Classification of Arabic Spoken Digits with Gaussian Mixture Modeling

Nolan Gelinas

Abstract

While the field of machine learning (ML) has been around for over two decades, the past year of ML development has caused an explosion in the popularity of artificial intelligence (AI). ChatGPT has already revolutionized many industries, from software development to creative work in just over a year of being released to the public. However, this milestone achievement couldn't have been done without the years of research on more basic forms of ML that specialize in classification and optimization. The foundational concepts taken from clustering algorithms, statistical modeling, and neural networks are all orchestrated together to create the modern AI platform ChatGPT. While they may seem like sorcery to some, generative predictive models can simply be described as 'probabilistic parrots', and the concepts I learned during this project can be applied to these models at a much smaller scale and scope.

This project follows the standard machine learning workflow of train, test, improve hyperparameters, and repeat. It uses statistical modeling to train a set of 10 Gaussian mixture models, 1 for each spoken Arabic digit, then uses these models to find the most likely value of an unseen spoken digit. While the scope of this project is limited to the classification of spoken Arabic digits, the applications of this workflow are nearly limitless; almost any type of data, from image to sound to text, can be represented as a multidimensional set of numbers. A similar form of statistical modeling and likelihood classification can be applied to group a wide range of datasets, from social media profiles to detecting songs or videos that violate copyright.

Important Definitions

- MFCC: Mel-frequency cepstral coefficient
- Phoneme: A singular sound in a word. Distinct from a syllable, which is often comprised of multiple phonemes.
- Gaussian Mixture Model (GMM): A set of **K** **d**-dimensional clusters, represented by probability, mean, and covariance.
- **K**: The number of clusters in a GMM. Also, the number of phonemes in the spoken digit.
- **d**: Number of dimensions. In this dataset, d is equal to the number of MFCCs used to train the model.
- Likelihood: The probability of observing data given a fixed statistical model
- Covariance (Σ): Describes how random variables move in relation to each other. In this project, covariance is represented as a d-by-d matrix where the i^{th} diagonal entry is the variance of the i^{th} dimension

Problem Statement

The primary goal of this project is to train a Gaussian mixture model to identify spoken Arabic digits. We are given a dataset that has separate training and testing data, which we use to create and measure the performance of our model, respectively. The words are distinguished by our model through the different phonemes present in the spoken digit. Based on the way the audio data is digitized into a set of MFCCs, the different sounds of each phoneme should create a distinct clustering in 13-dimensional space. We can take advantage of this by creating clusters that approximate the shape of each group of MFCCs. Doing this allows each digit to be represented by k sets of 3 variables, the clusters weight, center, and covariance. This means that unseen data can be probabilistically classified by calculating the likelihood of the new data belonging to each digit's GMM. The highest likelihood should yield the most likely digit that was spoken.

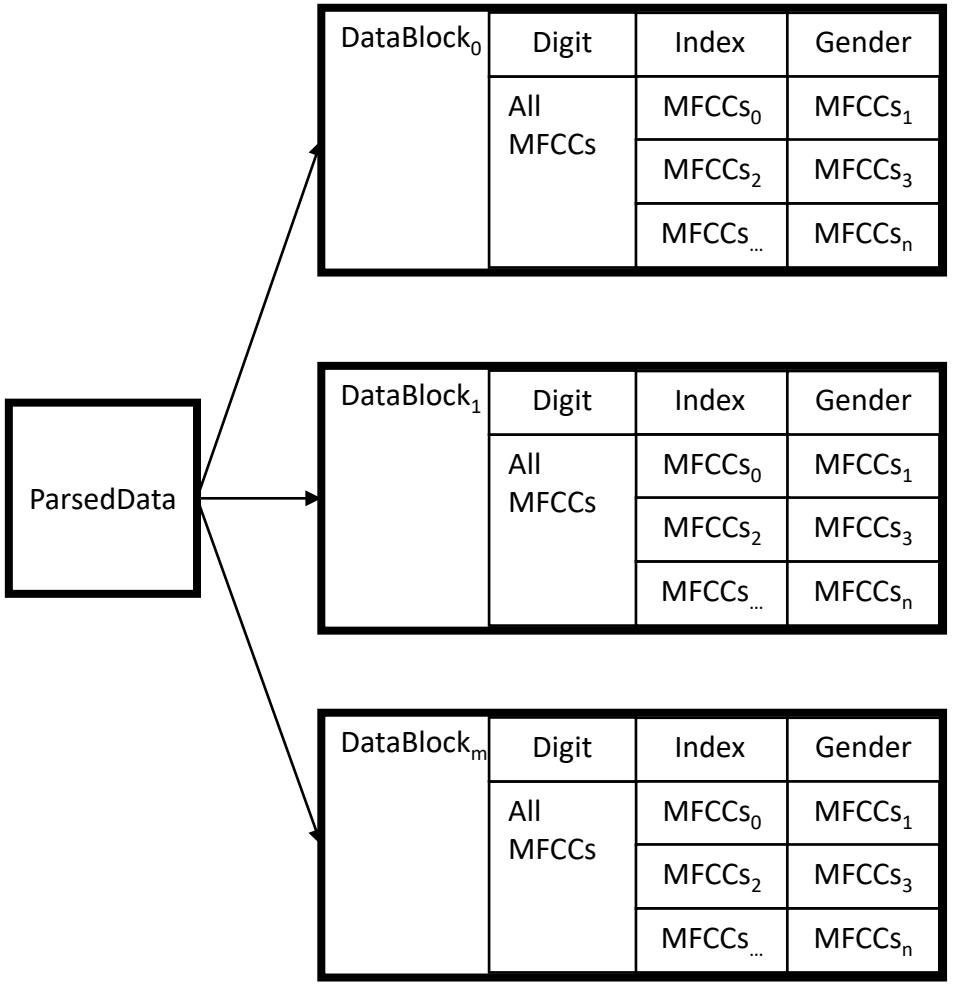
Phonetic Arabic words for each digit

0: Sifir
1: Wahad
2: Ithnayn
3: Thalatha
4: Araba'a
5: Khamsa
6: Sittah
7: Seb'a
8: Thamanieh
9: Tis'ah

The Dataset

The data used to train my model is extrapolated from the Spoken Arabic Digit dataset, which is comprised of 44 male and 44 female speakers. Each spoken digit, or utterance, has 25-40 speech frames, which are snapshots of one's speech. Each frame contains 13 Mel-frequency Cepstral coefficients (MFCCs), which are coefficients representing the amount of each frequency recorded during that snapshot. Every speaker says every digit 10 separate times to provide more data variability for our model. 6600 utterances are assigned to the training set, while the remaining 2200 utterances are reserved for testing the model.

The metadata for the dataset is not directly encoded in the dataset. Instead, each line represents the 13 MFCCs from the current frame, each of which are separated by a whitespace line. The metadata can be extrapolated by keeping track of how many blocks have occurred and alternating between male and female speakers. After 10 of these per gender, the digit is incremented. Due to the relative complexity of this metadata structure, I created a custom parser which follows this pattern and stores the data in a DataBlock object, which is simply a container for storing the MFCCs and the corresponding utterance metadata. Storing this metadata alongside the MFCCs makes it easier to filter, sort, and classify the data, which is useful when analyzing subsets of the data.

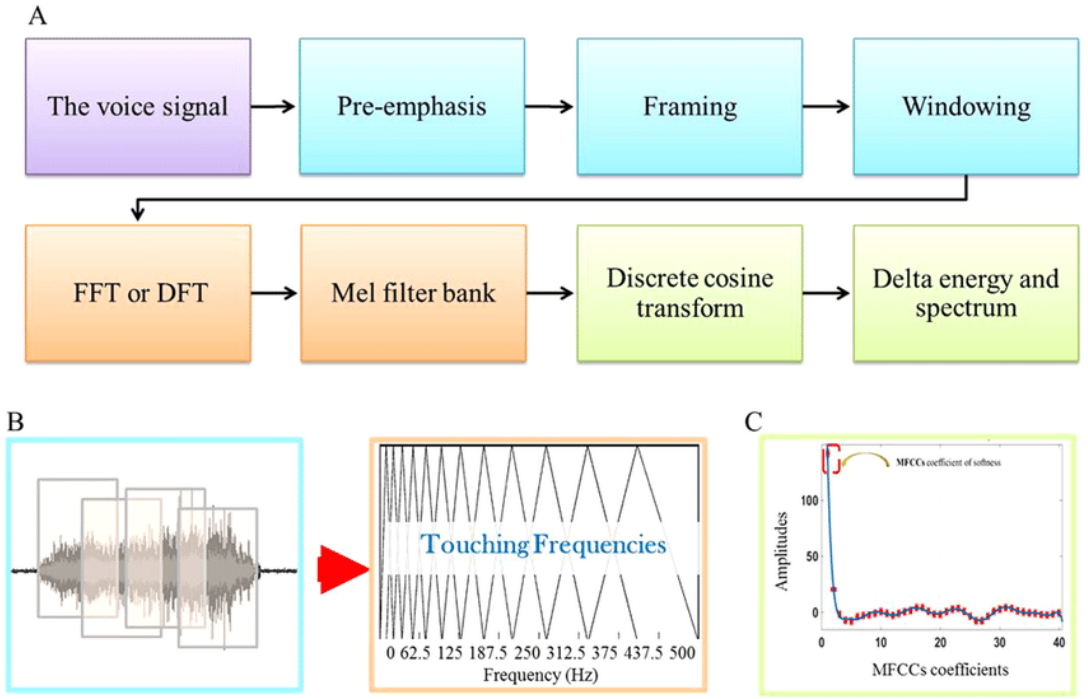


The ParsedData object holds all *m* utterances and contains functions to filter the data by gender, index, and digit. Each DataBlock holds the metadata for each utterance as well as *n* MFCC lists, each of size 13 or fewer, depending on the indices used to train. The DataBlock object can filter by MFCC indices to make the filtering process easier.

What is an MFCC?

Mel-frequency Cepstral Coefficients are often used in speech recognition applications because they're specifically designed to represent frequencies that are most common in human speech. To extrapolate an MFCC from an audio file, many mathematically operations are performed on the input data. First, the audio split up into discrete windows, usually 20-40ms in length. Then, the Fourier Transform is applied to each window to convert it from the time domain to the frequency domain. Next, converted to the Mel scale, which is short for melody scale because it is supposed to be representative of human hearing. Afterwards, the logarithm of the data is taken, and another Fourier Transform is applied to get the final coefficients.

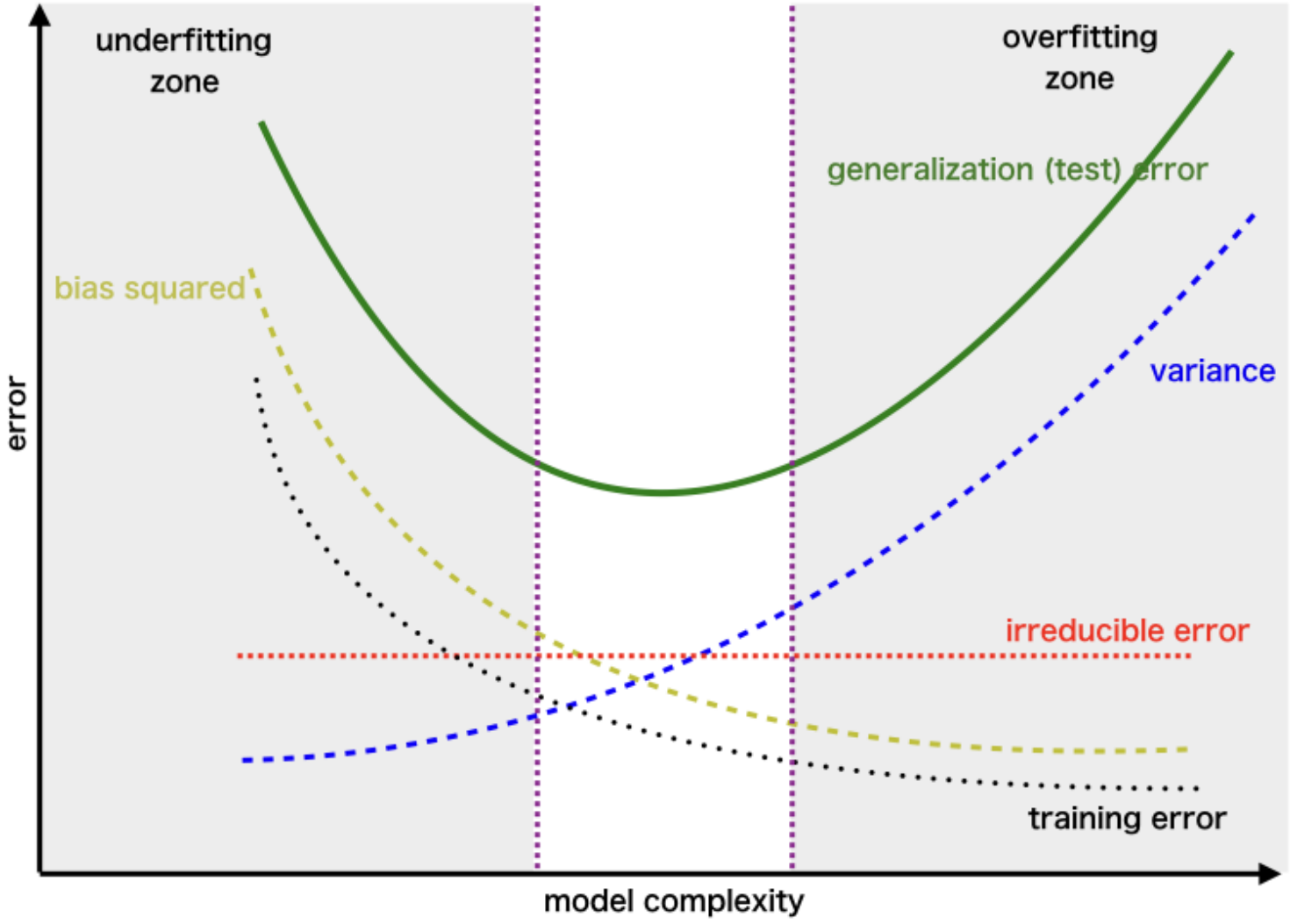
The MFCCs of an audio signal capture the essential spectral characteristics of an audio signal and are effective for representing the features of speech. They're often used in speech recognition scenarios because they capture distinct phonemes of the input data. While MFCCs can be used to recognize the order of the phonemes in advanced applications, our model does not take the order of the phonemes into account.



Credit to Abdenaceur Abdouni, R. Vargiolu, and Hassan Zahouani

Bias-Variance Tradeoff

The bias-variance tradeoff explains the issues with underfitting and overfitting. When bias is high, the model doesn't fit the training data well and we see high error on both the training and testing set. When variance is high, the model is fitting the training data so well that testing on unseen data has high error because it isn't the exact same shape as the training data. The ideal models strikes a balance between high bias and high variance to achieve the lowest error on new data.



Credit to GeeksForGeeks

The Model

To properly fit the Gaussian mixture models (GMMs) to the training data, many considerations need to be made about the hyperparameters passed into the model. A hyperparameter is a parameter that dictates how a model is trained. Besides acquiring more data or using a different modeling approach, tuning hyperparameters are one of the only ways to tune a model and improve performance. For my model, I used 5 hyperparameters.

1. Which MFCC indices to include
2. K-Means or Expectation Maximization for clustering
3. Covariance calculation types, which are spherical, diagonal, or full
4. Tied or distinct covariances
5. Mapping each digit to the number of clusters expected

Once one GMM is trained for each digit, it is represented as **k** sets of the following 3 variables: π, the relative strength of the $\mathbf{k}^{\mathbf{th}}$ Gaussian Model (GM), μ, the mean (or center) of the GM, and Σ, the covariance matrix of the GM. I then use these 3 variables to recreate the GM with a multivariate normal PDF, which is explained further in the maximum likelihood classification section.

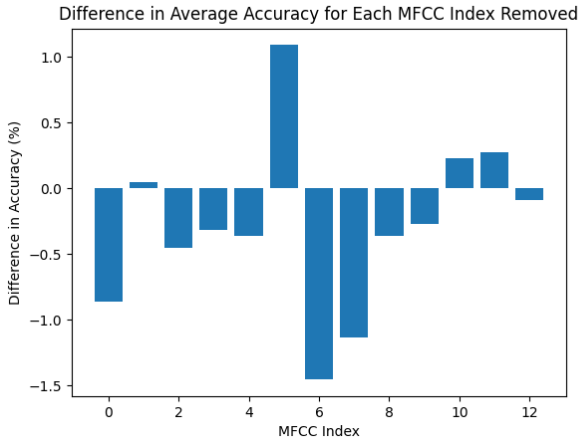
Hyperparameters

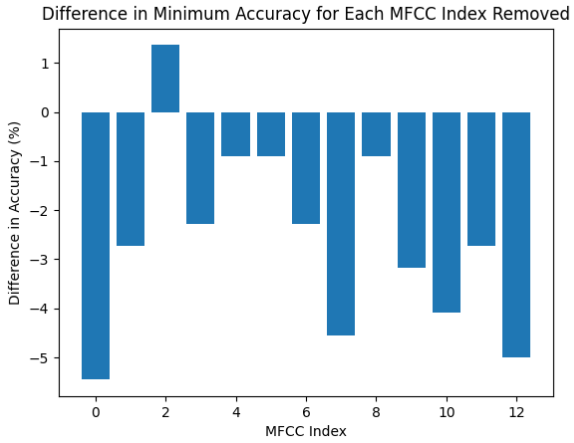
Hyperparameter 1: MFCC Indices

To choose the best MFCC indices to keep in my data, I used a leave-1-out algorithm that compares my average classification accuracy against the new average accuracy after removing each of the MFCC indices. Using all 13 MFCCs yielded an 88.91% average accuracy score. After 1 iteration of my greedy algorithm, I saw over a 1% increase by withholding index 5 for an average accuracy of 90.0%.

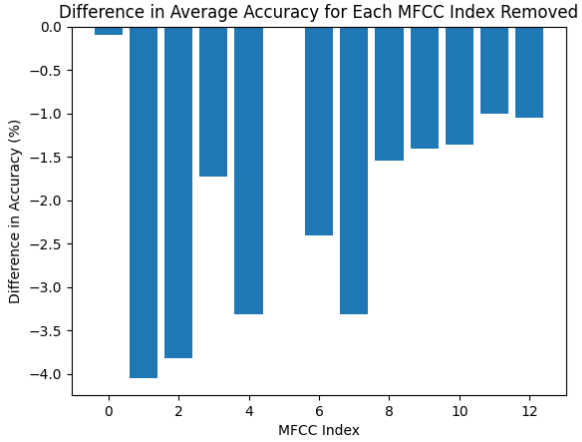
Unfortunately, after continuing this algorithm without MFCC index 5, I saw a decrease in average accuracy after removing any other index. I tried removing index 0 because it has the smallest loss in average accuracy to see if the algorithm could still feasibly continue iteration but saw further decreases in the average accuracy. However, this is still an impactful discovery because it marginally improves the speed of generating a confusion matrix due to less dimensions to fit and provided an increase in modeling accuracy.

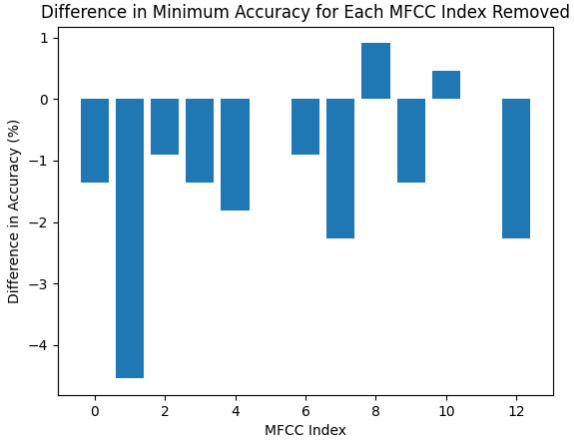
Including all MFCC indices:





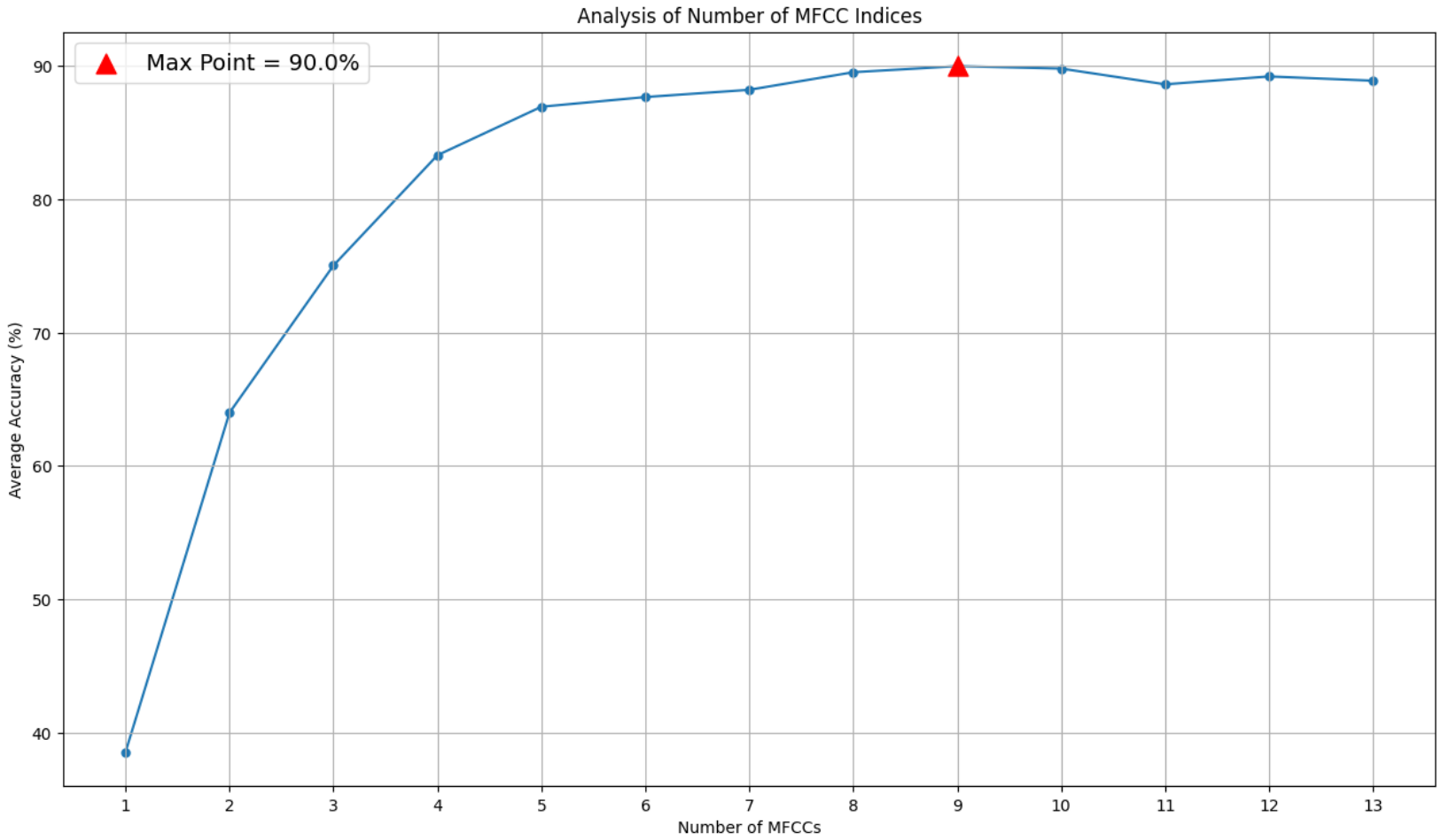
After Removing MFCC index 5:





Continuing to Explore MFCC Indices

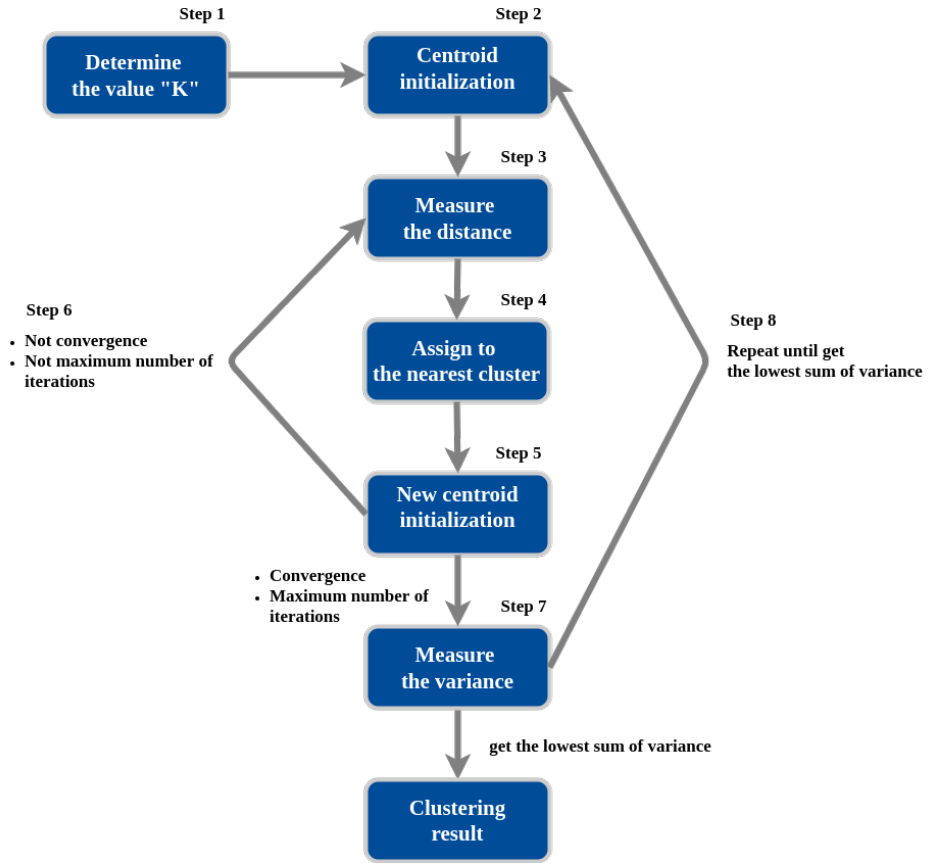
After attempting a leave-1-out algorithm starting with all MFCCs, I decided to try the opposite approach with a greedy algorithm. Instead of starting with all MFCC indices and removing one, I start with none of the indices and try adding each MFCC index by itself. I keep track of the index that produces the best accuracy, then permanently add that index to my list. I continue this process of testing an additional index in my MFCC list and adding the index with the highest accuracy. This method yields slightly better results, because instead of the leave-1-out algorithm that works backwards and quickly gets stuck, the greedy approach always finds another index to add until all MFCC indices are used. We can see from the results that the best number of MFCC indices is 9, which compared to the 12 indices from the leave-1-out algorithm is an improvement in training and testing speed for no loss in accuracy.



K-Means Algorithm

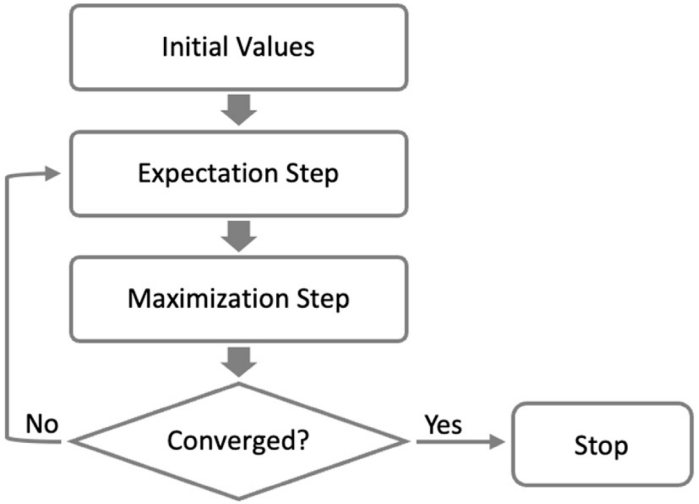
K-Means is one of the most basic clustering algorithms. It follows 4 basic steps: first, **k** initial centroids are chosen, which are simply points in **d**-dimensional space. Next, for each centroid, the distance between that centroid and every point is calculated. Then each point is assigned to the centroid that is nearest to that point. Finally, new centroids are calculated by taking the average value of all points belonging to the $\mathbf{k}^{th}$ cluster. Steps 2-4 are repeated until the centroids stay in the same location between iterations, which is known as convergence. This algorithm is useful because it is easy to implement, it is fast to run compared to other clustering algorithms, and it is easy to visualize and monitor the progress of the algorithm via subdimensions and heuristics.

There are some pitfalls of K-Means, primarily that it is highly sensitive to the initially chosen centroids. It also assumes that all clusters are hyperspheres, which is often not the case for real world data. K-Means is also very sensitive to outliers in the data, because these large distances will skew the newly calculated centroids in step 4. Finally, K-Means can only produce hard clusters, which means that each point is assigned to only 1 centroid at a time. This can cause issues with clusters that are highly overlapping.



Visualization of the Scikit-Learn K-Means algorithm, which is used in this project.
The additional variance minimization steps ensure the best clustering by trying many different initial centroids.
Credit: Arif R

Expectation Maximization Algorithm



Flowchart of Expectation Maximization
Credit: Jose Silva, Paulo Vaz, Pedro Martins, and Luis Ferreira

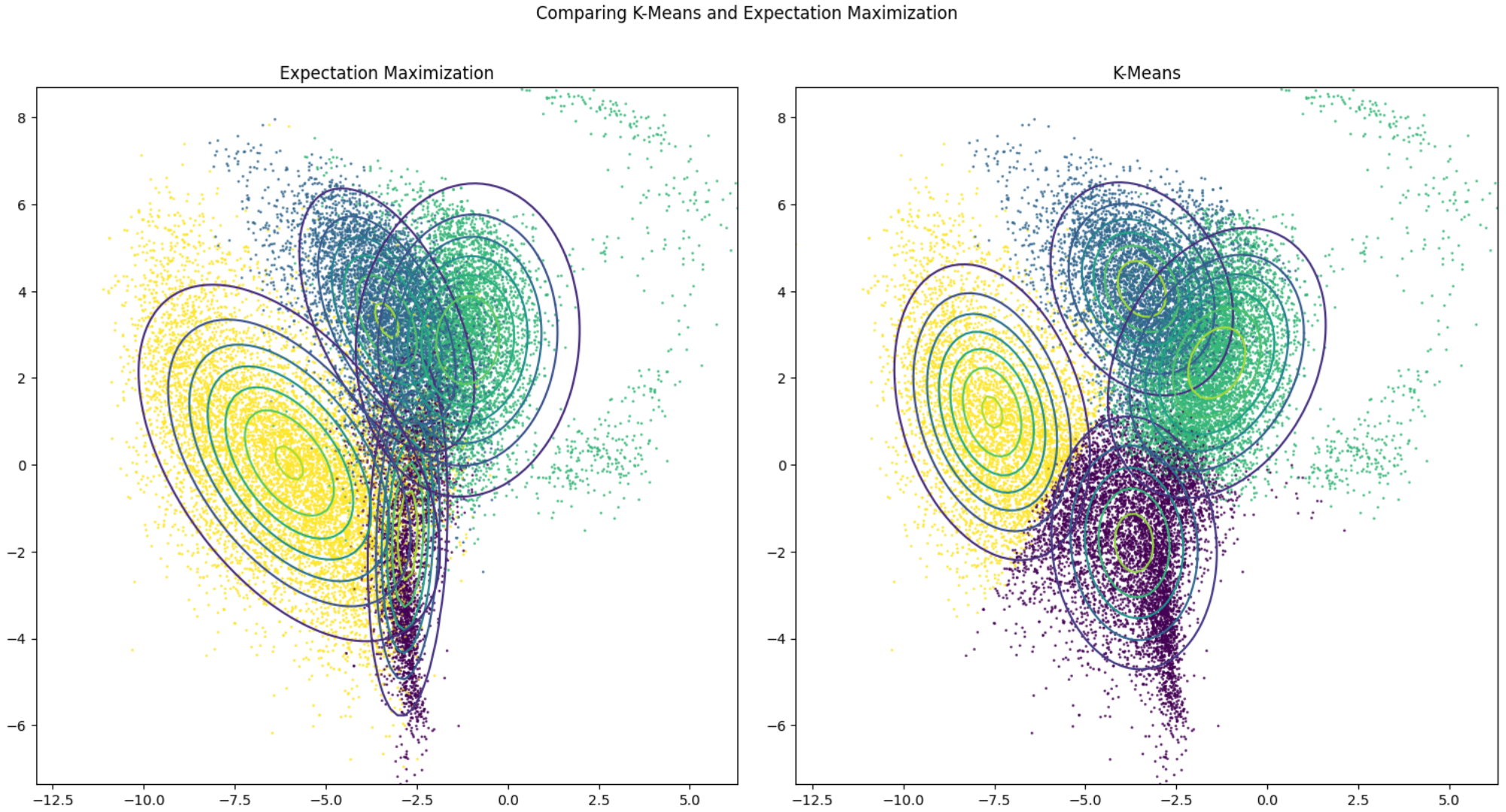
$$N_k = \sum_{n=1}^{N} r_{n,k} \qquad \pi_k = \frac{N_k}{N} \qquad r_{n,k} = \frac{\pi_k \eta(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{k} \pi_j \eta(x_n|\mu_j, \Sigma_j)}$$

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{n,k} x_n \qquad \Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{n,k}(x_n - \mu_k)(x_n - \mu_k)^T$$

The Expectation Maximization (EM) algorithm is another type of clustering algorithm that uses statistical analysis to cluster data. Like K-Means, the first step of this algorithm is to choose k initial centroids. Then, perform the E-step, which means calculate $r_{n,k}$ for each combination of **n** (data point) and **K** (cluster). This value represents the probability that x_n was drawn from the **k**$^{\textbf{th}}$ mixture component. Next, run the M-step by calculating N_k, the number of observations in each mixture component, μ_k, the mean of the mixture component, Σ_k, the covariance of each mixture component, and π_k, the probability of each mixture component. These values yield **K** new mixture components, which we use to repeat the E-step. The E and M steps are repeated until convergence, which means that the new mixture components are equal to the current ones.

This algorithm is still sensitive to the initially chosen centroids, but it can model almost any shaped cluster because during each iteration it considers the covariance of the cluster. It can also be tuned much more than K-Means because covariance constraints can be applied to the M step, which can make the model more rigid or more flexible depending on the requirements of the dataset.

Visualizing K-Means versus EM



We can see that EM more accurately creates clusters in this scenario because it can consider the shape of the clusters while it runs. It also yields better looking clusters because instead of the hard edges seen in K-Means, which is because K-Means only looks at the distance from a centroid to each point, while EM classifies points based on the highest probability of belonging to a cluster.

Hyperparameter 2: K-Means or EM

Deciding between K-Means and Expectation maximization was a difficult process, as some of my testing showed that K-Means performed better than expectation maximization for certain digits. However, after attempting to choose between K-Means and EM on a per-digit basis, I found that this significantly reduced overall model accuracy. While I can't be sure what exactly caused this, I believe it is due to some GMMs overpowering others, making the model classify a larger portion of the data as a certain digit than it otherwise would have. I found this to be the case when attempting to tune the covariance constraints on a per-digit basis as well, so I decided to only tune the number of clusters, **k**, for each digit and apply the rest of the hyperparameters to all digits equally.

After trying to tune hyperparameters based on each digit, I tested both K-Means and EM using all possible covariance constraints and found that distinct full covariance using the EM algorithm provided the best fit for my testing data. This is expected, because of the high dimensionality and variance in my input data. Since each person has somewhat different tones in their voice and some people say words differently than others, it is unlikely for the data to be spread along the axes in the 13-dimensional space. Therefore, it makes sense that EM performs better, because it takes into consideration the shape of the clusters as it is running the algorithm, as opposed to calculating the covariance matrix after the algorithm has completed.

Hyperparameters 3 and 4: Covariance Constraints

The 3rd and 4th hyperparameters are closely related, as they both apply different types of constraints on the covariance calculations. There are three different covariance types: spherical, diagonal, and full. These names describe the structure of the covariance matrix as well as the resultant level sets. Spherical covariance constraints create a hyperspherical level set when visualized and cannot accurately model clusters that are highly elliptical in any direction. Diagonal covariance constraints can stretch along each axis to better conform to clusters that are not hyperspheres, but if the data is oblique to all axis in **d** dimensions, diagonal will not be able to accurately conform to the data. Finally, full covariance constraints can conform to data stretched along any axis or any linear combination of the axes, giving it the most modeling flexibility.

Tied covariance constraints mean that, for each of the **k** clusters, the covariance is averaged together so that each resulting covariance matrix and level set is the same shape and size. Distinct covariance, on the other hand, allows each of the **K** clusters to conform as closely as possible to its subset of the data, meaning that each resulting covariance matrix and level set will be unique. Tied covariance leads to a more rigid model, which distinct covariance allows more flexibility.

$$\begin{bmatrix} \sigma^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma^2 \end{bmatrix}$$

Spherical

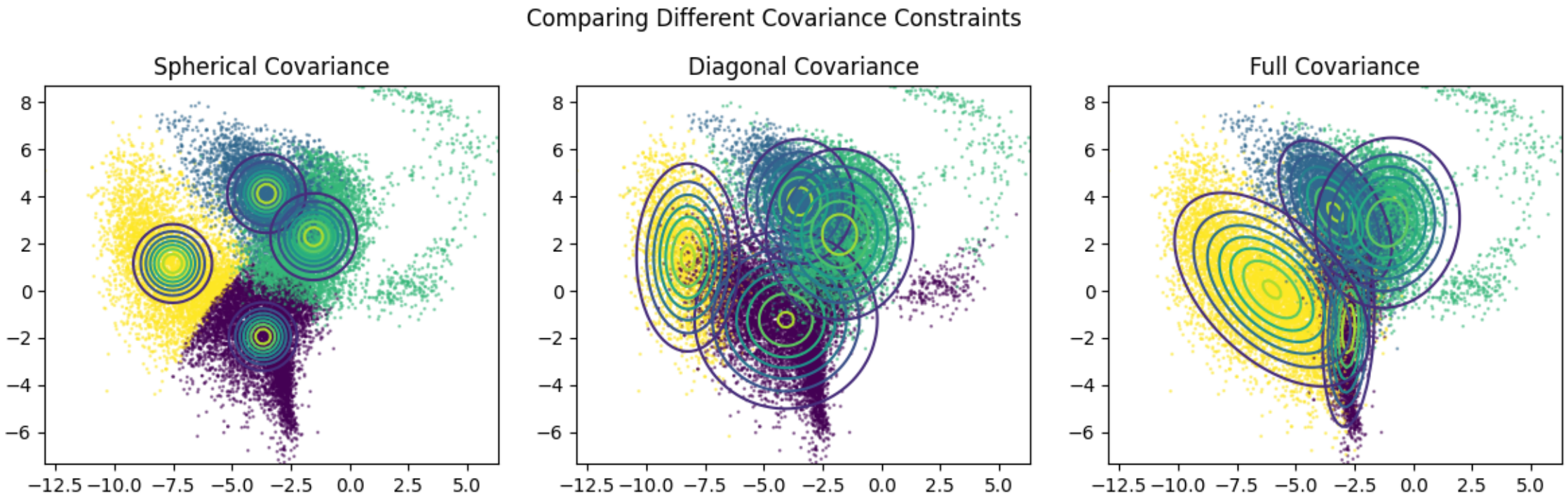
$$\begin{bmatrix} \sigma_0^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_d^2 \end{bmatrix}$$

Diagonal

$$\begin{bmatrix} \sigma_{00}^2 & \cdots & \sigma_{0d}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{d0}^2 & \cdots & \sigma_{dd}^2 \end{bmatrix}$$

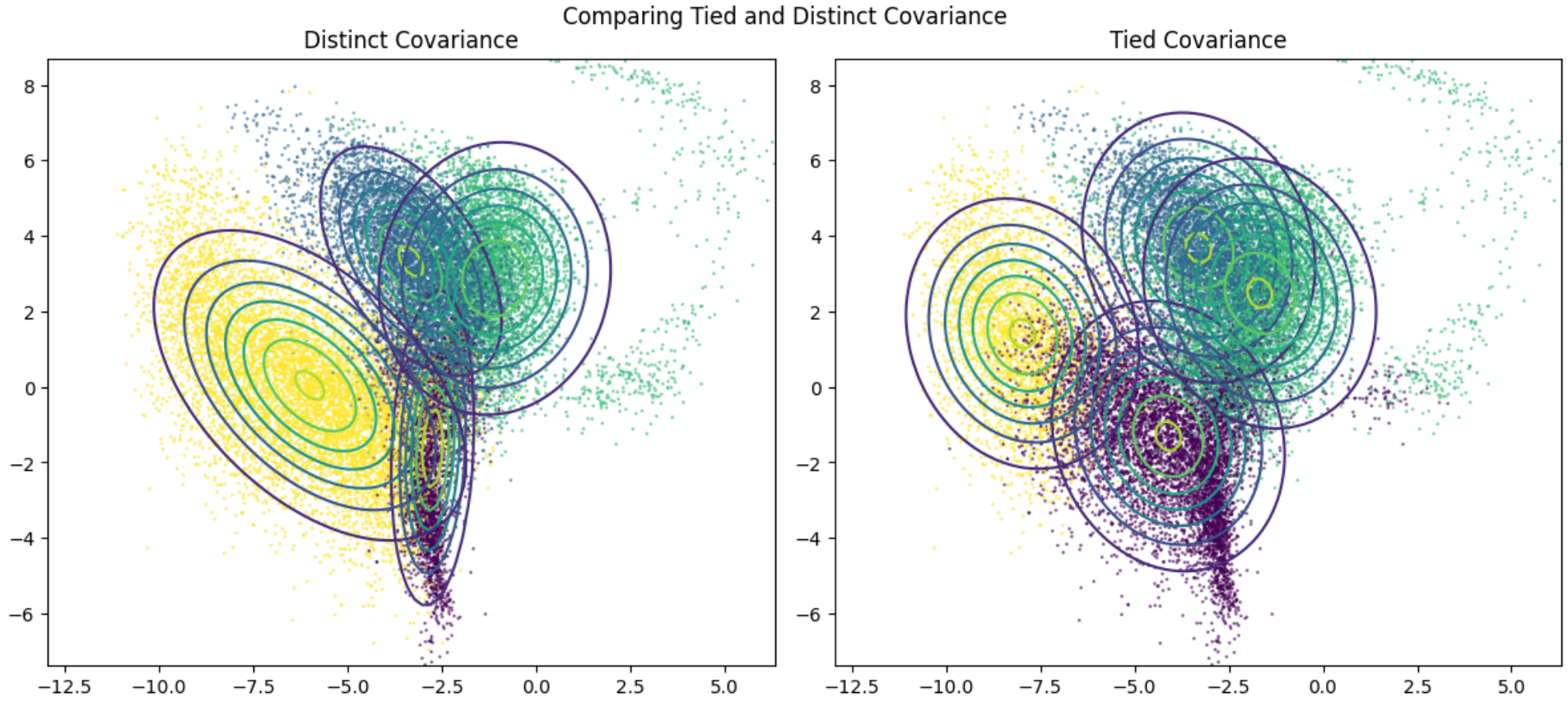
Full

Visualizing Spherical, Diagonal, and Full

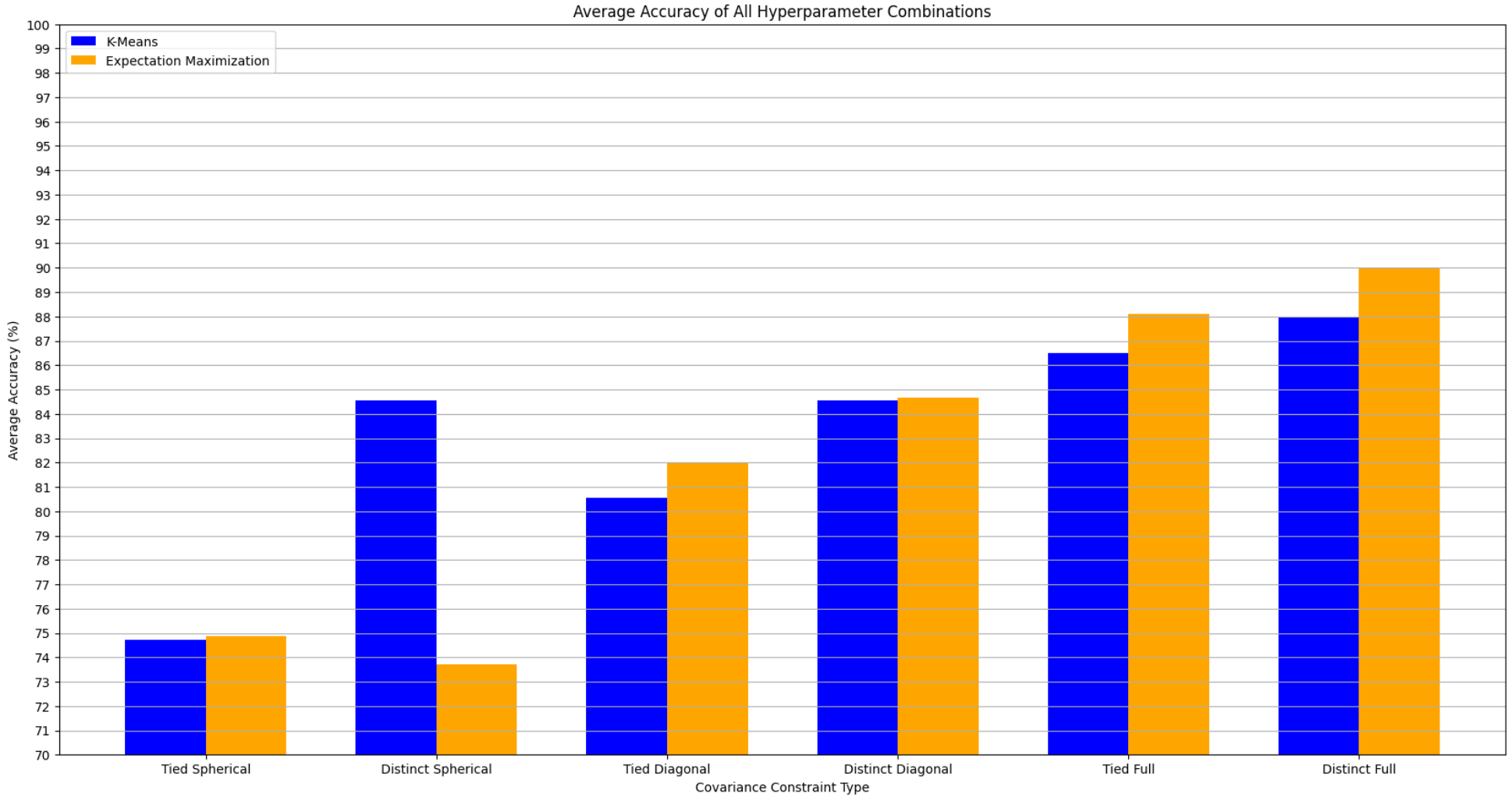


Each of these plots use EM on the same data. Looking at the resulting level sets, we can see that spherical performs very similarly to K-Means, with hard edges that don't account for cluster shape. Diagonal is better because it can fit covariance along the axes of the data, but full provides the most flexible clustering. The yellow cluster in the full scatter plot shows how full covariance can model data that isn't along an axis, which allows for more shapes to be accurately modeled.

Visualizing Tied Versus Distinct



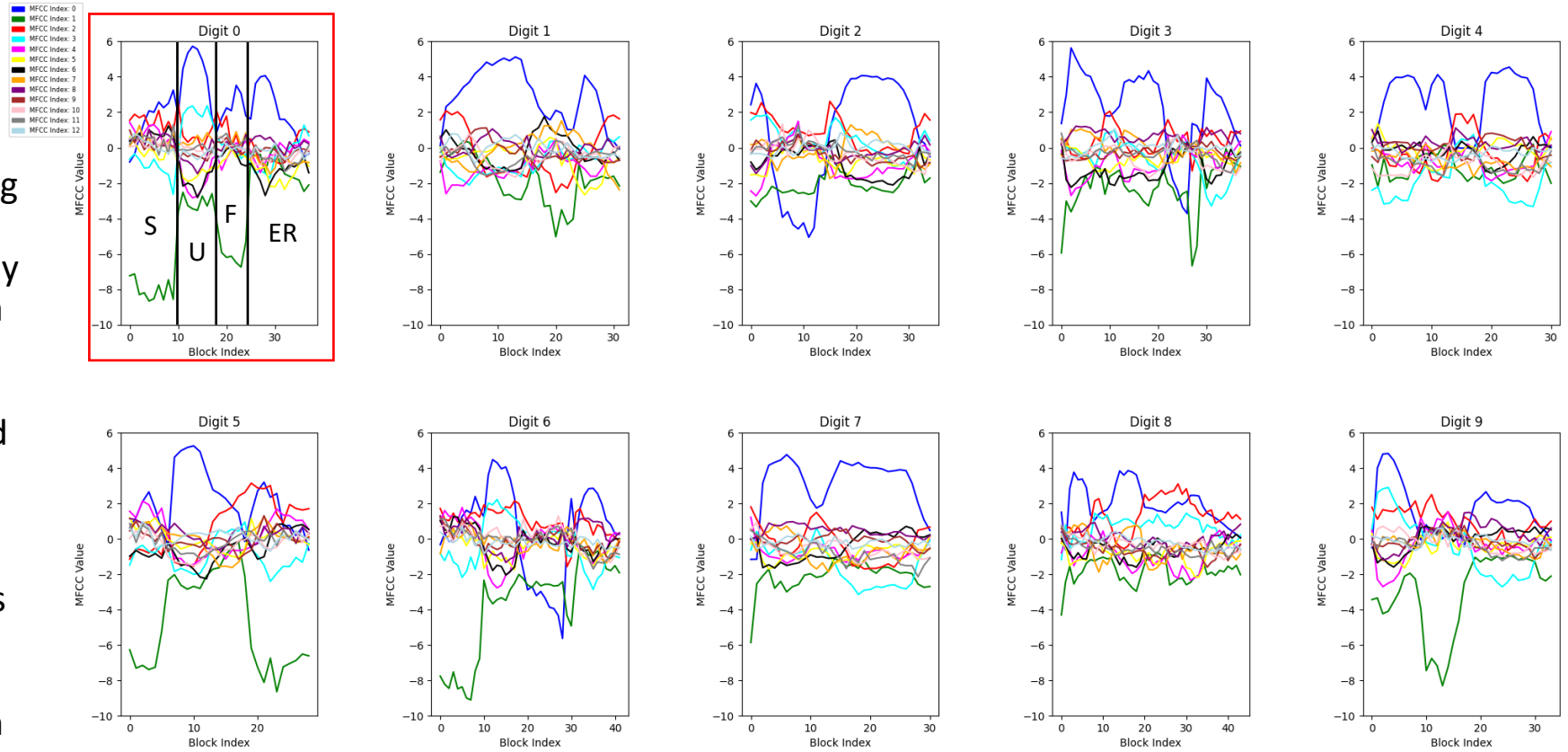
These plots show the differences in tied and distinct covariance. Each set of clusters are calculated using EM with full covariance to best illustrate the difference between tied and distinct. We can tell that distinct yields higher modeling flexibility because it can conform exactly to the shape of each cluster, while tied takes the average shape of all the clusters and combines them. In some cases, tied can give better test results because distinct can overfit the data.



This diagram shows my attempt to tune hyperparameters 2, 3, and 4. Distinct full using EM yields the highest accuracy using all MFCCs. This is likely because the input data has very complex but repeatable shapes, so a very flexible model is required to conform to the intricacies that come from the MFCCs. Distinct performs better than its tied counterpart, which is likely because the clusters coming from the MFCCs are all very differently shaped, so modeling each cluster separately yields better results. An outlier in this bar chart is distinct spherical with K-Means, which performs better than distinct diagonal and much better than EM distinct spherical. I'm not exactly sure why this is, but a potential explanation is that K-Means already assumes a spherical covariance while running the algorithms, so creating a spherical covariance matrix to classify with yields the closest model of how the algorithm determined its clusters.

Hyperparameter 5: K Mappings

Arguably the most important hyperparameters to choose are the **K** values for each digit. This value **K** decides how many clusters the clustering algorithm will attempt to find. For this project, the number of clusters is directly related to the number of phonemes in a word, since we are classifying each digit based on the phonemes alone. To get baseline **K** values for each digit, I plotted all 13 MFCCs over the digit's entire block. The number of times that the graph shows a clear shift should correspond to the number of phonemes present in the spoken digit. The plot of digit 0, outlined in red, shows an example of the approximate locations in the block index that the MFCCs show a noticeable shift. The resulting partitions correspond to the 4 phonemes of digit 0.



All MFCCs plotted for every block for each digit. For this example, the first instance of each digit is plotted.

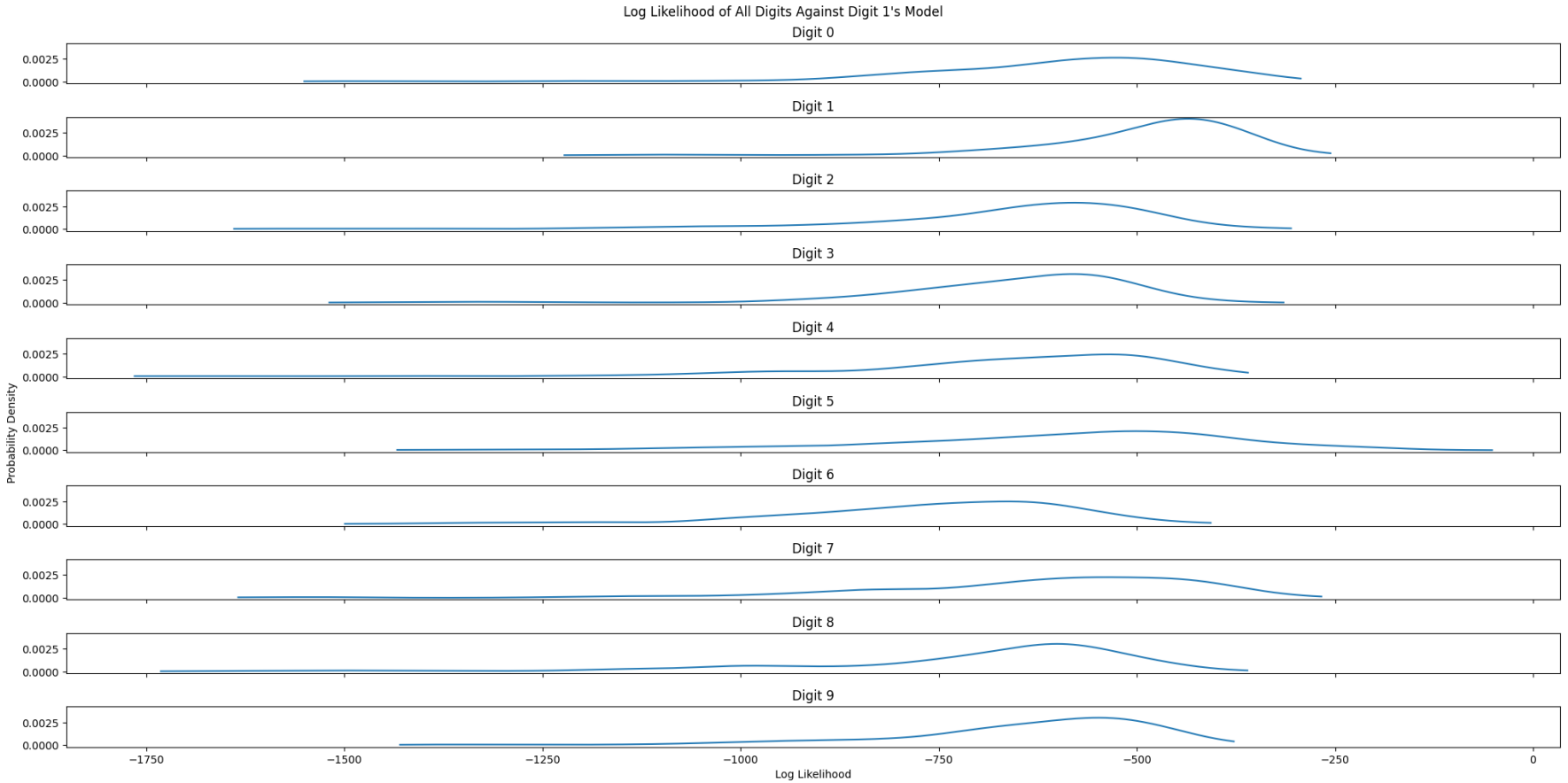
Maximum Likelihood Classification

Once a GMM has been created for each digit, the model is complete. All that is left to do is test the model's performance using the provided testing data. To utilize the GMMs for classification, one can apply maximum likelihood classification (MLC) with the following formula, which is calculated for each utterance on every digit's model. Using MLC is natural for this type of classification problem because the its equation uses the same parameters that are returned from each GMM. The digit with the highest likelihood corresponds to the classification of the tested utterance.

$$p(X|\Delta_d, \Pi_d) = \prod_{n=1}^{N} \sum_{m=1}^{M} \pi_{m,d} p(x_n|\Delta_{m,d})$$

X is the testing data, represented by a collection of **N** MFCCs
N is the number of frames, which ranges from 25-50 for each utterance
M is the number of GMM components (K value or number of phonemes)
d is the current digit represented by the GMM
$\boldsymbol{\pi_{m,d}}$ is the influence of the $\mathbf{m^{th}}$ GMM for digit **d**
$\boldsymbol{\Delta_{m,d}}$ is the $\mathbf{m^{th}}$ mean and covariance for digit **d**

Maximum Likelihood Classification Visualized



This figure shows how maximum likelihood classification works. Each subplot represents the log likelihood vs probability density. The algorithm takes the input digit, in this case 1, and creates a GMM based on the MFCCs of the input. It then calculates the log likelihood of each of the 10 digits to the input digit's model. The peak on each subplot shows the most probable log likelihood estimate of the input digit belonging to that set of clusters. To classify the digit, the algorithm simply returns the digit with the peak that has the highest log likelihood. In this instance, digit 1 would yield the highest log likelihood at about -450, therefore the algorithm would correctly classify the input digit as a 1. This visualization is the opposite of how the classifier for the project works; instead of fitting each digit to a single model, the real classifier takes a single, unseen digit and fits it to the 10 trained models. The principal of calculating the most likely digit remains the same, where the peak furthest to the right is the output of the classifier.

Training and Classification Speed

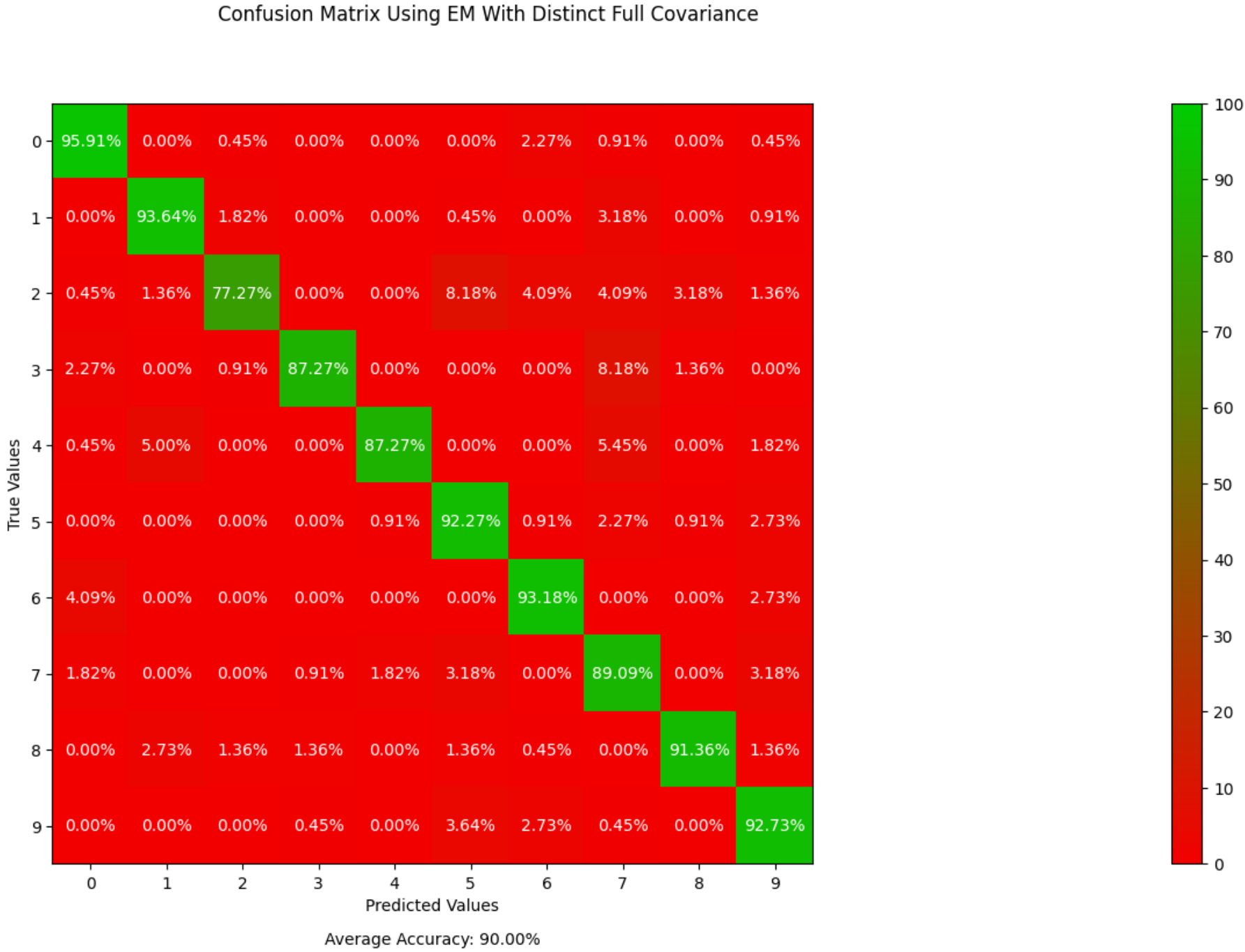
The speed of both training and testing the model proved to be very important throughout this project because I had to rerun my algorithm hundreds of times with different hyperparameters to increase my model accuracy. Fortunately, I learned this early on, and implementing timing calculations throughout my code so I can easily find bottlenecks and attempt to fix them. This paid dividends as I tested my model, because I found multiple bugs with my K-Means covariance calculations that I otherwise wouldn't have noticed. One specific bug I remember fixing was with my distinct spherical covariance calculation for K-Means, which took over 15 seconds per digit during training. I was able to quickly fix this bug and reduced the per-digit training time to less than 0.25 seconds.

The biggest impact to the speed of my algorithm was reducing the number of MFCCs. This is because the MFCC indices represent the dimensionality of the data, and lower dimensionality almost always leads to faster clustering calculations and maximum likelihood estimations. I also made minor improvements in the classification speed of my algorithm by replacing the Scikit-Learn multivariate normal PDF function with a custom PDF calculation that precomputes the log-likelihood of the covariance determinant, so it didn't have to be recalculated each time. This improved the average speed of my classification algorithm from about 2 seconds per row to about 1.25 seconds per row, depending on the covariance constraints and number of MFCCs.

Results

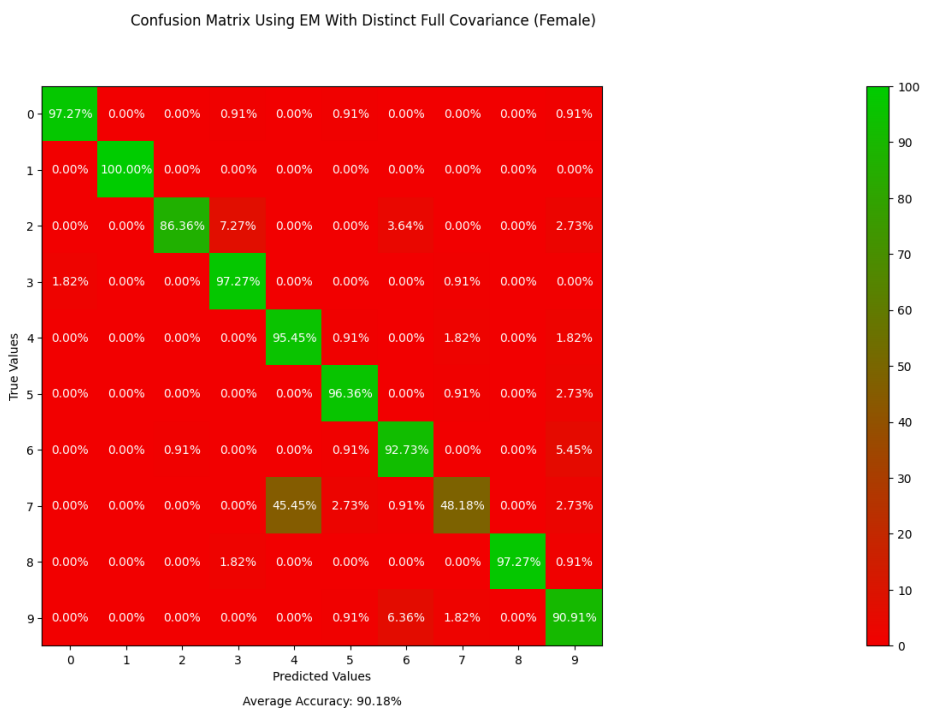
Confusion Matrix

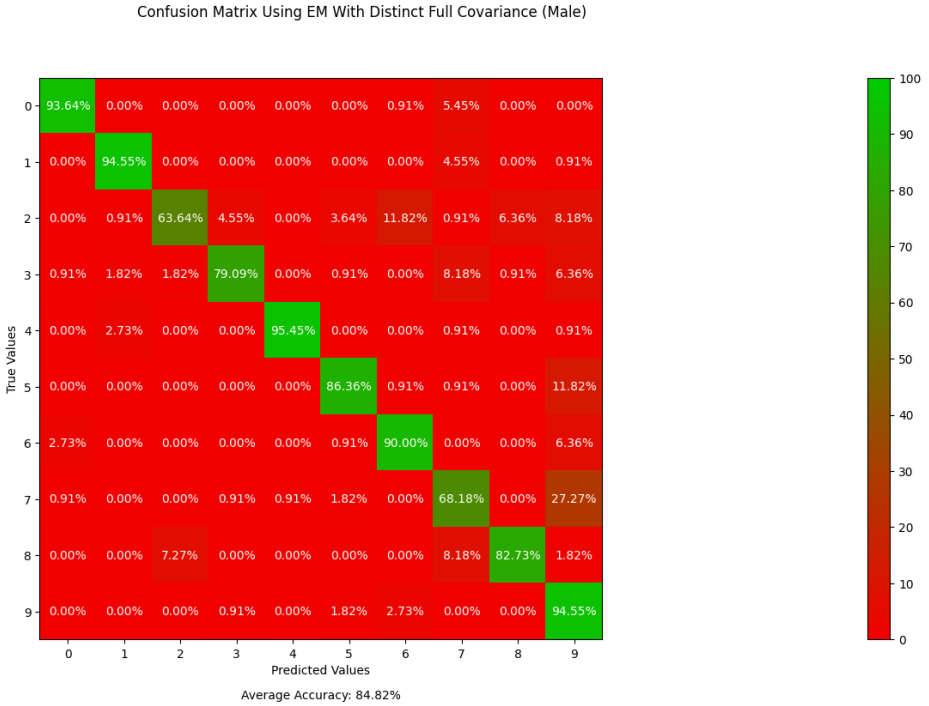
The confusion matrix is used to show classification accuracies of each digit. Every row shows the results of running the maximum likelihood classification of the testing data. Ideally, the identity diagonal would be all 100%, showing a perfect classification. This confusion matrix is using MFCC indices [1, 2, 4, 5, 6, 7, 8, 10, 12] (zero-indexed). We can see that digit 2 struggles to classify correctly and is often confused with 5. Digit 7 also causes confusion for many of the other digits, which likely means the model for 7 is somewhat underfit.



Exploring Gender Separation

Until now, the gender of the speaker was not considered in my model. After tuning all my hyperparameters to achieve the highest average accuracy including both genders, I decided to try modeling each gender separately to see if it improved my modeling. Classifying each gender separately somewhat improved the accuracy of my model because men and women produce different frequencies from their vocal cords and therefore will have distinct MFCC patterns when speaking the same digits. Separating genders allows the model to more closely conform to each genders frequencies instead of trying to generalize enough to accurately fit both. From the resulting confusion matrices, we can see that digit 7 causes a lot of trouble for both genders but is especially problematic with women. The model likely needs to be adjusted further for each gender to improve the separated accuracy.

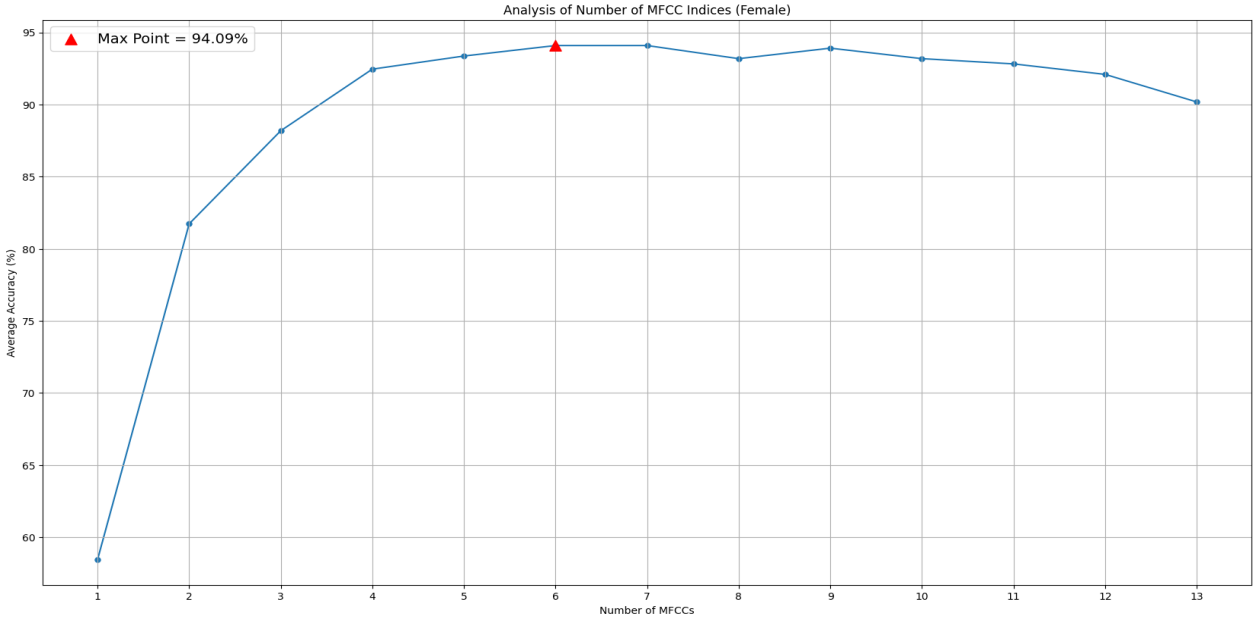


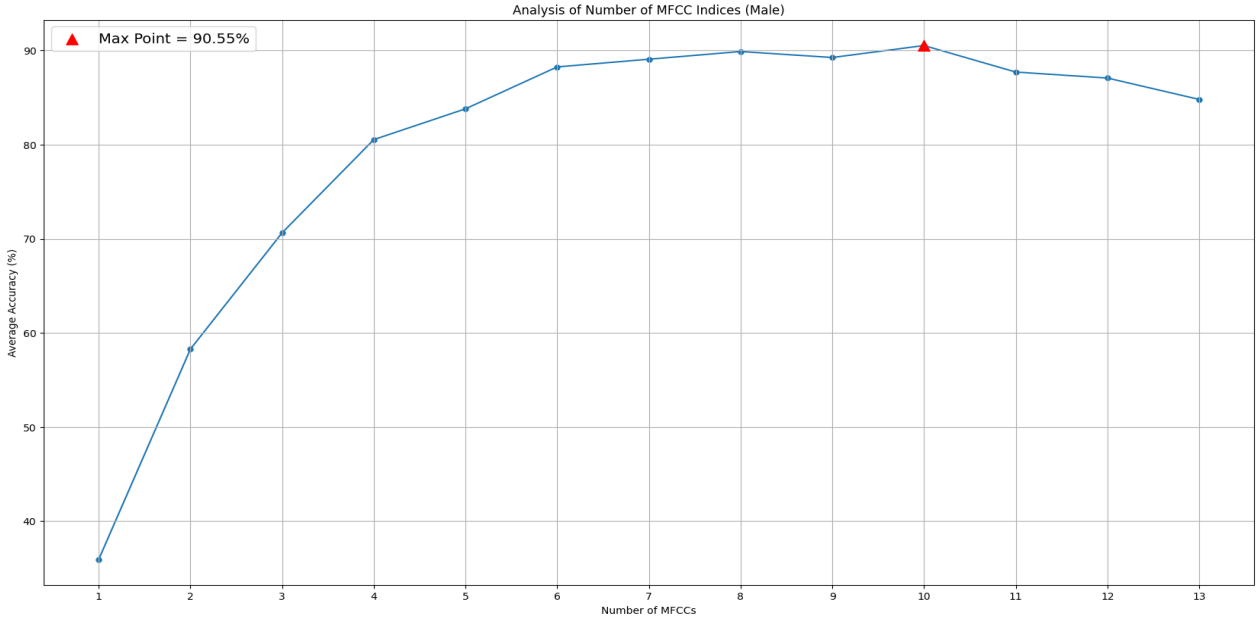


Recalculating MFCC Indices By Gender

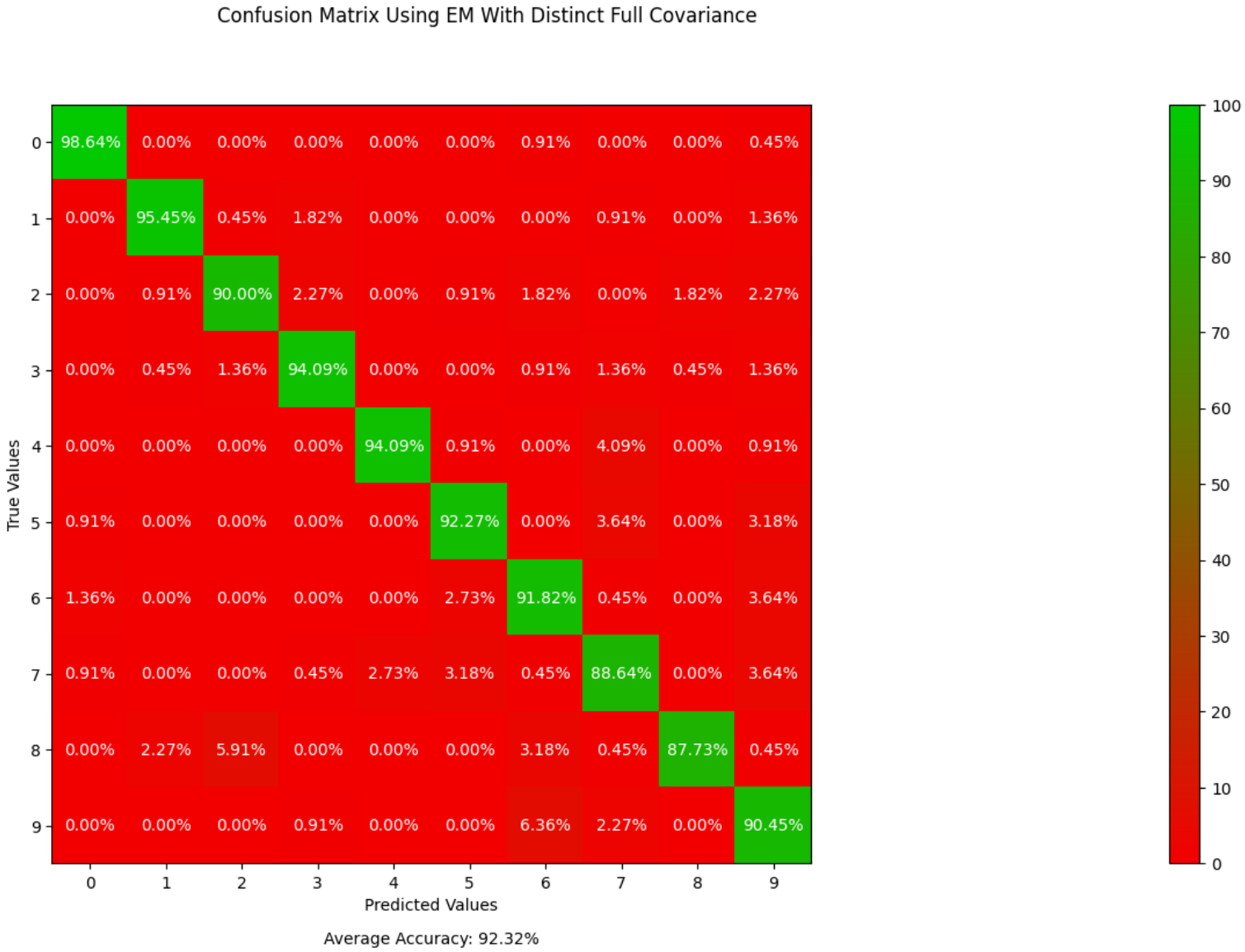
After discovering that the average accuracy of my classifier can be improved by separating the speaker's gender, I tried rerunning my greedy MFCC algorithm to see if further accuracy improvements could be achieved. To my surprise, after running the greedy algorithm once for each gender, I saw a significant improvement in average accuracy. This was especially apparent for female speakers, whose accuracy increased from 90.18% to over 94%! This is likely due to the tone of female speech vs male speech and leads me to believe that the women are more consistent than the men when pronouncing each digit.

With this information, I decided to create 2 separate models and test each utterance on the model corresponding to the speaker's gender.





Resulting Gender-Separated Confusion Matrix



Hyperparameter Conclusions

After completing this project, I've learned a lot about the difficulties of working with high-dimensional data, tuning hyperparameters to improve performance, and trade-offs that need to be considered when changing the hyperparameter values or structure. Analyzing data with greater than 3 dimensions proved difficult because it cannot directly be visualized. Visualizing sub-dimensions of the data can provide valuable insights into the overall structure of the data, which helps determine the most suitable initial hyperparameters for the model. Unfortunately, once dimensionality becomes too high, this qualitative approach is only feasible for choosing a ballpark starting value. For tuning the hyperparameters, informed trial and error seems to be the best way to improve the model accuracy. This is because changing 1 hyperparameter, even when it only applies to a single digit like the **K** value, can greatly impact the performance of other digit classifications.

I started tuning my hyperparameters by selecting initial **K** values. I did this by looking at the number of distinct shifts in the MFCC plots over an entire block for every digit, then tuned each digit's value individually by choosing the worst performing digit in the confusion matrix, incrementing or decrementing **K**, and checking the results. This approach was easy to implement and not particularly time consuming for the scope of this project, but likely wouldn't be possible given more classification options, such as 0 through 99 instead of 0 through 9. I found that the **K** value of each digit affected my model the most, likely because using an incorrect number of phonemes to model a word forces the clustering algorithms to create clusters that span across multiple 'real' clusters or model multiple clusters inside a single 'real' cluster. I continued this tuning process for every digit until I saw a decrease in accuracy by changing any of the **K** parameters, which to me meant that I found the correct number of phonemes for each word. Since this hyperparameter had such a drastic effect on my accuracies, I kept them the same after finding the apparent optimal values.

After tuning my **K** values, I tested all 12 combinations of clustering algorithms and covariance constraints and found that distinct full using Expectation Maximization gave the best average accuracy of just under 90%. The covariance constraints had a noticeable impact on the average performance, but in most cases, EM and K-Means performed similarly with EM doing better on average.

Finally, with the other hyperparameters set, I began tuning the MFCC indices. At first, I thought reducing the number of MFCCs used would reduce modeling accuracy because there is less data to train the model with, but I was completely wrong about this. In every case, fewer dimensions yielded a more accurate model, which makes me think that my model was overfitting when using all 13 dimensions.

Final Model and Reflection

My final system uses distinct full covariance constraints with the EM clustering algorithm. I classify digits using the log likelihood calculated from the maximum likelihood estimation formula. When creating the model, I split up my training data by gender and train 2 separate models with their own tuned MFCC indices. Then, when running the tests, I check the gender of the utterance and classify the digit using the corresponding model. While this method makes the training phase take twice as long and double the space, it improves classification speed and accuracy substantially, from below 90% to over 92.3%. While this increase may not seem like much, I've learned through this project that the last 10% of accuracy takes 90% of the effort, so an additional 2% increase in accuracy was very encouraging.

Given more time, I would have tried to improve digit 7, as it had the lowest accuracy in my final confusion matrix, and I had issues with this digit throughout the project. One possible solution for this issue could be to make the model for 7 more rigid to reduce the range of data classified as a 7. Something I would do differently next time is to think broadly about the problem before completing the check-ins, so I had less refactoring to do when compiling my individual functions from the check-ins. I did not write reusable or well-documented code for each check-in, which forced me to spend more time fixing and understanding it when incorporating the code into the rest of my system.

Digit	0	1	2	3	4	5	6	7	8	9
K	4	3	3	4	3	4	4	4	4	5

MFCC Indices											
	Male	0	2	3	4	6	7	8	9	10	12
	Female	1	2	4	6	7	10				

References

- **K-Means**
  - Bock, HH. (2007). Clustering Methods: A History of *k*-Means Algorithms. In: Brito, P., Cucumel, G., Bertrand, P., de Carvalho, F. (eds) Selected Contributions in Data Analysis and Classification. Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Berlin, Heidelberg.
  - Jin, X., Han, J. (2011). K-Means Clustering. In: Sammut, C., Webb, G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA.
- **Expectation Maximization**
  - Dellaert, F. (2002). The Expectation Maximization Algorithm.
- **Maximum Likelihood Classification**
  - John, S., Varghese, A.O. Analysis of support vector machine and maximum likelihood classifiers in land cover classification using Sentinel-2 images. Proc.Indian Natl. Sci. Acad. 88, 213–227 (2022).
- **Bias-Variance Tradeoff**
  - Geman, S., Geman, D.L., & Jordan, M.I. (1992). Learning from Data: A Short Course. Addison-Wesley.
  - Neal, B., Mittal, S., Baratin, A., Tantia, V., Scicluna, M., Lacoste-Julien, S., & Mitliagkas, I. (2018). A Modern Take on the Bias-Variance Tradeoff in Neural Networks. ArXiv.
- **Spoken Arabic Digit**
  - Bedda,Mouldi and Hammami,Nacereddine. (2010). Spoken Arabic Digit. UCI Machine Learning Repository.
- **Matplotlib**
  - Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95.
- **Numpy**
  - Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., & Oliphant, T.E. (2020). Array programming with NumPy. Nature, 585, 357-362.
- **Scikit-learn**
  - Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
- **Scipy**
  - Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., & SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

Collaborators

I worked with the following people to create this project and slide doc. We compared our code, hyperparameters, model performance, and slide doc structure. I found it very helpful to have friends who could look at my code when I encountered problems or to give me pointers on testing hyperparameter combinations when I didn't achieve the accuracy that I wanted. We also shared ideas on how to further improve performance, such as the greedy MFCC algorithm for picking which MFCC indices to include, which vastly increased my average accuracy.

Ashley Hong

Jerry Worthy