

RECOGNIZING SPOKEN DIGITS

Table of Contents

<i>page</i> 3	<i>page</i> 7	<i>page</i> 16	<i>page</i> 19	<i>page</i> 24
<hr/> <i>Problem Description</i> <hr/>	<hr/> <i>Data Modeling</i> <hr/>	<hr/> <i>Maximum Likelihood Classification</i> <hr/>	<hr/> <i>Classification Performance Results</i> <hr/>	<hr/> <i>Conclusions</i> <hr/>
 <i>page</i> 27	 <i>page</i> 30			
<hr/> <i>References</i> <hr/>	<hr/> <i>Collaborations</i> <hr/>			

Problem Description

Classifying Spoken Digits

Our goal for this project is to develop a model to classify recordings of digits spoken in Arabic. For example, given a recording of someone saying zero in Arabic, we'd like our model to predict that they said zero. Our model will do this for the digits zero through nine.

This model is a small **step to creating a voice recognition system**. If we can develop a model to predict which digit someone is saying, the model could be developed further to recognize words. With voice recognition, humans can **interact with computers with speech**, which provides convenience and accessibility. We could also use computers to **transcribe many audio files** much faster than a human could.

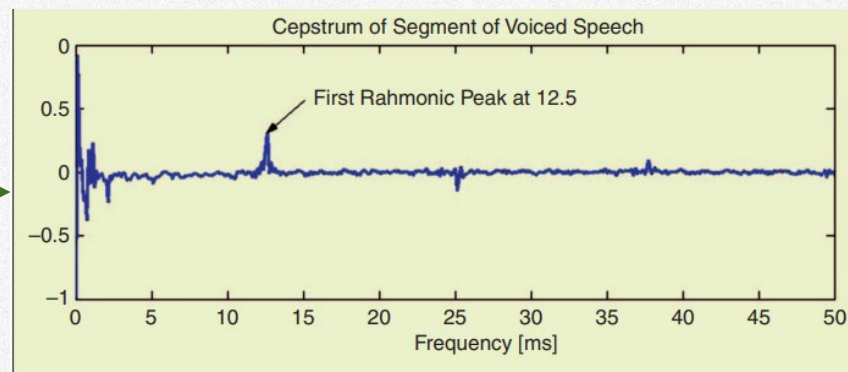
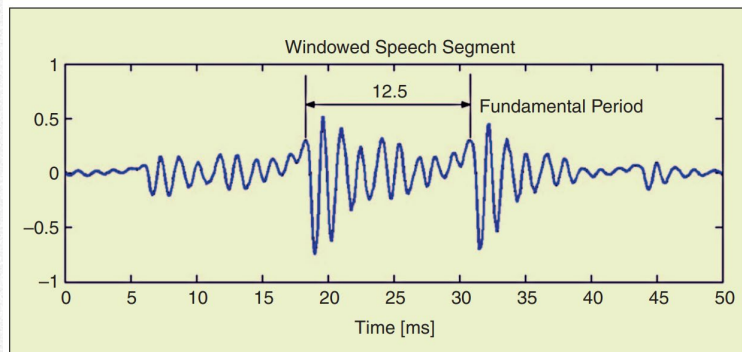
This approach can also apply to other problems that involve the classification of any multi-dimensional numerical data, even data with a varying length from sample to sample. The modeling framework we apply would be most effective if the data forms clusters. An example application is automated corrosion detection in remanufacturing, which used the same framework as this project¹.

1. Gibbons, Tom & Pierce, G. & Worden, K. & Antoniadou, Ifigeneia. (2018). A Gaussian mixture model for automated corrosion detection in remanufacturing.

Processing and Formatting the Data

The data started as **raw recorded voice samples** and was converted to **Mel-Frequency Cepstrum Coefficients (MFCCs)**², which are values found to be useful for speech recognition³.

These coefficients are calculated by first taking the discrete-time Fourier Transform of a windowed speech segment, then taking the log magnitude, and finally applying an inverse discrete Fourier transform. The resulting spectrum has peaks that represent key frequency components of the original signal³.



2. 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.

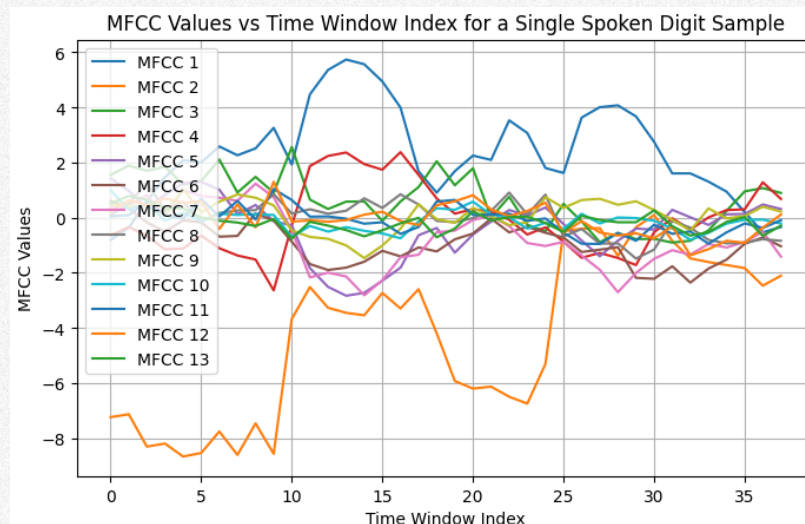
3. Alan V. Oppenheim and Ronald W. Schaffer, "From Frequency to Quefrency. A History of the Cepstrum," *IEEE Signal Processing Magazine*, 21 (5), pp. 95-99;106 (September 2004)

Our Dataset

After preprocessing, the dataset has 8,800 samples. Each sample contains the data for someone saying one of the ten digits. These samples were split into a training and testing set provided in the Spoken Arabic Digit Data Set from the UCI Machine Learning Repository². We create our model using the training set and use the test set to evaluate the model's performance.

Each sample has a varying number of time slices or frames, where each frame consists of 13 MFCC values. The MFCC values tend to form groupings over time, as you can see below. For example, there appears to be a group between time window indices 10 and 17. Each grouping theoretically represents a phoneme, which are distinct units of sound that make up a word.

Each spoken digit consists of a different set of phonemes. We will **create a model for each digit that captures the unique set of phonemes in that digit.**



Data (Feature) Modeling

We Can Use Gaussian Mixture Models to Classify Our Data

For this project, we will use **Gaussian Mixture Models (GMM)** to model our training data and make **classification predictions on our test data**.

A **GMM is a weighted combination of Gaussian distributions** represented mathematically using the equation below⁴. M represents the number of components, which are each a Gaussian distribution. Each Gaussian distribution has a weight, mean, and covariance, and is meant to model a phoneme. The model of each digit will then represent a set of phonemes.

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

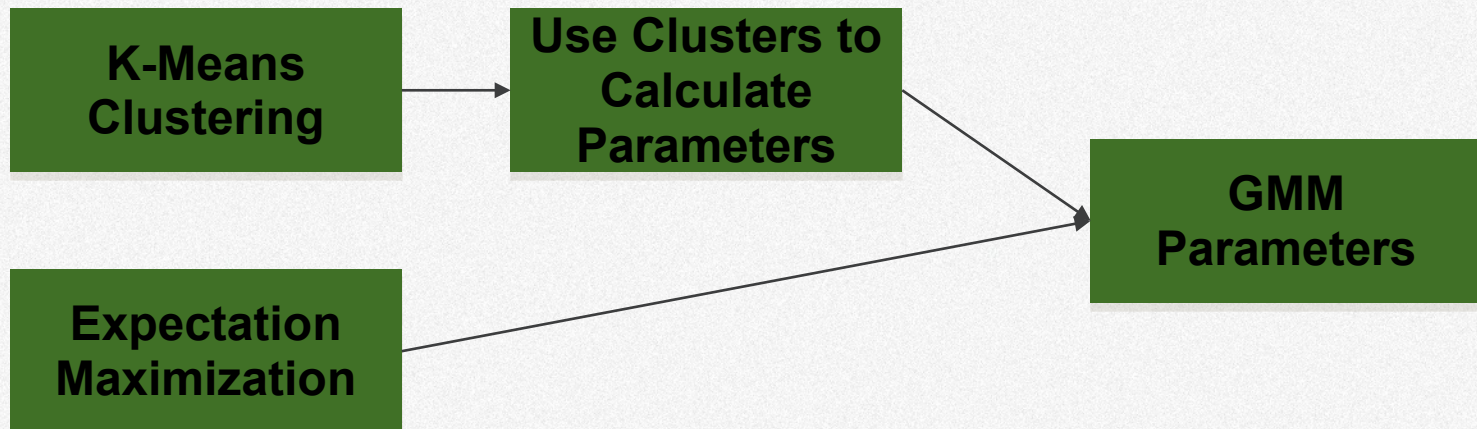
To use this model, we must pick the number of components we want, and then for each component, we'll need to estimate its weight, mean, and covariance. We can choose the same number of components as there are phonemes for each digit. In summary, we will have a **GMM for each digit**, and **the GMM will represent the digit's unique set of phonemes**.

4. Reynolds D. (2009) Gaussian Mixture Models. In: Li S.Z., Jain A. (eds) Encyclopedia of Biometrics. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-73003-5_196

There are Different Methods to Find the GMM Parameters

Given that we are going to use a GMM to model our training data, we need to find the weights, means, and covariances for each component of each digit. The two methods of finding these parameters I explored are **K-Means clustering** and **expectation maximization (EM)**. Since we'll have ten different GMMs, we'll repeat whichever method ten times, once for each digit.

I'll talk about each of these in the following slides.



K-Means Clustering

K-Means clustering⁵ can be used on a set of samples to extract k clusters separated from one another in multi-dimensional space.

The K-Means process starts with k groups, each of which is a single random point. Next, each sample is assigned to the group with the closest mean. The means are then recalculated, and the samples are reassigned to the closest group again. These two steps are repeated until the clusters converge. The formulation of this algorithm is presented below⁶ and implemented with a python package from sklearn⁷.

Algorithm 1 <i>K</i> -means clustering algorithm
Require: K , number of clusters; D , a data set of N points
Ensure: A set of K clusters
1. Initialization.
2. repeat
3. for each point p in D do
4. find the nearest center and assign p to the corresponding cluster.
5. end for
6. update clusters by calculating new centers using mean of the members.
7. until stop-iteration criteria satisfied
8. return clustering result.

5. MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam & J. Neyman (Eds.), *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297). California: University of California Press. [Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and ... - Google Books](#)

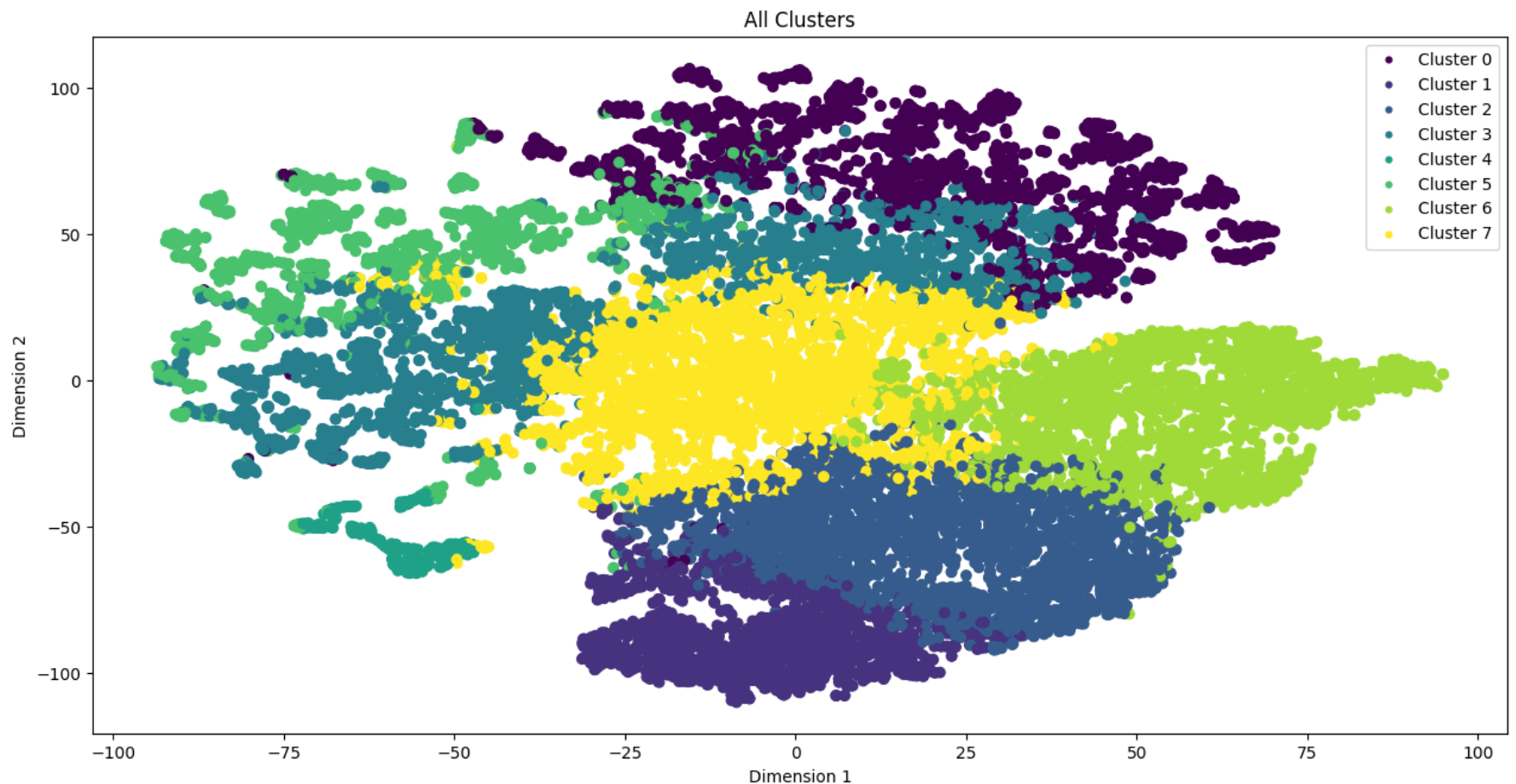
6. Jin X., Han J. (2011) *K*-Means Clustering. In: Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning*. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_425. |

7. "Sklearn.cluster.kmeans." *Scikit Learn*, <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>. Accessed 11 December 2021.

Clustering Frames Based on MFCC Values

We can use K-Means clustering to cluster the frames for a given digit. Each cluster theoretically represents a phoneme. Below, you can see a set of seven clusters for digit two, since it could be reasonable to expect seven phonemes for the Arabic word for two.

These clusters were found in 13-dimensional space since there are 13 MFCC values. For visualization purposes, the clusters were mapped to two dimensions using a method called t-SNE⁸.

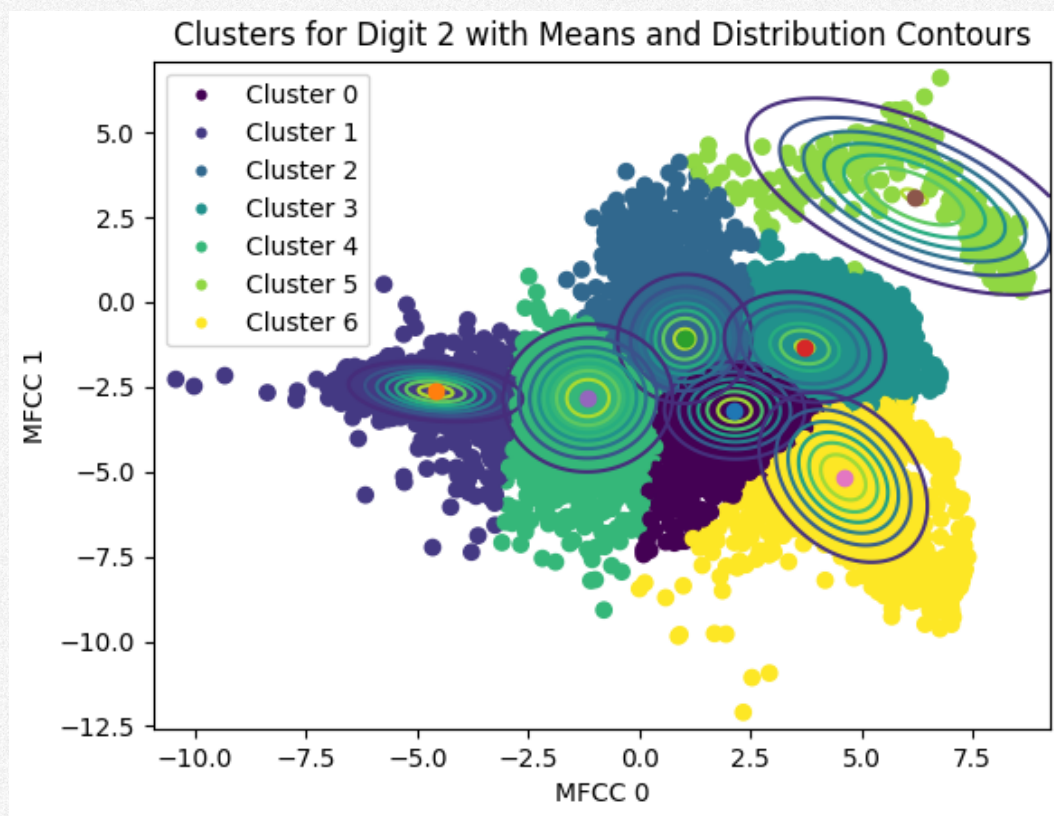


8. "Sklearn.manifold.TSNE." Scikit Learn, <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>. Accessed 11 December 2021.

GMM Parameter Estimation with K-Means

For each cluster in a digit, we can find the probability of that cluster, the mean, and the covariance. For example, for digit two, the seven clusters with their means and covariance shapes can be seen below. Here only two MFCC coefficients are used so we can visualize the contours of the distribution⁹ in 2D.

The GMM for each digit is then the weighted combination of these gaussian models for each cluster.



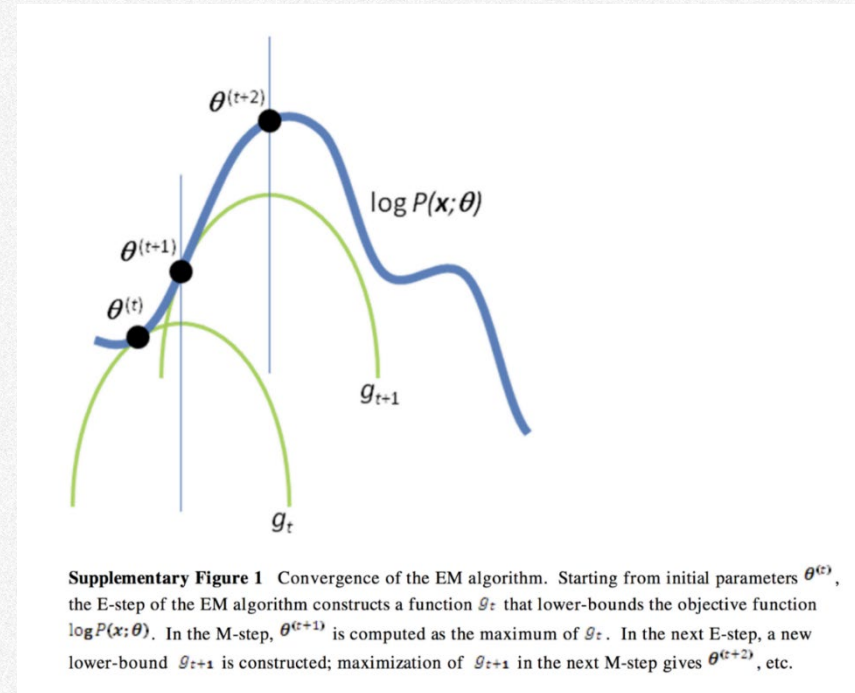
9. "Scipy.stats.multivariate_normal." Scipy.stats.multivariate_normal - SciPy v1.7.1 Manual, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multivariate_normal.html. Accessed 11 December 2021.

GMM Parameter Estimation with Expectation Maximization

We can also find the gaussian mixture model parameters for all clusters in a digit using expectation maximization (EM)^{10,11}. EM optimizes the GMM parameters directly as opposed to K-Means, which is used for clustering, and then we must manually calculate these parameters.

The diagram on the right¹² visualizes the process of EM. The parameters are first initialized, either randomly or using K-Means. Then two steps are repeated for each iteration t :

1. A lower bound function g^t is computed for the likelihood
2. Pick new parameters θ^{t+1} that maximize this lower bound



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

11. Dellaert, F. (2002). The expectation maximization algorithm (Tech. Rep.). Georgia Institute of Technology

12. "Chan, Cliburn, and Janice McCarthy. "Expectation Maximization." *Expectation Maximization - Computational Statistics in Python*, https://people.duke.edu/~ccc14/sta-663-2016/14_ExpectationMaximization.html.

GMM Parameter Estimation with Expectation Maximization

The mathematical formulation of this process is presented below ¹³.

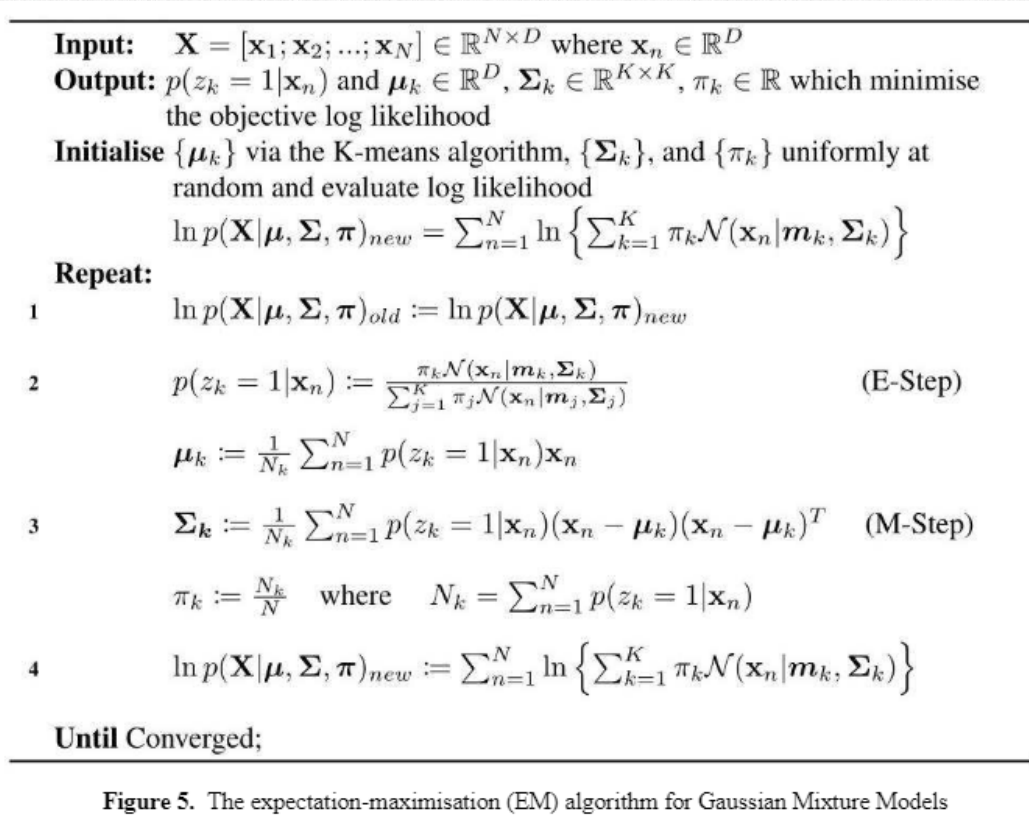


Figure 5. The expectation-maximisation (EM) algorithm for Gaussian Mixture Models

The implementation of this algorithm is taken from a Scikit Learn python package for GMMs¹⁴.

13. Gibbons, Tom & Pierce, G. & Worden, K. & Antoniadou, Ifigeneia. (2018). A Gaussian mixture model for automated corrosion detection in remanufacturing.

14. "Sklearn.mixture.gaussianmixture." Scikit Learn, <https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html>. Accessed 11 December 2021.

GMM Parameter Estimation with K-Means vs. Expectation Maximization

K-Means¹⁵:

- Advantages: Cheap to compute, scales to large data sets, guarantees convergence
- Disadvantages: Must choose k manually, is dependent on initialization, and doesn't scale well with the number of dimensions

Expectation Maximization¹⁶:

- Advantages: Guaranteed that likelihood will increase with each iteration. Numerically stable. Directly optimizes GMM parameters.
- Disadvantages: Slow convergence. converges to local optima only.

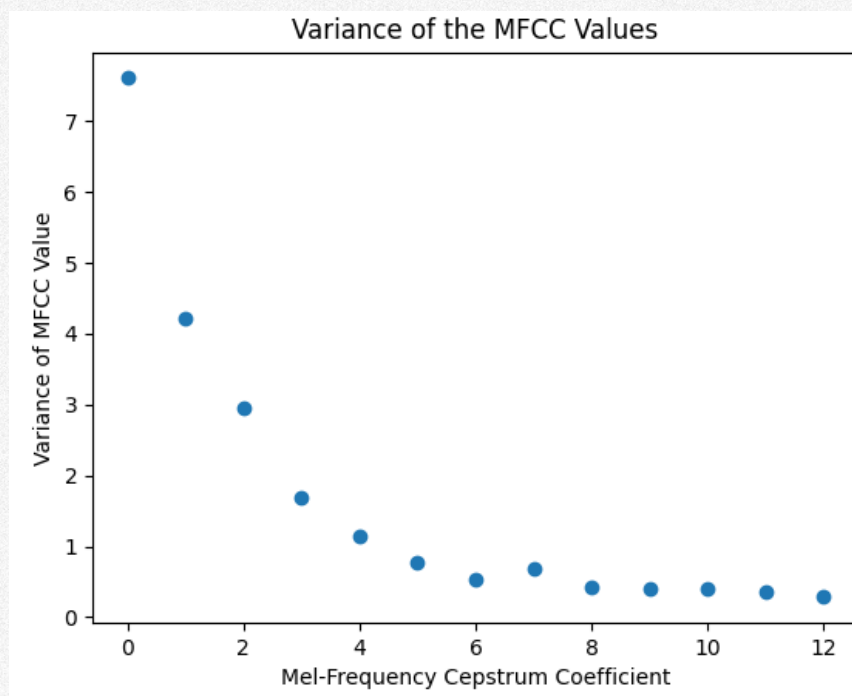
Summary: K-Means converges faster and is cheaper to compute; EM directly optimizes GMM; both dependent on initialization

15. Couvreur, Christophe. "The EM Algorithm: A Guided Tour." (1997). (5) (PDF) [The EM algorithm: A guided tour \(researchgate.net\)](https://www.researchgate.net/publication/312511111)

16. Pasi Fränti, Sami Sieranoja. How much can k-means be improved by using better initialization and repeats? Pattern Recognition, Volume 93, 2019, Pages 95-112, ISSN 0031-3203, <https://doi.org/10.1016/j.patcog.2019.04.014>.

More Modeling Options: Which MFCCs to Include

Do we include all 13 MFCCs, or do we include a subset of them? The more MFCCs used, the more information our model will have. However, with more MFCCs, we will have higher dimensional data, which could make EM or k-means parameter estimation more difficult. Some MFCCs may end up not being important, so by removing them, we could reduce computational complexity while not losing important information. By looking at the variances of the different MFCC values, we can see that the **first couple MFCC's vary significantly more than the later ones**. The MFCCs that change more might be those that are more indicative of phoneme transitions.



More Modeling Options: Number of Mixture Components



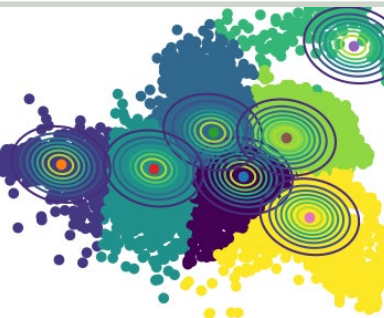
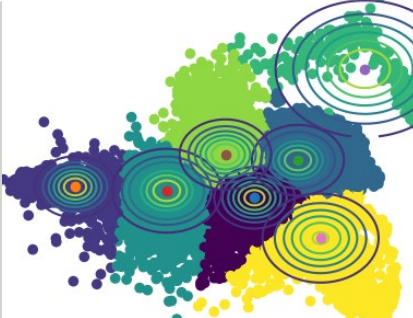
So far, we've assumed that we'll make the number of mixture components equal to the number of phonemes for the given digit, but we can experiment with this. In addition to each phoneme, we might want to have clusters represent the transitions between each phoneme. For N phonemes, we'll have $N-1$ transitions between phonemes, giving a total of $2N - 1$ clusters.

I tried to estimate the number of phonemes in each digit by listening to the pronunciations and got the results in the table below. As we'll see later, these are not ideal, and I found other cluster numbers to be better.

Digit	0	1	2	3	4	5	6	7	8	9
Phonemes (N)	4	5	4	6	4	5	4	3	6	4
Clusters ($2N-1$)	7	9	7	11	7	9	7	5	11	7

More Modeling Options: Restrictions on Covariance Matrices

We can constrain the covariance matrix for each GMM component to be a full matrix, a diagonal matrix, spherical, or constrain all covariance matrices for the clusters of each digit to be the same. By making the model more rigid, the model needs to estimate fewer parameters. This could improve the parameter estimates, especially if there is insufficient training data. The **options I explored are listed below, in order from the most flexible at the left to the most rigid at the right**. Let D equal the number of MFCCs to include and M the number of components to use.

Covariance	Full	Diagonal	Tied	Spherical
Restrictions	None	Diagonal Matrix	Equal across components	Diagonal with equal entries
Parameters to Calculate	$M \cdot (D/2 + (D^2)/2)$	$M \cdot D$	$(D/2 + (D^2)/2)$	M
Visualization				

More Modeling Options: Conditioning on Gender

We can also create **two models for each digit: one for the female speakers and the second for the male speakers**. Since gender affects the frequency of a person's voice, gender may be a confounding variable in the dataset. Making two models for each digit could provide a finer-grained prediction that could take this into account.

When we have a new digit sample to predict, we can evaluate the likelihood of all 20 models instead of 10 and predict the digit that is associated with the most likely model.

In a research paper published in 2016, the researchers used MFCC values to classify male and female speakers, with the best performing model getting a classification accuracy of around 90%¹⁷. These results indicate that **MFCC values in our dataset will depend on gender and that using separate models for each gender will provide more precision**.

Gender Identification using MFCC for Telephone Applications – A Comparative Study

Jamil Ahmad, Mustansar Fiaz, Soon-il Kwon, Maleerat Sodanil, Bay Vo, and *Sung Wook Baik

17. Ahmad, Jamil, et al. "Gender identification using mfcc for telephone applications-a comparative study." *arXiv preprint arXiv:1601.01577* (2016). Accessed 12 Dec 2021.

Maximum Likelihood Classification

Maximum Likelihood Classification

Now that we have a model based on our training data, we can now use our model to classify a new spoken digit sample using a maximum likelihood approach.

Maximum likelihood classification means to **predict the digit that maximizes the likelihood of our observed data**. This classification approach is **well-suited for our problem** because we can easily calculate the likelihood of our observed data for a GMM. It also works well because our data samples have varying lengths, so the likelihood can be found with the product of the probabilities for each time frame of the sample.

The formula below¹⁸ describes the likelihood of observed data given the GMM parameters for a single digit. For a new sample, we take each frame and find the probability that the frame is in any of the gaussian mixture model components. The likelihood of the sample is the product of all the probabilities that each frame is in the model. To get our prediction, we can **calculate the samples' likelihood for each digit and then select the digit with the highest likelihood**.

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

$$\Delta_{m,d} = (\mu_{m,d}, \Sigma_{m,d})$$

Mean, and covariance for the mth GMM component

Challenges With Underflow

Generally, the probabilities of the frames will be small. They become even smaller when we take their product for maximum likelihood classification. **Extremely small values can be difficult for a computer to store and can result in the loss of information and precision.**

To overcome this problem, I **used averaged log-likelihood¹⁹ instead of the product of the probabilities for each frame.** Averaged log-likelihood consists of taking the log of the previous expression and dividing it by the number of frames. The log function will transform small values to reasonably sized negative values monotonically so that our classification result will be the same as before. Taking the log of the expression also transforms the product in the maximum likelihood classification to a sum. Average log-likelihood removed any issues with underflow that I previously had and **significantly improved the performance of my model.**

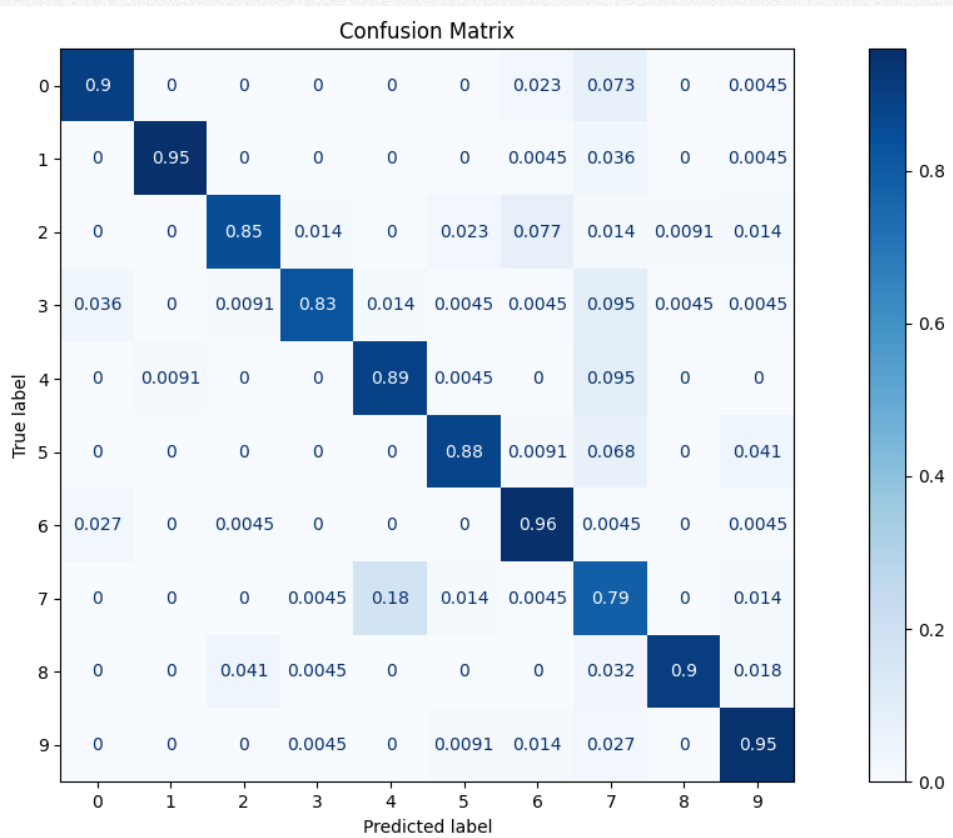
$$\ln p(\mathbf{X}|\Delta_d, \Pi_d) = \ln \prod_{n=1}^N \sum_{m=1}^M \pi_{m,d} p(\mathbf{x}_n | \Delta_{m,d}) = \sum_{n=1}^N \ln \sum_{m=1}^M \pi_{m,d} p(\mathbf{x}_n | \Delta_{m,d})$$

Average log-likelihood -> $\propto \frac{1}{N} \sum_{n=1}^N \ln \sum_{m=1}^M \pi_{m,d} p(\mathbf{x}_n | \Delta_{m,d})$

19. "Sklearn.mixture.gaussianmixture." Scikit Learn, <https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html>. Accessed 11 December 2021.

Classification Performance Results

Initial Classification Results from Both Methods are Great



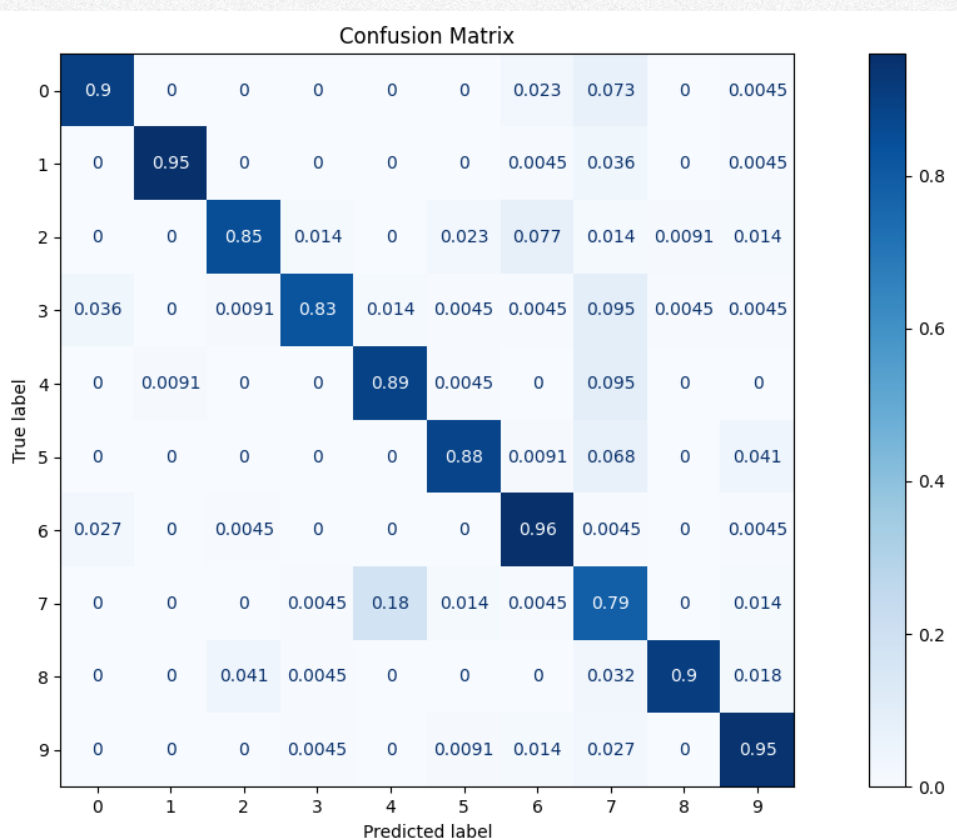
Using all 13 coefficients and GMM parameters from expectation maximization, the model's accuracy is 89.8%.

To the left is the confusion matrix²⁰ for these hyperparameters. The confusion matrix represents how well the true labels match with the predictions from the model. For example, we can see that for the digits with a true label of 7, the model predicted them correctly 79% of the time. The model predicted true 7's as a 4 18% of the time. Ideally, all the non-diagonal entries are zero, and all the diagonal entries are 1.

20. "Sklearn.metrics.confusionmatrixdisplay." Scikit Learn, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html#sklearn.metrics.ConfusionMatrixDisplay.from_predictions. Accessed 11 December 2021.

Explaining the Classification Results

You can see that the model performs best on digits 1, 6, and 9. It performs worst on digits 2, 3, and 7.



From the confusion matrix, you can see that the model often predicted digit 7 as 4. The pronunciations of digits 7 and 4 sound very similar qualitatively and seem to share two or three phonemes. Because the phonemes for these two digits are similar, this makes sense why the model often predicted 7 as 4.

The model performs well on digits 1, 6, and 9 likely because the phonemes in those digits are much different from other digits. For digit 1, this is the only one with a “w” sound, and for digits 6 and 9, their first syllable seems unique compared to the other digits.

Number of MFCC Values to Include

I experimented with various combinations and numbers of MFCC values included in the model. Every combination I experimented with performed worse than using all the MFCC values. Based on the previous graph that showed the variance of the MFCCs decreases from the 0th to 12th, so I tested combinations of MFCCs that started at zero and increased to a certain number. The results are averaged across ten runs and used diagonal covariance. The results indicate that **as the number of MFCCs included decreases, the accuracy generally decreases. The best performing model was using all 13 MFCCs.**

Indices of MFCCs Included	Average Accuracy
0-12	90.3
0-11	89.5
0-10	90.2
0-9	87.9
0-8	86.6
0-7	85.1
0-6	82.7
0-5	80.9
0-4	79.6
0-3	72.7
0-2	72.4
0-1	49.3
0	36.2

Number of Mixture Components

For each digit, we need to pick how many components to use. A colleague found different numbers of phonemes than I did, so I ran experiments to compare the performance of the number of clusters they picked versus what I had selected. These results are for EM and were averaged across all covariance types and ten runs for each covariance type.

Below are the cluster counts that my colleague had found.

Digit	0	1	2	3	4	5	6	7	8	9
Phonemes (N)	4	3	4	4	4	3	3	4	5	4
Clusters (2N-1)	7	5	7	7	7	5	5	7	9	7

I found that the cluster numbers that my colleague had produced resulted in better accuracies than mine. I also tried using a uniform number across all digits. I picked 7, which is the average number of clusters that my colleague found. I found that **the uniform number of clusters produced the best results.**

	Averaged Accuracy
Using my Set of Clusters	87.4
Colleague's Set of Clusters	88.7
Using 7 clusters for Each Digit	89.4

Expectation- Maximization Performs Better than K-Means

Below are performance results for GMMs estimated using the k-means and expectation-maximization method. These values are the accuracy evaluated on our testing set. Each of these results was averaged across ten different model initializations.

You can see that for all but the spherical covariance, the expectation-maximization method performs better than the k-means method. Averaging all these results shows that **EM performs better than k-means**, particularly for diagonal covariance.

GMM Parameter Method	Full Covariance Accuracy	Diagonal Covariance Accuracy	Tied Covariance Accuracy	Spherical Covariance Accuracy	Averaged Accuracy
K-Means	88.1	86.2	87.1	75.8	84.3
Expectation Maximization	88.5	89.4	88.1	74.6	85.2

Spherical Covariance Performs Poorly

These are the same results on the previous slide, with each result averaged over ten runs.

When comparing the four different types of covariances, I found that the **GMMs using spherical covariances performed significantly worse than the three other types of covariances**. No covariance type stood out for high performance, but **on average, the full covariance performed slightly better than diagonal and tied**. This is likely because the spherical covariance is not flexible enough to effectively model the data.

GMM Parameter Method	Full Covariance Accuracy	Diagonal Covariance Accuracy	Tied Covariance Accuracy	Spherical Covariance Accuracy
K-Means	88.1	86.2	87.1	75.8
Expectation Maximization	88.5	89.4	88.1	74.6
Averaged	88.3	87.8	87.6	75

Conditioning on Gender Results in Significant Improvements

Instead of using a GMM for each digit, we can try using two GMM's for each digit, one for each gender. For each digit, we can fit the first model to the data from the male speakers for that digit and fit the second model to the data from the female speakers.

The results below are the accuracies for various models, one set without separating on gender, and the other with separating on gender. Each result is averaged across ten runs. I found that **adding this modeling option improved the performance, particularly for the diagonal covariance and spherical covariance.**

Separating on Gender?	EM, Full Covariance	EM, Diagonal Covariance	EM, Tied Covariance	EM, Spherical Covariance	Average Accuracy
No	89.6	87.9	89.0	79.1	86.4
Yes	89.5	90.5	89.9	82.4	88.1

Conclusions

What modeling choices were important?

- **Spherical covariance type** – this modeling choice significantly affected the model's performance. Specifically choosing between spherical and the rest of the covariance types resulted in a large change in performance of around 10 percent. This is because constraining the covariance to spherical causes the model be too rigid. The model is then not flexible enough to model the data sufficiently well.
- **Choosing which MFCC values to use** – including all the MFCC values achieved the best accuracy, but when removing a significant amount of the MFCC values, the performance decreased greatly. When using the first 5 of the 13 coefficients, the performance was 10 percent lower than when using all 13 coefficients. When reducing the number of MFCCs to use, there is less information for the model to use. This information is important since the accuracy decreased as fewer MFCCs were included.
- **Using average log-likelihood** – using average log-likelihood instead of computing the product of the likelihoods for each frame drastically improved the model's performance. Before utilizing average log-likelihood, the performance of my model would not reach past 40%, but afterward, the performance reached around 90%. As mentioned previously, this is due to issues with underflow and using log-likelihood fixes these issues.

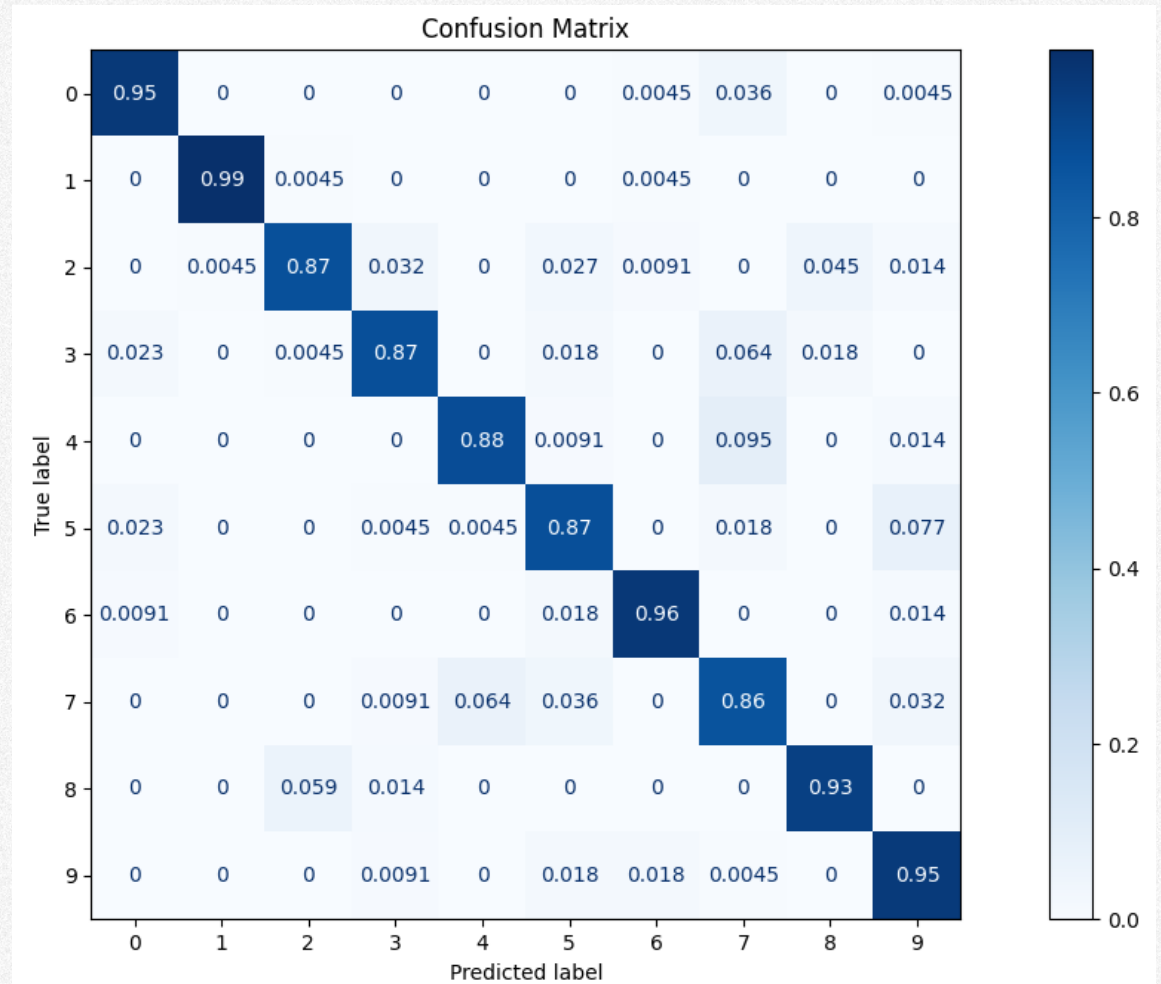
What modeling choices were not important?

- **Covariances types not including spherical** – the results showed small changes in performance when picking between full, diagonal, and tied covariances. This is because all these models are flexible enough to model the data well. When increasing the model's complexity from diagonal to full, for example, the increased model flexibility doesn't result in any significant changes in performance, which indicates that the full covariance doesn't result in overfitting.
- **Conditioning on gender** – this modeling choice was somewhat important and produced the best model. However, when looking at the average accuracy, adding this model feature only changed this metric by less than two percent. This is likely because since there are already many clusters in the GMM, those clusters could implicitly capture the different MFCC values for each gender. Adding another gender just adds more slightly more precision to the model.
- **Number of mixture components** – changing the number of mixture components resulted in changes of accuracy around 1%. Having a varying number of mixture components for each digit versus having the same number for each digit did not result in a significant change in accuracy. With four clusters for each digit, the accuracy for EM diagonal covariance was 87%. I think this is because the data for each digit can be modeled effectively with four or more clusters, so changing around the number past that point will essentially split up larger clusters into multiple smaller clusters and result in slight improvements in accuracy.

The Best Performing Model Was EM, Diagonal Covariance, Separating on Gender

I found that the best performing system that I experimented with was using **expectation maximization** to find the **GMM** parameters, using a **diagonal covariance matrix** for the **GMM**, and then creating a model for each gender.

This model achieved an accuracy of 90.5% when averaged across ten different initializations. The model associated with the confusion matrix to the right achieved a test accuracy of 91.2%.



What are Great Things About the System?

- **It trains and makes predictions quickly.** For EM and diagonal covariance, it takes around 2.3 seconds to train and 5ms to make a prediction, which is much faster than a human.
- **The methods we use to find GMM parameters are unsupervised,** meaning we don't need to label which frames are in which phonemes, for example. The data collection process simply involves collecting audio clips and preprocessing them.
- **This model works for any language.** We could apply this same model to digits for any other language and it would work just as well since the only assumption we are making is that we can determine words based on a collection of phonemes/clusters. I don't speak Arabic, but I can create this model that can classify these spoken digits for me, or for any other language that we collect data for.

What Would I do to Improve the System?

- **This model does not take any temporal information into account.** For example, if we were trying to classify “map” and “pam” using this model, the model would not be able to effectively differentiate them. This is because the phonemes for these two words will be the same but in a different temporal order. If we tried to use this for voice transcription with words instead of ten digits, then the model would likely perform poorly because of this. Something I might try is having multiple GMMs for each digit, where each GMM would be trained with a temporal subset of the frames for each digit. For example, one GMM would be trained with the first half of the frames for each sample and another GMM could be trained with the second half of the frames for each sample so that the temporal information is utilized in the model.
- **The model will perform better if we have more data.** More data provides more information for the model to be trained and improves the model’s generalizability when deploying it. Although I didn’t find significant issues with overfitting, more data can only improve the model’s performance.

Lessons Learned

What would I do differently next time?

- Trying to figure out the number of phonemes on my own did not work well. It seemed that my intuition for picking the number of clusters did not make much of a difference in the results. For specific problems such as this, having domain knowledge is important, so next time I might ask an expert about how to model the phonemes. Along with this, I think reading all of the relevant literature before working on the project would help with domain knowledge.
- Explore as many modeling options as I can. I think including a GMM for different time frames would be interesting to try next time. I did find a `BayesianGaussianMixture` package that I was interested to try but ran out of time.

What worked well, and you would do the same way next time?

- Using implementations from existing sources/packages worked very well. Particularly the `GaussianMixture` package from `scikit`, which implemented expectation-maximization and average log-likelihood.
- Collaborating was useful. It helped me validate results and discuss ideas for what modeling options to try. I learned from others about what works well for them and about modeling options that I didn't think of trying.

References

This is a compiled list of references put into categories. All of these (except some of the python packages) were referenced on their relevant slide as well.

Background and Contextual Information

Gibbons, Tom & Pierce, G. & Worden, K. & Antoniadou, Ifigeneia. (2018). A Gaussian mixture model for automated corrosion detection in remanufacturing.

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.

Alan V. Oppenheim and Ronald W. Schafer, "From Frequency to Quefrency. A History of the Cepstrum," IEEE Signal Processing Magazine, 21 (5), pp. 95-99;106 (September 2004)

GMM Model Estimation

Chan, Cliburn, and Janice McCarthy. "Expectation Maximization." Expectation Maximization - Computational Statistics in Python, https://people.duke.edu/~ccc14/sta-663-2016/14_ExpectationMaximization.html.

Reynolds D. (2009) Gaussian Mixture Models. In: Li S.Z., Jain A. (eds) Encyclopedia of Biometrics. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-73003-5_196

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam & J. Neyman (Eds.), Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, pp. 281–297). California: University of California Press. [Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and ... - Google Books](#)

Jin X., Han J. (2011) K-Means Clustering. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_425.

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

Dellaert, F. (2002).The expectation maximization algorithm(Tech. Rep.). Georgia Institute of Technology

"Chan, Cliburn, and Janice McCarthy. "Expectation Maximization." Expectation Maximization - Computational Statistics in Python, https://people.duke.edu/~ccc14/sta-663-2016/14_ExpectationMaximization.html.

Gibbons, Tom & Pierce, G. & Worden, K. & Antoniadou, Ifigeneia. (2018). A Gaussian mixture model for automated corrosion detection in remanufacturing.

Couvreur, Christophe. "The EM Algorithm: A Guided Tour." (1997). (5) (PDF) [The EM algorithm: A guided tour \(researchgate.net\)](#)

Pasi Fränti, Sami Sieranoja. How much can k-means be improved by using better initialization and repeats? Pattern Recognition, Volume 93, 2019, Pages 95-112, ISSN 0031-3203, <https://doi.org/10.1016/j.patcog.2019.04.014>.

Ahmad, Jamil, et al. "Gender identification using mfcc for telephone applications-a comparative study." *arXiv preprint arXiv:1601.01577* (2016). Accessed 12 Dec 2021.

Maximum Likelihood Classification

Tantum, Stacy. "Course Project: Recognizing Spoken Digits." Applied Probability for Statistical Learning. Fall 2021. Accessed 13 Dec. 2021.

"Sklearn.mixture.gaussianmixture." Scikit Learn, <https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html>. Accessed 11 December 2021.

Visualizations Taken From Other Sources

“Chan, Cliburn, and Janice McCarthy. “Expection Maximization.” Expection Maximization - Computational Statistics in Python, https://people.duke.edu/~ccc14/sta-663-2016/14_ExpectationMaximization.html.

Python Packages

NumPy, <https://numpy.org/>. Accessed 11 December 2021.

“Matplotlib.pyplot.” Matplotlib.pyplot - Matplotlib 3.5.0 Documentation, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html. Accessed 11 December 2021.

“Scipy.stats.multivariate_normal.” Scipy.stats.multivariate_normal - SciPy v1.7.1 Manual, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multivariate_normal.html. Accessed 11 December 2021.

“Sklearn.manifold.TSNE.” Scikit Learn, <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>. Accessed 11 December 2021.

“Sklearn.metrics.confusionmatrixdisplay.” Scikit Learn, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html#sklearn.metrics.ConfusionMatrixDisplay.from_predictions. Accessed 11 December 2021.

“Sklearn.mixture.gaussianmixture.” Scikit Learn, <https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html>. Accessed 11 December 2021.

Sklearn.cluster.kmeans.” Scikit Learn, <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>. Accessed 11 December 2021.

Collaborations

Collaborations

I talked about ideas with: Matthew Giglio, Shaan Gondalia, Eric Qi. I mostly talked with Matthew about types of modeling options. We talked about different packages to utilize and how to present results.

I shared code with: Matthew Giglio. We went over how to calculate different covariances when using K-Means clustering and the implementation for that.

I compared results with: Matthew Giglio. We compared results for different models to check that we were both getting reasonable results.

Who I helped overcome an obstacle: Matthew Giglio, Kevin Tu. I helped Kevin start on the project and describe how we use a GMM for each digit and then use K-Means or EM to find the parameters for each GMM. I helped Matthew with

Who helped me overcome an obstacle: Matthew Giglio. Matthew helped me by sharing what he had done for estimating the number of phonemes/clusters for each digit.