An Empirical Study: Separating Gaussians
How Covariance Estimations Affect Learnability

Chang Rey Tang

March 7, 2018

1 Introduction

The Gaussian distribution is arguably the most famous and widely applied continuous probability distribution in statistics. It's flexible properties have been widely applied across a variety of fields not only in statistics, but from social sciences to engineering. With the help of the central limit theorem, which states that the mean value of independent observations of a random variable from independent distributions converge to the Gaussian, it is extremely useful in a number of cases to estimate real-valued random variables whose distributions are not known.

The popularity of this distribution has spawned numerous studies to further understand new applications of the Gaussian. Specifically, we can apply the Gaussian to help us understand latent properties of sub-populations within the overall population. The most common example of this is finding the underlying distributions of male and female heights within the overall distribution of heights within an area. If we viewed the distributions of male and females heights each as their own Gaussian then what we are really seeking to do is to separate these two Gaussians. This is the problem that we will be tackling in the next few sections. We want to see how this problem may become harder or easier as the number of underlying sub-populations, dimensionality of the data, and a number of other factors change.

2 Problem

The problem of separating Gaussians is no easy task. At the core, it is a *parameter estimation problem*: we are given access to samples that are drawn from an overall distribution D, viewed to be a mixture of Gaussians, and we seek to reconstruct the parameters of the original distribution from these samples. This means that the original distribution can be seen as a combination of different component Gaussians, each with their own mean vector and covariance matrix, and a weight w associated with each. The mixture weights, whose probabilities sum up to 1, describe the proportion which each component affects the overall population distribution.

2.1 Separating Gaussians

To formalize the problem of separating Gaussians we can view the natural model of the encompassing population, $p(x)$, as a combination of K component Gaussians: $p(x) = w_1 p_1(x) + w_2 p_2(x) + ... + w_k p_k(x)$. This is commonly known as the Gaussian Mixture Model (GMM) which is a parametric probability density function represented as a weighted sum of Gaussian component densities. The $p_1(x), p_2(x), ..., p_k(x)$ are the Gaussian densities that represent the subpopulations that we are trying to separate. Each component has their own mean, μ_i, covariance matrix, Σ_i, and mixture weight, w_i, which describe it's Gaussian distribution and the influence it has on the encompassing population distribution. This means that the main problem of separating Gaussians is accurately estimating the parameters w_i, μ_i, Σ_i for $i = 1, ..., K$.

2.2 Expectation-Maximization Algorithm

A popular approach to estimating the parameters of the GMM from training data is the iterative Expectation-Maximization (EM) algorithm that has taken its inspiration from the K-Means clustering model. First, we will formalize the GMM parameters we are attempting to estimate over K components as ρ.

$$\rho = \{w_i, \mu_i, \Sigma_i\}, i = 1, ..., K$$

With these parameters in mind, we can view the GMM as the weighted sum of K component Gaussian densities given below:

$$p(x|\rho) = \sum_{i=1}^{K} w_i f(x|\mu_i, \Sigma_i)$$

Each component Gaussian is represented as a D-variate Gaussian function, $f(x|\mu_i, \Sigma_i)$, which we all know and love:

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

The Gaussian Mixture Model can be seen as an improved version of the basic K-Means Vector Quantization algorithm, because it not only has a mean μ_i but also an associated covariance matrix Σ_i which makes it a true PDF. The Vector Quantization (VQ) algorithm is already very powerful for identifying the density of high-dimensional data but each data point is just represented by it's closest centroid. This means VQ is based on distances while GMM is based on probabilities. Given GMM is a true PDF, it can be seen as a Bayesian classifier.

The most popular method of estimating ρ is utilizing Maximum Likelihood Estimation (MLE). MLE attempts to find our GMM parameters ρ by maximizing some likelihood function given T number of observations $X = \{x_1, x_2, .., x_T\}$. In our case, the GMM likelihood function we are attempting to maximize is:

$$p(X|\rho) = \Pi_{t=1}^{T} p(x_t|\rho)$$

Since this is a non-linear function of the parameters ρ we can't use direct maximization. However, this is where the Estimation-Maximization algorithm comes in to help us *iteratively* obtain the Maximum Likelihood parameter estimates. VQ still plays a role in the EM algorithm because the initial model ρ is usually obtained using the VQ algorithm. You can initialize μ_i using VQ but the initial values for Σ_i is usually just an identity matrix and the weights w_i is 1/K. The basic idea of EM is to begin with a initial model ρ and to estimate a new model ρ' such that $p(X|\rho') \geq p(X|\rho)$ therefore slowly maximizing the GMM likelihood function. The new model ρ' becomes the initial model for the next iteration and the process is repeated until some convergence threshold is reached.

The EM algorithm is essentially a bound-based optimizer for the GMM likelihood function. In the Expectation-step it constructs a lower bound of the log-likelihood function which is correct at the current parameters. This means that the GMM has a "soft" cluster membership such that each data point belongs to some component p_i to some percentage amount. In other words, each point contributes to every single component p_i to some extent, which is very different from the K-Means algorithm described before such that each data point has a component membership of either true or false. The Maximization-step is what moves the parameters ρ to the new ones ρ' that maximize the lower bound constructed in the E-step. On each iteration of EM, we use the *a posteriori* probability for a component to re-estimate the parameters such that we guarantee a monotonic increase in the model's likelihood value. This is called the maximum a posteriori probability (MAP) estimate of an unknown quantity and calculates the conditional probability that is assigned after the observation is taken into account. The *a posteriori* probability for component i is formalized by:

$$P(i|x_t, \rho) = \frac{w_i f(x|\mu_i, \Sigma_i)}{\sum_{k=1}^{K} w_k f(x|\mu_k, \Sigma_k)}$$

Using this *a posteriori* probability we can update the GMM parameters:

$$\text{Mixture Weights: } w_i' = \frac{1}{T} \sum_{t=1}^{T} P(i|x_t, \rho)$$

$$\text{Means: } \mu_i' = \frac{\sum_{t=1}^{T} P(i|x_t, \rho) x_t}{\sum_{t=1}^{T} P(i|x_t, \rho)}$$

The last parameter we need to update is the covariance matrix Σ_i but this is where we have more than one option and also the main parameter we will be studying in this paper.

2.3 Covariance Type

There are several *covariance types* we can choose from to update the parameter Σ_i and this configuration is usually determined by the amount of data available and the specific application of the GMM. This is one of the most important

concepts in GMM because this is what distinguishes it from K-Means. The addition of the covariance matrix is what makes it a true PDF and a Bayesian classifier. The covariance matrices can either be full rank, diagonal, spherical, or tied and we will be testing how each of these perform and scale with relation to training dataset size T, dimension d, and number of components K.

2.3.1 Full Covariance

The full rank covariance matrix configuration usually fits data the best but it is very costly in high-dimensional feature spaces. It is especially good at fitting component Gaussians that are dependent on each other because the covariance matrix itself includes the relationship that these components have with each other. As we can see in Figure 1, it produces a pretty tight fit on the training dataset. A possible drawback of this approach is that it can lose the ability to generalize the learned model because of the tight fit.

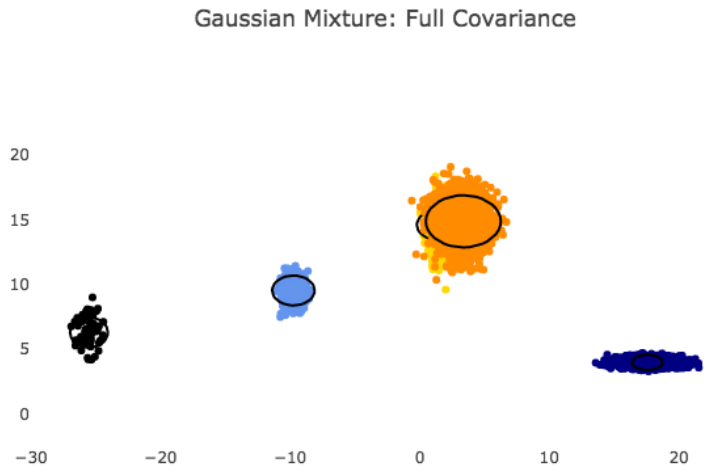

Figure 1: T = 10000, d = 5, K = 5

2.3.2 Diagonal Covariance

The diagonal covariance matrix configuration is seen as a happy medium between a good fit on the data and handling high-dimensional data. Since it only captures the diagonal (so the off-diagonal correlations are 0) it means that it works well in cases that the components are sufficiently independent from one another. Also because the component Gaussians are acting together to model

the overall mixture density, full-rank covariance matrices might not be necessary even if the features are not statistically independent. In most cases, the linear combination of diagonal covariance basis Gaussians can model the correlations between features. We can see in Figure 2 below that diagonal covariance matrices still fit the training data pretty well even if it is a looser fit than the full-rank covariance matrix.

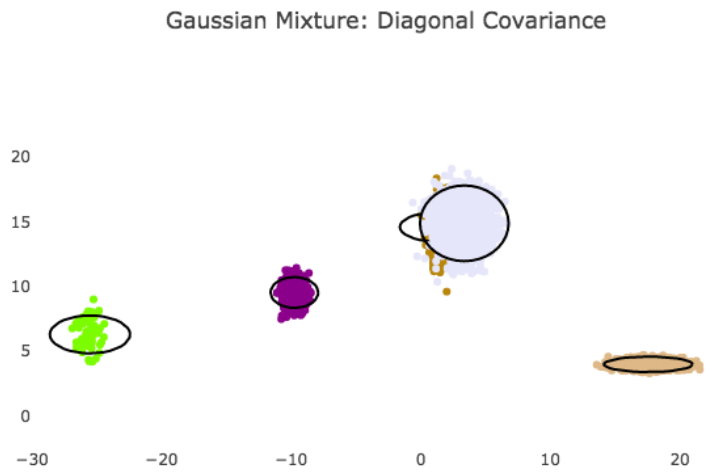

Figure 2: T = 10000, d = 5, K = 5

2.3.3 Spherical Covariance

The spherical covariance matrix configuration is especially good at fitting spherical Gaussians, which means that the probability distribution has spherical (circular) symmetry. The covariance matrix is once again diagonal like the diagonal covariance matrix configuration but this has the added property that the variances are equal. This usually gives it a very loose fit on the data unless the underlying density of a component is also spherical. This loose fit on data could possibly help it successfully generalize to future data. We can see in Figure 3 that the fit is very inclusive.

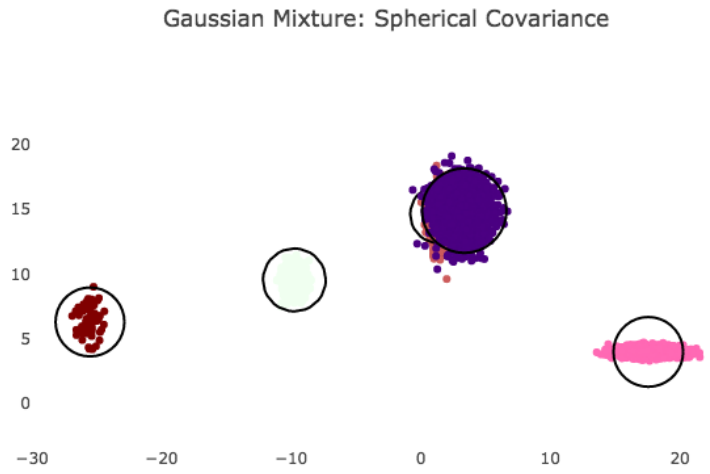

Figure 3: T = 10000, d = 5, K = 5

2.3.4 Tied Covariance

The last covariance type is the tied covariance configuration and this adopts the view that parameters can be shared (tied) among the individual Gaussian components, such as having a common covariance matrix for all components. This is good in situations such that all the data is similarly shaped and it'll find a good middle ground between all components such that it can generalize to all of them. In Figure 4 below, you can see that all the components share the same covariance matrix which looks similar to the spherical covariance configuration but it also has a tighter fit on the training dataset.

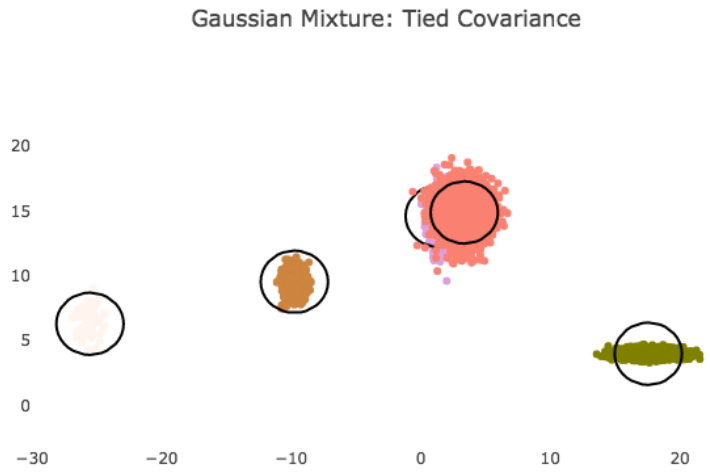

Figure 4: T = 10000, d = 5, K = 5

3 Hypotheses

I have three different hypotheses to study the effect of covariance types on the learnability of GMMs under various circumstances:

1. As the number of samples T used in the training dataset increases, the best performing GMM will move from the full-rank covariance configuration to the diagonal and tied configurations.

2. As the dimension of the data d increases, all GMM configurations except the full-rank covariance configuration will scale well (perform well)

3. As the number of components fitted K increases, all GMM configurations will start decreasing in performance but the tied covariance configuration will perform the best as it assumes shared parameters among components

4 Methodology

In order to test each hypothesis, I set up three different experiments that will be detailed in the next few sections. First, I will give a short overview of the technologies used to conduct these experiments. After that, I present the processes used for synthetic data generation, overview the definition and measurement

of GMM performance, and then formalize the experiment parameters before presenting the experiments. All of the code for conducting the following experiments can be found at `github.com/changreytang/gmm-empirical-study`.

4.1 Tools Utilized

4.1.1 Python

I chose to use Python as my programming language of choice because it has a lot of support from the Machine Learning community with numerous ML libraries. It was the perfect match for this project specifically because it allowed me to efficiently and quickly set up the experiments without spending a lot of time getting caught up in implementation details.

4.1.2 SKLearn

Sci-kit Learn is an easy to use machine learning library that has support for learning Gaussian Mixture Models with the Expectation-Maximization algorithm. It also has the perfect amount of flexibility in their GMM implementation so that I could easily set up the experiments.

4.1.3 *GaussianMixture* Parameters.

`sklearn.mixture.GaussianMixture` is the class I used to learn the GMM and it has some important parameters which helped me conduct the experiments:

- `n_components`: the number of mixture components. This is an important independent variable in the experiments and what the third hypothesis will be testing.
- `covariance_type`: the covariance configuration of the GMM. This is the core variable we will be analyzing in these set of experiments.
- `tol`: the convergence threshold. This dictates when the EM iterations will stop as the lower bound average gain is below this threshold. I keep it at the default value `1e-3`.
- `reg_covar`: the non-negative regularization added to the diagonal of the covariance matrix to assures it always positive. To account for situations with a high number of components I set this to `1e-3`.

4.2 Synthetic Data Generation

I chose to generate synthetic data for my experiments and not to perform it on real world data because I needed strong control over the shape of the dataset. It would have been ideal to conduct experiments on real world datasets that happened to fit my specifications but I knew it wouldn't be pratical. In order to test my hypotheses, I required an extremely controlled environment because

the goal is to evaluate and compare the behavior of the GMM utilizing different covariance configurations on a highly controlled dataset. I needed to be able to control the dimensionality, underlying Gaussian components, and size of the dataset. Attempting to find the exact real world data that fit our experiment descriptions wouldn't have been feasible.

My synthetic data generation method `_generate_gmm_data` takes in four parameters that allowed me to control the shape of the dataset:

- `points`: the number of samples to create
- `components`: the number of underlying component Gaussians to generate data from
- `dimensions`: the dimensionality of the data
- `seed`: an integer passed to `np.random.seed()` that guaranteed the same shaped dataset would be created if the same integer is passed. This seed value is what the `np.random` methods used to initialize the generator.

The first three parameters is what allows us to control most of the independent variables in our experiment.

4.3 Measuring Performance

The three hypotheses are all about the *performance* of the GMM with different configurations and situations so we need a solid definition of what performance is. I chose to define performance as three factors:

- *Accuracy*: During the synthetic data generation phase we label each data point with the ID of the component that it was generated from. We then split of the generated data such that 30% becomes the training data and the other 70% becomes the testing data. This is how we evaluate the accuracy of the model. We will be attempting to predict which component each data point in the testing dataset came from using the model that was trained with the training dataset. Thus, accuracy is defined as the number of correctly labeled data points in the total testing dataset.
- *Number of Iterations*: This is the number of EM iterations it took to learn the model. The lower this is the better.
- *Time to Learn (Elapsed Time)*: The amount of time it took to learn the model is a good identifier on whether the particular GMM will be able to scale to larger, higher-dimensionality datasets.

4.4 Experiment Parameters

4.4.1 Independent Variables

- Number of samples in training dataset: T = 0.3*(number of samples generated)

- Dimensionality of dataset: d
- Number of components: K
- Covariance type/configuration: full, diagonal, spherical, or tied

4.4.2 Dependent Variables

- Accuracy (percentage)
- Number of EM iterations
- Elapsed time, time to learn model (seconds)

4.5 Experiments

The experiments will be set up so that we can test how our independent variables `sample_size`, `number_of_components`, and `dimensionality` affect the dependent variables `accuracy`, `number_of_iterations`, and `elapsed_time` for our 4 covariance configurations of GMMs. Detailed below are three experiments that will be run on each of the GMM covariance configurations under the same exact conditions so that we can effectively compare their performance. We will then analyze the overall interactions between `sample_size`, `number_of_components`, and `dimensionality` to understand the underlying structure and bottlenecks of separating Gaussians.

4.5.1 Experiment One: Sample Size

First, we will test how sample size affects the dependent variables in our experiment. We will set the `dimensionality = 5`, `number_of_components = 100`, and `sample_size = 500` initially. Then we will scale the `sample_size` from 500 to 10,000 ($\Delta = 500$) and see how that affects the running time and accuracy of the GMM.

4.5.2 Experiment Two: Dimensionality

Second, we will test how dimensionality affects the dependent variables in our experiment. We will set the `dimensionality = 2`, `number_of_components = 100`, and `sample_size = 10000` initially. Then we will scale the `dimensionality` from 2 to 50 ($\Delta = 4$) and see how that affects the running time and accuracy of the GMM.

4.5.3 Experiment Three: Number of Components

Third, we will test how the number of underlying component Gaussians affects the dependent variables in our experiment. We will set the `dimensionality = 5`, `number_of_components = 5`, and `sample_size = 10000` initially. Then we will scale the `number_of_components` from 5 to 1005 ($\Delta = 200$) and see how that affects the running time and accuracy of the GMM.

5 Results and Discussion

5.1 Experiment One: Sample Size

5.1.1 Results

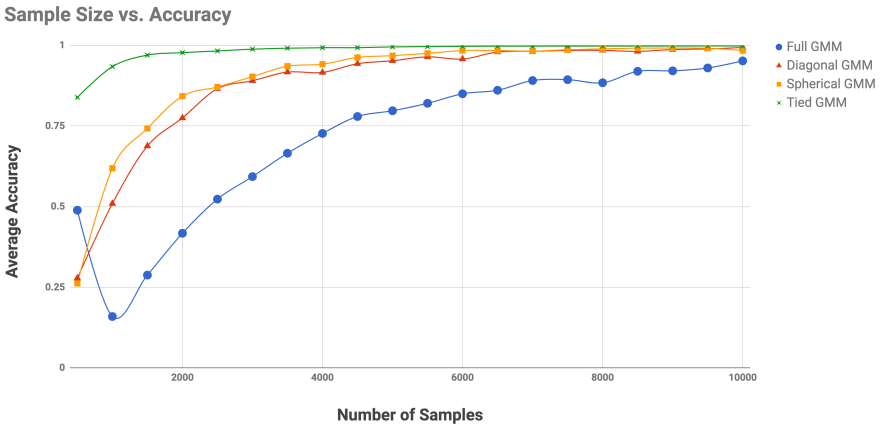

Figure 5: Number of Samples vs. Accuracy

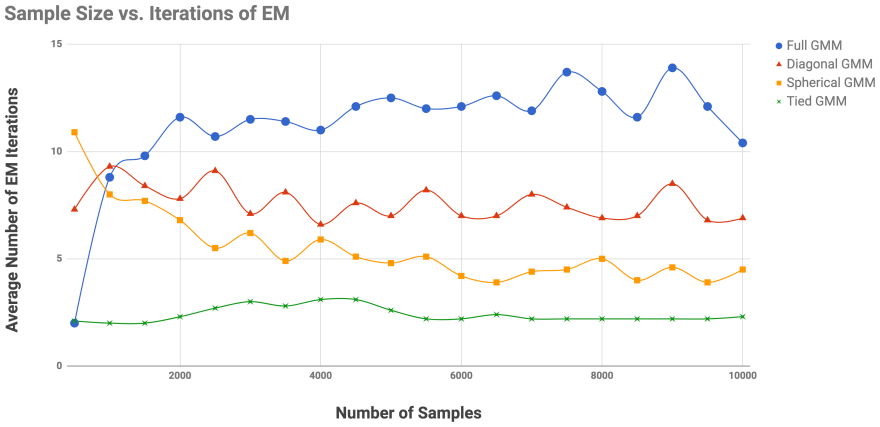

Figure 6: Number of Samples vs. Iterations of EM

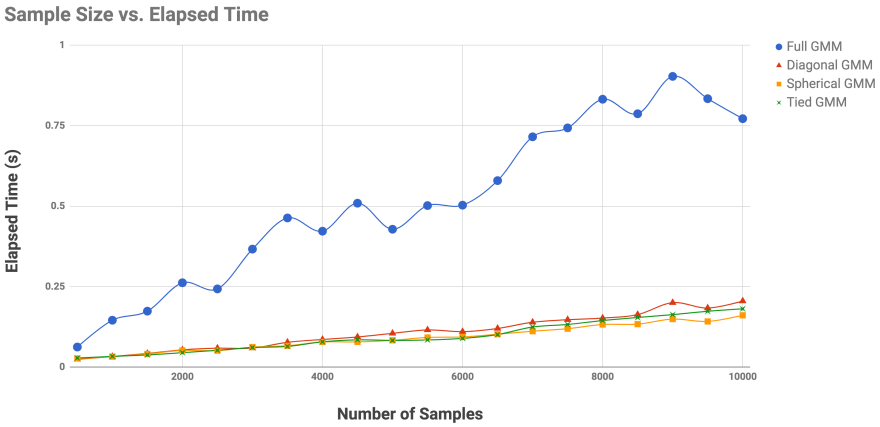

Figure 7: Number of Samples vs. Elapsed Time

5.1.2 Discussion

The initial hypothesis we are trying to test with this experiment is: *As the number of samples T used in the training dataset increases, the best performing GMM will move from the full-rank covariance configuration to the diagonal and tied configurations.* The reasoning for this initial hypothesis stemmed from the notion that the full-rank covariance should be able to perform better than all other configurations when the number of samples are low but the other configurations should catch up as the samples increase.

We can see clearly in Figure 5 that the results were quite different from what we expected. The Full GMM only performed better than the Diagonal GMM and Spherical GMM (not even the Tied GMM) initially. Even more surprising is that we see a drop in Full GMM accuracy when we increase from 500 to 1,000 samples. One of the only reasons I can hypothesize for this phenomenon is that the Full GMM fit the training so tightly that it didn't perform well on the testing dataset. This is a strong argument because the other three configurations are all looser fitting than the Full GMM and perform more accurately starting from 1,000 samples. Since the Full GMM eventually catches up in accuracy at 10,000 samples, we can observe that it needs to see more samples so that the tight fit will truly learn the component Gaussian parameters.

Another illuminating factor we can see in the data is the high performance of the Tied GMM. Due to this behavior, I have a strong reason to believe that there was some unforeseen shared component parameter assumption in the synthetic data generation phase.

Other than those two surprising observations, both Figure 6 and 7 show expected behavior. The `elapsed_time` and `number_of_iterations` should get

significantly worse for the Full GMM in comparison to the other configurations because it is estimating the full covariance matrix for each iteration.

5.2 Experiment Two: Dimensionality

5.2.1 Results

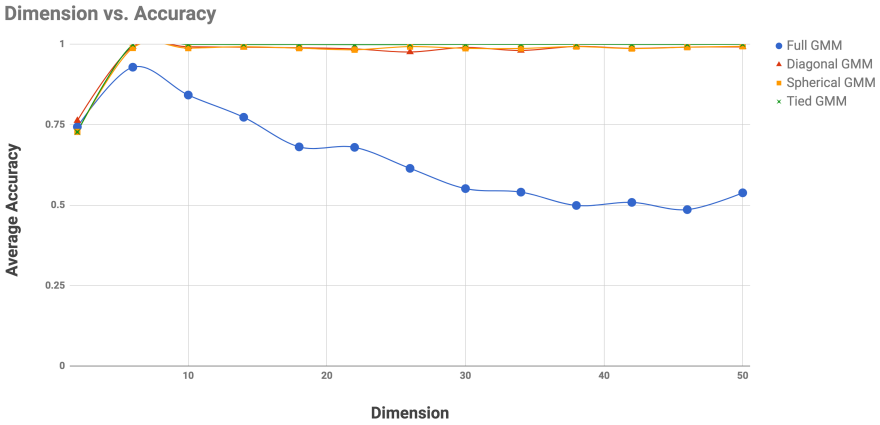

Figure 8: Dimension vs. Accuracy

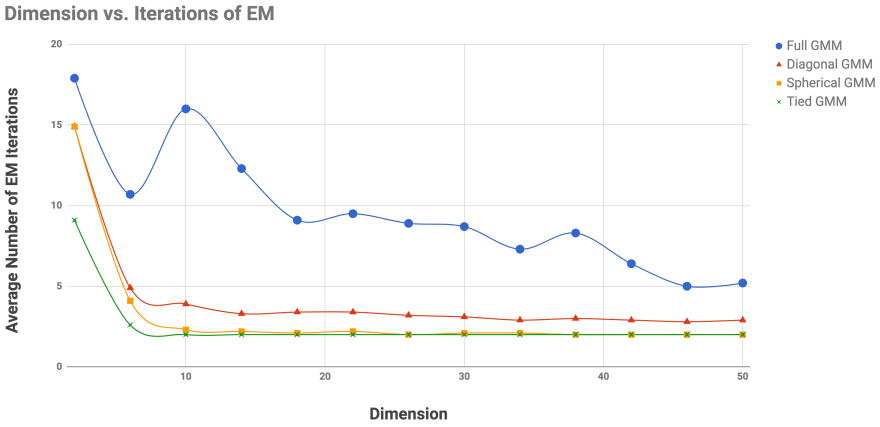

Figure 9: Dimension vs. Iterations of EM

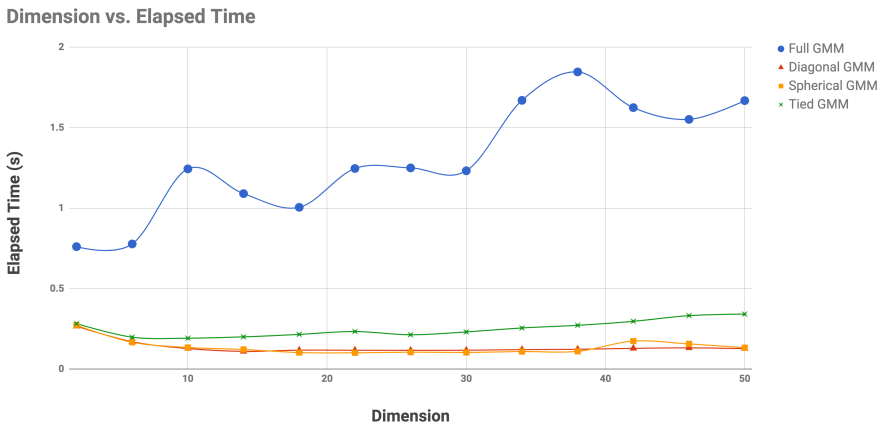

Figure 10: Dimension vs. Elapsed Time

5.2.2 Discussion

The initial hypothesis we are trying to test with this experiment is: *As the dimension of the data d increases, all GMM configurations except the full-rank covariance configuration will scale well (perform well).* The reason for this initial hypothesis is because we know that the Full GMM doesn't perform well on high-dimensional data. As we can see in Figure 8, this is very much true as we see the accuracy of the Full GMM drop as the dimensionality of the data increases. At d = 50, the Full GMM could only achieve about 50% accuracy on the testing dataset. Figure 10 also follows the assumptions of my hypothesis with the `elapsed_time` of the Full GMM increases with the dimensionality.

However, one interesting observation we can see in Figure 9 is that as dimensionality increases we actually see less and less iterations of EM. This means that even though the overall amount of time it takes to run one iteration is increasing, we are converging to the threshold in less iterations. We are observing a property of higher dimensional data which states we can more easily meaningfully separate data points in higher dimensions.

5.3 Experiment Three: Number of Components

5.3.1 Results

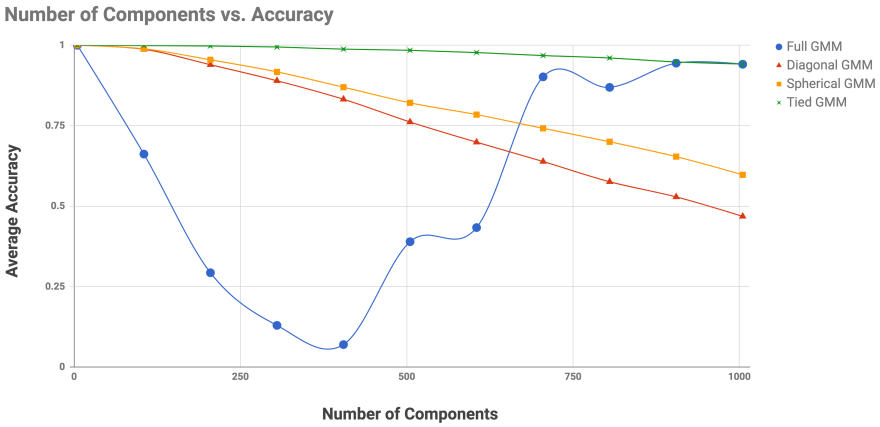

Figure 11: Number of Components vs. Accuracy

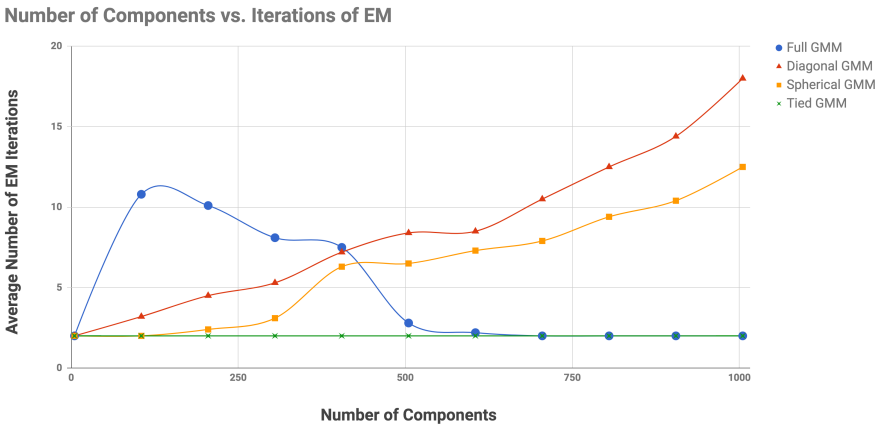

Figure 12: Number of Components vs. Iterations of EM

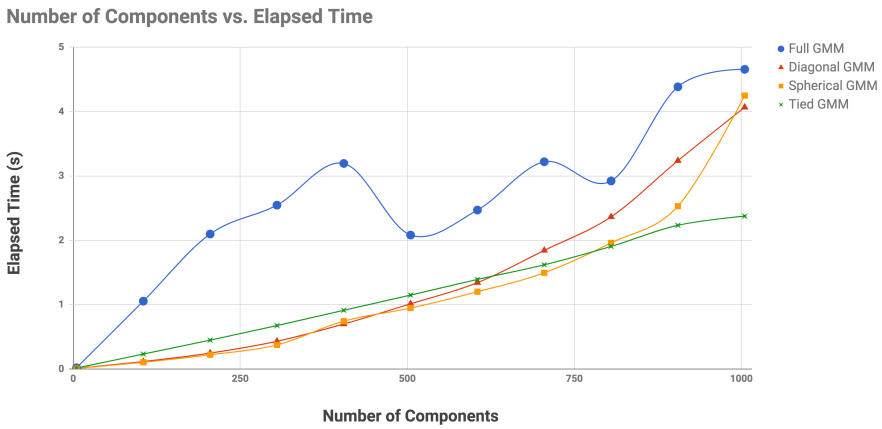

Figure 13: Number of Components vs. Elapsed Time

5.3.2 Discussion

The initial hypothesis we are trying to test with this experiment is: *As the number of components fitted K increases, all GMM configurations will start decreasing in performance but the tied covariance configuration will perform the best as it assumes shared parameters among components.* As we can see throughout Figure 11, 12, and 13, the Tied GMM does indeed perform best as the number of components increase. It should perform the best because even if the number of components increase, the performance isn't affected due to the fact it assumes shared parameters between the components. This means it won't have an increase in calculations when components increase.

Even though our hypothesis was correct, we can observe some interesting and unexpected behavior from the other GMM configurations. In Figure 11, we can see a sudden dip in Full GMM accuracy from 5 to 405 components and then an increase in accuracy from there until 1000 components, even matching the accuracy of the Tied GMM as the end. This behavior is very erratic and seemingly has no explanation but it could be due to unforseen variables in the synthetic data generation phase.

5.4 Overall Discussion

Overall, the experimental results have been really illuminating for the 3 hypotheses that we initially intended to prove or disprove. The first hypothesis was seen to be false, possibly due to the fact that the Full GMM was overfitting on the training data compared to the other looser fitting GMM configurations. The second and third hypotheses were seen to be true *even though* we did get to observe some anomalies in behavior from other observations in the results.

I believe that a reason for these unexplained phenomenons in the experimental results is due to the synthetic data generation phase using the `np.random` module. There could be an underlying order to the "random generation" that allowed for the Tied GMM to perform so drastically better than all the configurations and lead to complications in the Full GMM.

In the end, the EM algorithm for the GMM works pretty well with a large number of components and high-dimensional data. It may take a lot of configuration to help your GMM to perform well but I believe that it solves the problem of separating Gaussians pretty well. You just need to be conscientious of the possible underlying shape of the dataset you are trying to learn in order to help the process of fine-tuning the configurations to produce the best fit.

6 Conclusion

The problem of separating Gaussians is still a hard one. There has already been numerous published literature in this area of research but we still have so much to learn about solving this problem. Possible future work in this area is to test these hypotheses on real world data and see how these assumptions perform. I hope this empirical study on the effects of different covariance matrix estimation techniques on GMM performance shows how complex this problem can be.