Recognizing Spoken Digits

Ben Matz
ECE 480 Fall 2022

Introduction - The Dataset

In 2008, a group of researchers at University of Badji-Mokhtar collected a dataset of 8800 spoken Arabic digits. Using some signal processing techniques, the researchers decomposed these frequencies into 13 Mel Frequency Cepstral Coefficients (MFCCs). These represent the perceived difference between pitches as opposed to the true distance measured in Hertz (Hz) (Bogert, 1963).

https://educativ.net/universities/algeria/universite-badji-mokhtar-de-annaba/

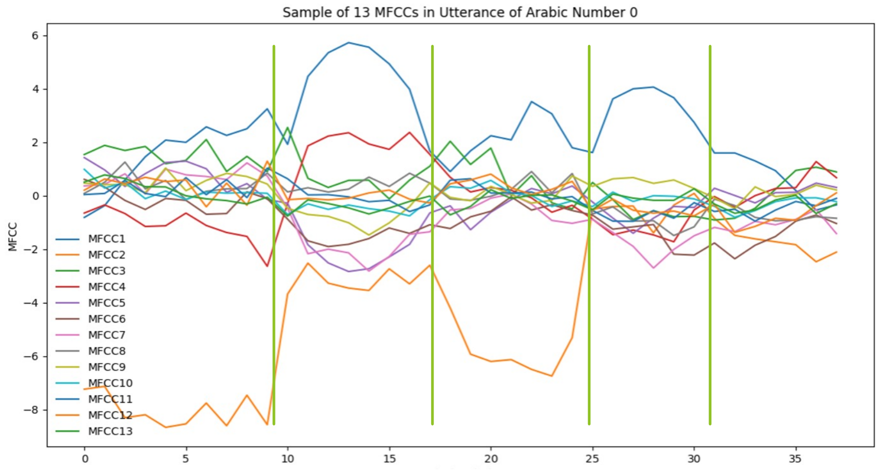
UCI

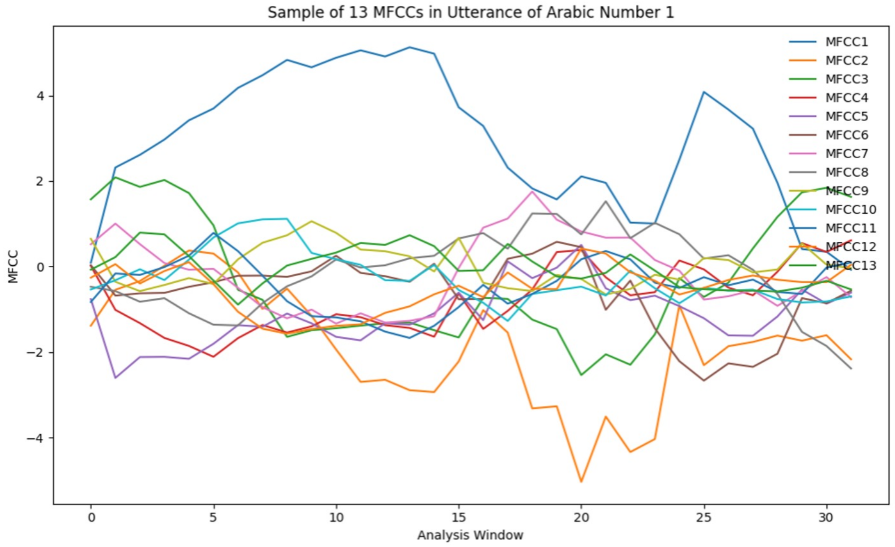
Machine Learning Repository

https://archive.ics.uci.edu/ml/index.php

Introduction - MFCC vs. Analysis Window

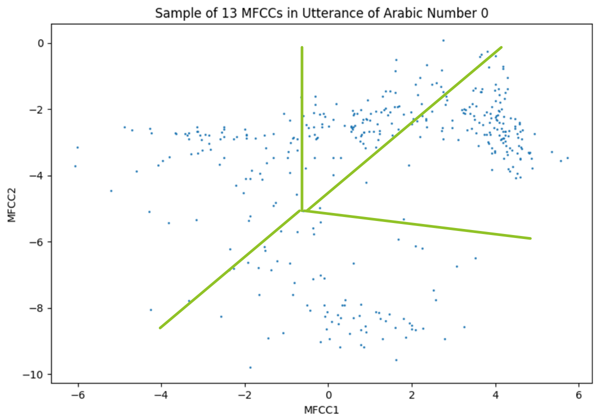
These graphs show what the 13 different MFCCs look like for one utterance of zero and one. Over a series of 25-40 analysis windows, the change can be visualized and broken down into a few phonemes. The phonemes in the zero last roughly eight windows (depicted with the lines) and allow us to break the number up into different "sounds." One can see how the first few MFCCs vary much more than the last few which tend to stay flat over the course of the utterance. One can also see distinct differences between the patterns for the two digits.

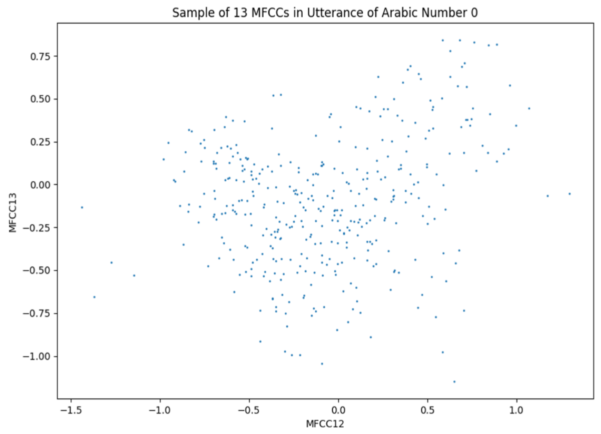


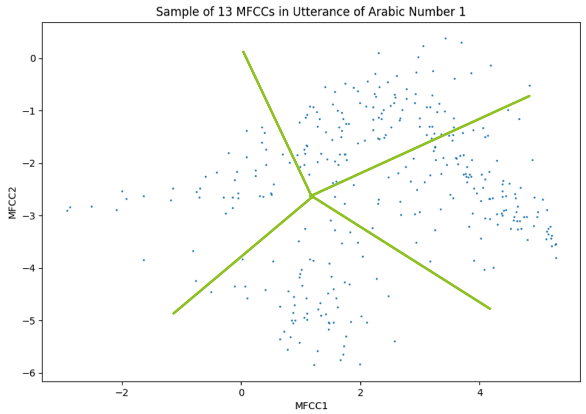


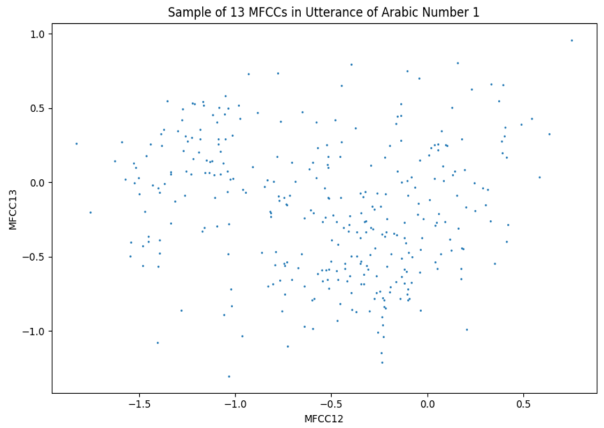
Introduction - Pairwise MFCC Scatterplots

These graphs show a pairwise comparison of MFCCs 1 and 2 or 12 and 13 for the digits zero and one. On the right half, with the later MFFCs, there is not much of a pattern. On the left, there are more distinct clusters (potential assignments annotated), showing that the first few coefficients are the best to base the prediction off of.





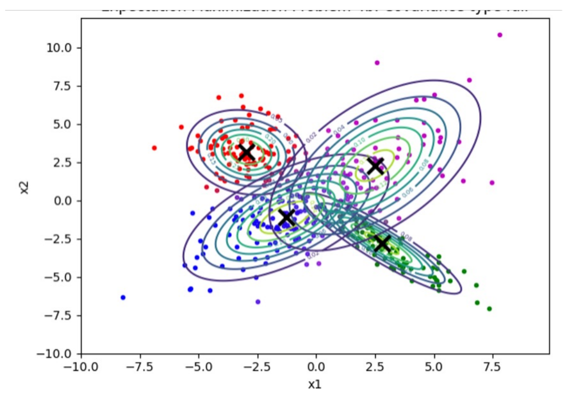




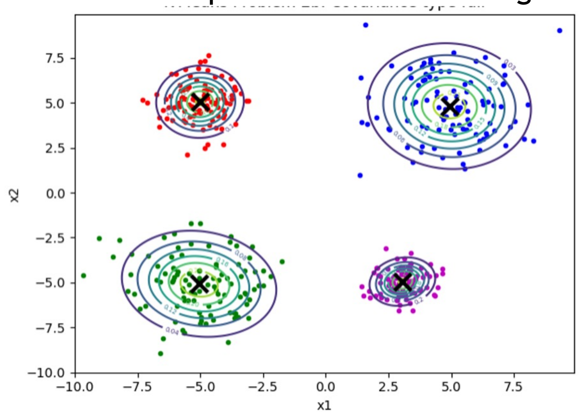
Introduction - The Problem

I was tasked with creating a model to accurately predict a spoken digit by using the dataset. This problem has many applications as natural language processing (NLP) is an emerging field. The largest application at the moment are virtual assistant devices such as Amazon Alexa and Siri. However, it could also be used for translation and speech-to-text technologies. In this project, I will explore K-Means (Lloyd, 1957) and Expectation Maximization (EM) clustering (Dempster, et al. 1958).

Example EM Clustering

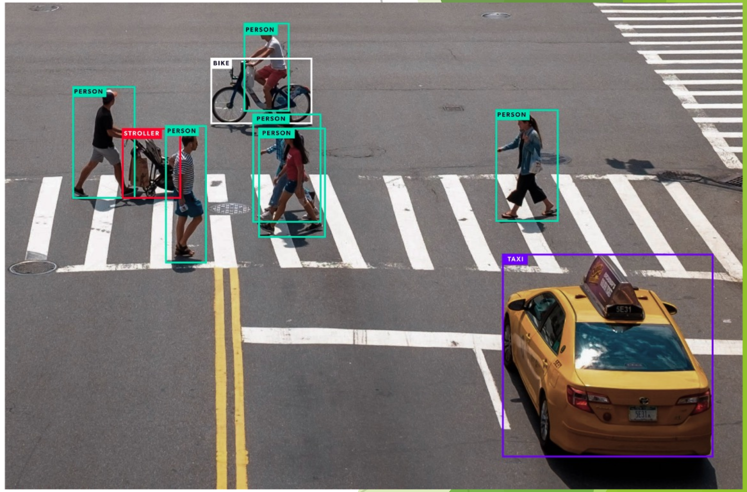


Example K-Means Clustering



Introduction - Additional Applications

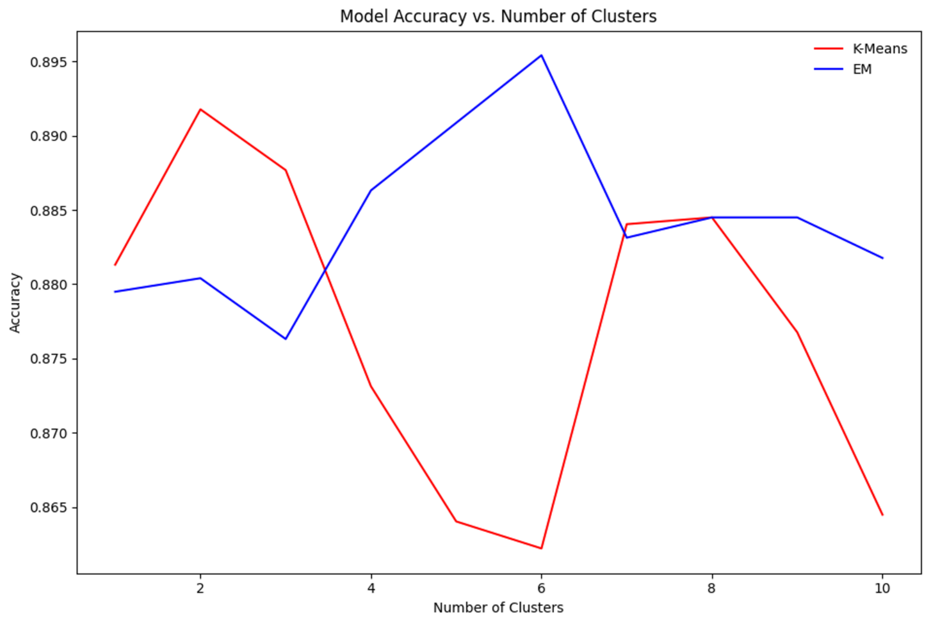
The gaussian mixture model (GMM) (Duda & Hart, 1973) and maximum likelihood estimation (MLE) (Fisher, 1921) modeling framework is generally used in classification problems. One specific type is image recognition and computer vision. If there's a constrained set of outputs, one can make a GMM and then use MLE to select the most likely output given an image. This paradigm could also be useful applications like Gradescope that read handwriting.



https://tryolabs.com/guides/introductory-guide-computer-vision

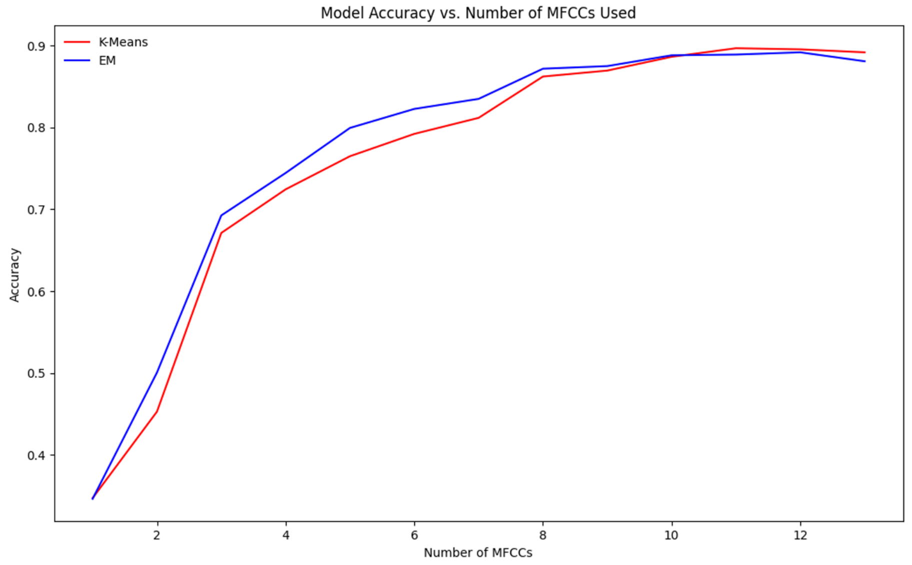
Modeling Choices - Number of Clusters

This is one of the more obvious choices when modeling. While less clusters means less computational intensity, it generally means less accuracy. However, too many clusters can cause overfitting when the models are applied to the testing data. As shown in the figure to the right, K-Means Clustering accuracy peaks around 2 or 3 clusters while EM peaks around 5 or 6 clusters.



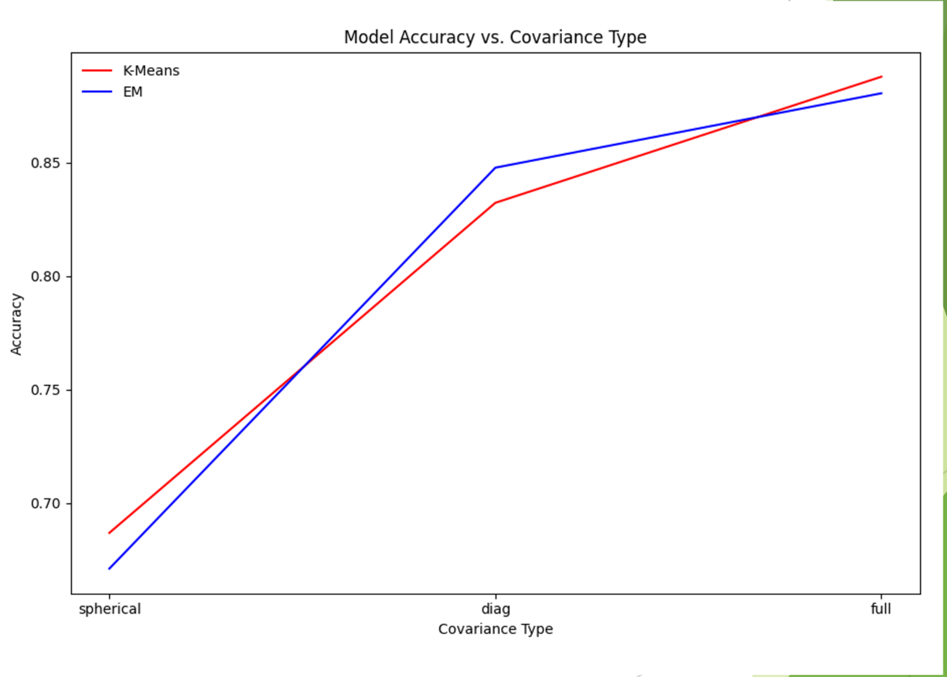
Modeling Choices - Number of MFCCs

The data has 13 MFCCs per spoken digit. However, not all MFCCs are created equally. Generally speaking, the first few MFCCs have the most variance while the last few barely change over the course of the utterance. While less MFCCs means less computational intensity, it generally means less accuracy. In extreme cases, too many MFCCs can cause overfitting. As shown in the figure to the right, both clustering techniques begin to level off in accuracy round 10 or 11 coefficients



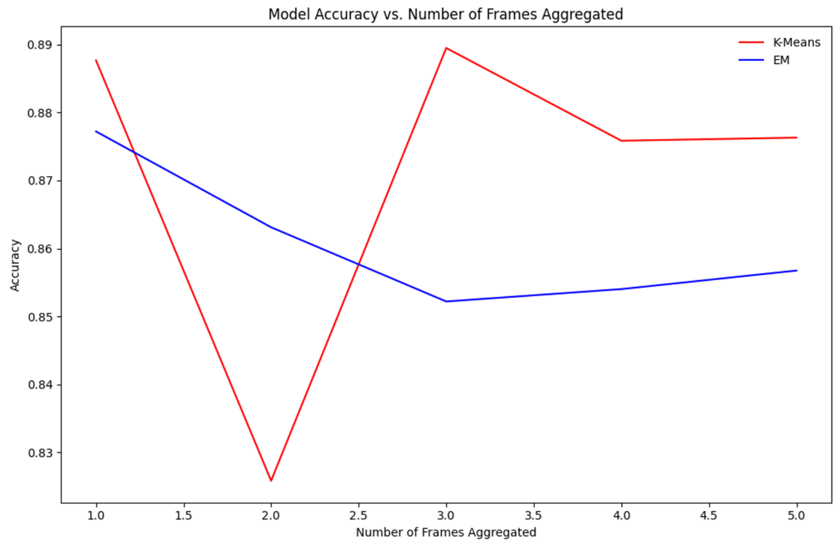
Modeling Choices - Covariance Type

I explored three covariance types: spherical, diagonal, and full. Let N represents the number of digits and D represents the number of MFCCs used. Then, to estimate covariance matrices, spherical type requires N numbers, diagonal requires N*D numbers, and full requires $N((D^2 + D)/2)$. As expected, full matrices gave the best results. However, accuracy only increased from 83% to 88%. So, if a lower fidelity model is needed, it’s almost certainly worth the tradeoff for a D/2 times increase in runtime (assuming full dimensionality).



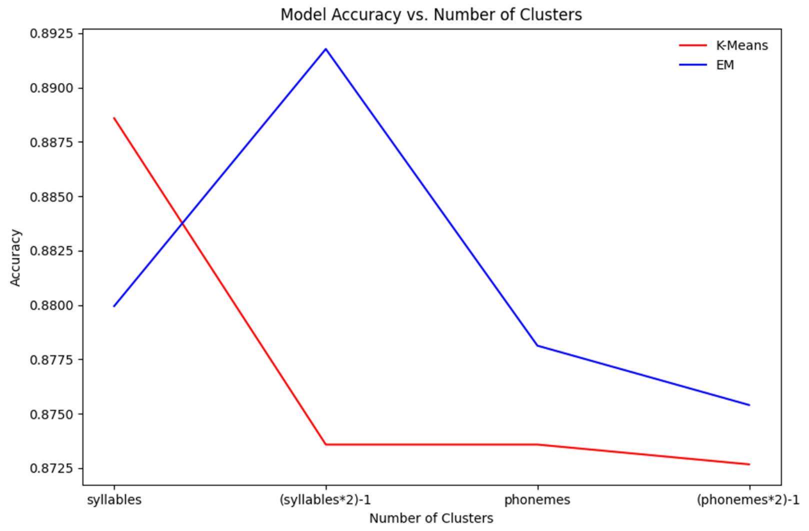
Modeling Choices - Aggregated Frames

In my first modeling attempts, I took each frame individually. However, one could instead choose to aggregate frames and treat n consecutive frames as one value. As shown in the graph, no aggregation improved either of the models. K-Means stayed fairly constant in accuracy aside from the outlier at n = 2 whereas EM's accuracy generally decreased the more that frames were combined. However, the upside is that aggregating data frames decreases the amount of points put into the model which can decrease runtime.



Modeling Choices - Number of Clusters Depending on Digit

While earlier modeling efforts kept the number of clusters constant across all digits, this doesn’t make the most sense. In theory, clusters should map to syllables or phonemes. Here, I tried 4 settings where the number of clusters was equal to the number syllables, phonemes, or either with transitions (2n - 1). More clusters generally means more computational burden and in this case, that didn’t appear to be worth the squeeze as accuracy did not significantly increase over standard parameters.

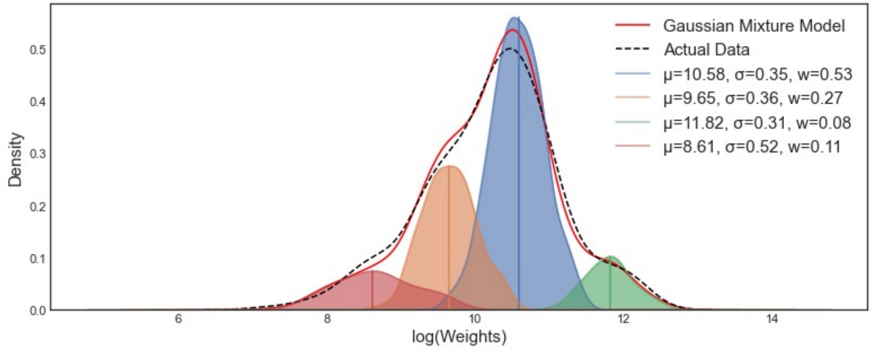


ML Classification – Gaussian Mixture Models (Duda & Hart, 1973)

A Gaussian Mixture Model with m mixture components/clusters is a function

$$f(x) = \pi_1 f_1(x) + \dots + \pi_m f_m(x)$$

where the $\Sigma\pi_i = 1$. Each π_i represents the proportion of observations in cluster i. In the context of this project $f_i(x)$ is a multivariate normal distribution with mean μ_i and covariance matrix $\mathbf{\Sigma}_i$.



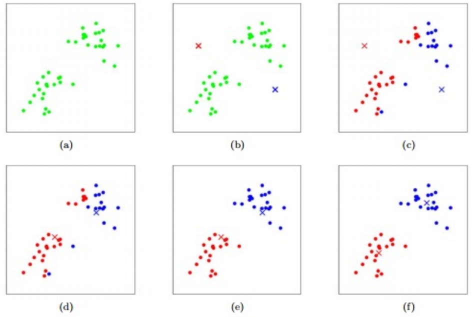
https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

ML Classification – K Means (Lloyd, 1957)

K-Means clustering is an unsupervised learning technique where the goal is to minimize the overall distance between points and their cluster centers. Given the cluster centers μ_m and cluster memberships, this total distance can be calculated as:

$$D = \sum_{k=1}^{K} \sum_{j=1}^{J} z_{j,k} d(\mathbf{x_j}, \mu_k)$$

where $z_{j,k}$ is an indicator function that equals 0 if $\mathbf{x}_j$ is not in cluster k and 1 if $\mathbf{x}_j$ is in cluster k. d is a function that calculates distance between $\mathbf{x}_j$ and μ_k. After initializing means, for each iteration, we assign points to their nearest cluster, recalculate the mean of each cluster, and then repeat (as the means have now changed). We keep iterating until reaching or approaching a minimum.



https://stanford.edu/~cpiech/cs221/handouts/kmeans.html

ML Classification – Expectation Maximization (Dempster, et al. , 1958)

Expectation Maximization is a clustering algorithm that uses an MLE algorithm to predict the parameters (θ) of the clusters. After initializing θ–which in the case of this problem includes $\mu_m, \mathbf{\Sigma}_m$, and π_m–EM can be broken up into two steps.

First is the E-Step. For the current model parameters θ, we must evaluate the responsibilities for all data $\mathbf{x}_n$. A responsibility $r_{n,m} = \frac{\pi_m Norm(\mathbf{x}_n|\mu_m,\mathbf{\Sigma}_m)}{\sum_{j=1}^{m} \pi_j Norm(\mathbf{x}_n|\mu_j,\mathbf{\Sigma}_j)}$ represents the probability that $\mathbf{x}_n$ was drawn from component m where $0 \leq r_{n,m} \leq 1$ and $\sum_{m=1}^{M} r_{n,m} = 1$.

Second is the M-Step where we estimate θ for the current responsibilities.

$N_m = \sum_{n=1}^{N} r_{n,m}$ represents effective number of observations in component m.

$\mu_m = \frac{1}{N_m} \sum_{n=1}^{N} r_{n,m}\mathbf{x}_n$ represents effective mean of component m.

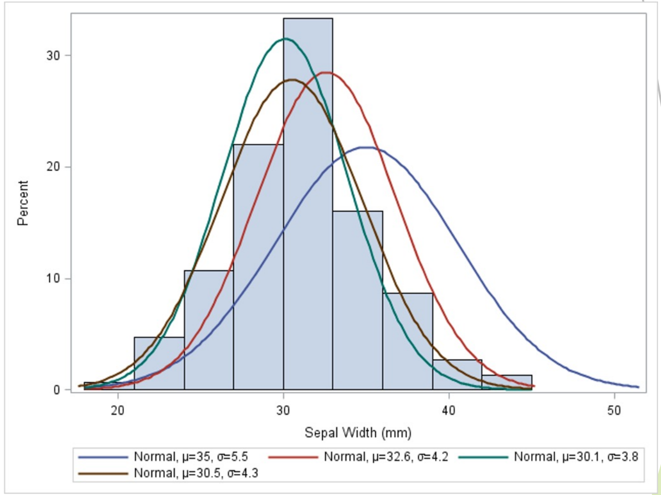
$\mathbf{\Sigma}_m = \frac{1}{N_m} \sum_{n=1}^{N} r_{n,m}(\mathbf{x}_n - \mu_m)(\mathbf{x}_n - \mu_m)^T$ represents effective covariance of component m.

$\mathbf{\Pi}_m = \frac{N_m}{N}$ represents effective probability of being in component m.

We iterate through these two steps until we reach convergence in either θ or the likelihood function.

ML Classification – What and Why Maximum Likelihood Estimation (MLE)

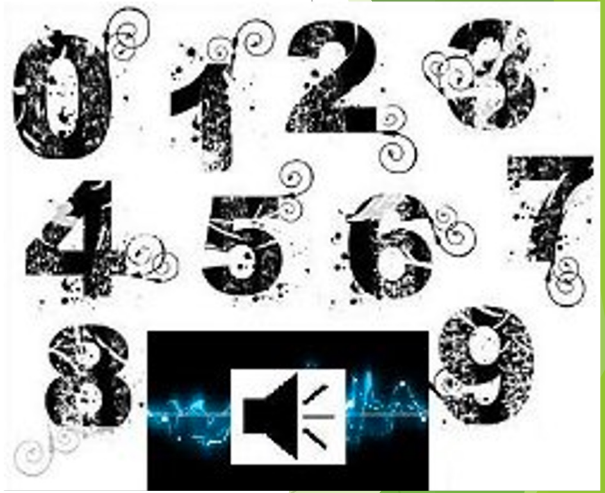
As the name suggests, MLE attempts to maximize the likelihood: p(X|Θ). This is the probability that we see the data given some model parameters. MLE is frequently used in classification models because it allows us to calculate which model the data is most likely to have come from. Using probability density functions (PDFs), we can calculate a data’s probability density for each model and then pick the maximum density as our selection.



https://blogs.sas.com/content/iml/2011/10/12/maximum-likelihood-estimation-in-sasiml.html

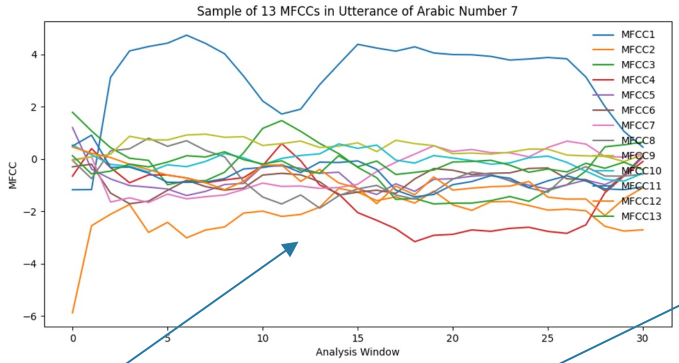
ML Classification - MLE in this Problem

For this project, we have a constrained set of outputs (0-9): a typical classification problem. So, we can represent each digit as a multivariate Gaussian normal distribution model with a D-dimensional mean vector and a DxD covariance matrix (D = number of MFCCs). Then, when given a new utterance, we can iterate over all of the digit models, calculate the likelihood of the data for each digit and pick the most likely number as our prediction.



https://archive.ics.uci.edu/ml/datasets/Spoken+Arabic+Digit

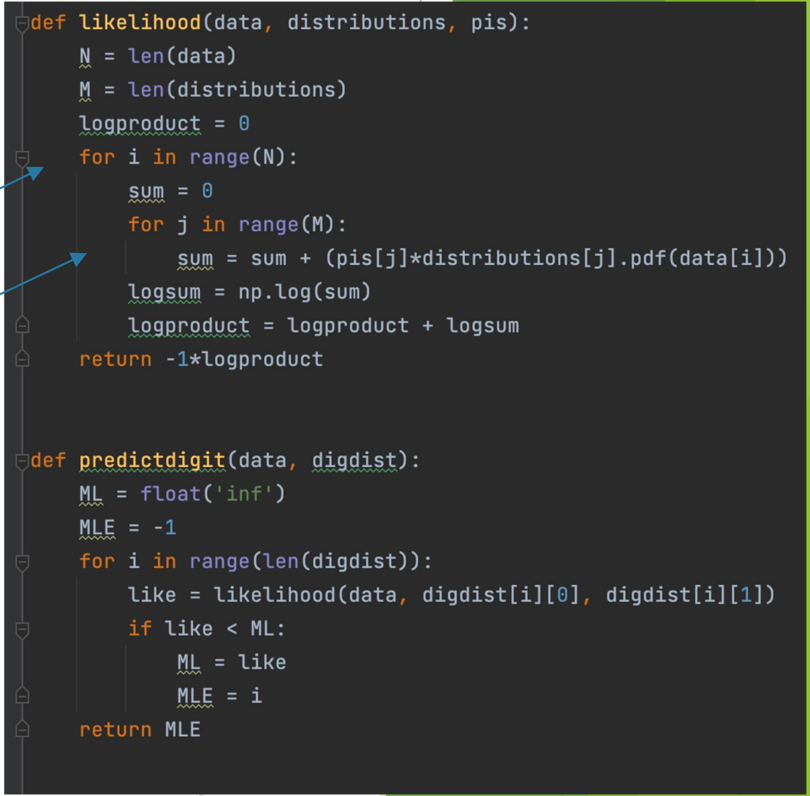
ML Classification - The Math Behind MLE



Where $\mathbf{X}$ is a series of N cepstral coefficients and we are given an M-component mixture model with mean μ_d, covariance matrix $\mathbf{\Sigma}_d$, and responsibility π_d, the likelihood of the data for a digit d is given by the equation:

$$p(\mathbf{X}|\mu_d, \mathbf{\Sigma}_d, \pi_d) = \sum_{i=0}^{N} log(\prod_{j=0}^{M} \pi_{j,d} * p(\mathbf{x}_n|\mu_d, \mathbf{\Sigma}_d)) = \sum_{i=0}^{N}\sum_{j=0}^{M} log(\pi_{j,d} * p(\mathbf{x}_n|\mu_d, \mathbf{\Sigma}_d))$$

and our goal is to find the maximum value of $p(\mathbf{X}|\mu_d, \mathbf{\Sigma}_d, \pi_d)$ over all d, giving us our predicted digit.



```
def likelihood(data, distributions, pis):
    N = len(data)
    M = len(distributions)
    logproduct = 0
    for i in range(N):
        sum = 0
        for j in range(M):
            sum = sum + (pis[j]*distributions[j].pdf(data[i]))
        logsum = np.log(sum)
        logproduct = logproduct + logsum
    return -1*logproduct


def predictdigit(data, digdist):
    ML = float('inf')
    MLE = -1
    for i in range(len(digdist)):
        like = likelihood(data, digdist[i][0], digdist[i][1])
        if like < ML:
            ML = like
            MLE = i
    return MLE
```

ML Classification - Challenges of MLE

The main issue I ran into with MLE was numerical underflow. When multiplying together so many small probability densities, they will eventually tend towards zero. Consequently, when I first implemented my MLE function, it was very bad at predicting digits. This error was fixed by using log-likelihood so that the product was rewritten as a sum of logs and I didn't have to worry about underflow. I decided to utilize a negative log-likelihood calculation so that my likelihoods would be positive. However, it was still performing very poorly. After some time, I realized that flipping the sign required a flip of the equality and I instead should have been predicting the digit that gave the *minimum* negative log-likelihood. After this fix, my model worked much better and the MLE function did not require any more tweaking.

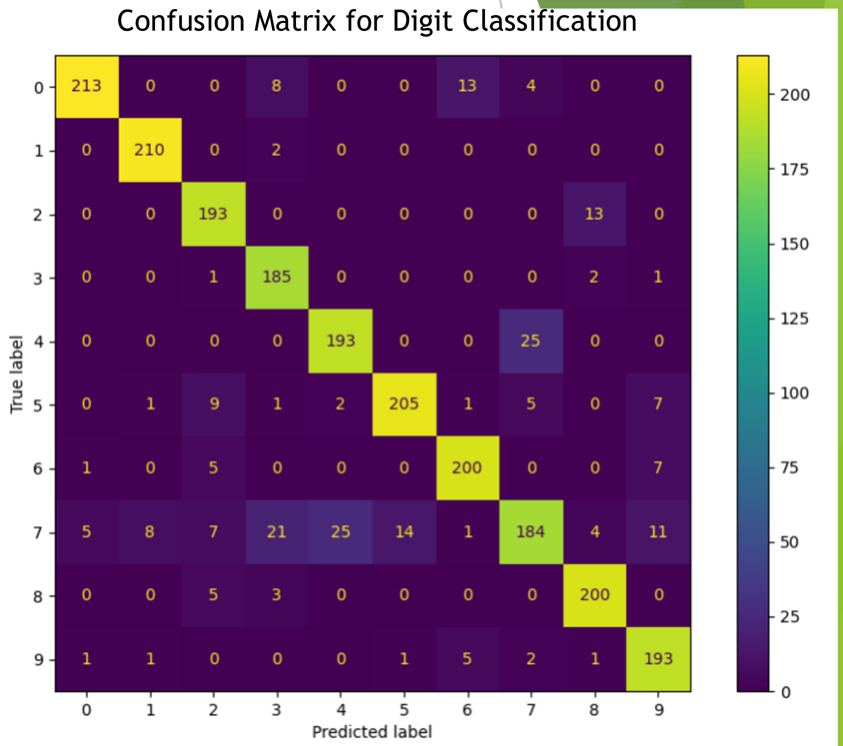
Results - Confusion Matrix (Final Parameters)

Overall Accuracy: 1976/2200 (89.8%)

Best Predicted Digit: 0 (96.9%)

Worst Predicted Digit: 7 (83.6%)

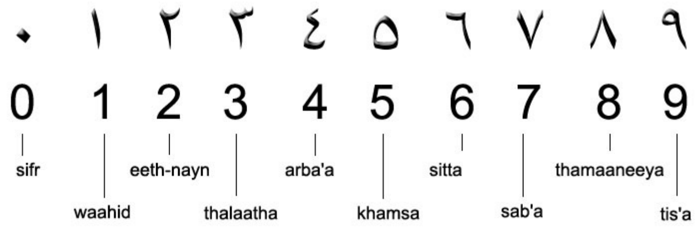
Most Confused Digits: 4 and 7 (11.4% for each digit)



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	213	0	0	8	0	0	13	4	0	0
1	0	210	0	2	0	0	0	0	0	0
2	0	0	193	0	0	0	0	0	13	0
3	0	0	1	185	0	0	0	0	2	1
4	0	0	0	0	193	0	0	25	0	0
5	0	1	9	1	2	205	1	5	0	7
6	1	0	5	0	0	0	200	0	0	7
7	5	8	7	21	25	14	1	184	4	11
8	0	0	5	3	0	0	0	0	200	0
9	1	1	0	0	0	1	5	2	1	193

Results - Prediction Explanations

The best predicted digits tended to be unique ones. For instance, 0 (sifir)--the only digit with a “Long I” sound--was predicted with near perfect accuracy. 1 (wahad)--the only digit with a w, h, or d sound--was a close second. Conversely, 4 (araba’a) and 7 (seb’a), the most confused digits, both have the “ba” sound in them. 2 (ithnayn) and 8 (thamanieh), also often confused, both share the “th” sound. Also, the four best predicted digits are all two syllables, which likely plays a factor with constant cluster sizes. The overall prediction trends for modeling choices were depicted in the “Modeling Choices” section.



https://seasonsali.wordpress.com/2010/12/11/arabic-numerals/

Conclusions - Important Modeling Choices

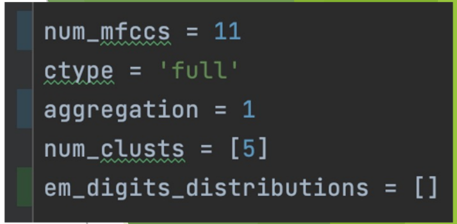
The two most important modeling choices were covariance type and number of MFFCs used. Not only did these have massive impacts on accuracy, but also on the computational intensity. As mentioned in the covariance slide, the simplest method of estimation is nearly D^2 times faster than the most complicated. However, this comes at the cost of poor accuracy (around 70%). Covariance has a huge impact as full covariance can allow for “tilting” of GMM components.

Decreasing the number of MFCCs used greatly lessens the dimensionality of the problem which also causes great speedups. However, with just 1 MFCC, we get about 35% accuracy. Yet, with 7 MFCCs, we can get over 80% accuracy. Number of MFCCs used makes a large difference because generally the more information we have, the better model we can make. However, once we got to 10 or 11 of the 13 MFCCs, we began to see the effects of overfitting, and performance leveled out.

Conclusions - Less Important Modeling Choices

Most of the other modeling choices had small effects on accuracy of just a few percentage points. For frame aggregation, this makes sense as data itself is not changed, but rather how it’s represented. So, while this potentially speeds up the model, it shouldn’t impact performance too much. What surprised me was the lack of impact that the number of clusters had. While there were clear sweet spots, these led to meager improvements of just a percentage point. Beyond a certain size, when the number of clusters was increased, it’s possible that the models just split an old cluster into multiple parts. Of course, when the cluster number gets larger we also run into the problem of overfitting.

Conclusions - Final Model Parameters



```
num_mfccs = 11
ctype = 'full'
aggregation = 1
num_clusts = [5]
em_digits_distributions = []
```

Clustering Technique: EM - allows for a better fit to the data than K-Means does.

MLE Classifier: Negative Log-Likelihood - avoids numerical underflow while keeping likelihood values positive.

First n MFCCs Used: 11 - gives us a very good estimate of clusters without unnecessary computational slowdown or overfitting.

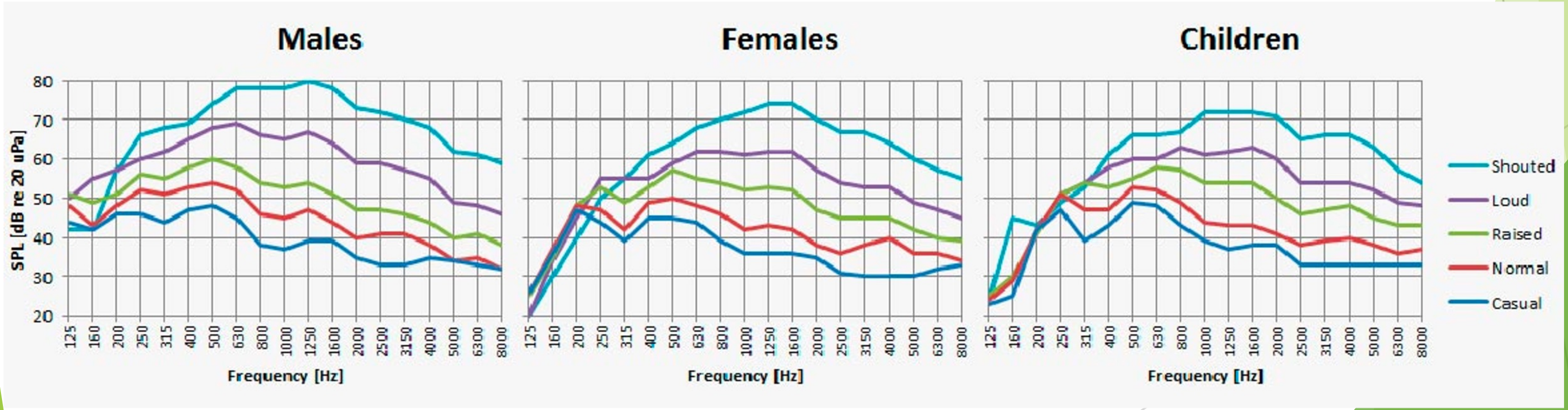
Number of Clusters: 5 (for all digits) - gives the best accuracy in EM benchmarking.

Covariance Type: Full - while slower than diagonal, still gives a 5% boost in accuracy.

Data Frames Aggregated: None - decreases EM accuracy in benchmarking.

Conclusions - Analyzing the Final Model

The system I selected works very well and is fairly simple. In under five minutes, using 6,600 training utterances, it can generate a model that predicts digits with nearly 90% accuracy! This is with a pretty standard implementation of EM and MLE which can be easily replicated. However, it could be improved upon with some conditional models. If one split up the data into male and female speakers, they could expect to see an increase in accuracy as males tend to have lower voices and therefore lower frequency pitches. So, one would anticipate different models for the different genders.



https://www.dpamicrophones.com/mic-university/facts-about-speech-intelligibility

Conclusions - Lessons Learned

My key takeaway from this project was that I should ensure that I have a good conceptual grasp of the task before diving into the code. When I started this project, I first lumped all of the data together and made one big GMM representing it (which obviously makes very little sense). However, I soon realized my error which brought me to my next problem. Where was the digit number encoded in the data? That led me to takeaway number two: read the data set description. Finally, if I had a do-over, I'd probably discuss the project with my classmates. Often, two peers can trade conceptual understandings to and fill in each other's gaps.

Conclusions - What Went Well?

Using the knowledge and code that I developed in my homework sets was quite helpful. When it came time to create my GMMs and make an MLE function, I turned to the homeworks where I had already done this. With just a bit of tweaking, they were ready for use in digit recognition. Also, as far as debugging, I made abundant use of print statements. This made it clear where I was going wrong **spoiler alert, I probably went out of bounds in an array**. Finally, I started this project very early. This was great whenever I got stuck. Rather than sitting frustrated at the computer, deadline looming, I could close my laptop and sit on it for a couple of days. Usually, with some pondering between classes, I’d be able to come up with a solution.

References - Background

B. P. Bogert, J. R. Healy and J. W Tukey, “The Quefrency Analysis of Time Series for Echoes: Cepstrum, Pseudo-Autocovariance, Cross-Cepstrum, and Saphe Cracking,” Proceedings of the Symposium on Time Series Analysis, 1963, pp. 209-243.

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.

A. V. Oppenheim and R. W. Schafer, “From Frequency to Quefrency: A History of the Cepstrum,” DSP History, September 2004.

References - Math

Dempster, A.P., Laird N. M., and Rubin, D.B. (1958). “Maximum Likelihood from Incomplete Data Via the EM Algorithm”. *Journal of the Royal Statistical Society (1977)*: 1-38.

Duda, R.O. and Hart, P.E. (1973) “Pattern Classification and Scene Analysis”. *Wiley*, New York.

Fisher, R.A. (1921). "On the "probable error" of a coefficient of correlation deduced from a small sample". *Metron*. 1: 3–32.

Fisher, R.A. (1922). "On the mathematical foundations of theoretical statistics". *Philosophical Transactions of the Royal Society A*. 222 (594–604): 309–368.

Forgy, Edward W. (1965). "Cluster analysis of multivariate data: efficiency versus interpretability of classifications". *Biometrics*. 21 (3): 768–769.

Lloyd, Stuart P. (1957). "Least square quantization in PCM". *Bell Telephone Laboratories Paper*. Published in journal much later: Lloyd, Stuart P. (1982)

References - Visualizations

https://educativ.net/universities/algeria/universite-badji-mokhtar-de-annaba/

https://archive.ics.uci.edu/ml/index.php

https://tryolabs.com/guides/introductory-guide-computer-vision

https://blogs.sas.com/content/iml/2011/10/12/maximum-likelihood-estimation-in-sasiml.html

https://archive.ics.uci.edu/ml/datasets/Spoken+Arabic+Digit

https://www.dpamicrophones.com/mic-university/facts-about-speech-intelligibility

https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

https://stanford.edu/~cpiech/cs221/handouts/kmeans.html

https://seasonsali.wordpress.com/2010/12/11/arabic-numerals/

References - Package and Toolbox

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

J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

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

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17(3), 261-272.

Collaborations

I worked entirely independently on this project. Thanks for reading my SlideDoc!