3 Use-Cases for Gaussian Mixture Model (GMM)

Feature engineering, unsupervised classification, and anomaly detection with the versatility of the GMM algorithm

Viyaleta Apgar
Jul 27, 202311 min read

Gaussian Mixture Model (GMM) is a simple, yet powerful unsupervised classification algorithm which builds upon K-means instructions in order to predict the probability of classification for each instance. This property of GMM makes it versatile for many applications. In this article, I will discuss how GMM can be used in feature engineering, unsupervised classification, and Anomaly Detection.

What are Gaussian Mixture Models (GMM)?

Model Description

While the Gaussian distribution of a single or multiple variables of a dataset attempts to represent the entire population probabilistically, GMM makes an assumption that there exist subpopulations in the dataset and each follows its own normal distribution. In an unsupervised fashion, GMM attempts to learn the subpopulations within the data and its probabilistic representation of each data point [1]. This property of GMM allows us to use the model to find points that have low probability of belonging to any subpopulation and, therefore, categorize such points as outliers.

GMM essentially extends the multivariate Gaussian distribution to fit the subpopulation case by utilizing components to represent these subpopulations and alters the multivariate probability distribution function to fit the components. As a gentle reminder, the probability density function of the multivariate Gaussian looks like this:

Given k features, random variable X, feature means $\vec{\mu}$
and covaraince matrix Σ
the multivariate Gaussian distribution probability density function is

$$f(x_1, ..., x_k) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\Big(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})\Big)$$

Multivariate Gaussian distribution probability density function

In GMM, the probability of each instance is modified to be the sum of probabilities across all components and component weights are parameterized as ϕ. GMM requires that the sum of all components weights is 1 so it can treat each component as a ratio of the whole. GMM also incorporates feature means and variances for each component. The model looks like this:

Given K components with component weights ϕ_i, component means $\vec{\mu_i}$
and component covariance matrix Σ_i for a random variable X
and covaraince matrix Σ
the multivariate Gaussian distribution probability density function is

$$\mathcal{N}(\vec{x} \mid \vec{\mu_i}, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\Big(-\frac{1}{2}(\vec{x} - \vec{\mu_i})^T \Sigma_i^{-1} (\vec{x} - \vec{\mu_i})\Big)$$

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

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

GMM model formulation

Notice the parallels between multivariate distribution and GMM. In essence, the GMM algorithm finds the correct weight for each component is represented as a multivariate Gaussian distribution. In his post, Oscar Contreras Carrasco does a fantastic derivation of GMM [2].

The parameters of the model can be initiated randomly or by using a specific strategy and the component weights ϕ of the model are determined using repeated Expectation Maximization (EM) steps [1].

Model Algorithm

The first part of implementation of GMM is **initialization of the components**. GMM implementation consists of initialization step followed by iterative **Expectation Maximization** (EM) process which repeats until convergence:

Step 1: In the ***initialization step***, model parameters are initialized: *K* values from the dataset are randomly assigned as component means; component variances are calculated based on the randomly assigned means; all component weights are assigned a value of 1/*K*.

Step 2: In the ***expectation step***, we calculate the probability that each data point is generated by each component. The expectation for each datapoint-component pair is the weight of this specific component times the probability that our datapoint belongs to this component (given the component mean and variance) as a fraction of all other component probabilities, parameterized with their respective component weights. Essentially,

expectation attempts to find the likelihood that each point belongs to each component and will use this value to slowly adjust model parameters until convergence.

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

Formula for the expectation step

Step 3: In the ***maximization step***, we reset the component weights and means and recalculate the variances based on the γ value from expectation step. The new component weights are set as sum of expectation values across all datapoints for this component. The new mean value for each component is the average of all data points, weighted by the expectation value.

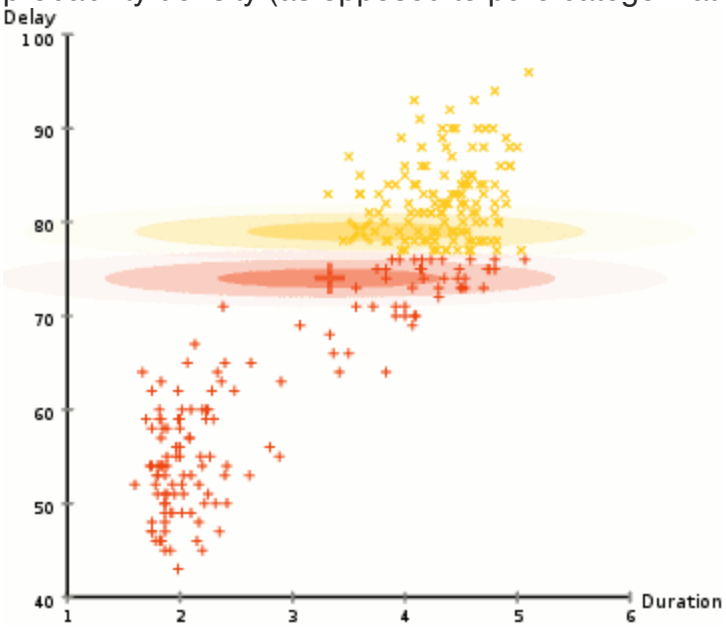
$$\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$$

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

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

Formulas for the maximization step

Just like in a k-means algorithm, it is appropriate to guess the number of components *K*, if the number of components is not previously available.
Below is a visual example of GMM convergence. Here, GMM demonstrates convergence on a two-dimensional dataset with two clusters. The algorithm behaves similarly to k-means but differs in that it estimates the probability density (as opposed to pure categorization of sample data points).

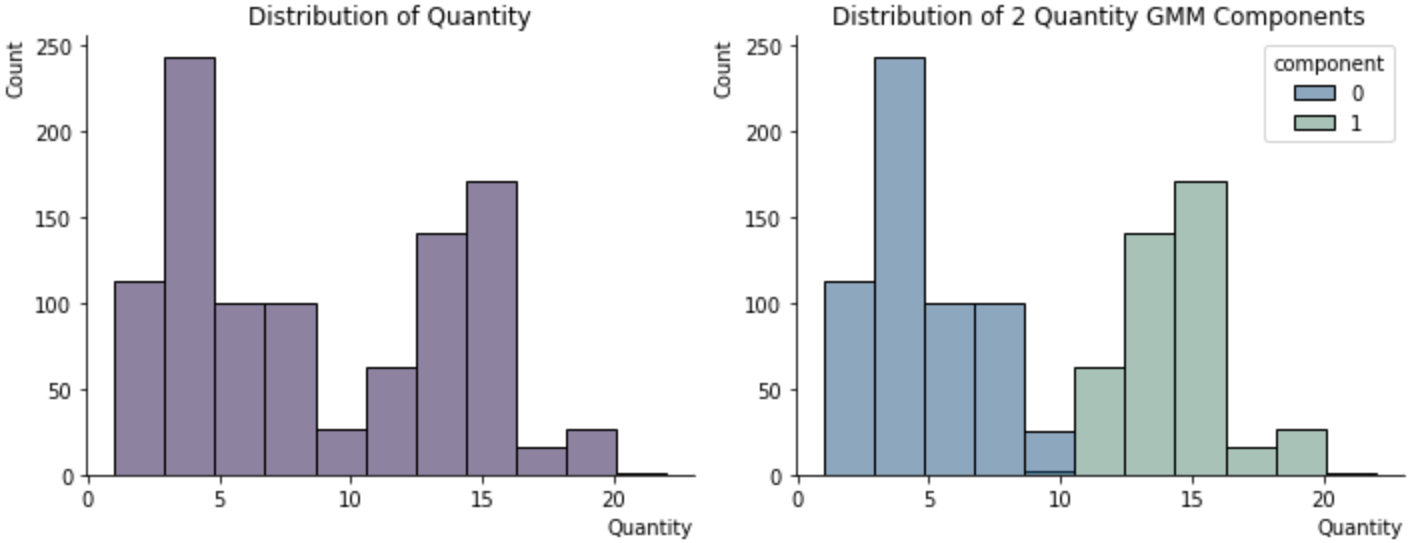


EM Clustering of Old Faithful data [3]

Let's see how this algorithm can be applied to our 3 use-cases.

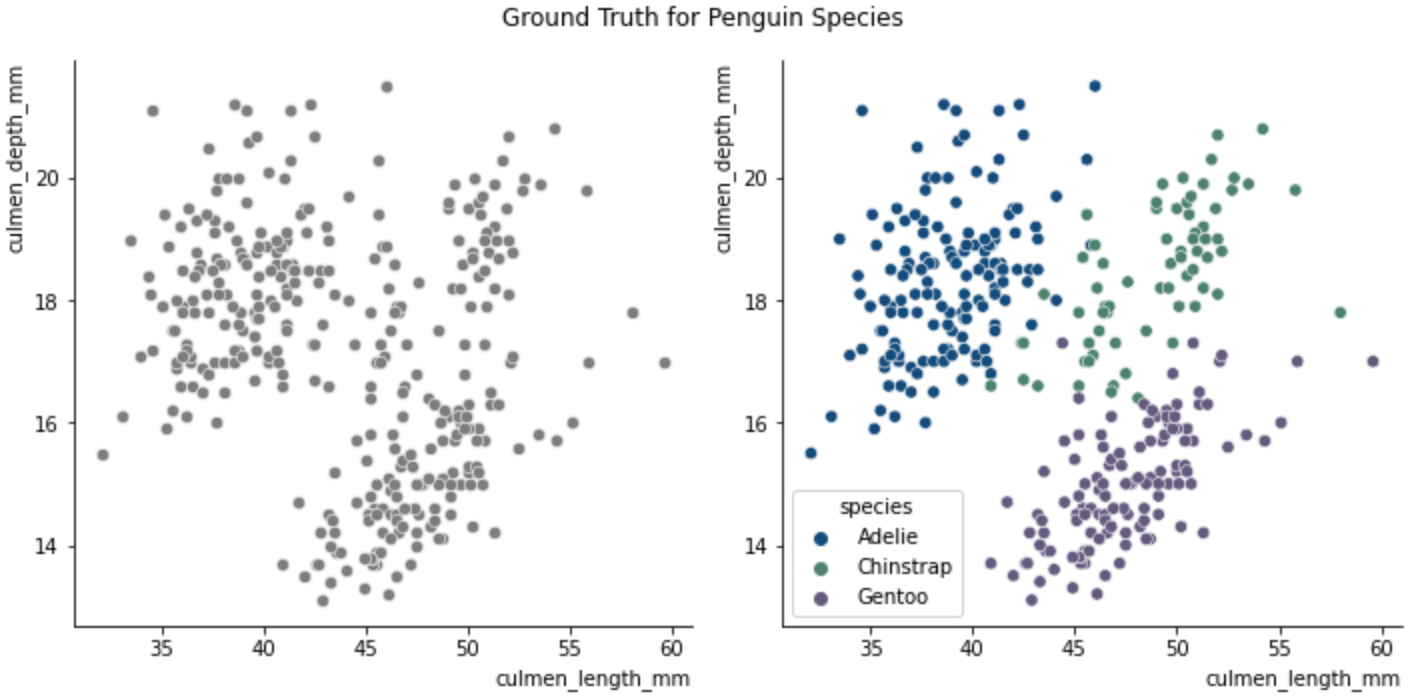
GMM for Feature Engineering
While some machine learning models (like the infamous XGBoost) can learn a variety of input feature distributions, others are more strict in their requirements. Linear and logistic regression, linear discriminant analysis (LDA), and multivariate Gaussian typically expect features to be normally distributed and may not function so well if the data is multimodal. There are other analytical and visual reasons we may want to deal with multimodality and GMM can help us do just that.
Let's take a look at an example of bimodal feature from a fictional bookstore dataset. I pulled this data from a Kaggle repository (found here) which contains data scraped from books.toscrape.com [7]. This dataset features typical bookstore information like book title, genre, price, and rating. It also contains book quantity, which

determines the total number of books the fictional book store has in stock. As luck would have it, book has a bimodal distribution. Let's see if we can use GMM as a Feature Engineering technique in order to create two separate features from the book quantity data.
I used the following code to implement this task in Python:
Let's take a look at the results. The left hand-side chart shows the original distribution of book quantity. The right hand-side chart shows the distribution of each predicted component, after the GMM transformation. Notice that the shape of the complete distribution is exactly the same but the two components split the original distribution at a specific point, creating two (mostly) normal histograms. If I was not satisfied with the point which segments two components, I can use the GMM predicted probability to adjust where component 1 ends and component 2 begins.



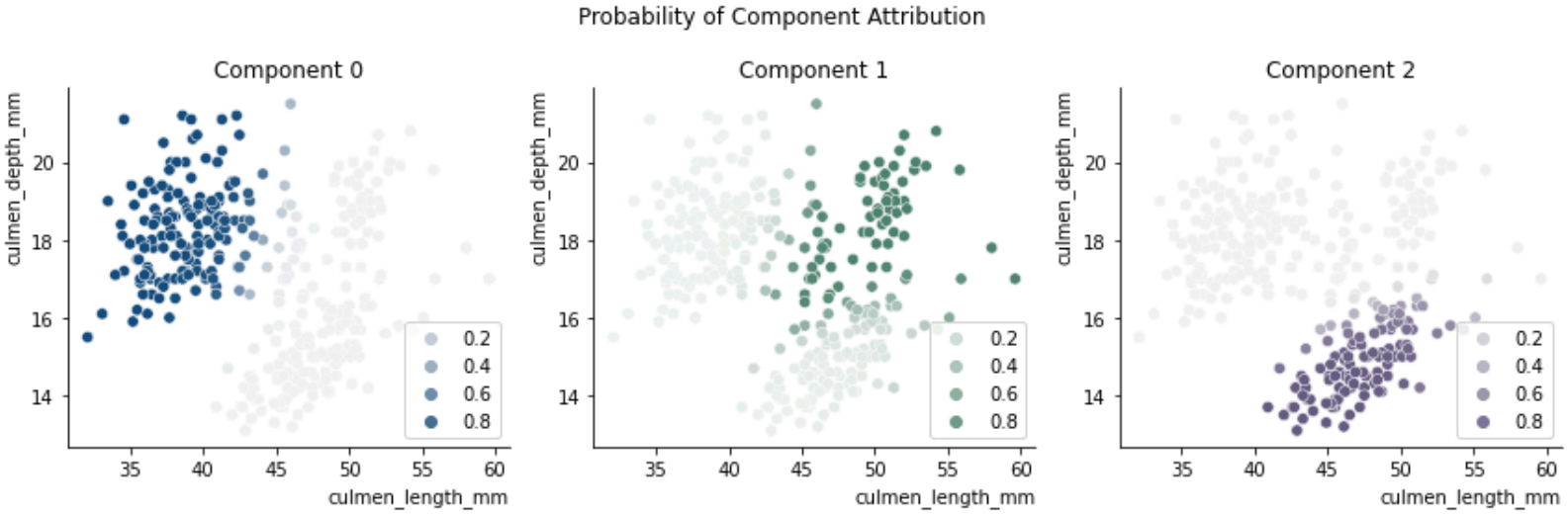
Distribution of Quantity before and after GMM [9]

GMM for Unsupervised Classification
Another use-case for GMM is unsupervised classification. In this sense, GMM works similarly to K-means algorithm but allows probabilistic determination of class belonging (unlike the K-means, where the output is a binary metric). This can be especially beneficial to use-cases that require custom thresholds for categorization or simply call for a probabilistic output.
For this example, I downloaded the penguins dataset (available on Kaggle) and selected two features for the purpose of visual demonstration: penguin culmen length and depth (culmen is the top ridge of penguin's bill) [8]. I removed null data points and created a scatterplot to depict the data.
When we take a look at the scatterplot, 3 potential groups stand out. If we color the ground truth categories, we see that, in fact, the three species of penguin do align with the 3 groups of data points. This example is quite rudimentary since in real world, we usually work with multidimensional data and there is no easy way to determine how many subpopulations exist in our dataset. Nonetheless, let's see how GMM can help us segment our data.



Scatterplot depicting penguin culmen length vs. depth [9]

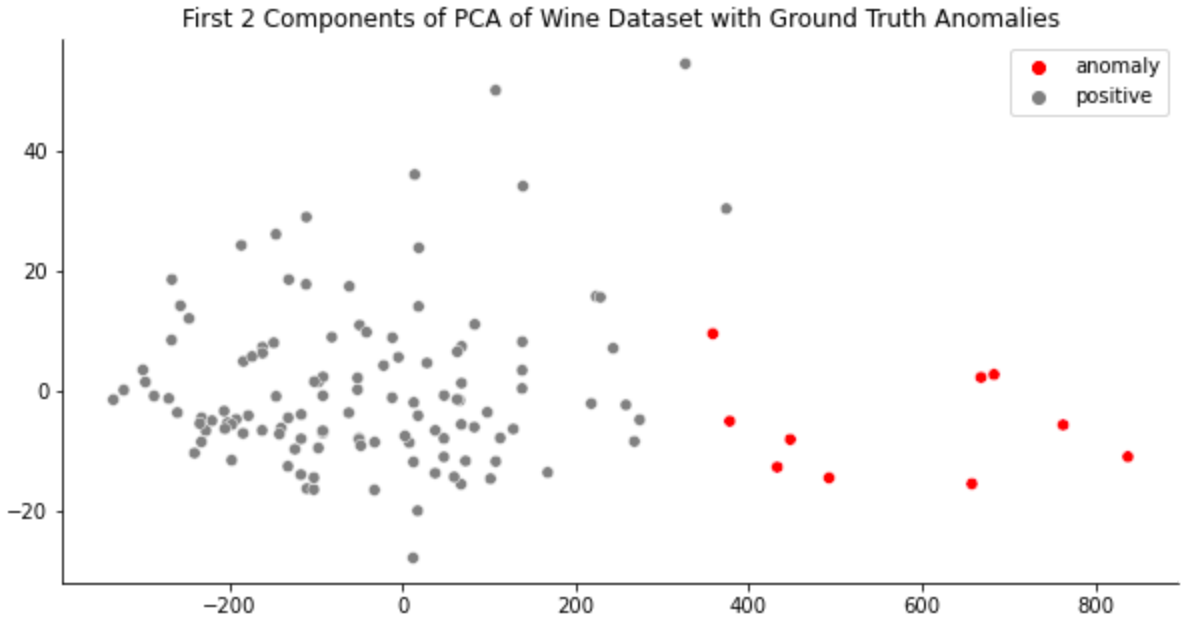
In the Python code snippet below, I downloaded the dataset, removed null values, selected the two features of interest, and fitted the GMM model to them. `sklearn` offers two options for predictions – predicting the class and predicting the probability of class belonging. Sum of probabilities for each data point is equal to 1 (per GMM algorithm constraint).
Let's take a look at the probability of each data point belonging to each component. The three scatterplots below show probabilities of instances per component. Points with higher transparency have lower probabilities and points depicted in brighter color have higher probability. In the chart below, we can see that GMM extrapolated greater uncertainty for points located between different components, as expected. We can use the `gmm.predict_proba()` function to retain control of class attribution.



Scatterplots depicting probability of class belonging for each predicted class [9]

GMM for Anomaly Detection
In my previous story, I shared the basics of anomaly detection and an application of a statistical approach to detect anomalies [6]. Namely, I used the mutlivariate Gaussian distribution to identify data points that have low probability of following the Normal distribution. The problem with this approach, however, is that our data is often more complex. Gaussian Mixture Model (GMM) attempts to solve the multimodality problem and is useful in cases where features form subpopulations of normally distributed feature relationships.
For this last example, I will use a wine dataset from the ODDS library [10]. This is the same dataset I used in my post where I applied multivariate Gaussian distribution to detect outliers. Let's see if I can improve on my previous result, where 41 instances where misclassified, 40 of which were falsely identified anomalies. The

first advantage over the multivariate Gaussian distribution method is that we can use all of our features and we are not restricted to only normally distributed features. The second advantage is that we can account for data subpopulations.
Perhaps the most prominent advantage of GMM is that this model can help in finding two types of anomalous data: anomalies which are outliers of a population (e.g. mistake in data entry) and anomalies which form their own group (e.g. credit card fraud behavior). The positive instances in the wine dataset are constructed as samples of two types of wine while anomalies in the dataset are constructed as a subsample of a third type of wine [10]. Therefore, we will likely find that our outliers make up their own group in this instance.
Since we have 13 features in our dataset, let's summarize the data into first two PCA components and plot them. In the chart below, we will see that the scatterplot features one dense area and about two dozen points scattered throughout. No distinct subpopulations appear in the data.



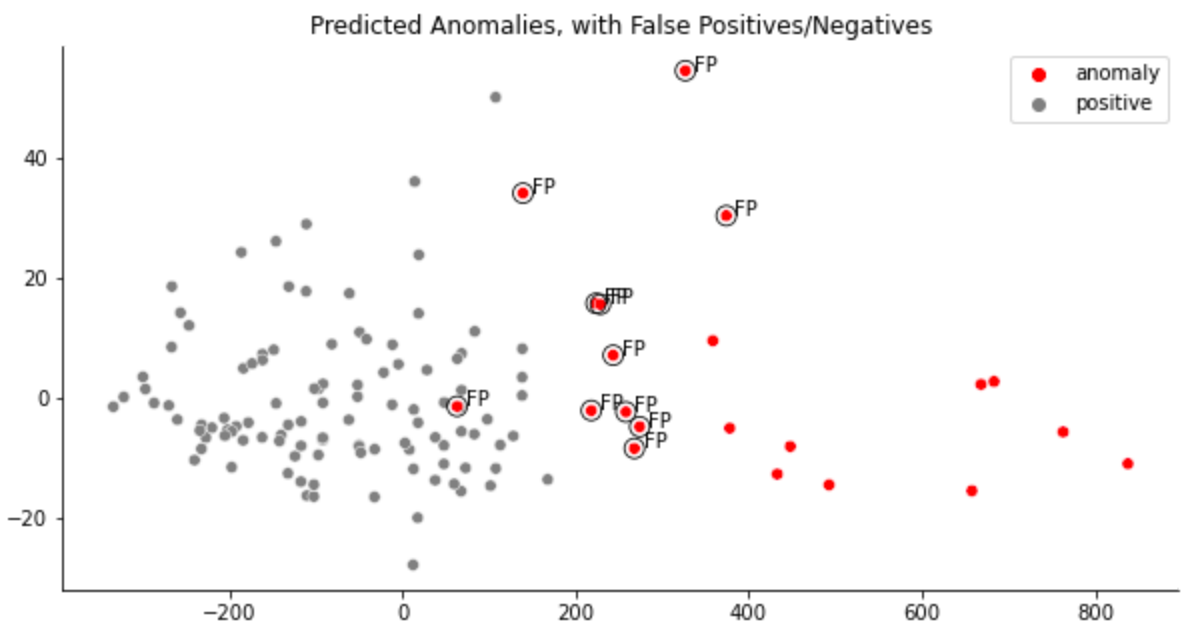
Scatterplot of first 2 PCA components, showing ground truth anomalies [9]

The code snippet below fits the GMM model with 2 components and standard input fields from `sklearn` library to our dataset. Hopefully, one component will identify positive instances while the other component identifies anomalies.
The predicted components are numbered randomly so I did some background magic to relabel the predictions. You can check out my Jupyter notebook for more guidance on how to do that.
The results look good! We have 0 false negatives and we have 11 falsely identified anomalies. If this was the real world, we would only have to manually check some 15% of data after running this model. This is already a significant improvement upon the multivariate Gaussian distribution approach.

	Predicted Positive	Predicted Anomaly
True Positive	108	11
True Anomaly	0	10

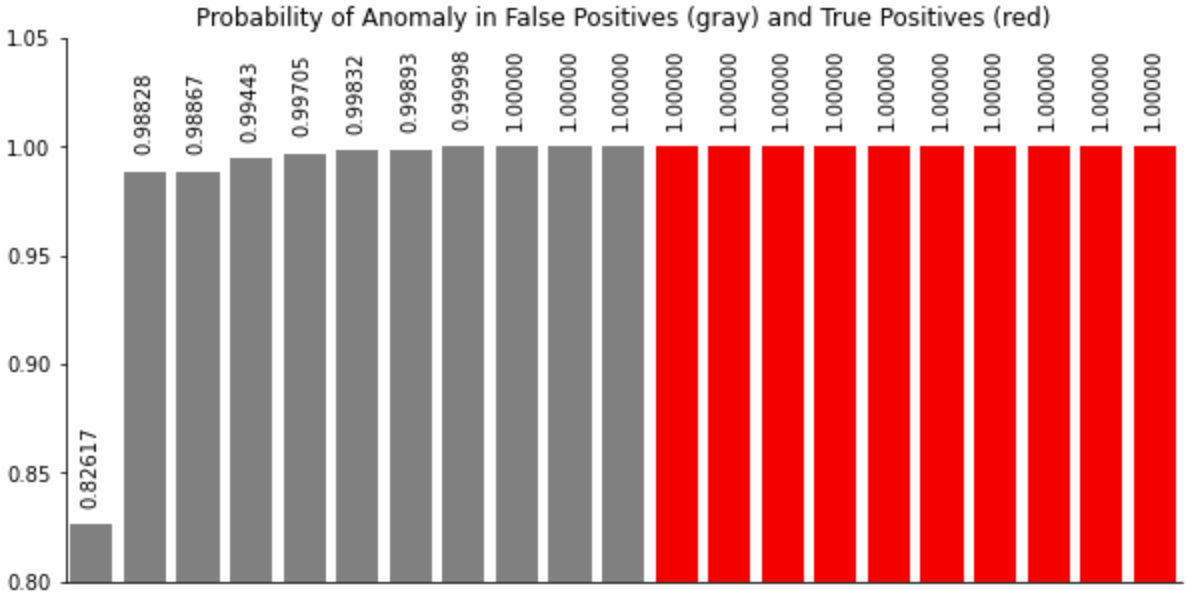
Confusion matrix of GMM results [9]

If we plot our results, we can see that the falsely identified anomalies happen to be closer to the most dense area. Again, we are using the first two components of PCA for visualization purposes only and GMM is doing a lot more to categorize our data points.



Scatterplot of first 2 PCA components, showing predicted anomalies [9]

We can probably improve upon this result by analyzing our predicted anomaly probabilities. Below, I have charted those probabilities for falsely identified anomaly instances and true anomaly instances (there were no predicted false negatives).



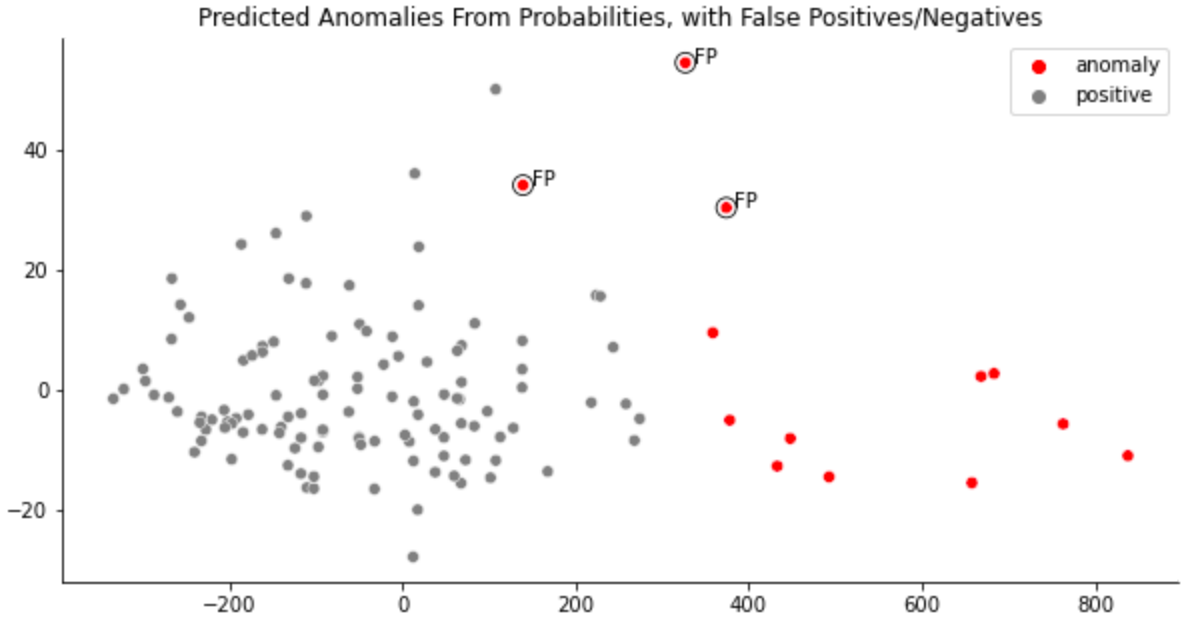
Bar chart showing probabilities of anomaly class belonging among falsely identified anomalies and true anomalies [9]

We can see that most of our falsely identified anomaly probabilities are lower than 0.9999. So, instead of using default classification, we can use the predicted anomaly probability and set a new threshold. If we set the threshold anomaly detection to be greater 0.9999, we will only have 3 falsely identified anomalies. This can be done with a single line of code: `components = np.where(proba>0.9999, 1, 0)` . In some instances, changing the threshold may add more false negatives to our final results. Lucky us, this is not the case.

	Predicted Positive	Predicted Anomaly
True Positive	116	3
True Anomaly	0	10

Confusion matrix of GMM results, after decreasing increasing probability threshold [9]

In the following PCA scatterplot, we can see that our model still predicts 3 falsely identified anomalies but with the new improvement, we would only need to manually check 10% of data instances for presence of anomalies. Given that 8% of data are actually anomalous, this is great!



Scatterplot of first 2 PCA components, after increasing anomaly probability threshold [9]

The power to predict class probabilities makes GMM is powerful and versatile tool in data science. Although it has the same limitations as its K-means cousin, GMM is helpful in feature engineering, more flexible unsupervised classification, and anomaly detection.