BUILDING AND ANALYZING AUTOMATIC SPEECH MODELS FOR SINGLE DIGIT RECOGNITION

Contents

1. Introduction

We aim to build Automatic Speech Recognition (ASR) systems which are capable of mapping an acoustic signal to a string of words. As part of this work, I will build a digit recognizer with the help of HTK toolkit, and use this model to test the system's robustness, adaptability and efficiency.

Typically, we expect ASR systems to be fast and responsive for most applications. Hence, it is crucial that every step implemented in its pipeline adds some significant value with respect to the computational resources used. I will begin by testing these steps, to find out their importance, if they must be used and if so what their best parameter settings are. This will allow me to settle on a model which is relatively efficient. I will use this model in my adaptability and robustness experiments.

Data plays a very key role in ASR. Various factors like the variation of noise in our input signals, to the accent of speakers due to their regional dialect, gender or other speaker class characteristics, must affect ASR systems greatly. Therefore, I will investigate the presence of these factors and how they affect the model's performance. This will ensure a better understanding of quality of data, and its collection. This is important as labelled data collection is very expensive.

These experiments will also allow me to develop better insights in the model's architecture and speculate potential improvements that could be made. My model is a set of generative models in form of Hidden Markov model (HMM) states and Gaussian probability density functions (PDF) competing to produce the source signal with the highest probability. This source signal is parameterized to compensate for the model's various limitations. Figure 1 shows this generative view of the model. (**holmes**).

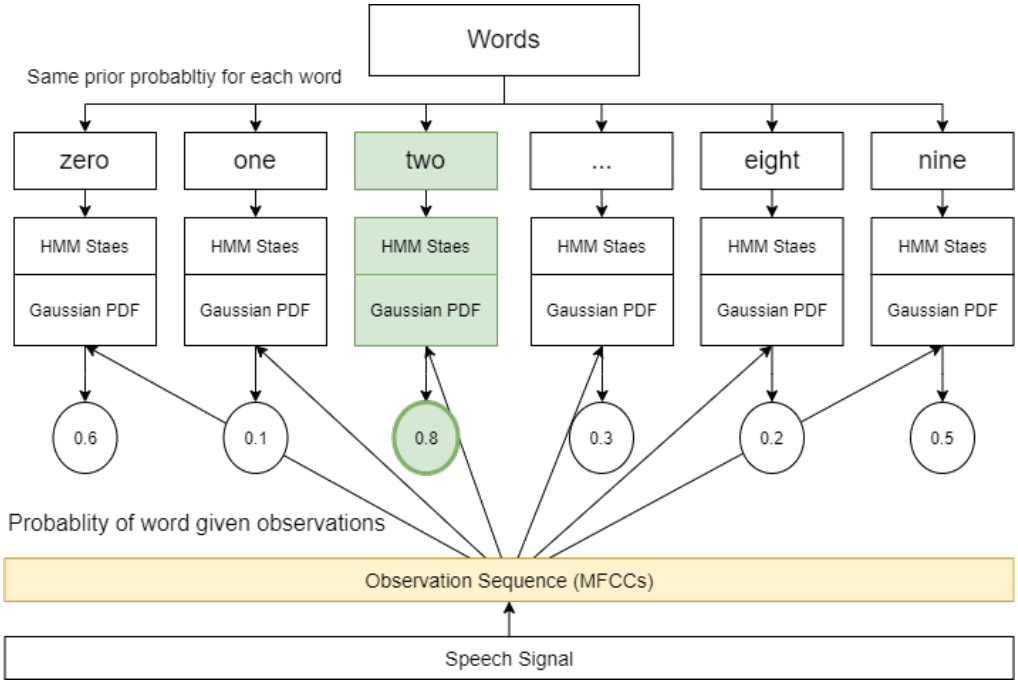

FIGURE 1. Generative view of an isolated digit recognizer. Word two is selected to represent the observed sequence as it can produced with highest probablity.

2. THEORY

2.1. **Data collection and acoustic features.** I used the data collected by students from previous years. They used a recording device to record their train and test files which they then manually labelled in wavesurfer. The labels were the digit names for the speech regions and junk for silence. They aimed to not cut off or omit the start/end of the word while labelling it. Next using HTK, waveforms are parameterized into sequence of feature vectors.

The spectrum of a voiced speech has a downward trend, where the lower frequencies have a higher value, also called spectral tilt. Hence, to obtain a flat spectral signal we apply a **pre-emphasis** to boost the higher frequencies. By default in HTK we set a value of 0.97 for pre-emphasis. (Young, 2006)

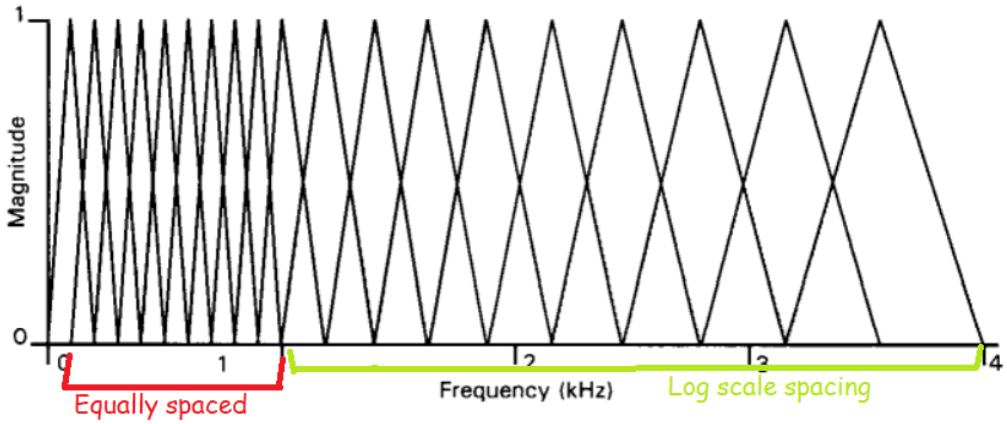

FIGURE 2. Triangular filters transforming output of FFT. Red highlight indicates linear spacing upto 1kHz

Next we extract a sequence of frames by using a window function. We apply the hamming window to reduce the discontinuities at the window edges. Not doing so might by introduce unwanted high-frequency components into the spectrum. (Jurafsky and Martin, 2009)

Next we want to extract the energy level at different frequencies, so we perform filterbank analysis. Using HTK we apply a fast Fourier transform (FFT) to extract them. Human hearing becomes less sensitive as the frequency increases, hence, we try to model this by transforming our FFT output onto the Mel scale as seen in Figure 2. These are then sampled into features by the weighted sum of the corresponding filter bank and the magnitude of the spectrum. (Jurafsky and Martin, 2009)

As we'll be using Gaussian's in our model, we need our features to be not highly correlated. Hence, we also need to perform ceptstral analysis on the output of our highly correlated filterbank coefficients. We obtain Mel frequency Cepstral coeficients (MFCCs) by applying discrete cosine transform and selecting the first N channels. The idea is to view the output from filterbank analysis as a source-filter model, where

the important speech information lies in the filter part of the model. In HTK we select the first 12 coefficients along with its energy. (**holmes**)

Lastly, because HMMs are time homogeneous but speech is not, we compensate our model by adding extra features using HTK called deltas and acceleration. These values capture the change in cepstral energy over time frames. By default we add the deltas and accelerations, resulting in 39 features. (Young, 2006)

2.2. **Training HMMs.** We use Hidden Markov model (HMM) to represent a word as a sequence of states. Each state has a transition probability from itself to another state. Each state represents a Gaussian mixture model. The Gaussian mixture model helps us best capture the multimodal nature of speech, due to factors like varying regional accents. Using the HMM, we can calculate the probability of generating the observed data. The model which has the highest probability of generating the observed data is said to represent the correct word as seen in Figure 1, and the goal of training is to ensure that our HMMs best represent the word. Jurafsky and Martin, 2009

Before we can apply any sophisticated algorithms, we must initialize the parameters. We start with Uniform segmentation. We align each observation with exactly one state by dividing the observations uniformly between each state. Figure 3 demonstrates with an example. The parameters of the Gaussian for a given state are computed by finding the mean and variance of the observations it was aligned to.

Clearly, the above solution can be improved. However, there is no direct way to have an analytical solution that estimates the parameters of our model. So we resort to iterative algorithms that aim to increase the likelihood of our training data.

After we have initialized the parameters by Uniform segmentation we apply Viterbi training, as seen in Figure 4. Using the observations and the model we perform a

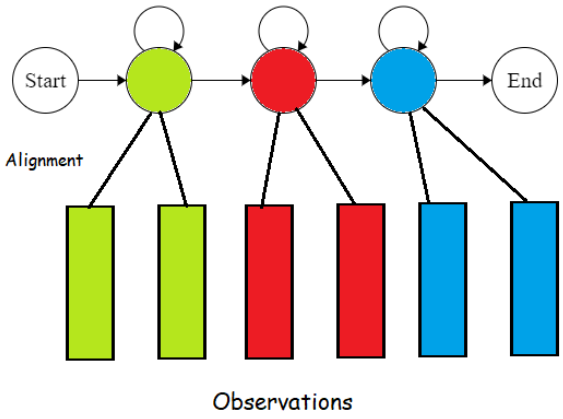

FIGURE 3. A HMM with 3 emitting states, aligned using Uniform segmentation. Colors represent observations used in calculating mean and variance for the respective colored state's Gaussian mixture model.

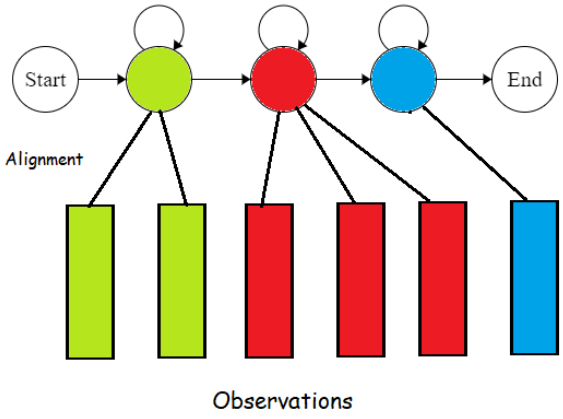

FIGURE 4. A HMM with 3 emitting states, aligned using Viterbi algorithm.

search to find the best alignment of states to the observation sequence. Once we find this alignment, we update the Gaussian parameters to reflect this alignment. We repeat alignment and reassignment until convergence. In HTK Uniform segmentation and Viterbi training counts as initializing the model. This model is a good initialization method, however it makes a very crude assumption of aligning each observation to strictly one state. (**holmes**)

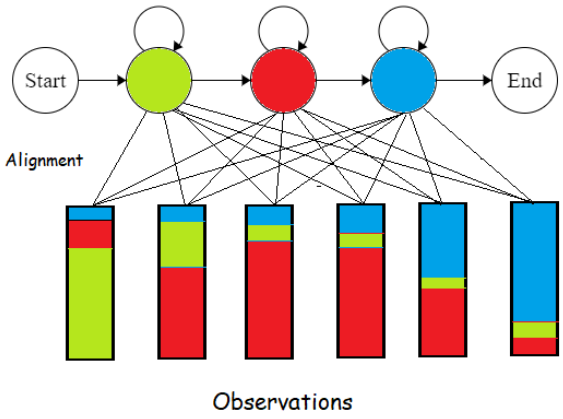

FIGURE 5. A HMM with 3 emitting states, being aligned using EM algorithm. Colors represents the weights of alignment between colored state and the observations.

We tackle this problem, and generalize Viterbi training in the Baum-Welch algorithm. In this algorithm each observation has a soft probabilistic assignment to each to state. This is shown Figure 5. Then we average over all the hidden variables, and the transition probabilities using the current model parameters. This is known as the expectation state. Next, we update the transition probabilities and Gaussian's parameters such that it maximizes the likelihood of our training data. This method of iteratively refining the model with respect to training examples of the word, guarantees to increase the likelihood of the training data. **holmes**

However, it doesn't guarantee increase in likelihood on the test data. Hence, we must ensure training set is representative. We also perform various manipulations as seen in the acoustic features section to our features which ensure their compatibility with both Gaussian mixture models and HMMs.

2.3. **Language Modelling.** Language models aims to compute how likely a given word sequence is. This is also known as the prior probability of word sequence. This is often calculated by building tri-gram model with backoff which is trained on a

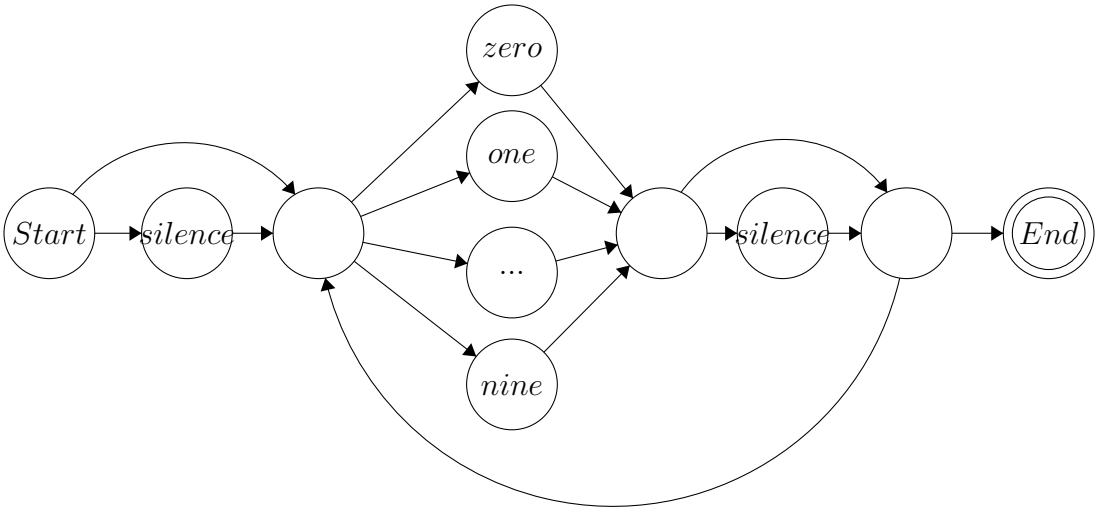

FIGURE 6. Grammar network for continuous digit recognition with silence allowed between spoken digits. Treated as huge HMM.

huge corpus. This are generally too big to visualized, but could be thought of a fine state network where each word has an arc leading to the other word with some probability. Traveling down the arc is the prior for that word sequence.

In our case of isolated, or continuous digit recognition the finite state of grammar network can be visualized easily. Figure 6 shows one such visualization. In our case all the prior probabilities are equal. HTK allows us to easily create grammars which are then converted to a finite state grammar network, like in Figure 6. (Young, 2006)

Most importantly, note that each of words zero, to nine and silence consist of a HMM. That means Figure 6 is just another large HMM with each arc coming out of the state have a transition probability of 1/(number of outgoing arcs from state). This is because our prior probabilities are the equal.

2.4. **Recognition using HMMs.** Our task is, given acoustic signals find the most likely word sequence. I.e.

$$\dot{W} = arg \max_{W \in L} P(O|W).P(W) \tag{2.1}$$

L is our language model, and W is all possible word sequences. As discussed earlier, in our case of digit recognition the $P(W)$ is the same for all (W) and so can be safely omitted.

The HMM consists of a lot of alignments between the HMM states and our observations. And $P(O|W)$ represents the sum over all of these alignments and their probabilities. However, computing all this is very slow and so we make an approximation of finding the probability of the most likely state sequence. By making this assumption we can use the Viterbi Algorithm to find the most likely state sequence.

It is a dynamic programming based algorithm to find the best path through a matrix where the vertical dimension represents the states (s_i) of the HMM and the horizontal dimension represents our observation sequence (o_i). Each point (o_i, s_j) represents the probability of observation o_i belonging to state s_i, and the probability of a path is the product of the probabilities along that path. and for each s_i we can compute all the paths from s_{i-1}. We do this recursively until we find the best path. Young, 2006

In HTK we use a techinque called token passing, as this generalizes the viterbi algorithm to word sequences, and allows us to add more pruning options. It is a parallel search algorithm where we pass a copy of token to all possible transitions. Moreover, whenever two tokens arrive at the same state, we only keep the token with higher probability. This simply generalizes to word sequences, as we saw in the previous section and in figure 6 that our language model is huge hidden markov

model, and so we can in the same manner pass tokens through this huge HMM. Now, when the best token reaches the end, its route taken represents the word sequence generated by our model for the observation. (Young et al., 1989)

Despite using Viterbi search the algorithm can seem computationally expensive and so we often add additional pruning methods. This usually involves getting rid of some tokens. One such method is called the beam search where we discard paths whose likelihood score is score lower than a given threshold with respect to the best partial path at that time. In HTK by default it uses a beam size of 250.

3. Experiments

I'll begin by investigating some key factors that affect my model first, in terms of computational resources and efficiency. Once, I have a good model at hand I'll be able to test it's robustness in environment with different issues in datasets.

The data used is students recordings of digits from all the previous years. The data has been checked for potential bad elements like missing training files, incorrect number of test files, recording of word "oh" instead of "zero", missing labels (lack of gender, microphone or accent) or transcription errors and such have been removed. This leaves me with a little over 300 users data. It's from this set of data all of my experiments pick data points from. Similarly, checks are made that no user is ever present in both training and testing partition.

3.1. **Number of states. Hypothesis:** As we increase the number of states representing a single word, the accuracy on the test data set should increase until it starts to flatten out, with diminishing returns.

Data: I will use a 70:30 train test split. It uses stratified splitting algorithm with respect to the gender, accent type and recording device. Hence, the distribution of

N States	Accuracy	Time Taken
3 states	74.7%	**01:13**
4 states	80.6%	01:17
5 states	86.1%	01:21
6 states	90.7%	01:27
7 states	92.6%	01:35
8 states	93.8%	01:50
10 states	95.7%	02:02
15 states	**96.8%**	02:42

TABLE 1. Performance with respect to number of states in HMM representing a word

gender, microphone type and accent type across the train test split is very even in ratio. This results 210 training participants and 90 test participants.

Results: Note, the number of states includes the start and the end state, hence, a 3 state only contains 1 emission state. The time taken represents the time taken for initialization, training and prediction by the model in minutes:seconds format.

Discussion: Refer table 1. As I predicted, increase in the number of states does increase the accuracy, however as it n states approaches higher values the amount of increase in accuracy starts to slow down. 10 states representing a word seems an appropriate choice as the accuracy is quite high with respect to the time taken. So, I'll use the 10 state model in my further experiments.

3.2. **Number of beams in Beam Search/pruning. Hypothesis:** The beam size is directly proportional to the time taken and the accuracy on the test set.

Data: The data is prepared in the exact same manner as in section 3.1.

Results: Note: The time taken represents the time taken to use an already trained 10 state model to make prediction on 2700 test files. Moreover, each experiment was performed thrice and the time taken was averaged.

Beam Size	Accuracy	Time Taken
10	59.4%	00:33
20	72.3%	00:38
50	89.4%	00:41
100	95.3%	00:42
200	95.7%	00:47
500	95.7%	00:59
1000	95.7%	01:21

TABLE 2. Performance with respect to number of beams in beam search.

Training Method	Accuracy	Time Taken
Viterbi training	95.3%	01:13
Baum Welch	95.7%	02:11

TABLE 3. Viterbi Training vs Baum Welch

Discussion: Refer table 2. As I predicted both time taken and accuracy increase with increasing beamsize. However, note after a beam size of 200, the performance of the model seems to make no additional gains in accuracy, hence, I'll be using the beam size of 200 for all future experiments. Also, it's interesting that the amount of time taken at 20, 50, and 100 beam size is very close.

3.3. **Viterbi training vs Baum-Welch. Hypothesis:** Baum-Welch Algorithm significantly improves the performance, but takes much longer to train.

Data: The data is prepared exactly as in section 3.1.

Results: A 10 state model with 200 beam size is used. The time reported represents the training, and prediction times.

Discussion: Refer table 3. I was quite surprised to see the Baum-Welch algorithm add very little improvement to the viterbi training, but take twice as long. I thought, maybe factors like the training dataset being representative of the test dataset, or the 10 state model already having high performance, or the huge size of training

Training Method	Accuracy	Time Taken	note
VT	93.0%	01:09	Randomly shuffled instead of stratified
BW	93.1%	02:13	Randomly shuffled instead of stratified
VT	83.3%	01:01	5 state and randomly shuffled
BW	84.2%	01:19	5 state and randomly shuffled
VT	81.7%	00:43	training data size 20, random users
BW	83.1%	00:52	training data size 20, random users

TABLE 4. Viterbi Training vs Baum Welch

dataset surprseed the benefits of the Baum-Welch Algorithm. So I performed further experiments, wherein I changed these specific factors.

Discussion2: Refer table 4. Despite my best attempts to show that Baum-Welch algorithm significantly outperforms the Viterbi training, it doesn't seem to do so. This might be specific to our task of isolated digit recognition being quite simple. However, note it does outperform the viterbi algorithm in every example tested, and because the training process generally takes place offline, we can choose to run a more computationally expensive algorithm to update our weights. Hence, choice of my model's training algorithm is Baum Welch for the future experiments.

3.4. **Time derivatives/ deltas. Hypothesis:** The more time derivatives we use the better the performance, however after some degree returns diminish.

Data: The data is prepared exactly as in section 3.1.

Results: _0 refers to the energy coefficient. Delta _D refers to the first order regression coefficients, Acceleration _A refers to the 2nd order regression coefficients, and Third _T refers to the third order. The time taken represents the time taken for initialization, training and prediction by the model in minutes:seconds format.

Discussion Refer table 5. MFCC's without any time derivatives performs much worse than the other models. This proves the importance of having these time derivatives. Also, surprisingly, MFCC_0_D_A_T performs worse than MFCC_0_D_A.

Diff Coefs	Accuracy	Time Taken
MFFC_0	87.2%	02:18
MFCC_0_D	94.2%	02:28
MFCC_0_D_A	95.0%	02:35
MFCC_0_D_A_T	94.4%	02:43

TABLE 5. Importance of time derivatives as Features.

HMM transitions	Accuracy
only forward connections	95.0%
Skip and back connection	95.4%

TABLE 6. HMM transition architecture

This could be due to the increased number of parameters that the we need for our gaussian mixture model, which provide very little new information. As we MFCC_0_D_A perform the best, these will be my choice for rest of the experiments.

3.5. **HMM transition architectures. Hypothesis:** Adding skip connections, and back connections to the HMM architecture for the word should help improve it's performance.

The reasoning is, to make the model more robust by allowing each state to absorb various impulsive noises in the training data. The backward skip allows this to happen without committing the model to transit to the following word. Young, 2006

Data: I have prepared the data in same manner as in section 3.1.

Discussion: Refer table 6 This offers some improvement in performance. Although, the true benefits of this transition architecture have to be tested out in a large vocabulary continuous word sequence model.

So far we have investigated different parameters of model and understood how they affect the performance. We have settled on a 10 state model, with 200 beam size,

Unshuffled & random		**Stratified data**	
Training Data size	**Accuracy**	**Training Data size**	**Accuracy**
13	84.51%	13	92.25
25	89.48%	25	93.00
49	91.36%	50	93.39
99	93.93%	99	93.93
199	94.30%	199	94.11

TABLE 7. Accuracy of model with respect to amount and quality of data.

with MFCC_0_D_A as our observable features trained with Baum-Welch algorithm for our isolated digit recognition task.

3.6. **Quantity and Quality of training data. Hypothesis:** Quantity of data improves performances, however quality of data plays a more important role.

Data:

(1) I split the entire dataset into two parts of 60:40% without any shuffling, or good measure. The 40% serves the test dataset. Next I incrementally pick from the remaining 60% of the data from roughly 10 training users to the complete 200 training users.

(2) I make a new split of the entire dataset into two parts of 60:40% and this time I use stratified sampling this sampling takes into account the user's recording device and their accents. The 40% serves the test dataset. Next I incrementally pick again in form of stratas from the remaining 60% of the data from roughly 10 training users to the complete 200 training users. Each of these increments consist of balanced attributes like accent and recording device.

Discussions: Refer table 7. We see the a general trend of increasing accuracy as the training dataset's size increases. However, in the unshuffled/random dataset

Trained On	Headset	Mobile	Desktop
Headset	**96.03%**	97.44%	91.16%
Mobile	92.76%	**97.97%**	92.90%
Laptop	91.98%	96.92%	**94.35%**

TABLE 8. Each Grid cell represents the accuracy when trained on device type in same row as the cell and tested on dataset represented by the grids column.

we see very bad accuracy when the training data size is 13, whereas when the those 13 samples are carefully chosen we see in the stratified sampling the accuracy being quite high. This shows that model is very sensitive to good data, and is not very data dependent if given well balanced training dataset.

3.7. **Recording device.** I roughly categories the types of devices used by students into 3 categories, headsets, mobiles and desktop/laptop. **Hypothesis:** When trained with a specific type of device the model doesn't generalize well to recordings from other device types. **Data:** I divide the dataset into the 3 above category. For each of this category I used stratified sampling with respect to gender and accent to maintain an even ratio in the train/test datasets. I trained on each of training sets, and tested the performance on each of the test set for each training. In each train test split, there are at least 30 training user and 10 test users, this amount is sufficient as shown in section 3.6

Discussion: Refer table 8. As expected, the best scores in each category is when the model was trained on the same dataset type. When trained on Headset, the model managed to perform well on mobile dataset. This might indicate that mobile dataset probably had less noise causing models trained on Headset and desktop to still perform well. However, headsets, or desktops seem to have some property unique to themselves that might is not capture when not trained on them.

4. Discussion and Overall Conclusion.

By conducting all the experiments above I got to test my model's robustness, adaptability and efficiency. I started by testing the model for its efficiency and importance of various techniques applied. Through my experiments, its clear that methods like beam search for pruning, delta coefficients as additional features have a clear advantage. We also found out that picking the number of states for an HMM is also a very important factor, however it greatly increases the cost of the model. This might particularly not scale very well. Overall, the model seems to be very quick with an average prediction time of 1/2 millisecond per observation.

We also saw that the model is very dependent on good quality of data, but not necessary large quantities of it. This means we must focus our efforts on collecting varied and testset representative training dataset. Moreover, as we collect new data we can continue training the model in an online fashion as we saw that it is quick at adaptablity too. We were also able to show that small details like the type of recording device can also play a huge role in the data. Hence, it's really important to at least have some small amount of data representing all possible types of variations in the test environment. These include the recording devices and the user's accents.

Lastly, all of these experiments show us the good and bad attributes of our model, which can we learn from to build better models. Learning that delta's are quite important, we can try to have recurrent neural network based architecture. With modern computing we can seek better ways of estimating the Gaussian's co variance matrix rather forcing it to use only the diagonals. Seeing how tri-gram language models take up large amounts of space we can build architectures that fit in on a mobile device. Therefore, we can build models that are even more robust, adaptable and efficient.

References

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing (2nd edition)*. Prentice-Hall, Inc.

Young, S. (2006). *the htk book (for htk version 3.4)*. cambridge University engineering departent.

Young, S., Russell, N., & Thornton, J. (1989). *Token passing: A simple conceptual model for connected speech recognition systems* (tech. rep.).

Holmes, J. N., Holmes, W. 2001 Speech synthesis and recognition. London: Taylor Francis.