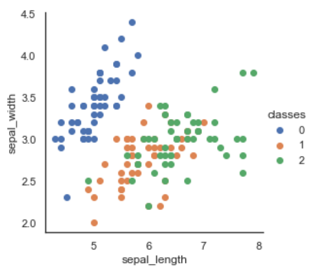
DISCUSSION

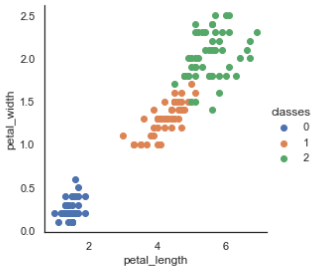
Experimental Design Introduction:
The objective of the experimental design was to implement and evaluate a Probabilistic Generative Classifier on Iris data. And use Gaussian Mixture Model to estimate the generating distribution for each class.

Experiment: *Process*
To achieve the objective for the exam, I implemented a code that can be divided into four main sections-

1. Defining Classes and Functions
   The first step in implementing was to import the necessary libraries and dependencies. I then proceeded with creating a class for Gaussian Mixture Model. I decided to implement the GMM as a class because classes provide a means of bundling data and functionality together. Creating a GMM class will create a GMM type of object, allowing new instances of that type to be made. Each class instance can then have attributes attached to it for maintaining its states. The GMM class consists of 3 main functions namely, *initialization and multivariate normal*; *fit function* (which is used to train the model); and *predict function*. The fit function uses the expectation-maximization algorithm to train the model and is later used in predicting the clusters. The training consists of two steps, Expectation and maximization. In the expectation step, the responsibilities matrix is created where rows are the samples from the dataset, columns represent clusters, the elements Rnk are the probability of sample n to be part of cluster k. In the maximization step, the value of means and covariances are set up step by step. The predict function uses the multivariate normal function using the covariance and means values calculated during training.
   After defining the class for the Gaussian Mixture Model, I created a function for the Probabilistic Generative Classifier. This function takes in the means, covariance, and prior values as arguments. The generating distributions (mean, covariance, and prior values) are then used to calculate the posterior of each class.
   The third function I created was to calculate the decision matrices. It uses the sklearn metrics library to operate to find the decision matrices.

2. Loading, visualizing, and splitting the IRIS Dataset
   The next step in the experiment is to load the IRIS dataset. After loading the dataset, I visualized the dataset's scatter plots for various classes concerning its features. Doing that, I realized that the classes were nearly linearly separated with petal sizes(as shown below).



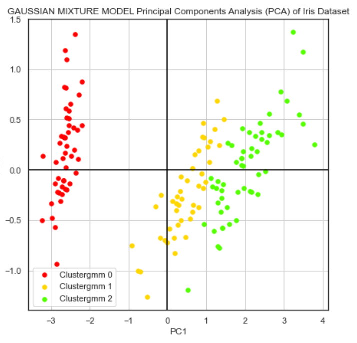


   But as GMM clustering is supervised, I separated the dataset into two parts- Feature set and Target set. The feature set contains the data about the sepal and petal sizes, and the target set contains data about class labels. Then, for splitting the data, I used a stratified K method to get the training dataset so that the training dataset can have attributes close to the original dataset. The validation and testing datasets were sampled randomly without any overlapping to prevent any data leakage.

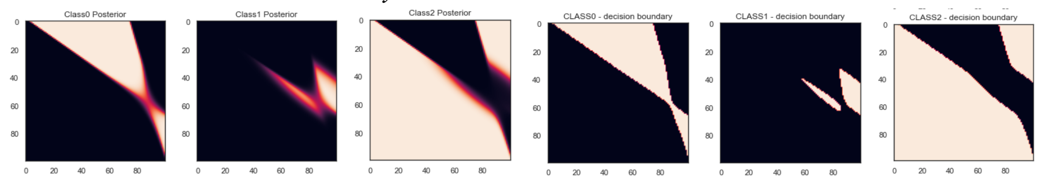
3. <u>Implementing GMM; PGC and getting Confusion Matrices</u>
   Now that we have the training data, we can initialize a GMM object. The initial arguments for GMM that need to be defined are the maximum number of iterations, the number of clusters, and the name of clusters. Once done, the GMM is used to train on the training dataset to get the *generating distributions for each class*. The generating distributions include means, covariances, and priors for each class. These generating distributions are then fed to the PGC function to get the PDFs and Posteriors for each class.
   The class labels (targets) are then calculated by the trained model and stored into an integer array. Once this is done, the confusion matrices for each dataset are calculated and printed using the sklearn-metrics library function for confusion matrices.

4. <u>Plotting the Output plots and images</u>
   Now as the GMM has been implemented, and the generating distributions from GMM have been used in PGC to get the posteriors for each class, I plotted the GMM clusters and PGC plots(posteriors and decision boundaries as shown below)-
   a. *GMM Cluster Plot*-
      As the numbers of features in the dataset are more (4d data), to avoid problems due to dimensionality while plotting, I remaped the data to a smaller coordinate system using PCA. By doing this, it becomes easier to plot a scatterplot for the clusters. As the data is going from only 4d to 2d, there will not be any significant loss of information.

      GMM cluster plot is as follows:



   b. *Posterior and Decision Boundary Plots*-

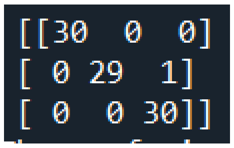


<u>**Conclusion**</u>**:**
In my experimental design, I used a Gaussian Mixture Model on each class of dataset and estimated generating Distribution for each of those classes. I then used the generating distributions to implement a Probabilistic Generative Classifier and evaluated it by plotting its posterior, decision boundaries, and confusion matrices (shown later in the report). I observed and found that the Gaussian Mixture Model performs better than other classifiers(such as K-means) because GMM takes into consideration both the means and covariances to cluster the data. Using a prior with GMM, as I did in the experiment, improves the model. I also found that the time taken to train the GMM increases exponentially for a large number of iterations.
The hyperparameters for this experimental design (along with their optimal settings based on iterative testing) are- number of clusters(3), maximum number of iterations(~ =100), dataset size splits.

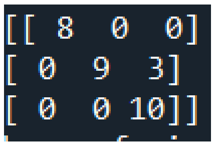
- The <u>*controlled variable*</u> in the code are- number of clusters, maximum number of iterations, cluster names, training data size, validation data testing size, testing data size PCA dimensions.
- The <u>*uncontrolled variables*</u> in the code are- means, Covariances, Priors, Posteriors.

- *Training/Validation/Testing Strategy*- For training the model, my approach was to use a training dataset that is big enough to contain enough data, so most optimal features could be selected, so I selected the size of training data to be 60% of the total dataset. The training data was sampled in a stratified manner so that it preserves the attributes of the original dataset. The reason for doing so was to train the model for all possible scenarios. For validation, the dataset was sampled randomly(20% of the total dataset) and without any overlap with the other datasets. Doing this prevents from having the problem of data leakage during cross-validation. For testing, the dataset was sampled randomly(20% of the total dataset) without any overlap on the training or testing data, to prevent the model from overfitting.
  Following this strategy provided the most optimal result with an accuracy on the test dataset of ~94%.

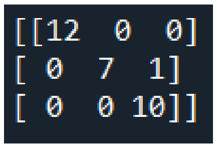
The final confusion matrix (for classes 0,1,2) of training, validation, and test data are as follows-



```
[[30  0  0]
 [ 0 29  1]
 [ 0  0 30]]
```



```
[[ 8  0  0]
 [ 0  9  3]
 [ 0  0 10]]
```



```
[[12  0  0]
 [ 0  7  1]
 [ 0  0 10]]
```