Maximum Likelihood Estimation for Classification of Spoken Arabic Digits

Feroze Mohideen

17 December 2018

Abstract

In this report, we investigate the efficacy of a generative model for accurate classification of spoken digits, primarily in the form of Gaussian Mixture Modeling. We discuss decisions made in model choice as well as parameter tuning, and present overall results. In addition, we offer motivation for other avenues of analysis given the temporal nature of our data.

Contents

1 Introduction

1.1 Problem Background and Motivation

The objective of this project is two-fold; firstly, we seek to accurately classify a set of spoken Arabic digits from the set {0-9}. Given data describing the frequency domain characteristics of a digit within that set at each frame that it persists in speech, our ultimate goal is to use this to translate a set of unknown frequency coefficients into its corresponding digit class. A tool of this caliber would very easily have enormous and immediate application, especially in the modern era; the selectivity for Arabic digits notwithstanding, speech translators have skyrocketed in popularity with the advent of virtual assistants in proxy of their human counterparts. Natural language processing has opened doors in a wide variety of industries including banking, entertainment, and retail, and is a core area in which research continues to be conducted by scientists worldwide.

1.2 Bayesian Inference

Aside from the scope of application that an efficient algorithm for speech recognition might have, the other pursuit of this case study is to demonstrate our knowledge of applying the generative approach to statistical classification, as we have been studying all semester. Specifically, given a model that we wish to find, we optimize the probability that our data was generated using that model. The basis of such an approach is inference; we assume that the underlying connection between training and test data, or "seen" and "unseen", is some joint probability distribution, and we make optimizations to our model such that the average loss over a set of arbitrary data elements from that distribution is minimized. Such describes the concept of statistical risk minimization.

The central approach under analysis within this work is the use of a Gaussian mixture model as a mixture distribution that represents the probability distribution of observations from the overall joint probability distribution population mentioned previously. More specifications as to the conceptual and mathematical details of GMMs, as well as their relation to K-Means, are discussed later in the model selection section.

The rest of this report concerns qualitative and quantitative descriptions of the dataset we were given, as well as discoveries and reflections that influenced model selection. The last piece of the work includes a brief exploration with the data and insights as to other areas of study, considering some limitations of possible approaches to the problem at hand.

2 Data Description

2.1 Raw Data and Cleanup

As alluded to previously, the specific data we have at our disposal is a collection of tokens describing sound bytes corresponding to a single utterance of a digit. The data was collected by the Laboratory of Automatic and Signals, from the University of Badji-Mokhtar, Annaba, Algeria [1]. Among the two provided files, **Train_Arabic_Digit.txt** and **Test_Arabic_Digit.txt**, there exist 8800 such utterances that represent 88 speakers who speak each digit from {0-9} ten times each, with the speakers being 44 males and 44 females all between the ages 18 and 40.

Within a token, the utterance is split into analysis frames, with the number of frames per utterance a variable number (intuitively, digits which take longer to enunciate are comprised of more frames). However, for all digits, each frame is represented by 13 Mel Frequency Cepstral Coefficients computed with a 16 bit sampling rate of 11025 Hz, hamming window, and 1-0.97^1 pre-emphasized filtering. These MFCC coefficients are generated through filtering of a spectrum of speech signals - usually produced from Fast Fourier Transformation (FFT) - through a group of triangular bandpass filters. The goal of such a procedure is to model the human auditory system that perceives sound using nonlinear bins of frequency. Further details of how our dataset was developed, although interesting, play little role in our decisions concerning model selection, and are thus left to the reader to explore in [2].

For convenience sake, decisions made and results obtained within this paper are derived from solely **Train_Arabic_Digit.txt**, which consists of 660 tokens for each spoken digit, with the first 330 blocks of each set representing male utterances and the rest female. Using the description of class distribution, I first extracted the data from the .txt file and afterward manually appended the following information to each line: (1) a frame number, to keep track of what frame I was in within a single token, (2) a token number, to keep track of which token of the 6600 total each line belongs to, (3) a digit, to serve as the ground-truth label of a token and therefore its corresponding frames, and (4) a gender, either "male" or "female". This format, coupled with the flexibility of the DataFrame structure of Pandas, a Python library, allowed me to easily extract entire frames, tokens, digits, and gender groups from the entire dataset. Further details about exactly how this was accomplished, along with all code written for this report, can be found in Appendix A.

From a high-level, the nature of the data at our disposal offers arguments for and against a clustering-based approach to classification. For one thing, words and therefore digits themselves are constructed from sub-word units called *phonemes*, the concept behind which motivates a grouping based approach for digits predicated on the existence of these sub-units [3]. For example, within the Arabic word for the digit "1" - *wahad* - one may isolate two such groupings: "*wa*" and "*had*" - although further decompositions of sounds within syllables are possible [4]. In fact, under such decompositions, the main issue and subject of later exploration arises - many groupings within words are simply not

unique, and cannot by themselves identify a digit. However, when considered in the context of other phonemes, and their relative occurrences with respect to time, the relationship is evident. The idea of incorporating temporal locality into model selection is explored later, but first we develop more quantitative intuition as to why groupings are even favorable.

2.2 Exploratory Data Analysis

After assembling the dataset, comprising of 6600 tokens evenly split across the 10 spoken digits, with around 260,000 rows, I proceeded to understand relationships within it in the form of visualization. The first task I sought to accomplish was to validate my intuition that the digits could be grouped into phonemes or something of the like, and hoped to see that supported by the data. A relevant plot to test that claim is one that maps the evolution in MFCCs across frames within a single token, like the one below, which only shows 3 MFCCs yet illustrates the same idea:

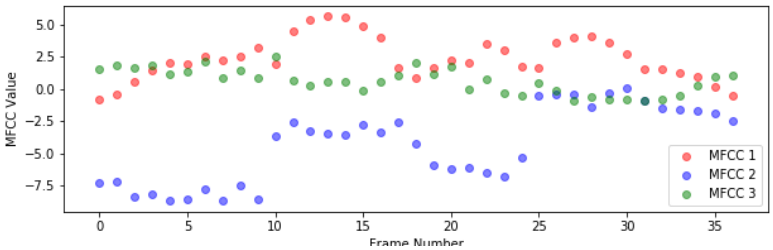


Figure 1: A plot of MFCC vs frame number for the first utterance of the digit "0". This specific utterance spans 37 frames.

As shown above, the formulation of distinct groupings within an utterance becomes somewhat apparent; one can assert that the combinations of MFCCs between frames 1-10, 10-16, 16-25, and 25-37 each represent distinct parts of the sound if not phonemes. In addition, another useful plot is one that relates each MFCC to others; perhaps there is some relationship between groupings of MFCCs for a single digit that conveys information about its identity. A relevant plot is displayed below:

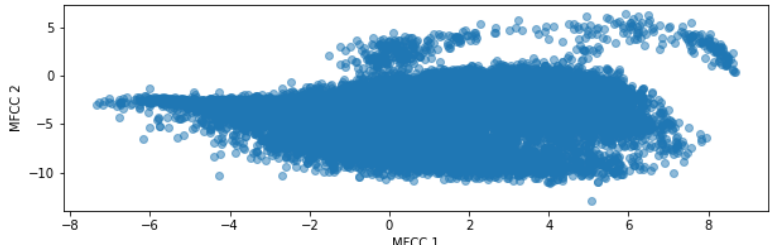


Figure 2: MFCC 1 vs MFCC 2 over all utterances of Digit "0".

The plot above doesn't support the notion of distinct groupings of MFCC values very well; rather, there instead exists one large group. None of the other

pairings of MFCCs offer support either; even if they had, there still exists the central issue of what the groups would represent. By neglecting temporal locality multiple different digits could theoretically look exactly the same when clustered. The distinguishing characteristic would be the unique transitions between clusters which lends itself to a different conceptual set of models entirely.

However, conclusions drawn from all of these 2D visualizations must be taken with a grain of salt. When selecting a model, one could choose to work with any combination of these features, or even add features if some core aspect of the problem would best be exposed by them. Following sections of this study focus on utilizing all MFCCs in 13-dimensional space, which as it turns out, leads to clusters more defined than the 2D example may predict, although the space is impossible to visualize.

3 Model Selection

In this section, I hope to break down my evolution of thought when considering models that incorporate maximum likelihood estimation. My first point of entry was understanding the K-means algorithm and how I could use clustering to my benefit, despite its limitations.

3.1 K-means Clustering

K-means is a traditional signal processing algorithm used to perform a summary of data by partitioning n observations into k clusters. Once these clusters are found, a naive classification approach would be to generate clusters for each class, then see which cluster a new data point is closest to, and label the point with the label of that cluster. Equivalently, we label the point with the label of the partition of the data space that the clusters produce. The formula for computing optimal clusters is the following minimization problem:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

where $\mu_\mathbf{i}$ is the mean of the elements in cluster S_i.

In our case, with 13 dimensional data and 10 sets of clusters (one set for each digit), one can opt to assign a new data point to the digit for which its set of frames are closest[1] to the digit's set of clusters.

However, the fact that the domain of the dataset is partitioned so discretely in this manner raises some issues as to incorporating maximum likelihood into a classifying model. Using k-means, when a new entry "falls" into a partition, we have no information as to how likely it is that the entry should be labeled with its corresponding label.

[1] "closest" in this case can refer to any distance metric, most commonly the L2

3.2 2D Clustering

Despite the limitations alluded to above and enumerated below, I decided to perform some further exploratory data analysis on features of the data in a way that we could visualize. If we again consider Figure 2, we can imagine a number of clusters to group the data, although the exact right number is difficult to ascertain due to its compactness. For a more quantitative measure, we can assess the *inertia* of a clustering, which is defined as the sum of squared distances of sample to their closest cluster center [5]. By comparing the inertia across optimal clusterings of different cluster center numbers, we can estimate which number is best suited for the data. One visual inspection can be done on the plot below:

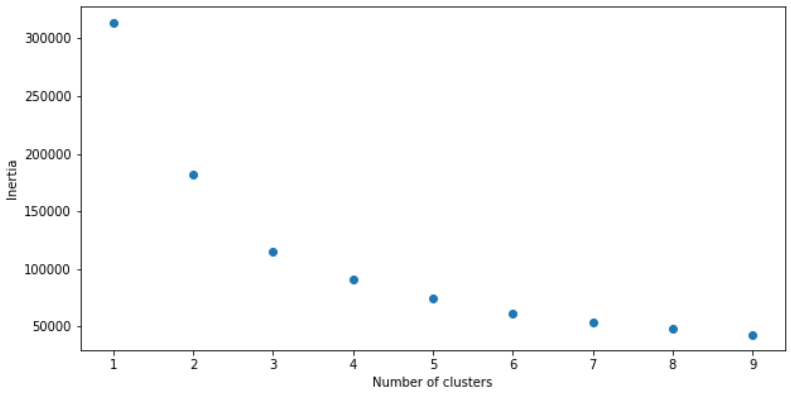


Figure 3: Inertia vs # of Clusters for 1-10 clusters.

However, it is important to note that the "best" number of clusters isn't simply the one with lowest inertia; in fact, we can achieve 0 inertia simply by making each pairing its own cluster. Rather, we seek a compromise, usually found at the "knee" of a plot like the one above; in this case, one may select 4 clusters as a result. When the number of clusters is selected, one can visualize how these clusters are arranged optimally as shown in Figure 4.
With a cluster arrangement for each digit, taking into account all 13 dimensions instead of only 2, one could proceed with the naive approach detailed above. However, for the purposes of this study, we instead move on to another approach that incorporates maximum likelihood estimation.

3.3 Motivation for a Better Solution

From an intuitive standpoint, we might expect that the clustering assignment for some points is more certain than others, but unfortunately, the K-means model has no intrinsic measure of probability or uncertainty of cluster assignments. This is similar to the concept of Maximum Likelihood Estimation vs Bayesian Inference; whereas the former provides a single value under which a probability distribution is maximized, with the latter we can obtain an entire posterior distribution that gives us much more information.

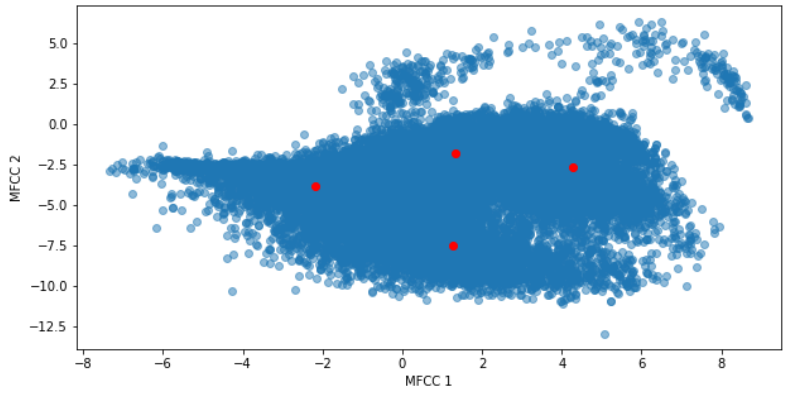


Figure 4: Identical to Figure 2, except overlaid with the 4 optimal cluster centers (red) that reduce its inertia.

One other limitation to K-means is its inability to change clusters' contributions to the partitioning of the data space. Due to the nature of its algorithm, one can imagine a cluster's "influence" on a domain as a series of concentric circles, with circles closer to the center indicating higher probability of elements within them belonging in its class, and the opposite for circles farther away. However, as is very often the case, arbitrary data may not be perfectly clustered into circles (or hyperspheres, to be more general), and thus K-means is not flexible enough to account for that. One may treat this issue by performing PCA or some kind of feature transformation such that the data aligns nicely within hyperspheres, but there are no guarantees as to the results of such an approach.

What we ultimately wish to achieve is a generalized K-means model that provides uncertainty in cluster assignment and non-circular clusters. These two features happen to to be essential to the next model choice we will explore, the Gaussian Mixture Model.

3.4 Gaussian Mixture Modeling (GMM)

Gaussian Mixture Modeling attempts to solve the problems that K-means fails to address by finding a mixture of (possibly) multi-dimensional Gaussian probability distributions that best model an input dataset. The algorithm reduces to K-means in the simplest case of finding clusters.

GMM finds its optimal clusters using Expectation-Maximization (EM), an algorithm that finds weights encoding the probability of membership in each cluster as its first step, then in the next, for each cluster, updates its location, normalization, and shape based on all data points, making use of the weights gathered from the E-step. The result places Gaussians instead of hard spheres on the

cluster center, though it may not converge under all random initializations[2]. This allows us flexibility in generating probabilities that an element belongs to a certain cluster, rather than a hard partitioning of the domain. As for arbitrary cluster shapes, GMMs account for this as well in its assignment of a covariance matrix for each cluster center.

Choosing the structure of the covariance matrix itself is no arbitrary task, however; the increase in degrees of freedom in the shape of each cluster comes with an increase in computational complexity. For the purpose of this study, however, we choose full covariance matrices, which allows each cluster to be modeled as an N-D ellipsoid with arbitrary orientation at the expense of longer runtimes.

Formally, the mathematical basis for Gaussian Mixture Modeling is to generate parameters for the following equations:

$$p(x) = \sum_{i=1}^{K} \phi_i \mathcal{N}(x \mid \mu_i, \sigma_i) \tag{1}$$

$$\mathcal{N}(x \mid \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right) \tag{2}$$

$$\sum_{i=1}^{K} \phi_i = 1 \tag{3}$$

where ϕ_i is the mixture component weight (or probability that this component exists) for each component, and μ_i and σ_i are the means and covariances of each component, respectively. Equation 3 follows from the fact that the sum of probabilities of each component existing must be 1. These parameters are calculated from the convergence of Expectation-Maximization, where the E-step parameters are generated from computing:

$$\hat{\gamma}_{ik} = \frac{\hat{\phi}_k \mathcal{N}(x_i \mid \hat{\mu}_k, \hat{\sigma}_k)}{\sum_{j=1}^{K} \hat{\phi}_j \mathcal{N}(x_i \mid \hat{\mu}_j, \hat{\sigma}_j)} \tag{4}$$

for all i, k, where $\hat{\gamma}_{ik}$ is the probability that x_i is generated from C_k.

In the M-step, we use those weights to re-initialize our parameters:

$$\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} x_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$$

$$\hat{\sigma}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} (x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$$

[2]Rather than using a random initialization for cluster centers and probabilities, a common approach, and the one followed in this report, is to initialize clusters using the result from K-means

and repeat until convergence [6].

Fortunately for us, the implementation of creating GMMs through EM has been packaged in pre-made libraries available across multiple programming languages [5], and only leave parameter selection and data input to the user, with the rest left as a "black box" [3]. I used such a package to create my GMMs, but first, I decided to play around with other traditional models the theory behind which I had more experience with; the essential goal of this pursuit was to bypass clustering entirely and test whether the classification problem of spoken digits could be solved under the idea of frame *independence*. The models I chose are not the central work of this report and are thus not described in as theoretical detail as the GMM or K-means; rather, they serve as a brief exploration for a curious mind.

4 A Brief Exploration (Part 1)

4.1 Random Forest Classification

The benefits of random forests are their lack of overfitting in general at the expense of the level of interpretability that a normal decision tree might have. To start, I split the dataset randomly into around $\frac{2}{3}$ training and $\frac{1}{3}$ test - in this case and in the next, I split all frames truly randomly because my goal was to see whether their locality within a token was significant. With an arbitrary hyperparameter of 150 trees in the forest and no cross-validation, I arrived at around 61% test accuracy. This is significantly better than the random-chance 10% probability, but still not too high.

4.2 SVM Classification

With the usual "catch-all" of an rbf kernel, an SVM seemed like a good guess for a model. Under the same train/test split as mentioned previously, I retained a test accuracy of 55%, which was again greater than random chance, but not especially high.

From these two sets of classification results, I came to the conclusion that frames should not be treated independently; rather, their role within tokens plays importance spacially and temporally. We move on to perfecting the GMM, and comparing its classification results to those under the assumption of frame independence.

5 GMM Parameter Selection

Parameter selection under the assumptions I laid above, with the GMM package used, reduced to selecting the number of components for each mixture model, which was not a trivial task. There were two optimization goals I could think of, and a number of ways of evaluating each one: (1) firstly, we could choose parameters for each digit such that the training data under those parameters has

[3] In fact, many of these libraries are open-source, allowing users to poke around in the back-end and see what kind of optimizations are made.

the highest likelihood, or (2) since our ultimate goal is accurate classification, which as specified in the next section relies on all 10 digit models, we could tune parameters according to final classification accuracy. Both approaches are considered, and are not necessarily mutually exclusive.

5.1 Information-Theoretic Criteria (BIC)

One way to select an appropriate model is to compare the Bayesian Information Criterion (BIC) for each model selection, with the one with the lowest BIC preferred. The BIC can be used to measure the efficiency of the parameterized model in terms of predicting the data using maximum likelihood. At the same time, its form penalizes model complexity, where complexity refers to the number of parameters of the model, which in our case, maps to the number of components. Its form is of the following:

$$\text{BIC} = \ln(n)k - 2\ln(\hat{L}).$$

where $\hat{L}$ is the maximized value of the likelihood function of the Model, n is the number of data points, and k is the number of parameters. The BIC has been shown to be an approximation of the Laplace approximation [7], and is useful though computationally expensive to calculate in our case, because the set of all component numbers over all covariance types is very large. As a result, Figure 5 shows a subset of possible BIC scores for the digit 0.

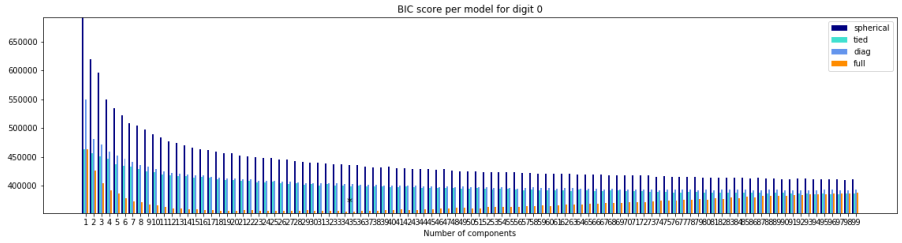


Figure 5: BIC score per number of components and covariance matrix type for covariance matrices of types in {"full", "diagonal", "spherical", and "tied"}. A larger version of this image can be found in Appendix B.

Although it is hard to make out in Figure 5, an asterisk is found at around 34 components, indicating the lowest BIC score for that model choice. Furthermore, the Figure demonstrates how low the BIC scores are for a "full" covariance matrix compared to the other types, sometimes drastically so - for this reason, we choose to only consider models with "full" covariance matrices so as to limit the ideal hyperparameter search space to only number of components.

In addition to BIC score, I chose to estimate model goodness-of-fit by monitoring the overall likelihood of the training data as the number of components increased; I expected to see some sort of drop-off where an optimal number could be found. Figure 6 shows an example of a visualization where no satisfying conclusions could be made.

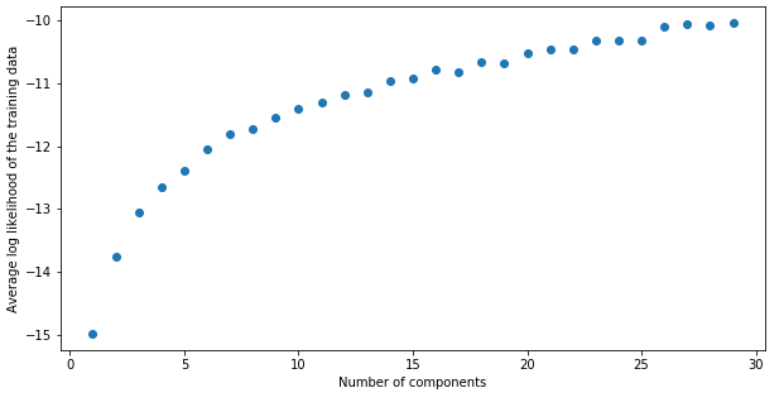


Figure 6: Average log likelihood of the training samples vs number of components used to generate the model for the digit "1".

To clarify, "log likelihood" of each sample refers to the probability of each sample given the model specified by the parameters generated from E-M, rewritten here:

$$\log(p(x \mid \vec{\mu}, \vec{\sigma})) = \log(\sum_{i=1}^{K} \phi_i \mathcal{N}(x \mid \mu_i, \sigma_i)) \tag{5}$$

The logarithm of the likelihood is used because it is computationally easier to derive, and is what the package returns in my code.

Concerning Figure 6, there is no real point at which the average likelihood stops increasing, so it was difficult to ascertain what number of components I should use. However, I realized shortly afterward that I had made a crucial mistake; by only making judgments based off the training data, I was susceptible to overfitting to it. What I needed instead, as any statistician determining the ideal hyperparameters needs, was a validation test to serve as a mock test set, and use the results on that validation set to base my assessments off of. I proceeded to do so, and using 5-fold cross-validation, I arrived at a more satisfying result, shown in Figure 7.

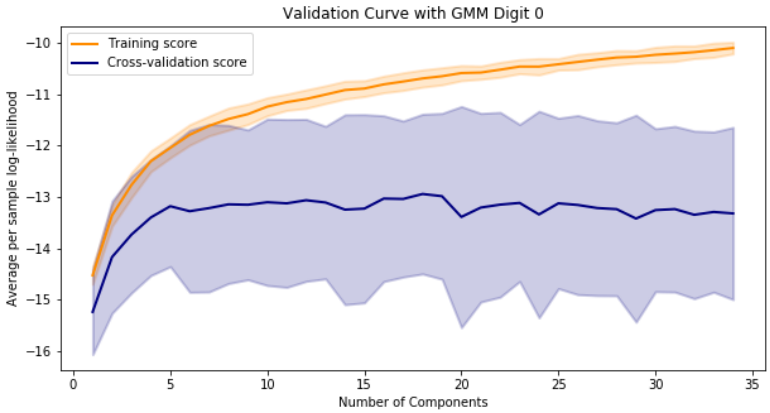


Figure 7: Average log likelihood of the training samples vs number of components with training and validation datasets for the digit "0". The shaded area represents variance and the lines signify means.

Figure 7 suggests that a fewer component model has just as much validity as one with more components, at least for the digit "0". For this reason, and for more detailed in the next subsection, I ultimately opted for a small component mixture model across all digits. Before exploring the next approach, however, it is appropriate to discuss how maximum likelihood estimate will ultimately be done across frames within the same token.

5.2 Maximum Likelihood Estimation with GMMs

Recall that the likelihood of a single sample given a GMM can be calculated by:

$$p(x \mid \vec{\mu}, \vec{\sigma}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(x \mid \mu_i, \sigma_i) \tag{6}$$

This gives us the likelihood of a single *frame* of the data, which as exhibited in the brief exploration, isn't very informative on its own. Rather, we must account for the spatial locality of frames within a token, and derive a maximum likelihood estimate for its entirety. Assuming independence of frames (the validity of such an assumption examined later), one can find the likelihood of the entire token by multiplying the likelihoods of each of its frames together:

$$p(t \mid \vec{\mu}, \vec{\sigma}) = \prod_{i=1}^{F} p(x_f \mid \vec{\mu}, \vec{\sigma}) \tag{7}$$

where t is the token and F is the number of frames. When working with log-likelihoods, the product in the above expression becomes a sum.

The procedure I followed was to compute this such token likelihood for a given test token under *all* models, and simply choose the one with the max likelihood. Technically speaking, the *score* function of the GMM package I used returned

an average log likelihood of the samples in the token, but the scaling factor of number of frames per token would not change the position of the maximum. Using this approach, I proceeded to testing a validation set in order make better judgments as to hyperparameter decisions. The test set was not touched until the very end.

5.3 Validation Accuracy

Since our ultimate goal was to classify digits correctly, rather than generate models that fit well for single digits, I used the validation set once again to simulate maximum likelihood estimation for entire tokens. However, a key detail is that I completed this *digit* by *digit*, with the following logic: for a given digit GMM, I expect that the likelihood for a token that matches the corresponding digit to be *high*, whereas the likelihood for a token that does not match the digit to be *low*. As such, a perfect classifier would be one where the likelihood under the first condition is maximized and that under the second minimized. I explored this theory by finding the average likelihoods of true positives and errors of tokens across a different number of components as shown in Figure 8. I call the difference in these numbers for each component value the "Feroze Information Criterion" (FIC), and sought a component number for each digit that maximized the FIC.

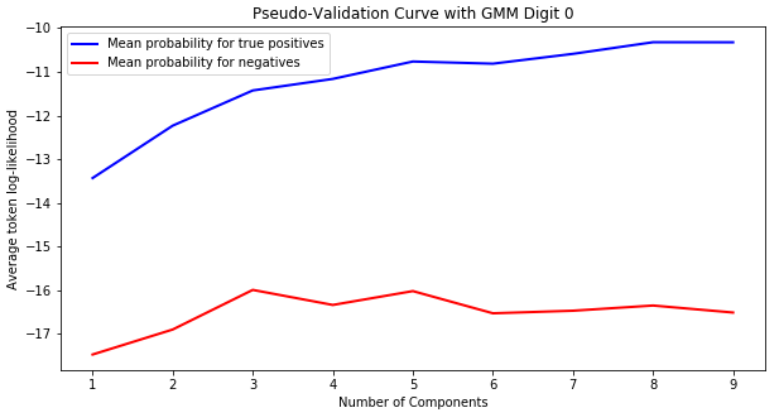


Figure 8: Average log likelihood of tokens that were correctly classified as digit "0" and those that were incorrectly classified vs number of components.

The main conclusion drawn from Figure 8, and one corroborated by earlier analysis, was that a low number of components is sufficient for modeling [4]. As such, I decided to use 4 components for all of my digit GMMs for simplicity sake, the results of which are explained in the following sections.

[4]The biggest barrier to hyperparameter search was the computational time required for some of the many-component GMMs; as a result, I often to explore a small range of components across a small range of digits. If I could start over, I would choose smaller train and validation sets to experiment on.

6 Classification Accuracy

Having decided that a low number of components for my GMMs was sufficient, I proceeded to perform maximum likelihood estimation on my test set as specified previously and compare my results to the ground-truth labels. The accuracy of the set of GMM classifiers was *much* better than I anticipated!

With a training-validation-test split of around 60-20-20, and 4 components chosen for each mixture model, I retained a test accuracy of **91%**. The summary statistics can be found in Table 1 and Figure 9, and the whole codebase for classification can be found in the Appendix.

Table 1: Precision and recall scores for each digit class

Class	Precision	Recall	F-score	Support
0.0	**0.99**	0.95	0.97	4833
1.0	**0.99**	0.98	0.98	4367
2.0	0.89	0.91	0.9	5256
3.0	0.93	0.98	0.95	5783
4.0	0.85	0.88	0.86	4640
5.0	0.88	0.96	0.92	4613
6.0	0.88	0.96	0.92	5948
7.0	0.83	0.74	0.78	5356
8.0	**0.97**	0.9	0.93	6715
9.0	0.93	0.87	0.9	4960
avg	0.91	0.91	0.91	52471

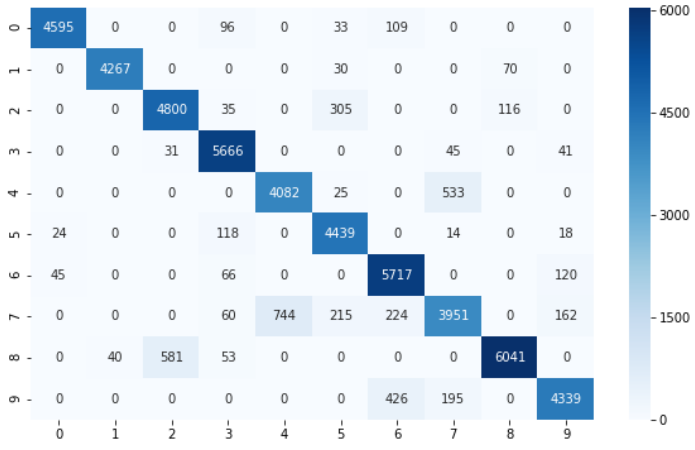


Figure 9: Heatmap displaying the confusion matrix across all digit classes.

As you can see, we reach almost perfect accuracy in classifying digits "0", "1", and "8", with high accuracy for the other digits as well. This may be due to the idiosyncrasies present within each of those digits. The digit "0", or *sifir*, is the only one of the 10 to contain an "f" sound, which may be its distinguishing characteristic. Similarly, "1" - *wahad* - is the sole owner of the "w" sound, and "8" - *thamanieh* - the sole owner of the "ni" or "nee" sound. These sounds or phonemes could represent MFCC clusters in 13-dimensional space that we did not predict in our 2D exploratory data analysis!

6.1 Gender as a Distinguishing Feature

Aside from further tuning hyperparameters, I sought another method by which classification results could improve. It seemed possible that male and female speakers could have different clusters in their MFCC patterns, and given the flexibility of my DataFrame, I moved to train and test on each gender's data separately. My results confirmed my suspicions; classification reports and confusion matrices by gender can be found in Appendix C.

With males, we reach **95%** test accuracy, with several digits being almost perfectly classified, and with females, we reach **92%** accuracy, although different digits are almost perfectly classified. This is an interesting insight; it may be the case that males and females have natural tendencies to pronounce different digits in different manners. Building off of that, a source of error that we may not have accounted for was the age groups that the speakers represent. It is certainly possible that an 18-year-old pronounces a digit through slang differently than a grown adult, so it may be helpful to know the distribution of ages as another uncertainty to integrate over.

7 A Brief Exploration (Part 2)

In the spirit of classification, I felt compelled to consider other approaches that may lead to better results, especially given the nature of the data. Although cluster classification proved to be very accurate, the temporal locality of frames almost necessitate a sequence-based model. After some research, it seems like some possibilities developed in literature include **Hidden Markov Models**, in which the system being observed is assumed to resemble a Markov chain with states; **Discrete Time Warping**, a method for measuring the similarity between two temporal sequences; and **Multi-Layer Perceptron**, a form of Artificial Neural Networks that is a very powerful classifier of patterns. Although all three methods are extremely interesting, for the purposes of this exploration I decided to focus on MLP accuracy testing.

7.1 The Multi-Layer Perceptron

Artificial Neural Networks like the MLP have been developed to model human neurons. They are excellent classifiers and can be used many pattern classification problems, especially when faced with noisy data. Today, many modern speech recognition systems use ANNs for classification of speech signals [8].

Although there is no deterministic formula behind MLPs, they all use layers of fully-connected nodes, with weights w_j along every edge to be trained. In addition, they typically use an activation function; in the Python package I used that builds MLPs [5], the default activation is RELU. The objective of the MLP is to minimize the *risk*, or average loss, over the training set through iterations of slight improvements to model parameter guesses. The loss I utilized was the cross-entropy loss, which can be represented mathematically as:

$$J = -\frac{1}{N}(\sum_{i=1}^{N} \mathbf{y_i} \cdot log(\hat{\mathbf{y}}_\mathbf{i})) \tag{8}$$

where $\mathbf{y_i}$ is the ground-truth label for a class (encoded as a one-hot vector), and $\hat{\mathbf{y}}_\mathbf{i}$) is calculated by the **softmax** function:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{9}$$

and $\mathbf{z}$ is the vector returned by the activations for each class. Before I could use the package, however, I had to restructure our dataset in a way that made sense to deploy an MLP.

7.2 Frame Dependence and Token Compression

In order to inform a new model about the temporal locality of frames within a given token, I had to modify the data such that frames were combined instead of separated. This presented an issue, however, because different tokens consisted of a different number of frames, so a combined MFCC vector would have a variable amount of elements, which the MLP can't work with. To solve this problem, I used padding, as is usually conducted in literature on neural networks; I added 0's to the end of every vector until its length was equal to the length of the maximum frame token vector. End-padding is preferable to symmetric padding in this case because of the linearly temporal characteristic of speech. From there, it was very simple to use a package and train an MLP with hyperparameters I arbitrarily chose.

7.3 Test Results

I utilize a $\frac{2}{3}$-$\frac{1}{3}$ train-test split before scaling both train and test data, as is also common in literature. The only hyperparameter I chose apart from the defaults was to use 100 hidden layers, because it has been shown that deep feed-forward networks are especially good for speech recognition [9]. The model surprisingly converged very quickly on the training data, with an example loss curve shown in Figure 10.

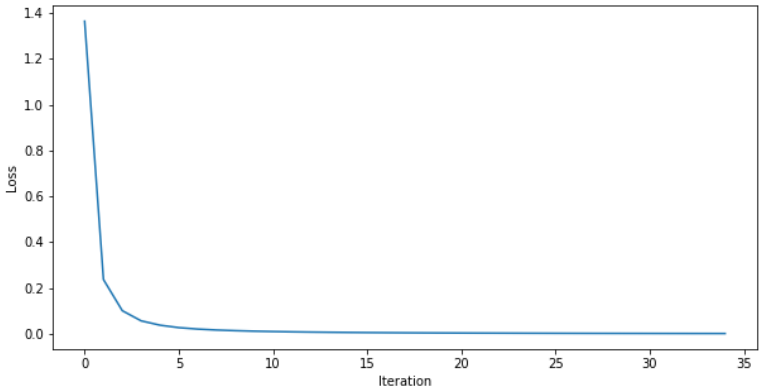


Figure 10: The model loss (in this case, cross-entropy loss) over iteration for males.

A full set of confusion matrices and classification reports can be found in Appendix D, but across the entire dataset, my test accuracy was **98.3%**, a marked increase from GMM classification. When I only considered males, however, my test accuracy was higher, at around **99.3%**. With only females considered, the test accuracy was a little lower, around **98.5%**. This slight drop compared to male accuracy affirms our earlier result, referencing that for GMM classification as well it seems harder overall to classify digits spoken by females!

In addition, it seems like our previously accurately classified digits in the GMM, like "8", have been made almost perfect classifiers in the MLP. In conclusion, the MLP represents a **discriminative model** whose results improve upon those of the **generative model** in the GMM, which is unsurprising given the temporal locality of frames that we were able to find patterns in. Other discriminative models under the same token compression and padding procedures could also be considered; we could newly revisit the Random Forest and SVM classifier mentioned earlier, but this has been left as an exercise to the reader.

8 Conclusion

I think the most surprising event that occurred during my analysis was the realization that GMMs could be used as an accurate classifier in the setting of spoken digit recognition. The 2D intuition was certainly misleading, with undefined clusters between select pairings of MFCCs, and I never would have guessed that in 13-dimensional space these clusters are more identifiable. In hindsight, the GMMs may have been a fairly useful model to use in the single spoken digit case, but it would be interesting to test its robustness against other complex cases, like entire sentences full of unique words.

Based off of literature, it seems that these models which don't take into account time sensitivity do worse in these more complicated cases. Furthermore, as I learned in my exploration, discriminative models like the neural network

take a very short time to train compared to the GMM, and retain much higher accuracy. In the future, I would love to further investigate other state-based systems like the HMM or techniques like DTW.

It was interesting to see the Validation curve in Figure 7 supporting the idea of low-component cluster models I hypothesized after generating Figure 1. The results indicate that the phoneme decomposition of digits is a valid strategy in this case, one in which transitions between phonemes are relatively unique. However, the same would not apply to other general sets of words, in which case a discriminative model as described above would prevail.

Overall, I really enjoyed this project, as it helped me build and test intuition as to the nature of data modeling, a hot topic in modern data analytics. I developed programming and testing skills that will no doubt help me in future studies and careers.

References

[1] Direction: Prof.Mouldi Bedda Participants: H.Dahmani, C.Snani, MC.Amara Korba, S.Atoui Adapted and preprocessed by : Nacereddine Hammami and Mouldi Bedda Faculty of Engineering, Al-Jouf University Sakaka, Al-Jouf Kingdom of Saudi Arabia e-mail: nacereddine.hammami '@' gmail.com mouldi_bedda '@' yahoo.fr Date: October, 2008

[2] L. Rabiner and B. H. Juang, Fundamentals of Speech Recognition, Prentice Hall, 1993

[3] R. Cox, C. Kamm, L. Rabiner, J. Schroeter, and J. Wilpon, "Speech and language processing for nextmillennium communications services", Proc. of the IEEE, vol. 88, no. 8, Aug 2000

[4] Dima Fayyad, 2018

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

[6] Gaussian Mixture Model. Brilliant.org. Retrieved from https://brilliant.org/wiki/gaussian-mixture-model.

[7] Wit, Ernst; Edwin van den Heuvel; Jan-Willem Romeyn (2012). "'All models are wrong...': an introduction to model uncertainty". Statistica Neerlandica. 66 (3): 217–236.

[8] Ahad, Abdul, Ahsan Fayyaz, and Tariq Mehmood. "Speech recognition using multilayer perceptron." Students Conference, 2002. ISCON'02. Proceedings. IEEE. Vol. 1. IEEE, 2002.

[9] Hinton, Geoffrey, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal processing magazine 29.6 (2012): 82-97.

Appendices

A Code

A.1 Data Extraction and Cleanup

```
# import packages
import pandas as pd
import numpy as np
import math

# Load in the data
data = pd.read_csv('Train_Arabic_Digit.txt', sep=" ")
data.columns = ["MFCC_1", "MFCC_2", "MFCC_3", "MFCC_4", "MFCC_5", "MFCC_6", "MFCC_7", "MFCC_8

# clean up data, add frame count, observation number, gender, and ground-truth label
arr = np.array(data['MFCC_13'])

frames = []
j = 0
for element in arr:
    if math.isnan(element):
        frames.append(np.nan)
        j = 0
    else:
        frames.append(int(j))
        j+=1
observations = []
j = 0
for element in arr:
    if math.isnan(element):
        observations.append(np.nan)
        j += 1
    else:
        observations.append(j)
data['Frame'] = frames
data['Token'] = observations

j = 0
g = 'male'
count_nans = 0
count_gen = 0
numbers = []
gender = []
for element in arr:
    if math.isnan(element):
        count_nans += 1
        count_gen += 1
        if count_gen == 330:
            if g == 'male':
                g = 'female'
            else:
                g = 'male'
            count_gen = 0
        if count_nans == 660:
            j += 1
            count_nans = 0
        numbers.append(np.nan)
        gender.append(np.nan)
    else:
```

```
        numbers.append(j)
        gender.append(g)
print(len(numbers))
data['Digit'] = numbers
data['Gender'] = gender

data = data.dropna(axis='rows')
data.head(50)
```

A.2 Exploratory Data Analysis

```
# try visualizations
import matplotlib.pyplot as plt
%matplotlib inline

# MFCC vs Frame Number
plt.figure(figsize=(10,3))
plt.scatter(data[data['Digit']==0.0]['Frame'].iloc[0:37], data[data['Digit']==0.0]['MFCC_1'].
plt.scatter(data[data['Digit']==0.0]['Frame'].iloc[0:37], data[data['Digit']==0.0]['MFCC_2'].
plt.scatter(data[data['Digit']==0.0]['Frame'].iloc[0:37], data[data['Digit']==0.0]['MFCC_3'].
plt.xlabel('Frame_Number')
plt.ylabel('MFCC_Value')
plt.legend(loc='best')

# MFCC 1 vs 2
plt.figure(figsize=(10,3))
plt.scatter(data[data['Digit']==0.0]['MFCC_1'], data[data['Digit']==0.0]['MFCC_2'], alpha=0.5
plt.xlabel('MFCC_1')
plt.ylabel('MFCC_2')
```

A.3 K-means Clustering

```
from sklearn.cluster import KMeans
import sys
from tqdm import trange

# inertia vs number of clusters
mfcc_a = 'MFCC_1'
mfcc_b = 'MFCC_2'
X = np.array(list(zip(data[data['Digit']==0.0][mfcc_a], data[data['Digit']==0.0][mfcc_b])))
inertias = []
cluster_centers = []
for k in trange(1,10):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(X)
    inertias.append(kmeans.inertia_)
    cluster_centers.append(kmeans.cluster_centers_)

plt.figure(figsize=(10,5))
plt.scatter(range(1,10), inertias)
plt.xlabel('Number_of_clusters')
plt.ylabel('Inertia')

#plot cluster centers on mfcc graph
ideal_k = 3
plt.figure(figsize=(10,5))
plt.scatter(data[data['Digit']==0.0][mfcc_a], data[data['Digit']==0.0][mfcc_b], alpha=0.5)
plt.scatter(list(zip(*cluster_centers[ideal_k]))[0], list(zip(*cluster_centers[ideal_k]))[1],
plt.xlabel('MFCC_1')
```

```
plt.ylabel('MFCC_2')
```

A.4 Random Forest and SVM Classification with Frame Independence

```
## Random Forest Classification
X = data.dropna().as_matrix(columns=['MFCC_1', 'MFCC_2', 'MFCC_3', 'MFCC_4', 'MFCC_5', 'MFCC
        'MFCC_8', 'MFCC_9', 'MFCC_10', 'MFCC_11', 'MFCC_12', 'MFCC_13'])
y = data.dropna().as_matrix(columns=['Digit'])
y = np.array(y)
y = y.ravel()


from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

clf = RandomForestClassifier(n_estimators=150)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)

## SVM Classification
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

clf = SVC(gamma='auto')
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

A.5 Information-Theoretic Criteria

```
import itertools
from scipy import linalg
import matplotlib as mpl
from sklearn import mixture


# Extract dataset
X = np.array(list(zip(X_train[X_train['Digit']==0.0]['MFCC_1'],
                      X_train[X_train['Digit']==0.0]['MFCC_2'],
                      X_train[X_train['Digit']==0.0]['MFCC_3'],
                      X_train[X_train['Digit']==0.0]['MFCC_4'],
                      X_train[X_train['Digit']==0.0]['MFCC_5'],
                      X_train[X_train['Digit']==0.0]['MFCC_6'],
                      X_train[X_train['Digit']==0.0]['MFCC_7'],
                      X_train[X_train['Digit']==0.0]['MFCC_8'],
                      X_train[X_train['Digit']==0.0]['MFCC_9'],
                      X_train[X_train['Digit']==0.0]['MFCC_10'],
                      X_train[X_train['Digit']==0.0]['MFCC_11'],
                      X_train[X_train['Digit']==0.0]['MFCC_12'],
                      X_train[X_train['Digit']==0.0]['MFCC_13'])))

# Bic score per model
lowest_bic = np.infty
bic = []
n_components_range = range(1, 100)
cv_types = ['spherical', 'tied', 'diag', 'full']
```

```
for cv_type in cv_types:
    for n_components in n_components_range:
        # Fit a Gaussian mixture with EM
        gmm = mixture.GaussianMixture(n_components=n_components,
                                      covariance_type=cv_type)
        gmm.fit(X)
        bic.append(gmm.bic(X))
        if bic[-1] < lowest_bic:
            lowest_bic = bic[-1]
            best_gmm = gmm

bic = np.array(bic)
color_iter = itertools.cycle(['navy', 'turquoise', 'cornflowerblue',
                              'darkorange'])
clf = best_gmm
bars = []

# Plot the BIC scores
plt.figure(figsize=(8, 6))
spl = plt.subplot(2, 1, 1)
for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
    xpos = np.array(n_components_range) + .2 * (i - 2)
    bars.append(plt.bar(xpos, bic[i * len(n_components_range):(i + 1) * len(n_components_rang
plt.xticks(n_components_range)
plt.ylim([bic.min() * 1.01 - .01 * bic.max(), bic.max()])
plt.title('BIC score per model')
xpos = np.mod(bic.argmin(), len(n_components_range)) + .65 +\
    .2 * np.floor(bic.argmin() / len(n_components_range))
plt.text(xpos, bic.min() * 0.97 + .03 * bic.max(), '*', fontsize=14)
spl.set_xlabel('Number of components')
spl.legend([b[0] for b in bars], cv_types)
```

A.6 Validation Curve

```
from sklearn.model_selection import validation_curve

X_0 = np.array(list(zip(X_train[X_train['Digit']==0.0]['MFCC_1'],
                        X_train[X_train['Digit']==0.0]['MFCC_2'],
                        X_train[X_train['Digit']==0.0]['MFCC_3'],
                        X_train[X_train['Digit']==0.0]['MFCC_4'],
                        X_train[X_train['Digit']==0.0]['MFCC_5'],
                        X_train[X_train['Digit']==0.0]['MFCC_6'],
                        X_train[X_train['Digit']==0.0]['MFCC_7'],
                        X_train[X_train['Digit']==0.0]['MFCC_8'],
                        X_train[X_train['Digit']==0.0]['MFCC_9'],
                        X_train[X_train['Digit']==0.0]['MFCC_10'],
                        X_train[X_train['Digit']==0.0]['MFCC_11'],
                        X_train[X_train['Digit']==0.0]['MFCC_12'],
                        X_train[X_train['Digit']==0.0]['MFCC_13'])))


param_range = range(1,35)
train_scores, test_scores = validation_curve(mixture.GaussianMixture(covariance_type='full'),
# train_scores, test_scores = -train_scores, -test_scores
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.title("Validation Curve with GMM Digit 0")
```

```
plt.xlabel("Number of Components")
plt.ylabel("Average per sample log-likelihood")
lw = 2
plt.plot(param_range, train_scores_mean, label="Training score",
             color="darkorange", lw=lw)
plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.2,
                 color="darkorange", lw=lw)
plt.plot(param_range, test_scores_mean, label="Cross-validation score",
             color="navy", lw=lw)
plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.2,
                 color="navy", lw=lw)
plt.legend(loc="best")
plt.show()
```

A.7 FIC

```
def find_best_component_num(X_train, X_val, digit, upper):
    # extract relevant training data
    X1 = np.array(list(zip(X_train[X_train['Digit']==digit]['MFCC_1'],
                           X_train[X_train['Digit']==digit]['MFCC_2'],
                           X_train[X_train['Digit']==digit]['MFCC_3'],
                           X_train[X_train['Digit']==digit]['MFCC_4'],
                           X_train[X_train['Digit']==digit]['MFCC_5'],
                           X_train[X_train['Digit']==digit]['MFCC_6'],
                           X_train[X_train['Digit']==digit]['MFCC_7'],
                           X_train[X_train['Digit']==digit]['MFCC_8'],
                           X_train[X_train['Digit']==digit]['MFCC_9'],
                           X_train[X_train['Digit']==digit]['MFCC_10'],
                           X_train[X_train['Digit']==digit]['MFCC_11'],
                           X_train[X_train['Digit']==digit]['MFCC_12'],
                           X_train[X_train['Digit']==digit]['MFCC_13'])))

    high_probs = []
    low_probs = []

    for i in trange(1,upper):
        # train each model w each number of components
        gmm = mixture.GaussianMixture(n_components=i,
                                      covariance_type='full')
        gmm.fit(X1)

        prob = []
        for i in X_val['Token']:
            temp = X_val[X_val['Token'] == i]
            temp = temp[['MFCC_1', 'MFCC_2', 'MFCC_3', 'MFCC_4', 'MFCC_5', 'MFCC_6', 'MFCC_7'
            X = np.array(list(zip(temp['MFCC_1'],
                                  temp['MFCC_2'],
                                  temp['MFCC_3'],
                                  temp['MFCC_4'],
                                  temp['MFCC_5'],
                                  temp['MFCC_6'],
                                  temp['MFCC_7'],
                                  temp['MFCC_8'],
                                  temp['MFCC_9'],
                                  temp['MFCC_10'],
                                  temp['MFCC_11'],
                                  temp['MFCC_12'],
                                  temp['MFCC_13'])))
```

```
            probs = gmm.score(X)
            prob.append(probs)

        #probs for TP and TN
        tp = []
        tn = []
        for j in range(len(prob)):
            if y_val[j] == digit:
                tp.append(prob[j])
            else:
                tn.append(prob[j])
        high_probs.append(np.mean(tp))
        low_probs.append(np.mean(tn))


    return high_probs, low_probs
high_probs, low_probs = find_best_component_num(X_train, X_val, 0, 10)
plt.title("Pseudo-Validation Curve with GMM Digit 0")
plt.xlabel("Number of Components")
plt.ylabel("Average token log-likelihood")
lw = 2
plt.plot(range(1, 10), high_probs, 'b-', lw=lw, label="Mean probability for true positives")
plt.plot(range(1, 10), low_probs, 'r-', lw=lw, label="Mean probability for negatives")
plt.legend(loc="best")
plt.show()
```

A.8 GMM Classification

```
comp=4
X_0 = np.array(list(zip(X_train[X_train['Digit']==0.0]['MFCC 1'],
                        X_train[X_train['Digit']==0.0]['MFCC 2'],
                        X_train[X_train['Digit']==0.0]['MFCC 3'],
                        X_train[X_train['Digit']==0.0]['MFCC 4'],
                        X_train[X_train['Digit']==0.0]['MFCC 5'],
                        X_train[X_train['Digit']==0.0]['MFCC 6'],
                        X_train[X_train['Digit']==0.0]['MFCC 7'],
                        X_train[X_train['Digit']==0.0]['MFCC 8'],
                        X_train[X_train['Digit']==0.0]['MFCC 9'],
                        X_train[X_train['Digit']==0.0]['MFCC 10'],
                        X_train[X_train['Digit']==0.0]['MFCC 11'],
                        X_train[X_train['Digit']==0.0]['MFCC 12'],
                        X_train[X_train['Digit']==0.0]['MFCC 13'])))
gmm_0 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_0.fit(X_0)
print("Finished training 0")
X_1 = np.array(list(zip(X_train[X_train['Digit']==1.0]['MFCC 1'],
                        X_train[X_train['Digit']==1.0]['MFCC 2'],
                        X_train[X_train['Digit']==1.0]['MFCC 3'],
                        X_train[X_train['Digit']==1.0]['MFCC 4'],
                        X_train[X_train['Digit']==1.0]['MFCC 5'],
                        X_train[X_train['Digit']==1.0]['MFCC 6'],
                        X_train[X_train['Digit']==1.0]['MFCC 7'],
                        X_train[X_train['Digit']==1.0]['MFCC 8'],
                        X_train[X_train['Digit']==1.0]['MFCC 9'],
                        X_train[X_train['Digit']==1.0]['MFCC 10'],
                        X_train[X_train['Digit']==1.0]['MFCC 11'],
                        X_train[X_train['Digit']==1.0]['MFCC 12'],
                        X_train[X_train['Digit']==1.0]['MFCC 13'])))
gmm_1 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_1.fit(X_1)
```

```
print("Finished training 1")
X_2 = np.array(list(zip(X_train[X_train['Digit']==2.0]['MFCC 1'],
                        X_train[X_train['Digit']==2.0]['MFCC 2'],
                        X_train[X_train['Digit']==2.0]['MFCC 3'],
                        X_train[X_train['Digit']==2.0]['MFCC 4'],
                        X_train[X_train['Digit']==2.0]['MFCC 5'],
                        X_train[X_train['Digit']==2.0]['MFCC 6'],
                        X_train[X_train['Digit']==2.0]['MFCC 7'],
                        X_train[X_train['Digit']==2.0]['MFCC 8'],
                        X_train[X_train['Digit']==2.0]['MFCC 9'],
                        X_train[X_train['Digit']==2.0]['MFCC 10'],
                        X_train[X_train['Digit']==2.0]['MFCC 11'],
                        X_train[X_train['Digit']==2.0]['MFCC 12'],
                        X_train[X_train['Digit']==2.0]['MFCC 13'])))
gmm_2 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_2.fit(X_2)
print("Finished training 2")
X_3 = np.array(list(zip(X_train[X_train['Digit']==3.0]['MFCC 1'],
                        X_train[X_train['Digit']==3.0]['MFCC 2'],
                        X_train[X_train['Digit']==3.0]['MFCC 3'],
                        X_train[X_train['Digit']==3.0]['MFCC 4'],
                        X_train[X_train['Digit']==3.0]['MFCC 5'],
                        X_train[X_train['Digit']==3.0]['MFCC 6'],
                        X_train[X_train['Digit']==3.0]['MFCC 7'],
                        X_train[X_train['Digit']==3.0]['MFCC 8'],
                        X_train[X_train['Digit']==3.0]['MFCC 9'],
                        X_train[X_train['Digit']==3.0]['MFCC 10'],
                        X_train[X_train['Digit']==3.0]['MFCC 11'],
                        X_train[X_train['Digit']==3.0]['MFCC 12'],
                        X_train[X_train['Digit']==3.0]['MFCC 13'])))
gmm_3 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_3.fit(X_3)
print("Finished training 3")
X_4 = np.array(list(zip(X_train[X_train['Digit']==4.0]['MFCC 1'],
                        X_train[X_train['Digit']==4.0]['MFCC 2'],
                        X_train[X_train['Digit']==4.0]['MFCC 3'],
                        X_train[X_train['Digit']==4.0]['MFCC 4'],
                        X_train[X_train['Digit']==4.0]['MFCC 5'],
                        X_train[X_train['Digit']==4.0]['MFCC 6'],
                        X_train[X_train['Digit']==4.0]['MFCC 7'],
                        X_train[X_train['Digit']==4.0]['MFCC 8'],
                        X_train[X_train['Digit']==4.0]['MFCC 9'],
                        X_train[X_train['Digit']==4.0]['MFCC 10'],
                        X_train[X_train['Digit']==4.0]['MFCC 11'],
                        X_train[X_train['Digit']==4.0]['MFCC 12'],
                        X_train[X_train['Digit']==4.0]['MFCC 13'])))
gmm_4 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_4.fit(X_4)
print("Finished training 4")
X_5 = np.array(list(zip(X_train[X_train['Digit']==5.0]['MFCC 1'],
                        X_train[X_train['Digit']==5.0]['MFCC 2'],
                        X_train[X_train['Digit']==5.0]['MFCC 3'],
                        X_train[X_train['Digit']==5.0]['MFCC 4'],
                        X_train[X_train['Digit']==5.0]['MFCC 5'],
                        X_train[X_train['Digit']==5.0]['MFCC 6'],
                        X_train[X_train['Digit']==5.0]['MFCC 7'],
                        X_train[X_train['Digit']==5.0]['MFCC 8'],
                        X_train[X_train['Digit']==5.0]['MFCC 9'],
                        X_train[X_train['Digit']==5.0]['MFCC 10'],
                        X_train[X_train['Digit']==5.0]['MFCC 11'],
                        X_train[X_train['Digit']==5.0]['MFCC 12'],
                        X_train[X_train['Digit']==5.0]['MFCC 13'])))
```

```
gmm_5 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_5.fit(X_5)
print("Finished training 5")
X_6 = np.array(list(zip(X_train[X_train['Digit']==6.0]['MFCC 1'],
                        X_train[X_train['Digit']==6.0]['MFCC 2'],
                        X_train[X_train['Digit']==6.0]['MFCC 3'],
                        X_train[X_train['Digit']==6.0]['MFCC 4'],
                        X_train[X_train['Digit']==6.0]['MFCC 5'],
                        X_train[X_train['Digit']==6.0]['MFCC 6'],
                        X_train[X_train['Digit']==6.0]['MFCC 7'],
                        X_train[X_train['Digit']==6.0]['MFCC 8'],
                        X_train[X_train['Digit']==6.0]['MFCC 9'],
                        X_train[X_train['Digit']==6.0]['MFCC 10'],
                        X_train[X_train['Digit']==6.0]['MFCC 11'],
                        X_train[X_train['Digit']==6.0]['MFCC 12'],
                        X_train[X_train['Digit']==6.0]['MFCC 13'])))
gmm_6 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_6.fit(X_6)
print("Finished training 6")
X_7 = np.array(list(zip(X_train[X_train['Digit']==7.0]['MFCC 1'],
                        X_train[X_train['Digit']==7.0]['MFCC 2'],
                        X_train[X_train['Digit']==7.0]['MFCC 3'],
                        X_train[X_train['Digit']==7.0]['MFCC 4'],
                        X_train[X_train['Digit']==7.0]['MFCC 5'],
                        X_train[X_train['Digit']==7.0]['MFCC 6'],
                        X_train[X_train['Digit']==7.0]['MFCC 7'],
                        X_train[X_train['Digit']==7.0]['MFCC 8'],
                        X_train[X_train['Digit']==7.0]['MFCC 9'],
                        X_train[X_train['Digit']==7.0]['MFCC 10'],
                        X_train[X_train['Digit']==7.0]['MFCC 11'],
                        X_train[X_train['Digit']==7.0]['MFCC 12'],
                        X_train[X_train['Digit']==7.0]['MFCC 13'])))
gmm_7 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_7.fit(X_7)
print("Finished training 7")
X_8 = np.array(list(zip(X_train[X_train['Digit']==8.0]['MFCC 1'],
                        X_train[X_train['Digit']==8.0]['MFCC 2'],
                        X_train[X_train['Digit']==8.0]['MFCC 3'],
                        X_train[X_train['Digit']==8.0]['MFCC 4'],
                        X_train[X_train['Digit']==8.0]['MFCC 5'],
                        X_train[X_train['Digit']==8.0]['MFCC 6'],
                        X_train[X_train['Digit']==8.0]['MFCC 7'],
                        X_train[X_train['Digit']==8.0]['MFCC 8'],
                        X_train[X_train['Digit']==8.0]['MFCC 9'],
                        X_train[X_train['Digit']==8.0]['MFCC 10'],
                        X_train[X_train['Digit']==8.0]['MFCC 11'],
                        X_train[X_train['Digit']==8.0]['MFCC 12'],
                        X_train[X_train['Digit']==8.0]['MFCC 13'])))
gmm_8 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_8.fit(X_8)
print("Finished training 8")
X_9 = np.array(list(zip(X_train[X_train['Digit']==9.0]['MFCC 1'],
                        X_train[X_train['Digit']==9.0]['MFCC 2'],
                        X_train[X_train['Digit']==9.0]['MFCC 3'],
                        X_train[X_train['Digit']==9.0]['MFCC 4'],
                        X_train[X_train['Digit']==9.0]['MFCC 5'],
                        X_train[X_train['Digit']==9.0]['MFCC 6'],
                        X_train[X_train['Digit']==9.0]['MFCC 7'],
                        X_train[X_train['Digit']==9.0]['MFCC 8'],
                        X_train[X_train['Digit']==9.0]['MFCC 9'],
                        X_train[X_train['Digit']==9.0]['MFCC 10'],
                        X_train[X_train['Digit']==9.0]['MFCC 11'],
```

```
                        X_train[X_train['Digit']==9.0]['MFCC_12'],
                        X_train[X_train['Digit']==9.0]['MFCC_13'])))
gmm_9 = mixture.GaussianMixture(n_components=comp, covariance_type='full')
gmm_9.fit(X_9)
print("Finished_training_9")

arr=np.array(X_token)
max_prob = []

for i in range(len(X_test['Token'])):
    # this method is inefficient in that it loops through tokens once for each of their frame
    # however, it doesn't not work
    temp = X_test[X_test['Token'] == X_test['Token'].iloc[i]]
    temp = temp[['MFCC_1', 'MFCC_2', 'MFCC_3', 'MFCC_4', 'MFCC_5', 'MFCC_6', 'MFCC_7', 'MFCC_
    X = np.array(list(zip(temp['MFCC_1'],
                        temp['MFCC_2'],
                        temp['MFCC_3'],
                        temp['MFCC_4'],
                        temp['MFCC_5'],
                        temp['MFCC_6'],
                        temp['MFCC_7'],
                        temp['MFCC_8'],
                        temp['MFCC_9'],
                        temp['MFCC_10'],
                        temp['MFCC_11'],
                        temp['MFCC_12'],
                        temp['MFCC_13'])))
    probs = np.array([gmm_0.score(X),
                        gmm_1.score(X),
                        gmm_2.score(X),
                        gmm_3.score(X),
                        gmm_4.score(X),
                        gmm_5.score(X),
                        gmm_6.score(X),
                        gmm_7.score(X),
                        gmm_8.score(X),
                        gmm_9.score(X)])
    max_prob.append(np.argmax(probs))

y_test = np.array(y_test)

cm = confusion_matrix(y_test, max_prob)
plt.figure(figsize=(15,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
print(classification_report(y_test, max_prob))
```

A.9 Token Compression and Padding

```
# separate by gender to reduce uncertainty
male_data = data[data['Gender']=='female'] # can change to any metric
male_mfcc =  male_data[['MFCC_1', 'MFCC_2', 'MFCC_3', 'MFCC_4', 'MFCC_5', 'MFCC_6', 'MFCC_7',

max_frame = male_data['Frame'].max()
# max_frame = 100
X = []
y = []
for token in male_data['Token'].unique():
    temp = male_data[male_data['Token'] == token]
    y.append(temp.iloc[0]['Digit'])
    frames = temp.shape[0]
    vec = []
    temp = temp[['MFCC_1', 'MFCC_2', 'MFCC_3', 'MFCC_4', 'MFCC_5', 'MFCC_6', 'MFCC_7', 'MFCC
    vec = temp.values.flatten()

    if frames < max_frame:
        num_zeros = 13*(max_frame-frames)
        vec = np.pad(vec, (0,int(num_zeros)), 'constant', constant_values=0)
    X.append(vec)

new_X = []
new_y = []
for i,j in enumerate(X):
    if len(j) == max_frame*13:
        new_X.append(j)
        new_y.append(y[i])
X = new_X
y = new_y

X = np.array(X)
y = np.array(y)
```

A.10 MLP Fitting and Scoring

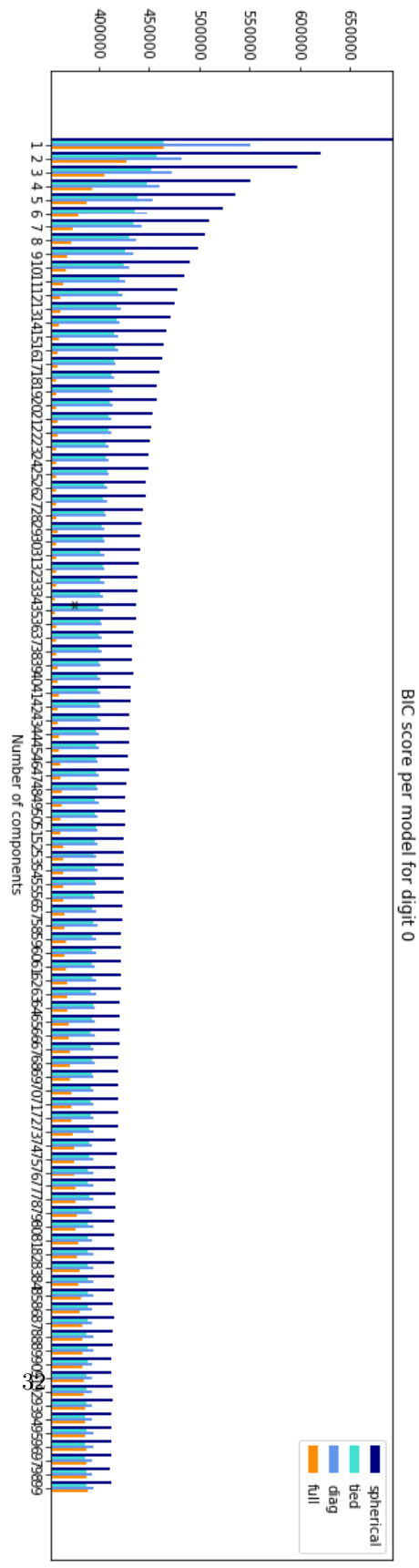
```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)

clf = MLPClassifier(hidden_layer_sizes=(100,), random_state=1, verbose=1).fit(X_train,y_train
print("Accuracy: " + str(clf.score(X_test,y_test)))
```

B BIC score per covariance matrix type and component number



C GMM Classification Results by Gender

Table 2: Precision and recall scores for each digit class for males.

Class	Precision	Recall	F-score	Support
0.0	0.95	0.96	0.96	1797
1.0	**0.97**	0.98	0.98	1946
2.0	**0.98**	0.89	0.94	3076
3.0	**0.99**	0.92	0.95	2700
4.0	**1.0**	0.97	0.99	2852
5.0	0.91	1.0	0.96	2251
6.0	0.93	0.98	0.95	2455
7.0	0.88	0.96	0.92	2406
8.0	0.89	0.97	0.93	3113
9.0	**0.98**	0.84	0.9	2257
avg	0.95	0.95	0.95	24853

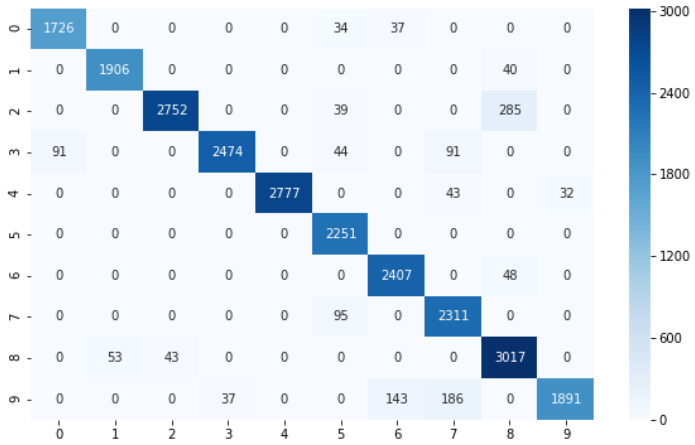


Figure 11: Heatmap displaying the confusion matrix across all digit classes for males.

Table 3: Precision and recall scores for each digit class for females.

Class	Precision	Recall	F-score	Support
0.0	**0.98**	0.92	0.95	2150
1.0	**1.0**	1.0	1.0	2015
2.0	0.91	0.95	0.93	2523
3.0	0.93	0.97	0.95	2892
4.0	0.86	0.86	0.86	3157
5.0	0.92	0.98	0.95	2319
6.0	0.9	1.0	0.95	2620
7.0	0.78	0.76	0.77	2387
8.0	**0.98**	0.89	0.93	3486
9.0	0.94	0.88	0.91	3024
avg	0.92	0.92	0.92	26573

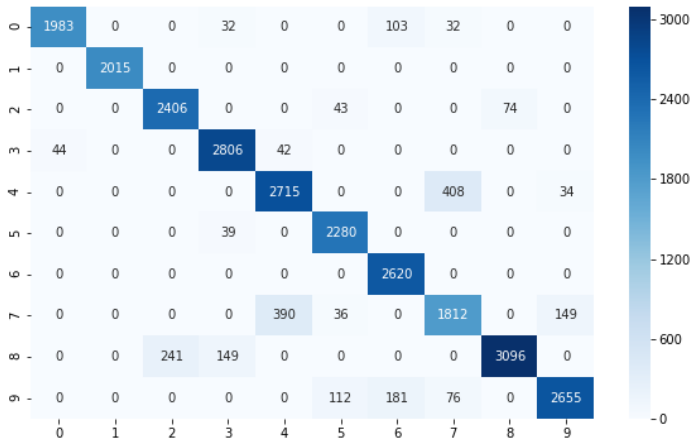


Figure 12: Heatmap displaying the confusion matrix across all digit classes for females.

D MLP Classification Results

Table 4: Precision and recall scores for each digit class over all data.

Class	Precision	Recall	F-score	Support
0.0	0.98	0.99	0.99	237
1.0	**1.0**	1.0	1.0	217
2.0	**1.0**	0.99	0.99	229
3.0	0.98	1.0	0.99	208
4.0	0.99	0.99	0.99	212
5.0	0.97	0.99	0.98	220
6.0	0.97	0.96	0.96	202
7.0	0.97	0.96	0.97	227
8.0	0.99	0.99	0.99	226
9.0	0.98	0.98	0.98	200
avg	0.98	0.98	0.98	2178

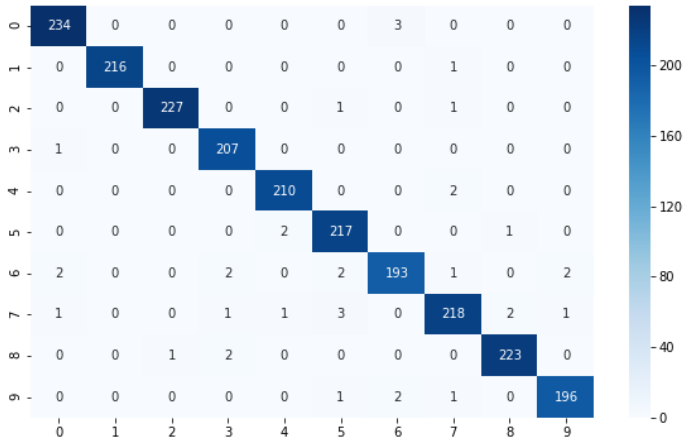


Figure 13: Heatmap displaying the confusion matrix across all digit classes over all data.

Table 5: Precision and recall scores for each digit class for males.

Class	Precision	Recall	F-score	Support
0.0	**1.0**	0.99	1.0	124
1.0	**1.0**	0.99	1.0	114
2.0	**1.0**	1.0	1.0	103
3.0	0.98	1.0	0.99	117
4.0	0.98	1.0	0.99	110
5.0	0.98	0.98	0.98	100
6.0	0.99	0.99	0.99	92
7.0	**1.0**	0.98	0.99	106
8.0	0.99	1.0	1.0	109
9.0	**1.0**	0.99	1.0	114
avg	0.99	0.99	0.99	1089

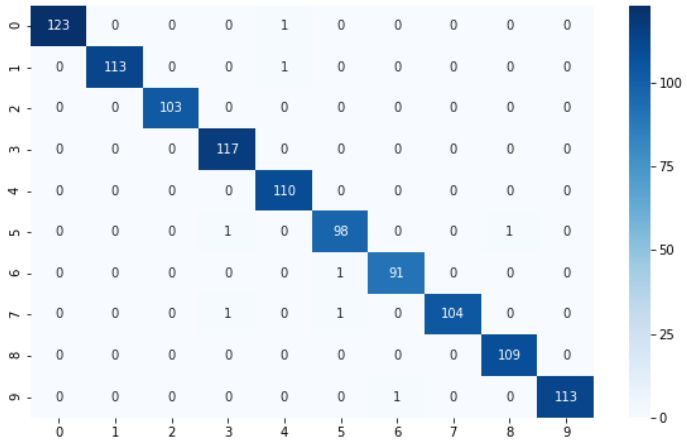


Figure 14: Heatmap displaying the confusion matrix across all digit classes for males.

Table 6: Precision and recall scores for each digit class for females.

Class	Precision	Recall	F-score	Support
0.0	0.98	0.98	0.98	124
1.0	0.98	0.99	0.99	114
2.0	0.99	1.0	1.0	103
3.0	**1.0**	0.98	0.99	117
4.0	0.98	0.99	0.99	110
5.0	0.98	0.97	0.97	99
6.0	0.98	0.98	0.98	93
7.0	0.98	0.98	0.98	106
8.0	**1.0**	1.0	1.0	109
9.0	0.98	0.98	0.98	114
avg	0.99	0.99	0.99	1089

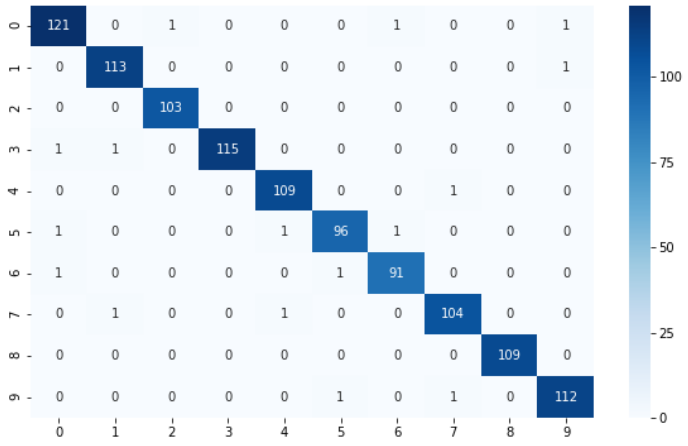


Figure 15: Heatmap displaying the confusion matrix across all digit classes for females.