

Course Project: Recognizing Spoken Digits in Arabic

Jake Vigeant

December 12, 2022

Problem Description

Project Goals

The project aims to identify the Arabic digits 0-9 as represented by a dataset of Mel-frequency Cepstral Coefficients (MFCCs) generated from spoken recordings

Each Arabic digit has its own set of unique phonemes that comprise its pronunciation. Thus, examining a data set of MFCCs that showcase the phoneme variances will allow identification to be done smoothly.

Identification will proceed by implementing Gaussian Mixture Models (GMMs) to fit the training dataset and then applying the models to the test dataset. From here, Maximum Likelihood Classification (ML) will be applied to classify each digit. The goal of this analysis is to examine the effects of different modelling choices on the fixed MLE classifier's accuracy.

Use of GMMs will also allow for a comparison between two methods of GMM parameter selection: K-Means clustering and Expectation Maximization.

How to Count in Arabic			
0	zero صفر ṣifr	one واحد wāḥid	1
2	two إثنان 'itnān	three ثلاثة talāṭah	3
4	four أربعة 'arba'ah	five خمسة ḥamsah	5
6	six سبعة sittah	seven سبعة sab'ah	7
8	eight ثمانية tamāniyah	nine تسعة tis'ah	9

Image from (Arabic Numbers: How to Count in Arabic)

Project Importance

Voice/recording classification is a growing technology space with large economic boons for the best solutions.

Determination of the most effective modelling options for MFCC-based classifiers will allow for optimal classification on voice recordings that can reliably be represented in this basis. Efficient voice recognition is an important technological hurdle as humans implement more Internet-Of-Things (IOT) devices into daily life. Buggy voice commands are common when dealing with car voice control or one of the various voice assistant products on the market (Alexa, Siri, etc.). As IOT devices like these become further integrated into daily life, there will be significant competition in the space. Thus, there is a substantial competitive and economic advantage to developing a system that models spoken data optimally to make classification as effective as possible.

MFCC based classifiers have numerous real-world applications in identifying the differences in patterns between two speakers and allowing new speech samples to then be classified by speaker efficiently. MFCC calculation can be used to produce a text-dependent speaker identification model (Ashish). This is a very valuable feature as voice assistant products look to adapt to the voices of their end users.

Interest of Others

The modeling techniques employed in this project are extensible to any data space that can be represented as feature vectors

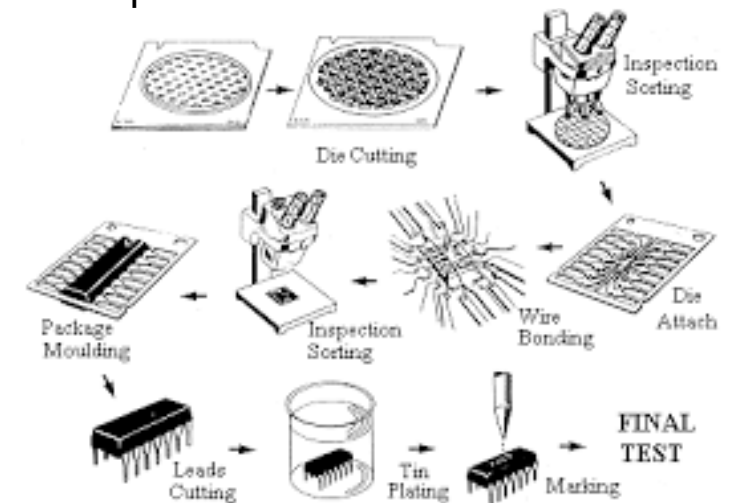
The modeling techniques explored in this project are extensible to many other data sets across various disciplines and pockets of Machine Learning. Aside the idea of MFCCs, any data set that can be represented as a series of feature vectors whose components provide a pattern that defines a classification problem can benefit from understanding the modeling techniques explored in this problem. Although more cutting-edge techniques exist in applying Machine Learning to speech recognition, the underlying process of fitting GMMs to a dataset to identify clusters that adequately fit them is still used in many problems. GMMs may be used to segment customer data into clusters, identify gene expression patterns in biological dataset, and model many other natural events which are hypothesized to occur under the normal distributions. So, examining the process of generating multiple GMM fits based upon different modeling assumptions and then comparing their respective ML classifier performances can provide a general procedure for efficiently approaching these related problems.


Specific Application – Manufacturing Process Monitoring

GMMs are used to monitor manufacturing process data and detect deviations from fitted confidence intervals so that errors can be detected

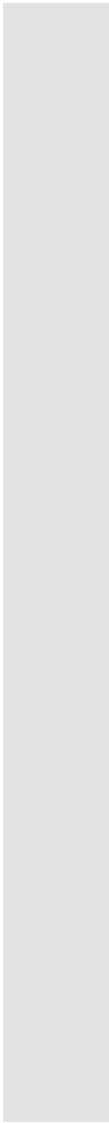
Research has studied applying GMMs to manufacturing process data to generate confidence intervals that can be used to gauge if the process is within its normal range of accuracy or needs to be interrupted and fixed. This is done by fitting a GMM to a dataset that conveys the quality of each manufacturing product and assumes that this is represented by a set of normal distributions. This was applied specifically to a semiconductor production process, and the GMM confidence intervals were found to fit the several quality-level clusters much more effectively than a non-Bayesian approach of representing the data's PDF (Chen), helping identify errors at a significantly higher rate.

This is a valuable application to solve, as streamlining production processes to reduce unusable products yields direct profit for corporations. Further, exploring and contrasting modeling options for fitting GMMs to data will allow clustering performance to be optimized.





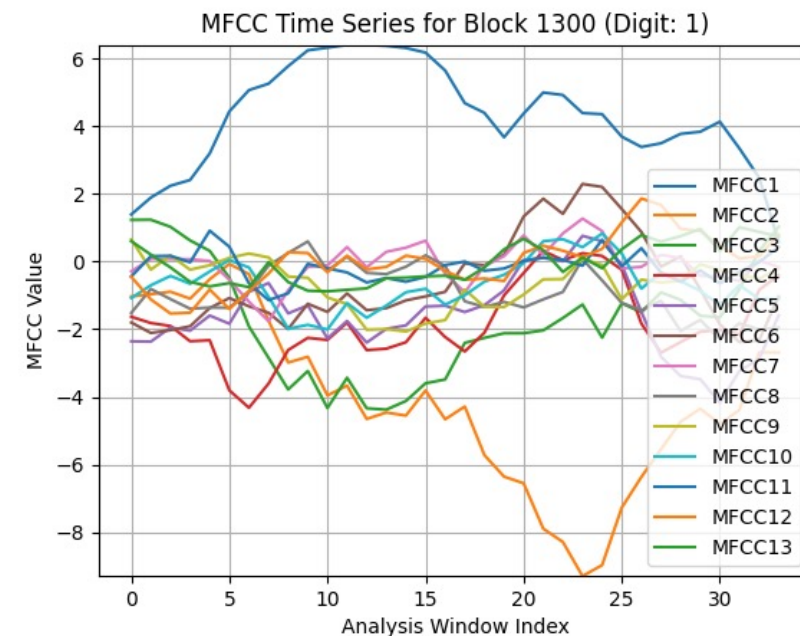
Data (Feature) Modeling



The Dataset

Data Source

This project uses the data contained in the UCI Machine Learning Repository Spoken Arabic Digit Dataset. The dataset is already partitioned into two separate testing and training sets. There is a 75/25 training/testing split. The training set will be used to fit each model, and then classifier performance will be gauged by applying the models to the testing set and identifying the accuracy in output labels to actual data labels. There are 10 digits, 10 repetitions per speaker, and 88 speakers (44 male 44 female). So, the dataset contains 8800 total recorded digits. Each distinct digit utterance is called a block, so there are 8800 total blocks. This means there are also 880 blocks per digit. The blocks are represented as a series of analysis frames. For each analysis frame, there is a vector representing the 13 MFCC values at that instant.



The above plot represents the time-series of each MFCC across 35 analysis frames for a block that represents a recording of digit 1.

Core Modeling Approaches

Defining Fundamental Modeling Tools

The Gaussian Mixture Model (GMM)

GMMs fit a finite number of Gaussian distributions to a dataset to model it

GMMs operate on the assumption that a given dataset can be modeled, although not necessarily was generated from, a set of Gaussian distribution components (GDC). GMMs generate a specified number of Gaussian distributions that model the dataset when combined. The GDCs each have a mean (μ) and covariance matrix (Σ) as shown in the PDF below. The iterative methods of deciding which points correspond to a which mixture component varies and will be focused upon in depth later in this project, but after points are assigned the mean and covariance of each component can be computed. The GMM model is then dependent on the full set of mean, covariances, and weights (w) for each component, represented by λ . The likelihood of a specific data point is found by getting its probability from each component and then performing a weighted sum as shown. This probability can be viewed as the likelihood it came from the entire GMM.

Gaussian Component PDF

$$g(\mathbf{x}|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2} |\Sigma_i|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \mu_i)' \Sigma_i^{-1} (\mathbf{x} - \mu_i) \right\}$$

GMM Likelihood

$$p(\mathbf{x}|\lambda) = \sum_{i=1}^M w_i g(\mathbf{x}|\mu_i, \Sigma_i).$$

Equations taken from Reynold's paper.

K-means Clustering (K- Means)

K-Means clustering is an unsupervised learning technique that assigns data points in a dataset into a specified number (k) of clusters

K-Means is one of the methods the project will employ to generate a GMM. After K-means has been applied, a Gaussian mixture component can be generated from each cluster's mean, covariance, and percent of total data points it holds (weight). K-Means works by selecting a set of k initial cluster centroids and then iteratively recalculating the position of the centroids as the mean of the points corresponding to each cluster until the cluster assignments do not appreciably change between steps or a set number of iterations has concluded. A given point is assigned to its nearest cluster based on which centroid it has the minimal squared distance to. This approach will yield a set of cluster assignments that have worked to minimize the error function below, dependent on the distance between each point and its cluster's center

Error Function

The diagram shows the formula for the objective function J, which is the sum of squared distances between data points and their assigned cluster centroids. The formula is $J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$. Annotations include: 'number of clusters' pointing to k, 'number of cases' pointing to n, 'case i' pointing to $x_i^{(j)}$, 'centroid for cluster j' pointing to c_j , and 'Distance function' pointing to the squared norm $\|x_i^{(j)} - c_j\|^2$. The entire expression is labeled 'objective function'.

$$\text{objective function} \leftarrow J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$$

number of clusters number of cases case i centroid for cluster j

Distance function

K-Means was implemented using the SciKit Learn K-Me package throughout the code base (SciKitLearn).

Image from Chaudhary

Expectation Maximization (EM)

EM is an unsupervised learning technique that generates clusters in a dataset that can then be used to describe a GMM.

As EM is much more algorithmically complex than EM, I will provide a more general description of its methodology and outputs. EM assigns each point to a random Gaussian component at initialization, and then uses the points assigned to each component in order to calculate each component's mean and variance at each step. As this generates a set of Gaussian PDFs the probability of each point belonging to each component can now be calculated, and this is used to reassign the points in successive steps. The likelihood a point was drawn from our model can be represented as the GMM's final state as mentioned on the GMM slide. Thus, the model can iteratively maximize this likelihood across the data set. At conclusion, the data can easily be partitioned into cluster based on the likelihood of each point coming from each mixture component. The components themselves and their weights also provide a GMM that has been fitted to the data. As EM iteratively fits Gaussians to the data clusters, the clusters depend also upon the variance – not just the means as is the case with K-Means.

Fabien article was read to help develop the above understanding of EM in a GMM context.

The SciKit-Learn GMM package implements EM to produce its models, so this was used in the code base (SciKitLearn).

The Classifier

The focus of this project is on modeling choices and their effects on a classifier, not classifier selection itself. Nonetheless, the ML classifier will be used in this project and it is valuable to understand it conceptually.

The Maximum Likelihood (ML) Classifier

The ML Classifier determines the likelihood that a data point was drawn from a set of hypotheses and then selects the hypothesis with the highest likelihood.

In this project, we will create a GMM in various ways from the training set, and then use the model to classify the testing set. The ML classifier will thus calculate the likelihood that a data point represents each of the 10 digits and then apply the label of the digit with the highest likelihood. The likelihood a given MFCC sequence is a specific is found by taking the product of the likelihood of each frame of data belonging to a given digit over the sequence's time. The likelihood of a given frame of 13 MFCCs corresponding to a digit is found by finding the likelihood of that sequence belonging to the GMM for that digit. This equation was presented on the GMM slide. So, taking a product over all the frames of the likelihood of each digit's GMM will yield the likelihood that a block belongs to the digit that is currently being examined. In the equation below, f is the sum over all the mixture components for a target digit, θ . We evaluate this for each digit 0-9.

General Maximum Likelihood Classifier

$$L(\theta) = \prod_{i=1}^n f(x_i; \theta)$$

Equation image from (Maximum Likelihood Estimation).

Why is ML good for Speech Recognition?

The ML classifier that the project implements is agnostic to the underlying model's design choices. So, it is easy to implement once and then only change the input model in order to explore many modeling options efficiently. The classifier is rooted in the intrinsic concept of a likelihood: a measure of confidence in a given hypothesis. So, making use of this is a good fit for a classifier as we should look at an input MFCC sequence and examine the hypothesis that the sequence represents each of the 10 digits. Then, selecting the most likely digit from this makes sense.

Additionally, many academic explorations into speech recognition utilize an ML-based classifier. Continuous speech recognition, a much more complex problem than the one this project addresses, implemented an ML classifier to group multiple spoken words together (Bahl).

Challenges in Implementing ML Classification

The ML Classifier was implemented from scratch.

There was no package I could find that provided a suitable implementation of the ML Classifier for this project. Given the mathematical complexity of the ML Classifier equation, it was difficult to implement from scratch and be confident that it worked as was expected. Selecting a classifier that already existed in SciKit Learn such as an SVM, decision tree, or a CNN would have made application of the classifier simpler, but those more complex classifiers require additional constraints on the data that would have been very hard to check and may have been even more time consuming to run.

The ML classifier also adds in additional summation products as more features are considered. So, if many data features are to be analyzed (many frames, speaker gender, etc.), the classifier's runtime will not scale very well. This runtime issue posed itself during classification.

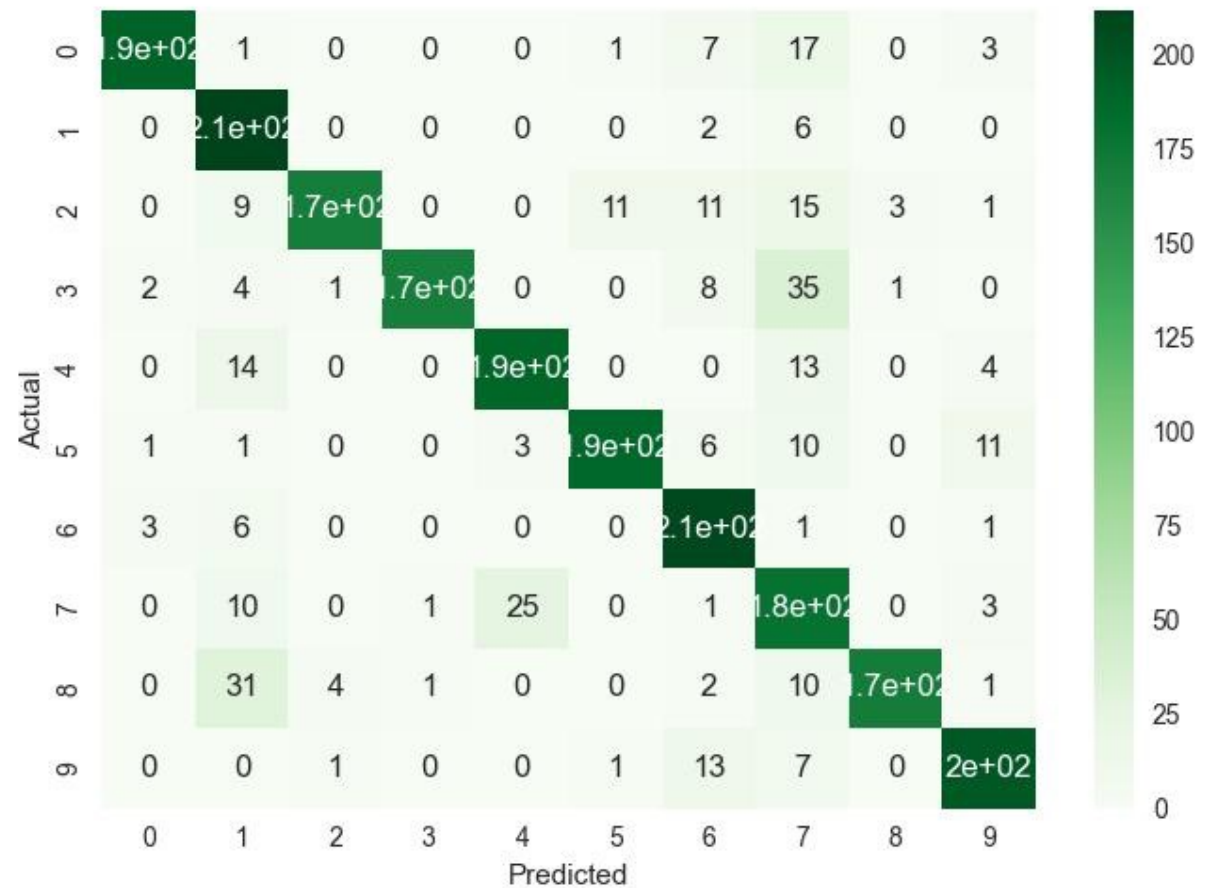
K-Means vs EM Classification Performance

Confusion Matrices

All confusion matrices generated with Matplotlib and Seaborn packages. Numpy also used to shape/process the data prior to plotting

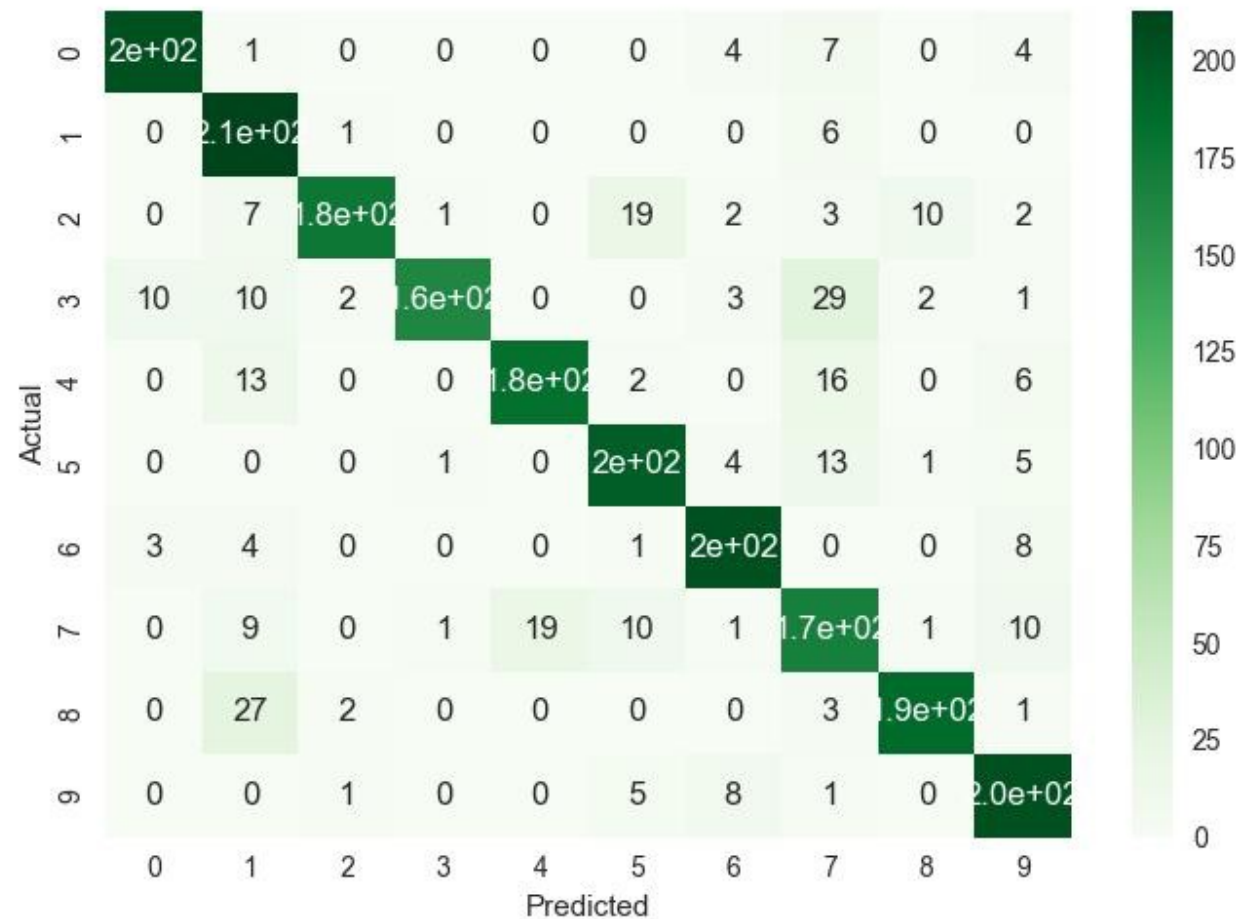
K-Means GMM Classification Results

Total Accuracy: 85.3182 %



EM GMM Classification Results

Total Accuracy: 86.36%



Interpretation

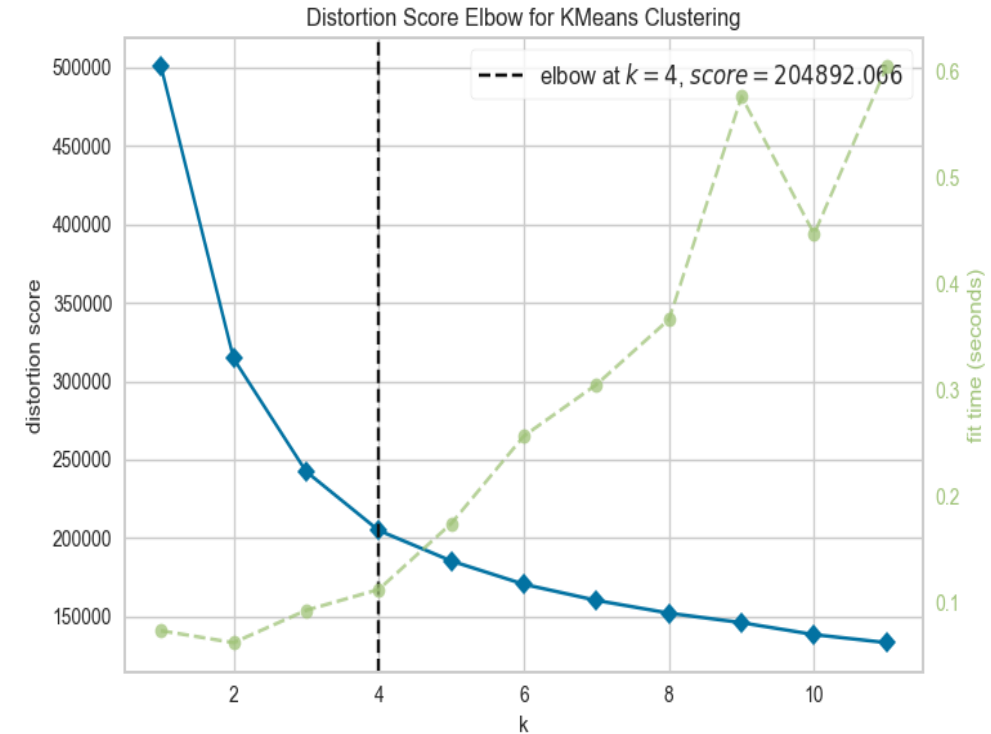
The EM GMM model performed better than K-Means Model. As generating the GMM with EM is a more complex task than running K-Means to get the mixture components, the modeling step took longer to run with EM. However, the majority of the runtime is due to the classification step, so this runtime difference can be neglected here. As can be seen from the accuracies listed on the previous slide, the EM GMM model was able to classify the correct digit 1.04 more percent of the time. This was as expected, as EM considers the variances in each cluster in addition to the means, while K-means can just generate clusters based on the means. For these reasons, the following additional modeling choices were implemented with the EM GMM model as a base.

The digit 1 was the easiest to classify across both models, with an accuracy rate of ~92% in the EM GMM model. This is likely because its phonemes follow a distinct variance structure, so its GMM is most unique compared to the others. The digit 3 seems the hardest to classify, with a lot of mis-labeling as 7 occurring. Interestingly, not nearly as many 7s are labeled as 3s. Digits with similar MFCC variation patterns would be easy to confuse, as they will have GMMs that appear similarly, thus similar likelihood. It is likely that 3 exhibits some features similar to 7 that are particularly strong, and 7 has these same features but also phonemes that are different enough from 3 to diminish the likelihood of 3's GMM.

Additional Modeling Choices

Selecting the proper number of components for each digit in K-means

K-Means requires an input number of clusters to work. So, to generate a GMM from K-Means for each digit's data, one must choose the number of cluster centers desired by the algorithm. This choice was not immediately apparent in looking at the dataset or project specifications, so an elbow plot was used to select a value of K where error has begun to level out for each digit. So, plots like the one to the right were generated for each digit and used to select K values. K elbowed at either 3 or 4 for each of the 10 digits.



Sample elbow Plot generated for the digit 0, suggests $k=4$ is a good choice for the resulting K-Means attempt.

A Scikit sub-package called Yellow Brick was used to generate the elbow plots (Bengfort).

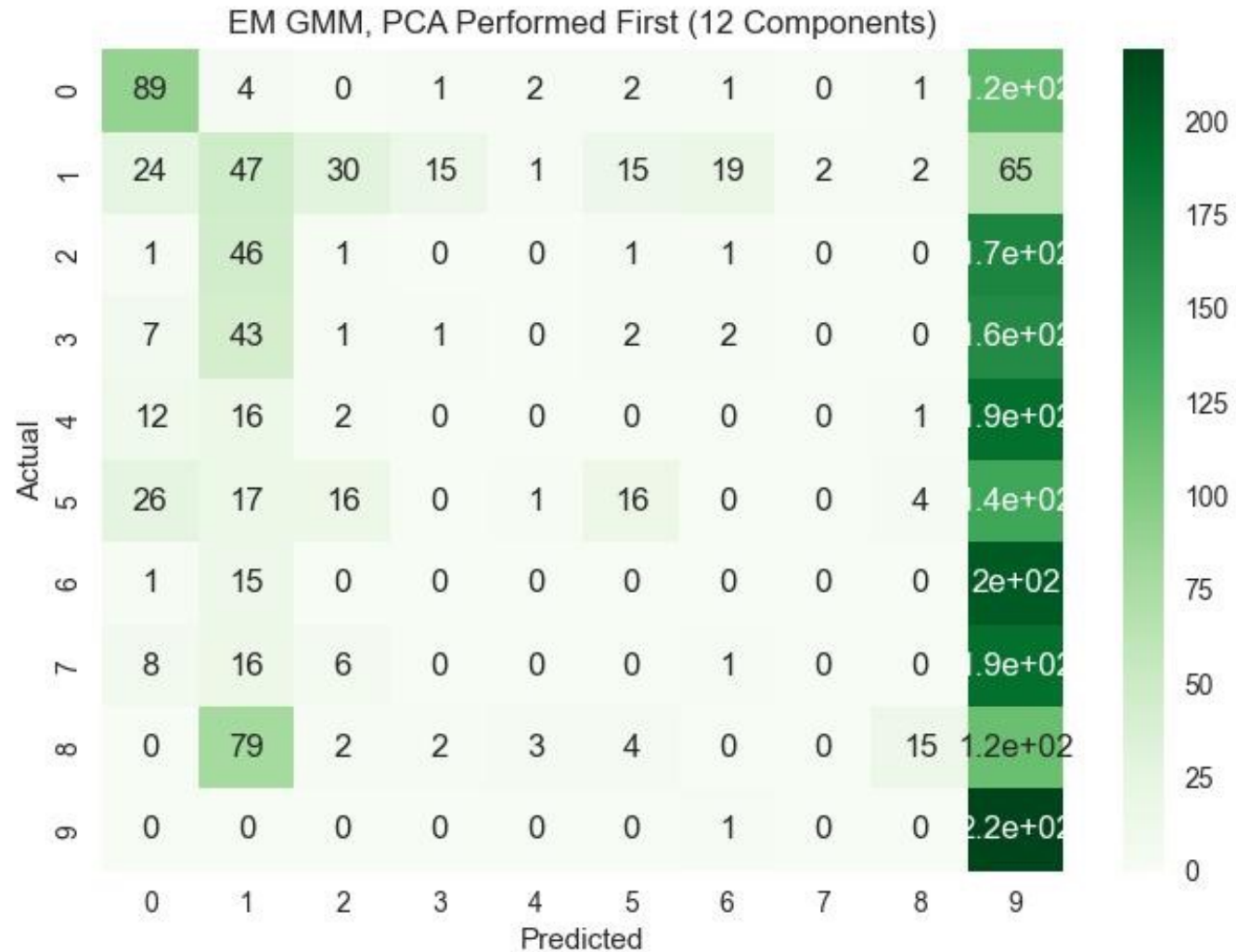
Selecting a Subset of Cepstral Coefficients

Motivation

Reducing the dimensionality of our dataset would improve code runtime significantly, so this would be preferred if there are cepstral coefficients that can be removed that do not diminish classifier accuracy very significantly. Principal component analysis was applied (SciKitLearn) and resulted in a very large runtime and a severe reduction in accuracy in projecting down to just 12/13 input features. More simplistic methods including examining the variances among each MFCC within a digit at the modeling phase and choosing 1 to exclude were also conducting.

PCA Results (12 Components)

- Total Accuracy: 17.64%



PCA Interpretation

The runtime was only sped up by roughly 5% with the 12 component PCA model. Accuracy is expected to decline further with each additional PCA component that is removed in our projection, so there is no use in attempting to diminish runtime further, as the classification results are not acceptable now. As the classification accuracy was so low, this gain in runtime is clearly not worth the tradeoff. So, other methods of diminishing MFCC dimensionality will be explored. It is interesting that the classifier selects 9 and 1 so much of the time with the PCA model. This implies that there is variance the PCA has removed that is integral in distinguishing the set of digits [0,2,3,4,5,6,7,8] from the set [1,9]. PCA has been shown to be weak on data where there is a weak correlation between the feature dimensions. So, the results suggest that the MFCCs may be very close to independent and removing each of them individually might yield better results.

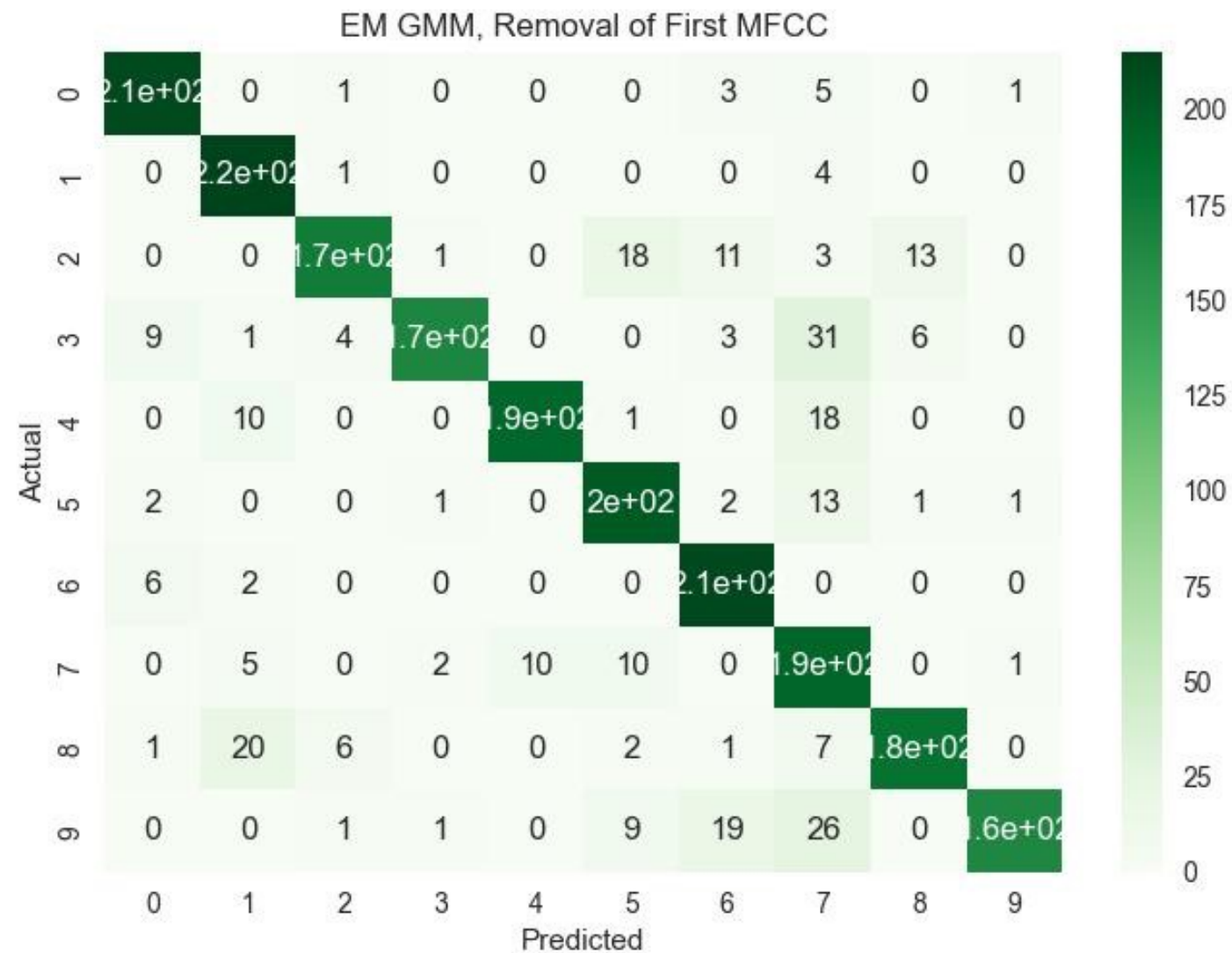
Choosing MFCCs to Remove by Hand

Motivation and Analysis

After PCA made classification much worse, I experimented with removing each of the MFCCs by hand to see if it was possible that one of them was particularly noisy or if there could be runtime gains achieved in removing one without impacting classification accuracy too much. After running the classification with each MFCC removed, it was evident that the first coefficient was noisy. The accuracy actually increased (by 0.32%) after it was removed from the GMM fitting step. The rest provided significant decreases in classification efficacy and only sped up the code's runtime marginally. So, it makes the model better in terms of both runtime and accuracy to remove the 1st MFCC value from the data.

Results

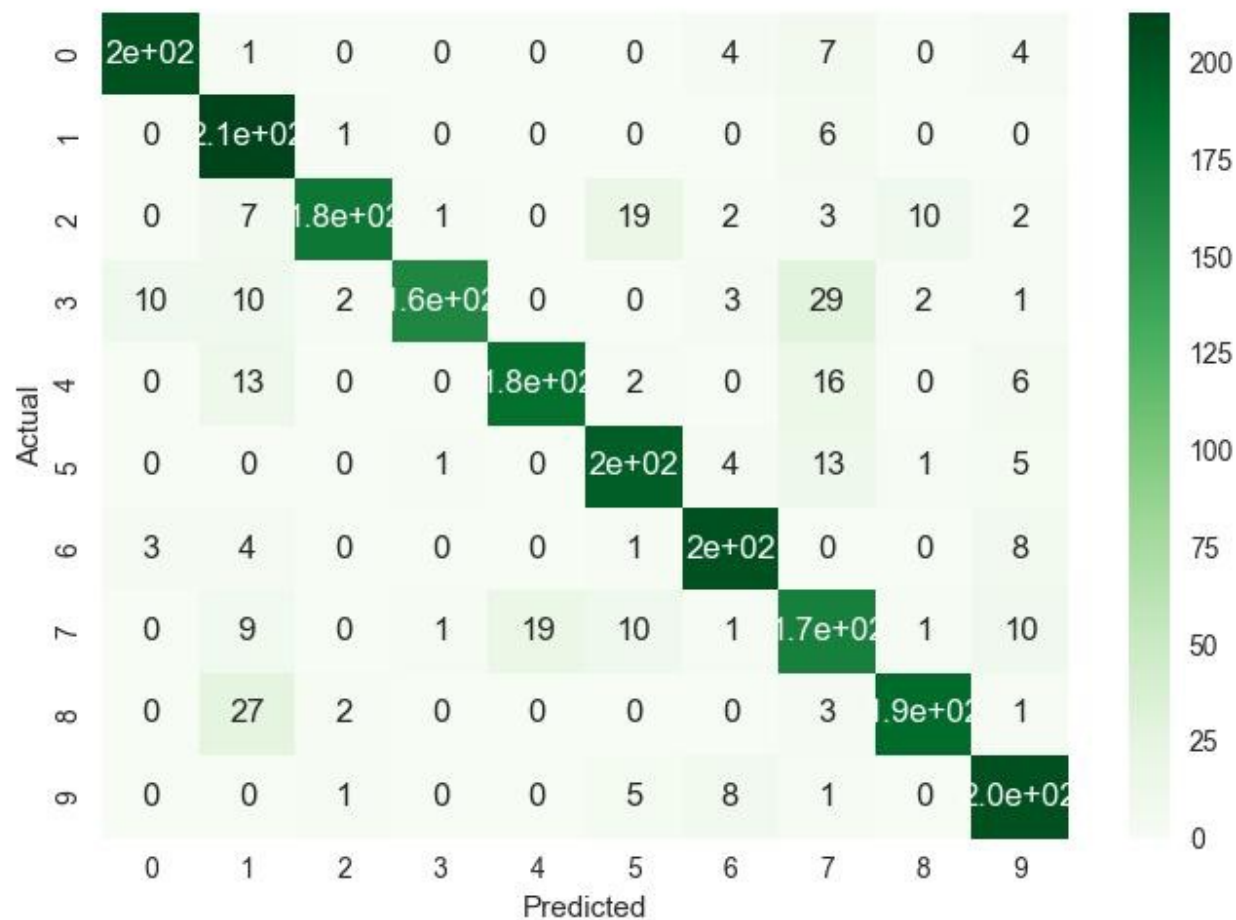
- Total Accuracy: 86.68%



Covariance Matrix Restriction

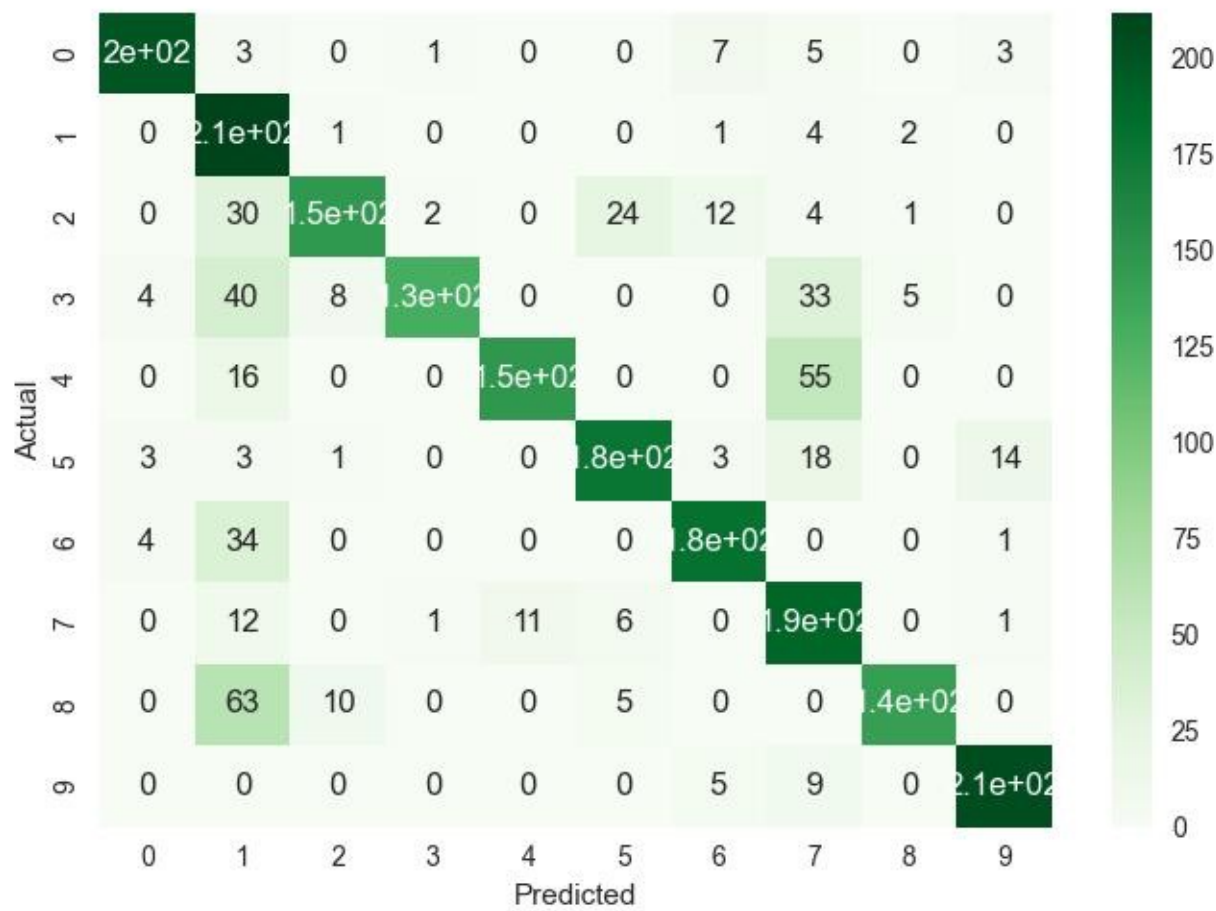
Full Covariance Matrix EM GMM Results

- Total Accuracy: 86.36%



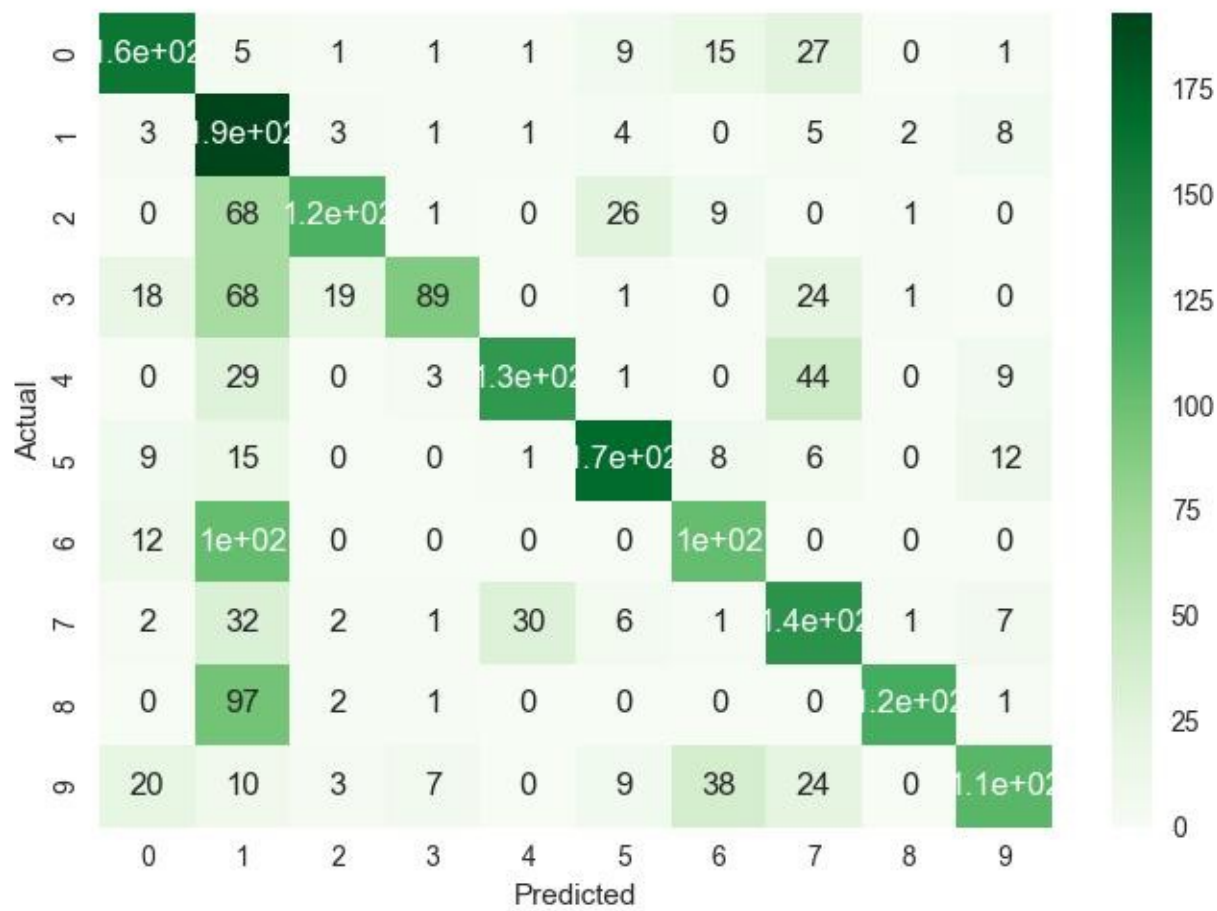
Diagonal Covariance Matrix EM GMM Results

Total Accuracy: 78.86%



Spherical Covariance Matrix EM GMM Results

Total Accuracy: 60.45%



Which is best?

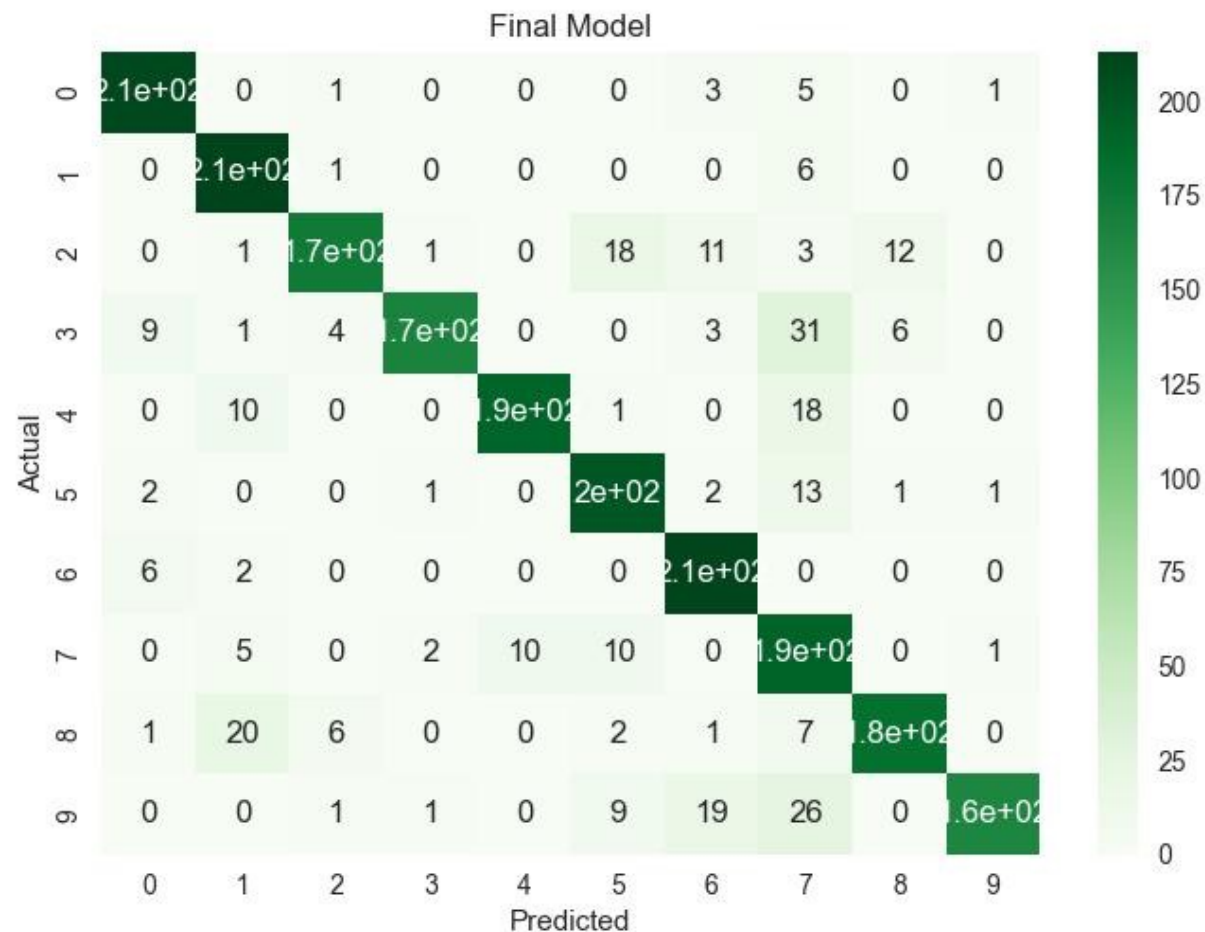
The code runtime is spent mostly on the classification step, not the GMM fitting step. As this only happens once per code run and is dwarfed in runtime by the classification, it makes sense to disregard the slight performance differences between the covariance matrix shapes in the SciKitLearn GMM implementation and select the matrix shape that leads to the greatest accuracy in classification. As expected, the full covariance matrix fit the data best. Spherical constraints were the worst, suggesting that the clusters (13D, so they can not be visualized) have higher-dimensional ellipsoid shapes instead of spherical ones where identical vectors along each axis are suitable for fitting. As the previous results show, the accuracy increases in classification with each increase in covariance matrix flexibility. So, as the runtime for the GMM fitting step is not a huge concern, selecting a full covariance matrix is a wise modeling choice.

Conclusions

Final Model

Based on the modeling choices explored, the best model achieved was the GMM EM model with a full covariance matrix, the 1st MFCC removed, and number of clusters decided with an elbow plot for each digit. The runtime between options explored was not appreciably different, and this combination of choices yielded the model with the greatest accuracy.

Accuracy: 86.6%



Important Modeling Decisions

The choice of K-Means or EM clustering to generate a GMM is an important modeling choice. There was a significant gain in model accuracy yielded just by changing this parameter of the classifier. This is because K-Means clusters based on less information: EM determines clusters based on both mean and covariance.

The choice of covariance matrix shape also impacts the results significantly. Less constrained covariance matrices yielded much better results, all while compromising very little on runtime. This is because less constrained matrices have more flexible shapes and can fit the data shape better in general. So, it is essential to examine the model results with each covariance matrix shape and determine which is most appropriate for the underlying data.

The choice of which MFCCs to include in the model – if one is to be removed - is also very important. My analysis of the impact of each MFCC on the model's accuracy shows that some of the MFCCs are actually noisy in terms of classification. Removing the first coefficient improved both runtime and accuracy.

Unimportant Modeling Decisions

The choice to remove a single MFCC did not substantially increase the accuracy or runtime of the model. Despite identifying a potentially noisy 1st MFCC, the results achieved in removing it may not warrant the time spent to analyze the model sans each MFCC and determine this information. So, unless dimensionality can be more significantly reduced (not the case in this project as accuracy started to plummet after 1 MFCC) successfully, this feature is not of cardinal importance to the model's overall performance. As the PCA results would suggest that the MFCCs are relatively independent, losing 1/13 feature points still provides enough variance for the classifier to work similarly.

What Went Well?

Overall, the classification results turned out well. Going in, I expected the EM Classifier to outperform the K-Means classifier, which was indeed the case. My method for selecting the number of components in each K-Means and EM fitting yielded GMMs that fit the data well, as the accuracy was ~86% for the final model. This is an acceptable results, given the data still contains significant variances - spoken digits from different people and genders.

My method of selecting an MFCC to remove also worked well, as removing the 1st MFCC resulted in gains in both accuracy and runtime.

My method of iteratively improving the model for subsequent modeling choice explorations was very effective and is something I would repeat, as this allowed me to isolate each new choice and work with the most efficient version of the model possible.

Lessons Learned

Allotting more time at the project's conclusion for running the code many more times and exploring more modeling options would have improved the project. I delve more deeply into a specific example of this on the next slide.

The PCA performed in the project was not effective at reducing the dimensionality compared to just removing an MFCC. This was surprising, as I anticipated that PCA would be able to maintain some variance from each of the 13 MFCCs in its 12 component form. I was not aware that there were conditions in which PCA would not be appropriately applicable prior to beginning the project. Seeing these results inspired me to research this and understand that PCA is not effective if the data features are wholly independent.

Frame Aggregation for Modeling

K-Means and EM required data input vectors of 13 dimensions each. As each block consists of several frame vectors like this, it was initially unintuitive to select a way to represent the blocks in a format that can be clustered. After experimenting with many approaches in code, it became clear that concatenating the frames together and fitting the GMMs to the concatenated block data made the most sense and yielded proper results. This was done in throughout the modeling procedures of this project. In order to reduce the amount of data, it would be possible to potentially identify the key frames by examining MFCC variance values and then fit/test the models based on only these frames. Next steps to improve model runtime could be to implement a variance-based frame selector that tries to trim the number of MFCC frame vectors in each block. This would speed up classification and potentially not impact accuracy very much. Given the runtime of the model, there was not enough time remaining at the end to adequately explore this modeling choice.

Collaborators

I discussed grouping the data and pseudocode with Monty Truitt on one occasion. We did not exchange code or other resources.

Citations

Arabic Numbers: How to Count in Arabic. *ArabicPod101.Com Blog*, 25 Oct. 2019, <https://www.arabicpod101.com/blog/2019/10/24/arabic-numbers/>.

Ashish, Ashish et al. "IDENTIFICATION AND VERIFICATION OF SPEAKER USING MEL FREQUENCY CEPSTRAL COEFFICIENT." *International Journal of Information Technology & Computer Sciences Perspectives* 3 (2014): 907-913.

Bahl, Lalit R., et al. "A Maximum Likelihood Approach to Continuous Speech Recognition." *Readings in Speech Recognition*, 1990, pp. 308–319., <https://doi.org/10.1016/b978-0-08-051584-7.50029-2>.

Bengfort, Benjamin, et al. (2022). *YellowBrick V1.5*. <https://doi.org/10.5281/zenodo.7013541>

Chen, Tao, et al. "Probability Density Estimation via an Infinite Gaussian Mixture Model: Application to Statistical Process Monitoring." *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 55, no. 5, 2006, pp. 699–715., <https://doi.org/10.1111/j.1467-9876.2006.00560.x>.

Chaudhary, Mukesh. "K-Means Clustering in Machine Learning:" *Medium*, Medium, 29 May 2020, <https://medium.com/@cmukesh8688/k-means-clustering-in-machine-learning-252130c85e23>.

Fabien, Maël. "Expectation-Maximization for GMMS Explained." *Medium*, Towards Data Science, 3 June 2020, <https://towardsdatascience.com/expectation-maximization-for-gmms-explained-5636161577ca>.

Maximum Likelihood Estimation: STAT 415. *PennState: Statistics Online Courses*, <https://online.stat.psu.edu/stat415/lesson/1/1.2>.

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

Reynolds, Douglas. "Gaussian Mixture Models - Leap Laboratory." *Gaussian Mixture Models*, http://leap.ee.iisc.ac.in/sriram/teaching/MLSP_19/refs/GMM_Tutorial_Reynolds.pdf.