a mixture of microphone types.
- Test set: the test data of 25 males Non-Native (NN) speakers using a mixture of microphone types **not** in training set A & B.



Graph 3: shows the Word Error Rate (WER) of using different quantity of training data with control over other variables like gender, microphone type and accent.

From Graph 3, we see that the performance of the model is better on training set A as the Word Error Rate (WER) is 10.69% which is slightly lower compared to the WER on training set B which is 11.01% and slightly higher in comparison, thus proving that the hypothesis given is correct. A reason to explain this result could be that as the quantity of the training data is higher in training set A which means that the model can learn on the different types (mixture of microphone types) and perform better compared to using a smaller amount of training data to train models. Thus, this is a very important factor that needs to be balanced as "using too little training data results in a poor approximation. An over-constrained model will underfit the small training dataset, whereas an under-constrained model, in turn, will likely overfit the training data, both resulting in poor performance. Too little test data will result in an optimistic and high variance estimation of model performance" (Brownlee, 2019).

4. Discussion and overall conclusion

To conclude, to create a digit recogniser, it involves completing the following steps:

- Collecting data and extracting MFCC's from it.

- Training and initialising the HMM model using uniform segmentation and Viterbi training and Baum-Welch algorithm so that the parameters can be learned and tuned.

- Creating a language model for modelling the output.

- Performing recognition on the models using the Viterbi algorithm so it can recognise digits correctly.

There are some factors which need to be taken care of as they may have an impact on the performance of the ASR system such as the gender and amount of training data used to train on. We should also ensure that the quality of training data and test data used to measure the performance of the system is high meaning it shouldn't have any noise which may affect the results as well as also ensuring that issues like overfitting and underfitting don't arise.

5. References

Dan Jurafsky and James H. Martin “Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition”, 2009, Pearson Prentice Hall, Upper Saddle River, N.J., Second edition, ISBN 0135041961.

Young, Steve, Evermann, Gunnar, Kershaw, Dan, Moore, Gareth, Odell, Julian, Ollason, Dave, Valtchev, Valtcho, and Woodland, Phil. The htk book, 2006. URL htkbook.pdf (cam.ac.uk)

Saunders, Gemma. “The Vocal Differences between the Genders - Openmic.” *Open Mic UK*, 25 Nov. 2019, URl https://www.openmicuk.co.uk/advice/vocal-differences-between-genders/

Wahrenberger, Jon W. “Is There a ‘Best’ Microphone for Speech Recognition Software?” *Evaluating Microphones for Speech Recognition*, URl https://speechrecsolutions.com/microphone_accuracy.htm

Brownlee, Jason. “Impact of Dataset Size on Deep Learning Model Skill and Performance Estimates.” *MachineLearningMastery.com*, 25 Aug. 2020, URl https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/