Recognizing Spoken Arabic Digits: A Generative Model Problem

Matthew Bloom

Background and Problem Description

This project focuses on automated speech identification for the spoken Arabic numerals 0-9. The data utilized is the Spoken Arabic Digit dataset of mel-frequency cepstral coefficients (MFCCs) from the UCI Machine Learning Repository, which contains 6600 training samples (660 for each digit) and 2200 test samples. Each sample contains the time series of the 13 MFCCs for an utterance of one of the digits in Arabic, organized in time by analysis frames and in dimension by the order of the MFCCs. The computation of the MFCCs themselves is not the focus of the project, but they are based on the log of the frequency spectrum for a given frame.

The project's goal is to build a probabilistic generative model to represent the MFCCs. Then, modeling choices are explored to find how they affect the performance of a classifier in correctly identifying the spoken digit based on the series of MFCCs. The general problem of speech recognition is interesting due to its variety of applications, such as in automatic translation. For this particular situation, examining modeling choices to produce a low-cost and accurate model for speech recognition could be interesting, for example, in the domain of healthcare, where language barriers can make care difficult. While this problem focuses on a relatively simple task where the domain is restricted to only 10 different utterances, the ideas examined here could be applied to a more robust model.

One similar scenario where applying a generative model could be appropriate would be the problem of out-of-distribution (OOD) detection in machine learning. The goal is to detect when an input is not from the training set of classes, which could be accomplished by modeling distributions for the in-domain classes and OOD classes and examining the differences between them.

Basic Model Choices

The project focuses on building a set of Gaussian Mixture Models (GMMs) as generative models of the distributions of the data for each digit (so we build 10 models, 1 for each digit). A generative model represents a model that represents a distribution which has a high likelihood that it produced the data. A GMM is formulated as depicted on the right, as a sum of M “mixture components,” each with a corresponding probability π, such that the probability density functions of all the components sum to 1. For a GMM, each mixture component is a normal (Gaussian) distribution *N* with its own mean and covariance matrix. In the context of the problem, each mixture component M represents a unique phoneme of each spoken digit.

$$\mathrm{GMM} = \sum_{m=1}^{M} \pi_m N(\mu_m, \mathbf{\Sigma}_m)$$

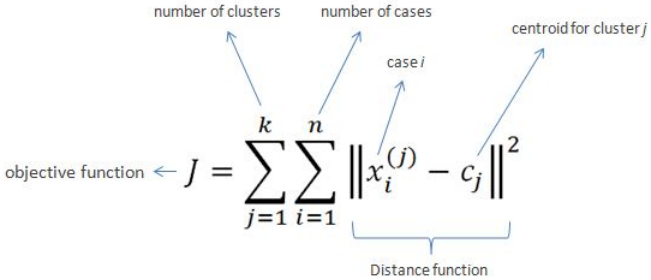
The initial model choice for the project centers around how to estimate the GMM parameters (means and covariances), with two methods: K-Means clustering and Expectation-Maximization (EM).

K-Means Clustering

K-Means is an algorithm used to group the data for each digit into groups (clusters). From those clusters of observations, the means and covariances are computed directly and used as the mixture component parameters. So, to estimate a GMM for a digit d with M mixture components, we aggregate the data together, use K-Means to form M clusters, then calculate the M means and covariances for each component.

The objective of the K-Means algorithm is to minimize the loss function on the right, which represents the squared distance between the observations *x* and their cluster centers *c*. The algorithm is as follows:

1. Select k, the number of clusters
2. Choose k points as the initial cluster centers
3. Assign all the observations to each cluster center based on which they are closest to, and calculate J.
4. Find the means of the observations in each cluster, assign as the new cluster centers, and calculate J again.
5. Repeat steps 3 and 4 until J converges.



K-means is advantageous in that it is fairly efficient and easy to implement, but faces a number of pitfalls. One is that k must be chosen ahead of time, while another is that the algorithm performs best when clusters are distinct and “clump-like.”

Expectation-Maximization

Expectation-Maximization is an algorithm that directly results in the GMM parameters (means, covariances, and probabilities). The algorithm's goal is to maximize the ln of the likelihood (log-likelihood) of the data given the GMM with the parameters $\boldsymbol{\theta}$ that it has estimated. The next slide will examine exactly how that likelihood is formulated and calculated. The EM algorithm is as follows for a GMM with M components, consisting of repetition of the Expectation and Maximization steps:

1. Choose initial values for the M means, covariances, and probabilities ($\boldsymbol{\theta}$).
2. E-step: for the current $\boldsymbol{\theta}$, evaluate the **responsibilities** for all observations, which are the set of M probabilities (for each point) representing the likelihood that a given observation was drawn from the mth mixture component.
3. M-step: for the current responsibilities, estimate $\boldsymbol{\theta}$
4. Repeat 2 and 3 until the $\boldsymbol{\theta}$ found or the log-likelihood converges.

Maximum Likelihood Classification/Classifier

In this project, we are not interested in the varying the choice of classifier, but we will be using a maximum likelihood (ML) classifier. ML classification involves evaluating the likelihood of the data conditioned on each one of a set of models representing classes (in our case, d=10 different GMMs representing each Arabic spoken digit). Then, the classifier chooses as its result the class corresponding to the model with the highest likelihood. ML classification is well-suited for this problem because we have a set of distinct classes with unique distributions that can be modeled by separate probabilistic models from which these likelihoods can be computed.

The first formula on the right describes the formula for the likelihood of an utterance conditioned on the model for a digit *d* used in ML classification for our problem. **X** represents the set of *N* frames of MFCCs for a given utterance, $\mathbf{\Delta}_d$ represents the set of GMM center and spread parameters for its M components (M sets of means and covariances for each digit), and $\mathbf{\Pi}_d$ represents the set of M mixture component probabilities for the GMM. p() represents the pdf for a component of the GMM evaluated on a single frame. So, in summary, the pdfs (scaled by their probabilities) for each of the M GMM components are evaluated on each frame and then summed to give the total likelihood of a frame. Then, the likelihoods are multiplied together for all *N* frames.

$$p(\mathbf{X}|\mathbf{\Delta}_d, \mathbf{\Pi}_d) = \prod_{n=1}^{N} \sum_{m=1}^{M} \pi_{m,d} p(\mathbf{x}_n|\Delta_{m,d})$$

$$\mathcal{L} = \ln(p(\mathbf{X}|\mathbf{\Delta}_d, \mathbf{\Pi}_d))$$

$$= \sum_{n=1}^{N} \ln \sum_{m=1}^{M} \pi_{m,d} p(\mathbf{x}_n|\Delta_{m,d})$$

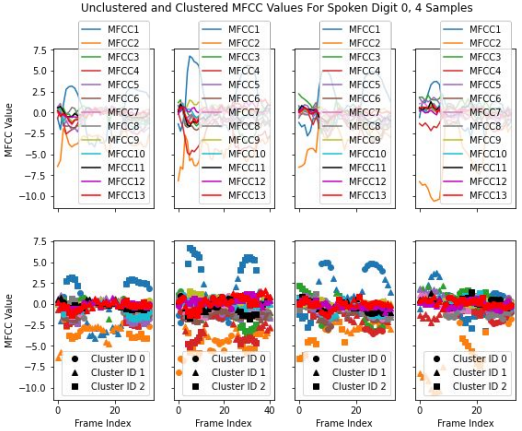
The main challenge when applying ML classification was the issue that the product across all *N* frames often includes terms that due to machine precision would drive the result to 0. In order to overcome this challenge, we convert the likelihood formulation to the log-likelihood $\mathcal{L}$ by taking the natural log. Using the log product rule, the product across all *N* frames becomes a sum of log probabilities, resulting in a sum of many negative numbers that does not suffer from the same machine precision issues.

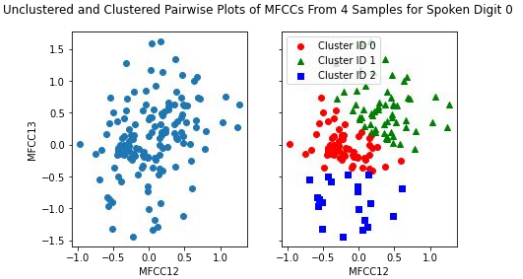
Determining Basic Model Hyperparameters

Before building the EM- and K-Means-based GMMs, a set of “basic” hyperparameters to be used by each set of models had to be determined. The first was the number of mixture components *M* to use for each digit *d*. This was matched with an estimate of the number of distinct phonemes (units of sound) in the utterances of each digit. The theoretical motivation behind this was that we would like to create a generative model where it is likely that each mixture component generated the MFCCs corresponding to a phoneme of the model’s digit. The other two hyperparameters that were tuned were the number of MFCCs to include for each utterance and the constraints on the covariance matrices for each mixture component. For this basic case, both set of models were left as flexible as possible; all 13 MFCCs were used for training and testing, and a “full” covariance matrix was utilized for each component, where the pairwise (co)variances for all 13 MFCCs were computed.

Determining Mixture Component Numbers

In order to estimate the number of phonemes and therefore the number of mixture components to use in the GMMs, the spoken digits for 0-9 were examined in 3 different contexts. The first was simply the background information of the phonetic pronunciation of each digit. The second was the set of time series plots of the MFCCs' values across the analysis frames for each utterance. Plots for 4 sample utterances for each digit were generated, first with the raw values of the MFCCs and then clustered using K-Means with the number of clusters corresponding to the number of phonemes estimated for the pronunciations. This allowed evaluation of whether the phonetics corresponded well to the clusters of numerical values observed and subsequent tuning if they did not. Finally, pairwise scatter plots of the MFCCs (for the 4 samples concatenated) were also generated and then clustered to remove the time component. This allowed examination of whether clustering performance appeared better when data was aggregated without respect to time. The results for digits 0-9 follow.

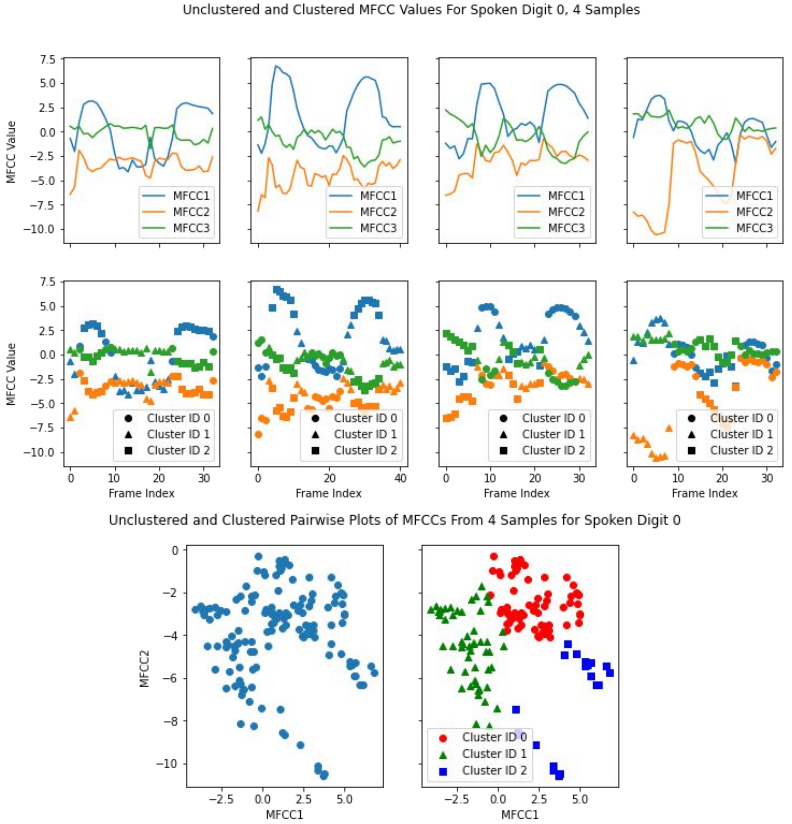




Note: on the slides that follow, on the time series plots only the first 3 MFCCs are included, and on the pairwise scatter plots we compare the 1st and 2nd MFCCs. This is to induce consistency and reduce cluttering of the values and legends (the time series plots are quite difficult to read on this slide). It also appears when examining the time series plots more closely that the most distinct variations are observed with the first few MFCCs, additionally giving solid theoretical motivation.

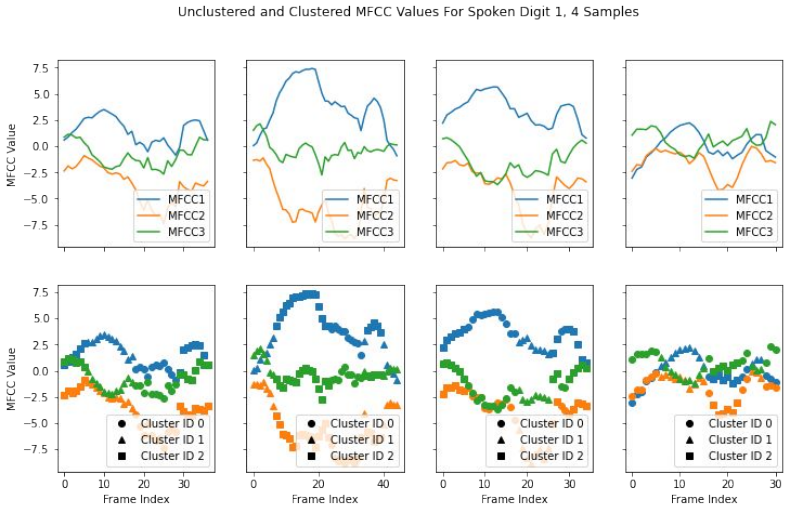
Mixture Component Numbers Continued: Digit 0

Here, when examining the time series plots, there appear to be **roughly 3 distinct peaks/valleys (phonemes)**. The clustered time series plots appear to be approximately grouped according to these phonemes, however the grouping seems to more closely follow the transitions between consistent *values*, which are usually consistent for the 1st and 3rd phoneme. For example, on the second plot, the 1st and 3rd peak were assigned to cluster ID 2. In terms of time, there is a bit of intermingling between the circles, triangles, and squares. On the pairwise plot where time is removed, the clusters are not entirely distinct, but we can make out a fairly good grouping of 3 clusters.



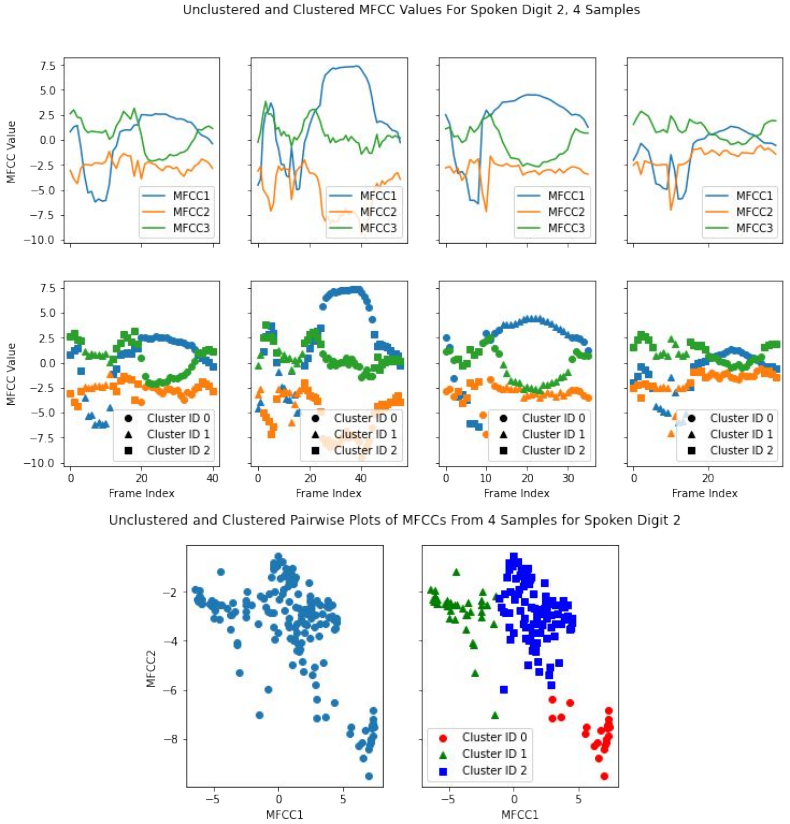
Mixture Component Numbers Continued: Digit 1

Similar trends to those seen observed for digit 0 appeared for digit 1 and those that follow, where there appear to be a number of roughly distinct phonemes in the time series plots. Again, on the pairwise plots, the clusters are not entirely distinct, but we can make out a rough grouping. This slight ambiguity between the clusters with K-Means might indicate that when the models are built, EM will perform better. The K-Means algorithm assigns hard boundaries between clusters, while EM assigns soft responsibilities corresponding to the relative probability that an observation was drawn from a mixture component. Therefore, when observations estimated as generated by components overlap, K-Means may result in biased estimations of the model parameters. For digit 1, based on the shape of the time series plots, **3 components seems appropriate.**



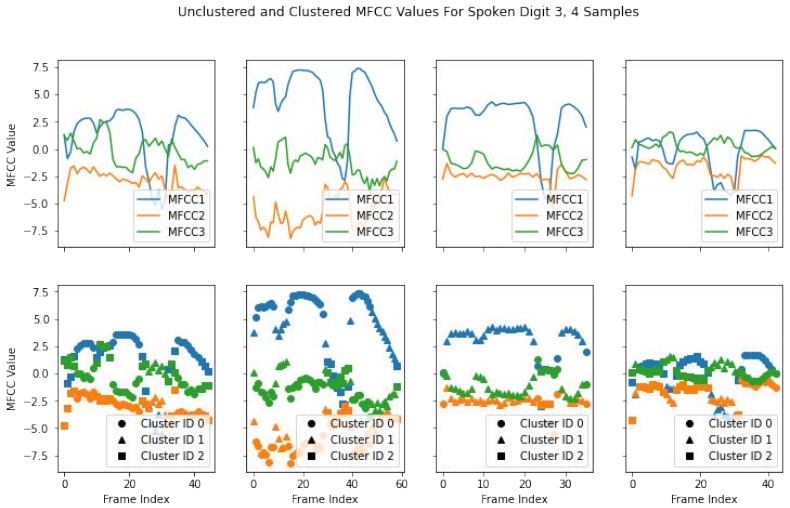
Mixture Component Numbers Continued: Digit 2

Here on the time series plots, there appear to be roughly **three phonemes identified**. The symbols indicating the cluster IDs change over the course of the analysis frames of each utterance as the data transition between the peaks/valleys. Again, the pairwise plots represent some overlap but do produce three approximate groups. An argument could certainly be made here that the plots appear to indicate 2 components is appropriate, but later testing found that accuracy significantly improved (~10%) upon adding a component.



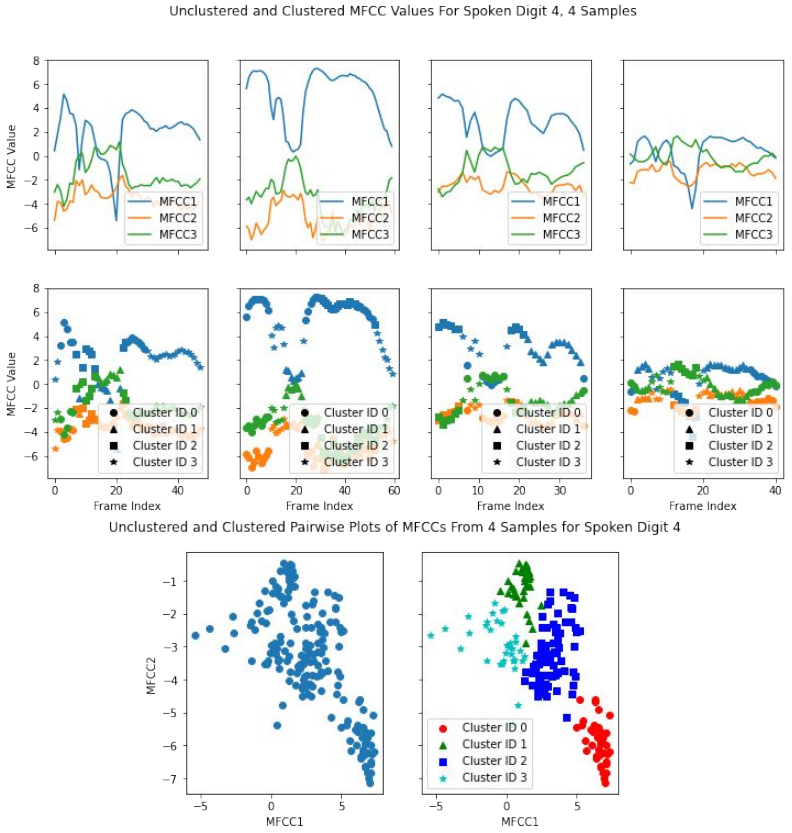
Mixture Component Numbers Continued: Digit 3

Here, while the time series plots appear to represent 4 peaks/valleys, the 3 peaks are fairly close in value. Additionally, the pairwise plot seems to lend itself more to 3 groupings (the top left, top middle, and bottom right). Therefore, **3 components were used for the digit 3 models.** In the slides that follow, similar intuition was used to determine the number of components to use for each digit, so the logic is not repeated 6 more times. The number of components used and any additional interesting trends are commented on.



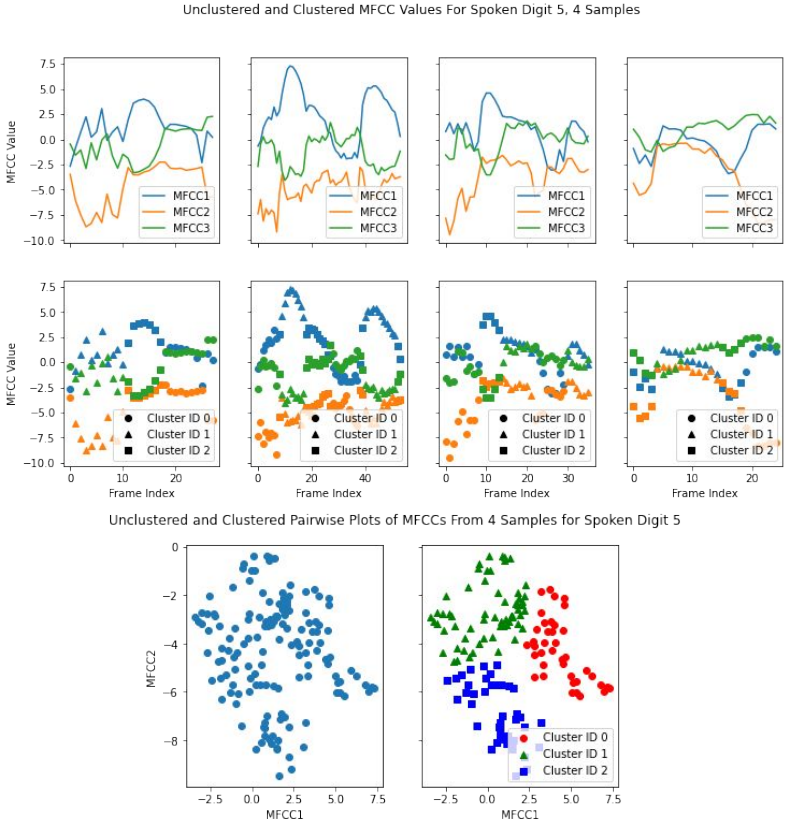
Mixture Component Numbers Continued: Digit 4

4 peaks were observed and therefore **4 components were used for the digit 4 models.**



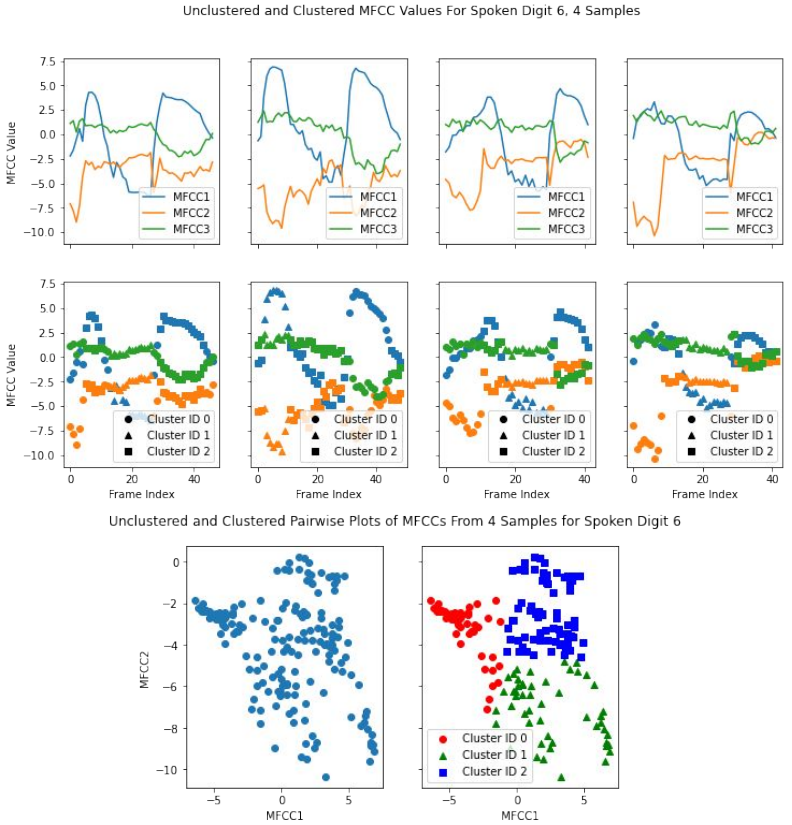
Mixture Component Numbers Continued: Digit 5

3 components were used for the digit 5 models.



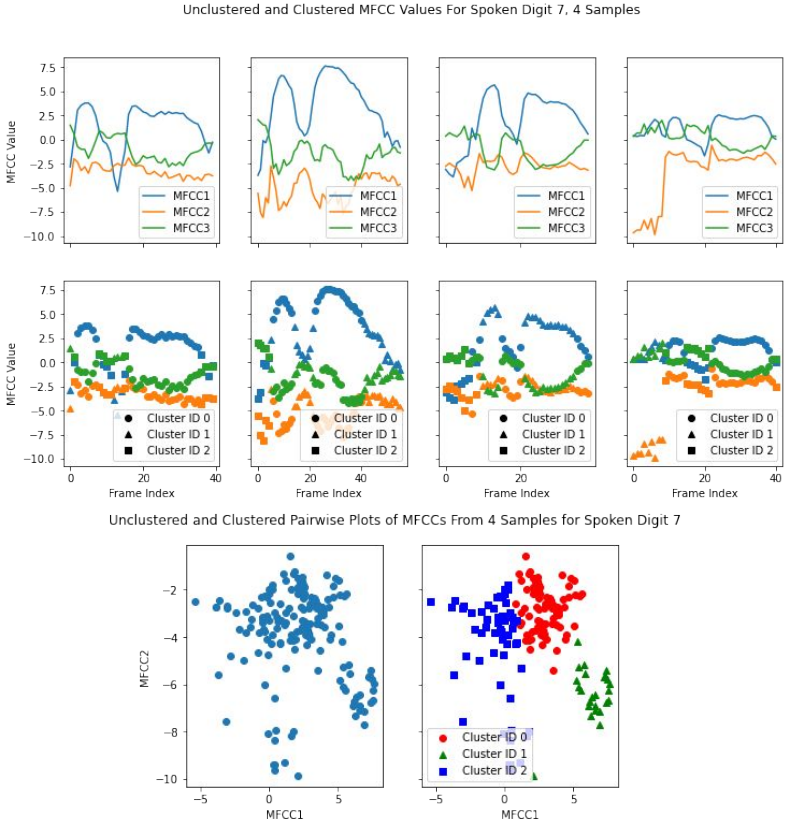
Mixture Component Numbers Continued: Digit 6

3 components were used for the digit 6 models. The transitions in time between phonemes are fairly smooth here, seen on the plot as a rough continuity from circles to triangles to squares.



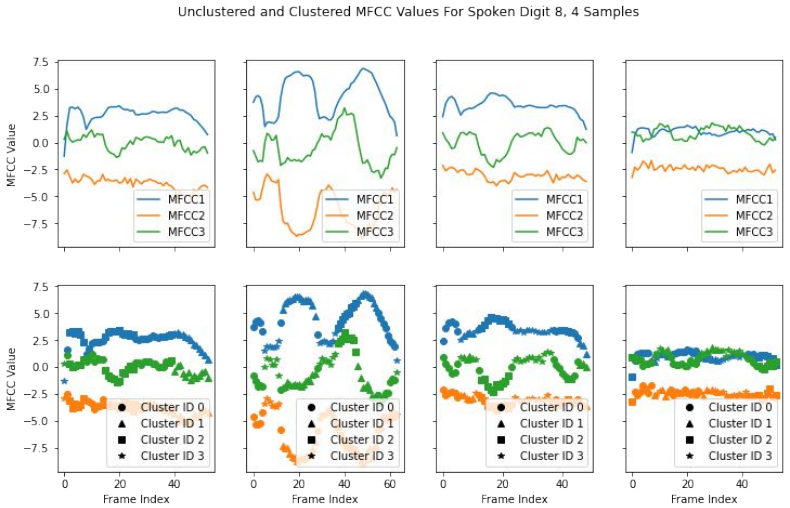
Mixture Component Numbers Continued: Digit 7

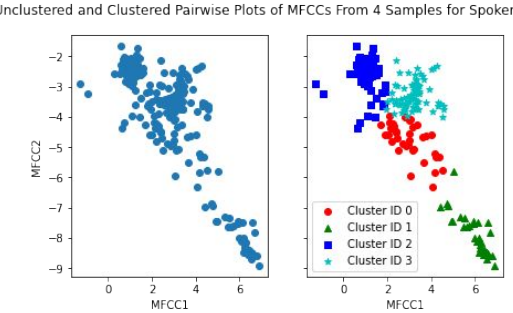
3 components were used for the digit 7 models.



Mixture Component Numbers Continued: Digit 8

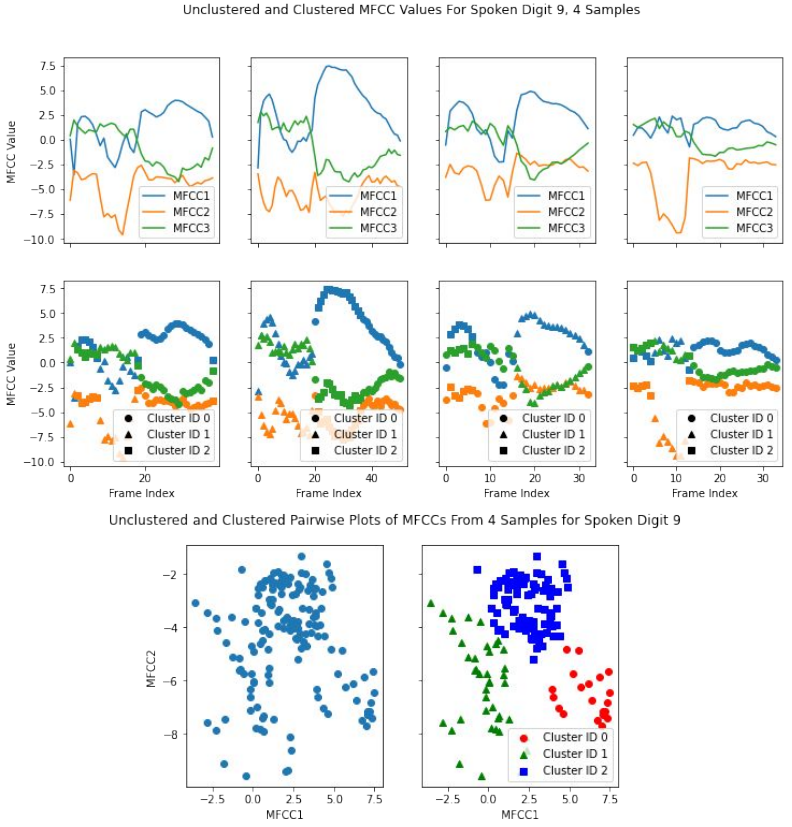
4 components were used for the digit 8 models. The number of components was supported both by the shape of the time series plots and the phonetics of the digit (‘thamanieh’)





Mixture Component Numbers Continued: Digit 9

3 components were used for the digit 9 models.



Mixture Component Number Determination Summary

This slide presents a table of the number of phonemes and therefore mixture components that were used in the models for digits 0-9. These numbers were used for all models examined in the following slides in order to maintain consistency.

Digit	#Estimated Phonemes/ Components
0	3
1	3
2	3
3	3
4	4
5	3
6	3
7	3
8	4
9	3

Basic Classification Performance

In the following slides, in order to evaluate performance we present confusion matrices as well as the overall probability that the classifier predicts the correct digit for the “basic” K-Means and EM-based GMMs. As mentioned earlier, each set of models utilizes the numbers of mixture components indicated on the previous slide, all 13 MFCCs for the corresponding digit, and “full” covariance matrices for each mixture component in each model. **The confusion matrix provides a visualization of how the classifier performs within each testing class (digits 0-9). For an entry in the confusion matrix at row i and column j, the matrix reports the probability that a test utterance with ground truth label i is given a predicted label of j.** So, on the diagonals, where i = j, we have the “true positive rate” for digit i: the probability that a sample labeled i is correctly predicted as i by the classifier. The probabilities on the off diagonals, where i != j, indicate the chance of a “confusion,” where a sample labeled i is incorrectly predicted as label j. These probabilities are computed by the number of predictions of label j for test samples with true label i divided by the total number of test samples with label i. For example, the entry at [0, 0] indicates the proportion of test samples for digit 0 that were correctly predicted, while the entry at [0, 9] indicates the fraction of digit 0 samples that were incorrectly predicted as digit 9.

The overall probability of a correct digit being predicted is simply the total number of test samples for which the true label matches the predicted label divided by the total number of test samples (for this dataset, 2200). Equations used to produce both performance metrics are presented on the right.

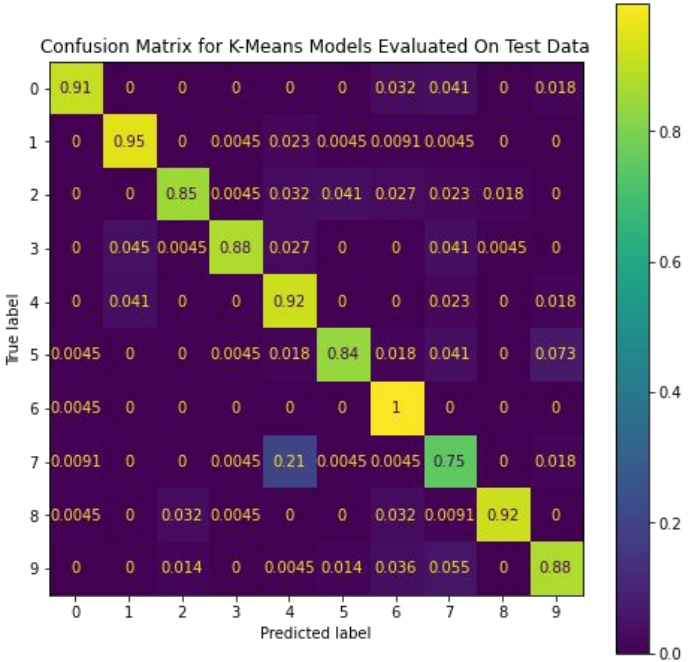
$$\mathrm{confuse}_{i,j} = \frac{\#\mathrm{predicted}_j \in \mathrm{true}_i}{\#\in \mathrm{true}_i}$$

$$p_{\mathrm{correct}} = \frac{\sum_{d=0}^{d=9} \#\mathrm{predicted}_d \in \mathrm{true}_d}{\#\mathrm{testsamples}}$$

Basic Classification Performance, K-Means GMMs

Even in this basic case with K-Means based GMMs, performance is quite good; overall accuracy is almost 89%. When examining the confusion matrix, we can see that the classifier performs worse in the sub-cases for some digits, the worst ones being 2, 5, and 7, which all have a “true positive rate” of 85% or less. The digit 7 is the worst, with only 75% of the samples being predicted correctly, and 21% being confused as digit 4. One possible explanation for this performance in particular could be the fact that the spoken digits are somewhat similar phonetically: “seb’a” and “araba’a” for 7 and 4, respectively. This could cause their MFCCs to have similar characteristics, and create difficulties in distinguishing between the digits/models. For 2 and 5, the confusions are more evenly spread among the other digits. One explanation for these confusions could be that at this point, we are not attempting to handle the differing number of analysis frames in the utterances for different digits, which could result in biases toward “longer” utterances. This is due to the accumulation of the sum of the log likelihoods in the ML classifier.

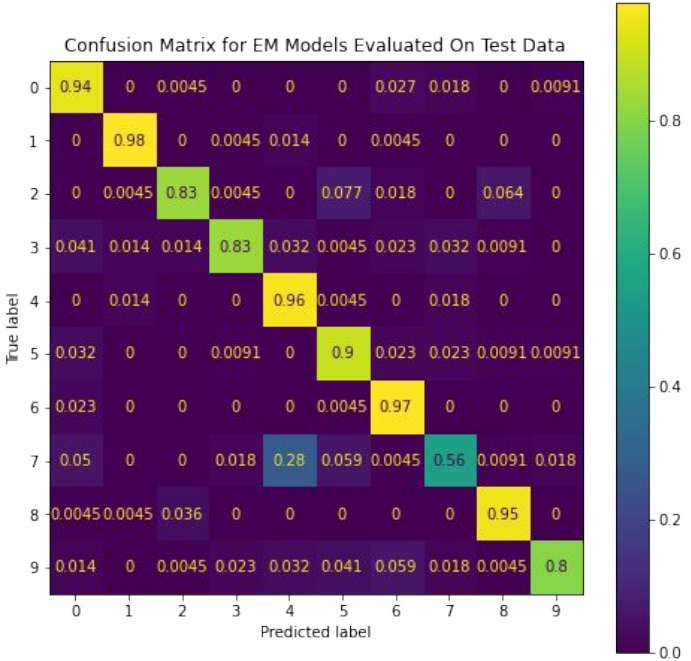
The performance is best for digits 1 and 6; both are above 95% correct (almost all of the samples for 6 were classified correctly, so its correct probability rounded to 1). An explanation for the performance in these subcases could be the uniqueness among the digits in phonetics; ‘wahad’ (1) does not appear phonetically similar to the other digits. The predictions for digit 6 could also be biased positively due to generally producing utterances with more analysis frames.



$$p_{\text{correct}} = 0.8895$$

Basic Classification Performance, EM GMMs

Overall performance for the classifier with EM-based GMMs is slightly worse compared to with K-Means-based GMMs. This contradicts my earlier hypothesis that performance would be notably better due to K-Means resulting in biased GMM parameters. For some digits, performance worsened, while for others it improved. For example, for 5, which previously had a true positive rate <85%, that rate increased to 90%. Compared to K-Means, the number of digits with true positive rates >=95% went from 2 to 4 (1, 4, 6, and 8). However, the amount of digits with true positive rate<=85% also increased from 3 to 4 (2, 3, 7, 9). The worst one, 7, also now has even worse performance; Only a little more than half of the utterances for the digit were classified correctly, and 28% were again confused for digit 4. One reason for this decay in performance could be the “ambiguity” afforded by the responsibilities in the EM algorithm vs. the hard boundaries of K-Means, which was predicted to help performance but may actually result in more confusions when boundaries are difficult to distinguish.

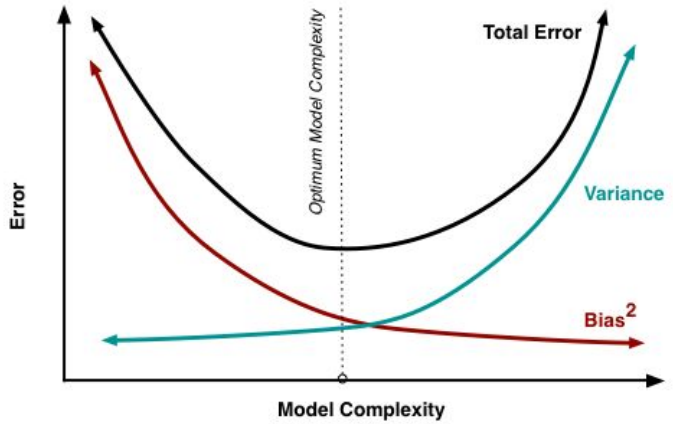


$$p_{\text{correct}} = 0.8732$$

Additional Modeling Choices

Additional modeling choices were explored in order to see how performance is affected and if performance can be improved. Most of these additional choices serve to move the models along the bias-variance tradeoff curve, as they vary the complexity of the models being built (by changing the number of parameters required to be computed). A more complex model is flexible and has low bias and high variance, while a less complex one is rigid and has low variance but high bias.

For each modeling choice, results are again presented in the form of confusion matrices and the overall probability of a correct digit. Additional visualizations and equations are presented when necessary. The classifiers used each time consist of K-Means GMMs with full covariances, except when noted otherwise.



A representation of the bias-variance tradeoff. Bias is a measure of the ability of an average model estimate h(x) to characterize the “true model” y(x) for the data across input datasets, and decreases as the number of parameters increases. However, at the same time, the variance increases, creating a tradeoff. Variance measures the model’s sensitivity to variations in the particular dataset. Ideally, a model balances the two, landing at the dotted line on the plot. A model with too little complexity may not vary much in its predictions, but they are unlikely to be correct on average and error will be high. A model with too many parameters might characterize y(x) well on average, but might fall apart when presented with noisy data, for example. This is similar to the concept of overfitting in machine learning.

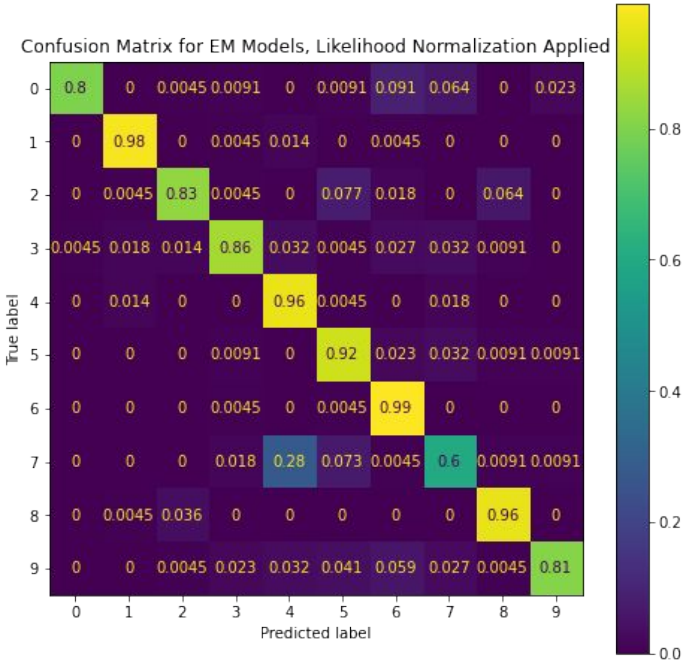
Handling Differing “Lengths” of Tokens: Likelihood Normalization

As was mentioned earlier, the varying amount of analysis frames in the utterances for each digit may result in biased likelihood estimates for a digit’s MFCCs given a particular model. For example, from examining the blocks in the test data, those for digit 6 tended to be around 50-60 frames long, while for digit 0 they had about 30-40. Due to the product across all N frames (or sum as in the log-likelihood classifier being utilized here), the results would be biased not due to the actual values in the frames, but due to how large N is for the utterances. In order to eliminate this effect, we examine normalizing the likelihood according to the number of frames N. For the non-log likelihood, normalizing the product would require taking the Nth root of the likelihood, but given the sum of log likelihoods, the only change to the classifier required is to divide the original likelihood by N, as shown to the right.

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \ln \sum_{m=1}^{M} \pi_{m,d} p(\mathbf{x}_n | \Delta_{m,d})$$

Likelihood Normalization Results, EM

Compared to the results without normalization on Slide 22, overall accuracy is (very) slightly lower. However, when examining the diagonals (representing the “true positive rate” for each digit) we see that the probability of producing a correct prediction has actually stayed constant or increased for every digit except for 0. The worst-performing digit, 7, went from a rate of 56% to 60% - still far from ideal, however, likely due to the phonetic similarity discussed earlier. So, it appears that for most digits the normalization improves EM results. The reason for the large drop (~14%) in accuracy for 0 could actually indicate that those results in particular benefited from the unnormalized likelihoods; due to 0’s phonetic similarity to 6 (“sifir” vs. ‘sittah”), a differentiator may have been the number of frames in the digits’ utterances. By examining the 0 column, we see that the normalized EM models rarely predict 0 in any sub-case at all.

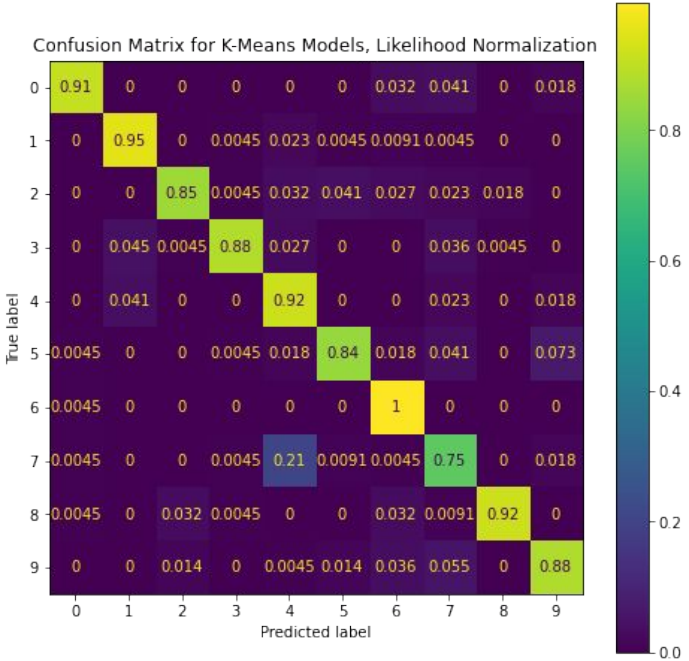


$$p_{\text{correct}} = 0.8714$$

Likelihood Normalization Results, K-Means

The performance for the K-Means GMMs with normalization was very comparable to without normalization. It resulted in a slight increase in overall accuracy (so slight, however, that with the rounded values in the confusion matrix the results appear almost the same). However, we would predict that this normalization across the length of the tokens would result in models that are more robust to test sets with more variance in analysis frame counts. If the models were exposed to higher variance datasets, the gap between the normalized and unnormalized classifiers would likely increase. Additionally, the increase in parameters needed is 0, meaning model complexity does not change, and the increase in computation required is trivial (a single divide per prediction). Therefore, even if the difference is small, at least for K-Means this modeling choice can be seen as “free” performance.

Based on these results, for all of the remaining modeling experiments, the ML classifier on slide 24 with likelihood normalization was utilized.



$$p_{\text{correct}} = 0.8900$$

Constraining the GMM Covariances

The effect of placing a constraint on the covariance matrix for each GMM component was explored in order to examine the effects of reducing model flexibility. This choice directly moves us to the left along the bias-variance curve on Slide 23, resulting in more rigid models with greater bias but smaller variance. The smaller variance afforded might be of benefit when the amount of data available is not enough to justify the most flexible models. Model parameter estimates may actually be improved with the lower flexibility models, instead of assuming the highest flexibility model without enough data and producing poor estimates. If performance remains comparable when reducing flexibility, lower computational cost and variance could be achieved by choosing those models.

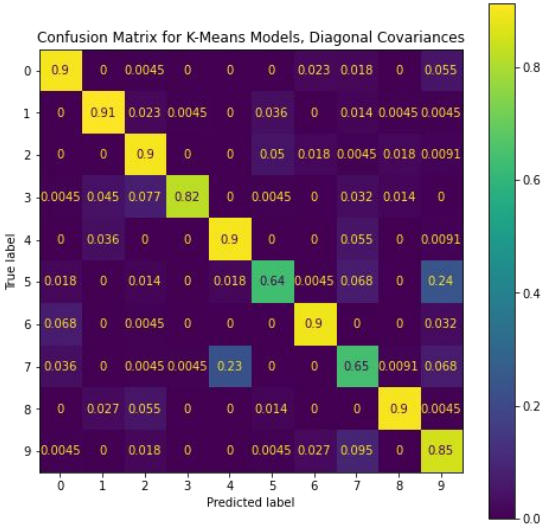
The constraints examined were no constraint (“full” covariances), assuming diagonal covariances for each component (implying independence among the MFCCs), and a tied full covariance, where the data is centered (mean subtracted) for each digit, a single full covariance matrix is computed, and that matrix is used as the covariance of all mixture components for the digit’s model.

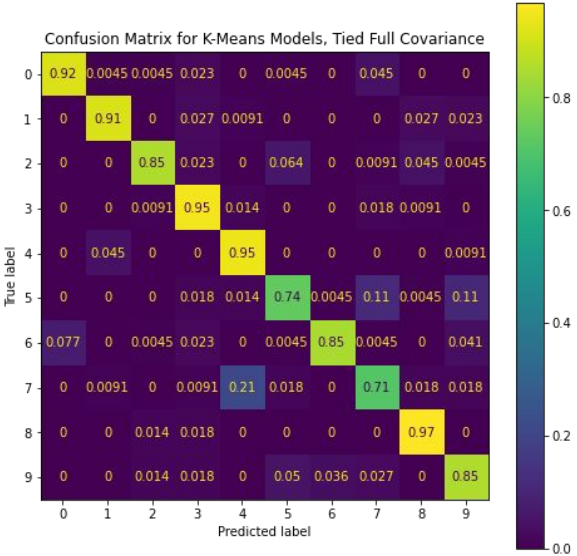
Constraint	# Parameters
Full (none)	(M/2)*182
Tied Full	91
Diagonal	M*13

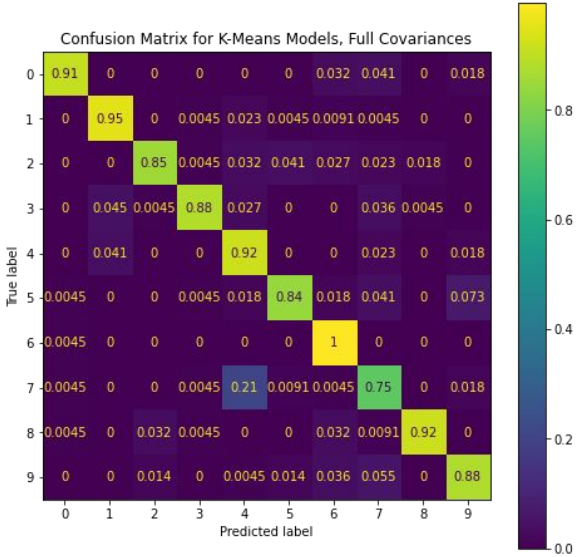
The above table visualizes the number of covariance parameters needed to be estimated for each digit’s GMM, assuming all 13 MFCCs are used (the covariance matrix is 13x13) and the model has M mixture components. Constraints are ordered from most to least flexible. The equation for each GMM and the classifier does not change, the covariance matrix used for each model simply changes according to the constraint.

Covariance Constraint Results

The results increase in model complexity from left to right. These results are discussed on the following slide!







$p_{\text{correct}} = 0.8364$

$p_{\text{correct}} = 0.8700$

$p_{\text{correct}} = 0.8900$

Covariance Constraint Discussion

As expected, overall accuracy across all digits decreased as model complexity decreased. However, notably, the overall decrease was only 2% when going from the most flexible models (full covariances) to the models with a tied full covariance. In the case of a model with M=4 mixture components, this represents a reduction from 364 to 91 covariance calculations required. When examining the individual rates in the confusion matrix for the “tied-full” models, true positive rates actually increased for some digits such as 3 and 4, but for others they dropped significantly. For the digits that suffered the most performance (such as 5 and 6), their individual full covariance matrices tended to vary more from the tied covariance matrix that was calculated, meaning their individual distributions had more significant differences compared to the pooled distribution. This meant they had “more to lose” when converting from individual to tied covariances. However, overall accuracy is high enough that this constraint appears to produce fairly suitable models, especially when less data is available, with the advantages of reduced parameter calculations and lower variance.

The diagonal covariance constraint for the same M=4 mixture components would result in an even further reduction from 91 to 52 covariance parameters. This results in a less acceptable drop in overall accuracy of over 5% to the point that overall accuracy is under 85% and the true positive rate for two digits (5 and 7) is 65% or lower. It appears that this constraint may be too much to produce satisfactory results except in cases of much lower observation counts; accuracy falls across the board, and confusions become more likely. The heaviest drops in accuracy for sub-cases (for 5, about 20%) are likely due to the presence of significant off-diagonal covariances that are eliminated by the constraints. However, considering the low amount of parameters needed, it is impressive that for most individual digits the accuracy is about 85-90%. Overall, though, this does not appear to be a suitable modeling choice for the particular problem situation given the heavy disadvantages and the number of test samples available, even despite the significant advantage in terms of reduced parameter count and variance.

Using MFCC Subsets

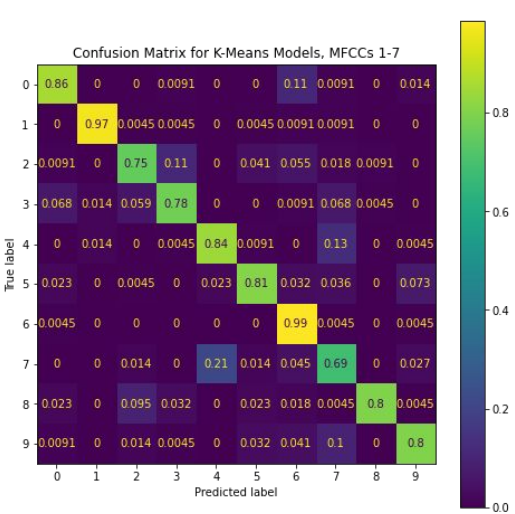
On slide 8, when examining the MFCC time series plots it was noted that the earlier orders of the coefficients from 1-13 appeared to contain more of the variation across the analysis frames. Using this intuition, the effect of including a reduced subset of the MFCCs when training (and therefore testing) the models was examined in order to see how many higher order coefficients could be safely removed without significantly negatively affecting performance. Similar to constraining the covariance matrices, this modeling choice reduces the number of covariance parameters that need to be calculated, moving the models along the bias-variance tradeoff. If D is the number of MFCCs included and therefore the dimension of the DxD covariance matrices, then for a GMM with M components $(M/2)(D^2+D)$ covariance parameters must be calculated.

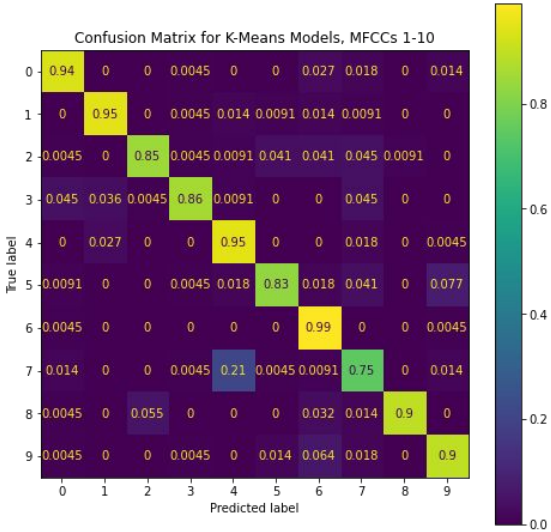
The table to the right indicates covariance parameter counts per model for the cases examined, where the first D MFCCs were used to train and test the model for a digit and the model contains M mixture components.

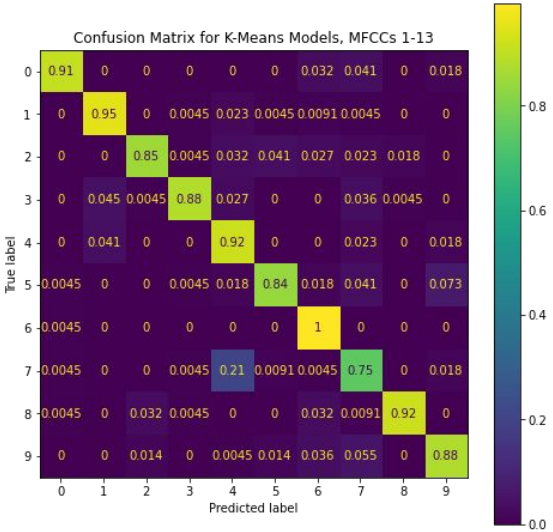
D	# Parameters
13	(M/2)*182
10	(M/2)*110
7	(M/2)*56

MFCC Subset Results

These results are discussed on the following slide!







$p_{\text{correct}} = 0.8282$

$p_{\text{correct}} = 0.8900$

$p_{\text{correct}} = 0.8900$

MFCC Subset Discussion

These results were quite impressive; 3 out of the 13 MFCCs could be completely excluded without affecting overall accuracy (to the 4th decimal point). Removing 3 more did result in more significant drops, but this would be expected when removing almost half of the training set's dimensions. Examining the confusion matrix for the model with the first 10 MFCCs, we see that some true positive rates (0, 4, 9) actually increase (and therefore confusion probabilities decrease). Some accuracy drops among the individual digits occur and some confusions become more likely, but these changes are mostly quite small. The advantages of this modeling choice are clear; parameter counts and variance decrease, with little negative effect on accuracy (to a certain extent). The main disadvantage is that the computational savings are clearly not infinitely free. When only including the first 7 MFCCs, there are 6 instances where the probability of a confusion is 10% or higher, overall accuracy is <85%, and 5 of the true positive rates are 80% or less. Overall, though, the savings when applying this modeling choice in moderation seem to make it a clearly suitable choice, especially if data is scarce. As an example, for a GMM with M=4 components, the amount of covariances that must be calculated drops from 364 to 220 when removing the last 3 MFCCs.

Conclusions: Modeling Choices

The most important modeling choice appeared to be the methods used to constrain the parameters, namely the covariance constraints and the limits on the MFCCs utilized. These modeling choices had such a significant impact on performance because as they were changed significantly, the models made strides along the bias-variance tradeoff as the number of components increased or decreased. If the number of parameters was constrained too tightly, accuracy dropped off significantly.

The less important modeling choices seemed to be the “binary choices”: selection of the method for determining GMM parameters (EM or K-Means) and normalization of the likelihood by analysis frame count. Neither of these had large effects on performance on way or the other, but that is not to say that they are unimportant. Using EM or K-Means seemed to have ramifications for how effective other modeling choices were, and result in biases to the model parameters. Likelihood normalization had a relatively small effect when comparing results after implementation with this test set, but it would be reasonable to predict that a dataset with more variation in lengths or lower signal-to-noise ratios could make its importance quite high.

Borrowing from the machine learning concept of “compound scaling,” I would combine the methods that produced the best overall performance. So, I would choose a set of K-Means based GMMs with a normalized ML classifier. Based on the constraints explored, unless data was extremely abundant I would use the first 10 MFCCs and a tied covariance matrix for each GMM.

Conclusions: Closing Thoughts

Considering the somewhat limited number of observations in the dataset and the fact that our methods concatenate the data, removing the sense of time from the spoken utterances, I would say that this classification system performs quite well for spoken digit recognition. With the methods tried here, it achieves an overall accuracy of almost 90%. I think the system is good in that it is easy to implement algorithmically; while not handling the time series when building the models may hurt robustness, it increases ease of use and reduces model size. I think one way to improve the system would be to implement a sense of grouping across time during K-Means and EM.

One modeling choice I wish I could have explored and would have with more time is the effect of speaker gender on the models. Next time, I definitely would have collaborated more with classmates, as while I am proud of my independent work, I wish I could have compared more results to mutually improve our experiences. I think my personal approach to coding, however, worked well, and I felt that my approach in terms of the classes' and functions' utilities was effective.

References

Dollar, P., Singh, M., & Girshick, R. (2021). Fast and accurate model scaling. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr46437.2021.00098

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

(image) Fortmann-Roe, S. (2012, June). *Understanding the Bias-Variance Tradeoff*. Retrieved December 12, 2022, from http://scott.fortmann-roe.com/docs/BiasVariance.html

Hammami, N., & Bedda, M. (2010). Improved tree model for Arabic speech recognition. *2010 3rd International Conference on Computer Science and Information Technology*. https://doi.org/10.1109/iccsit.2010.5563892

Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy. *Nature* 585, 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2

J. D. Hunter, "Matplotlib: A 2D Graphics Environment," in Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55.

P. S. Sisodia, V. Tiwari and A. Kumar, "Analysis of Supervised Maximum Likelihood Classification for remote sensing image," International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014), 2014, pp. 1-4, doi: 10.1109/ICRAIE.2014.6909319.

(image) Sahnoun, Y. (2019, October 25). *Arabic numbers: How to count in Arabic*. ArabicPod101.com Blog. Retrieved December 12, 2022, from https://www.arabicpod101.com/blog/2019/10/24/arabic-numbers/

(image) Sayad, S. (n.d.). *K-Means Clustering*. An Introduction to Data Science. Retrieved December 12, 2022, from https://www.saedsayad.com/clustering_kmeans.htm

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 881-892, July 2002, doi: 10.1109/TPAMI.2002.1017616.

T. K. Moon, "The expectation-maximization algorithm," in IEEE Signal Processing Magazine, vol. 13, no. 6, pp. 47-60, Nov. 1996, doi: 10.1109/79.543975.

Collaboration

I worked on my project report and code entirely independently.