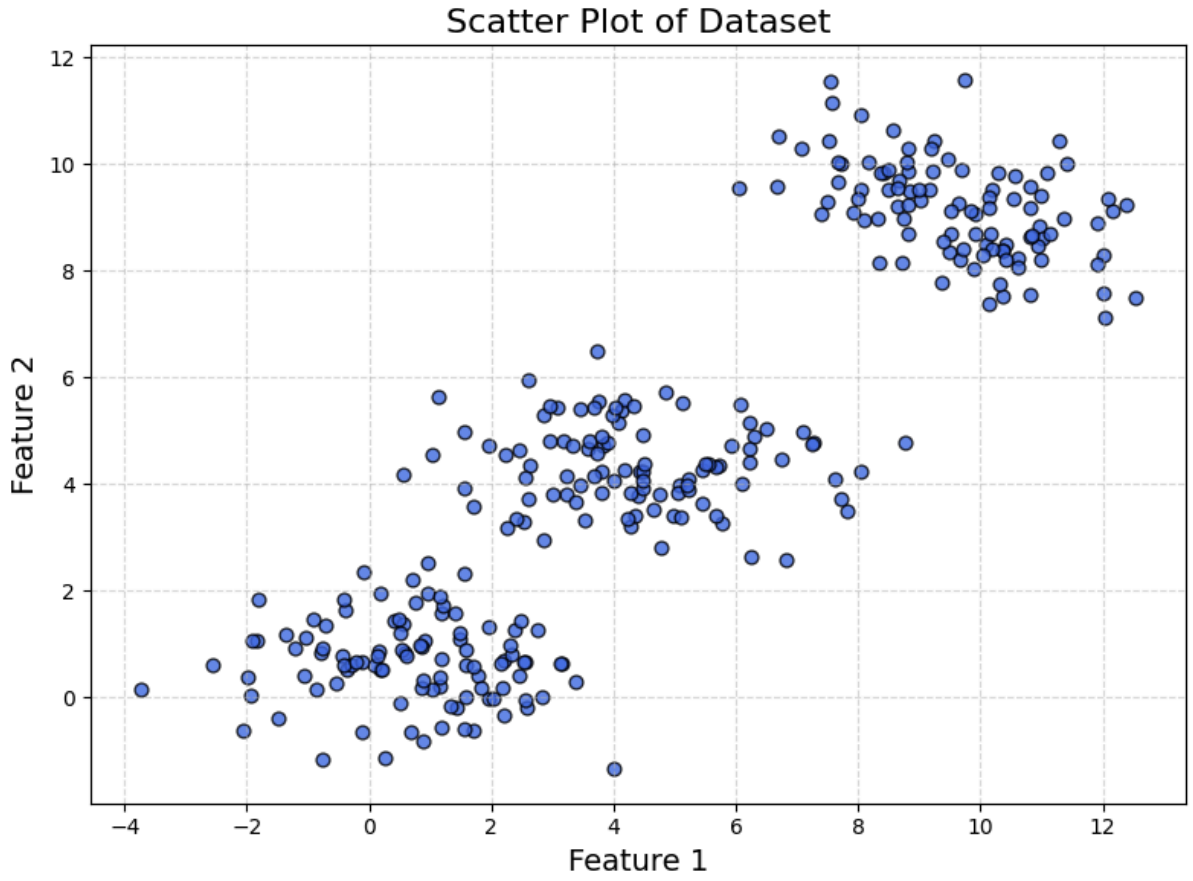
CMPE 544 Assignment 1 Report

Ömer Coşkun
2024700024

Expectation Maximization

This section is about the first part of the assignment - Expectation Maximization.

The scatter of the data is below:



In total, we have 300 data points, each having two features. As described in the assignment, we assume that this dataset is a mixture of three Gaussians. We aim to use Expectation Minimization algorithm to estimate the mean and the covariance matrices of these distributions.

The algorithm and the formulas used to implement the algorithm are the same as the ones provided on Moodle [1], therefore, we only provide the general layout of the algorithm and our results.

The Algorithm

The algorithm for EM is simple and the general layout is described below:

> **Step 1:** Initialize μ, Σ, and π to their initial values.
> **Step 2:** For each data point, compute the responsibility of each Gaussian. (*Given the current guesses of the clusters, what's the probability this point belongs to each one?)*
> **Step 3:** Update μ, Σ, and π to better fit the data, based on the responsibilities.
> **Step 4:** If the difference between the current and previous values of the log likelihood is small enough stop; else, go back to Step 2.

For the initialization of parameters, below methods were used:

> **μ:** k number of values were randomly picked from the data.
> **Σ:** 2x2 identity matrix for each k.
> **π:** ⅓ for each k, since there are 3 classes.

For the main training loop, most of the initializations provided very good results, even though sometimes the results were suboptimal when the initialization was really poor. It was observed that setting a stopping criteria (the difference between the current and previous values of the log likelihood) of 1e-4 and providing an upper bound of 100 loops to the main loop generally provided satisfactory results. Further improvements are also possible such as setting the initial means to the result of an k-NN algorithm and making the chance of getting a poor initialization even lower.

Results

Below we present one of the results that was found to be satisfactory.

Trained for: 13 loops
Log-likelihood at the end of the training: -1231.69

Estimated parameters:

k = 1

μ = [9.61, 9.17]

Σ = [(2.01, -0.64), (-0.64, 0.82)]

k = 2

μ = [4.38, 4.35]

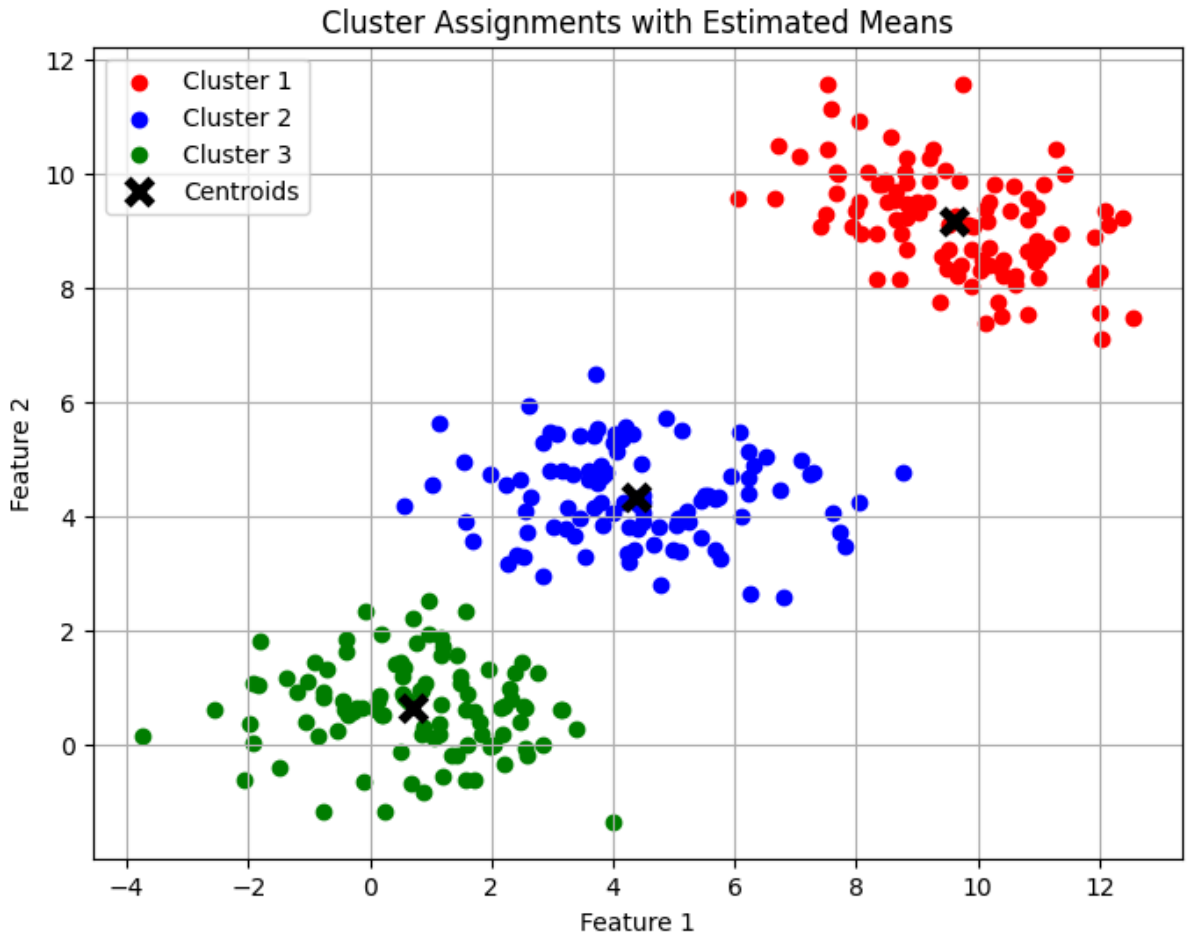
Σ = [(2.75, -0.12), (-0.12, 0.62)]

k = 3

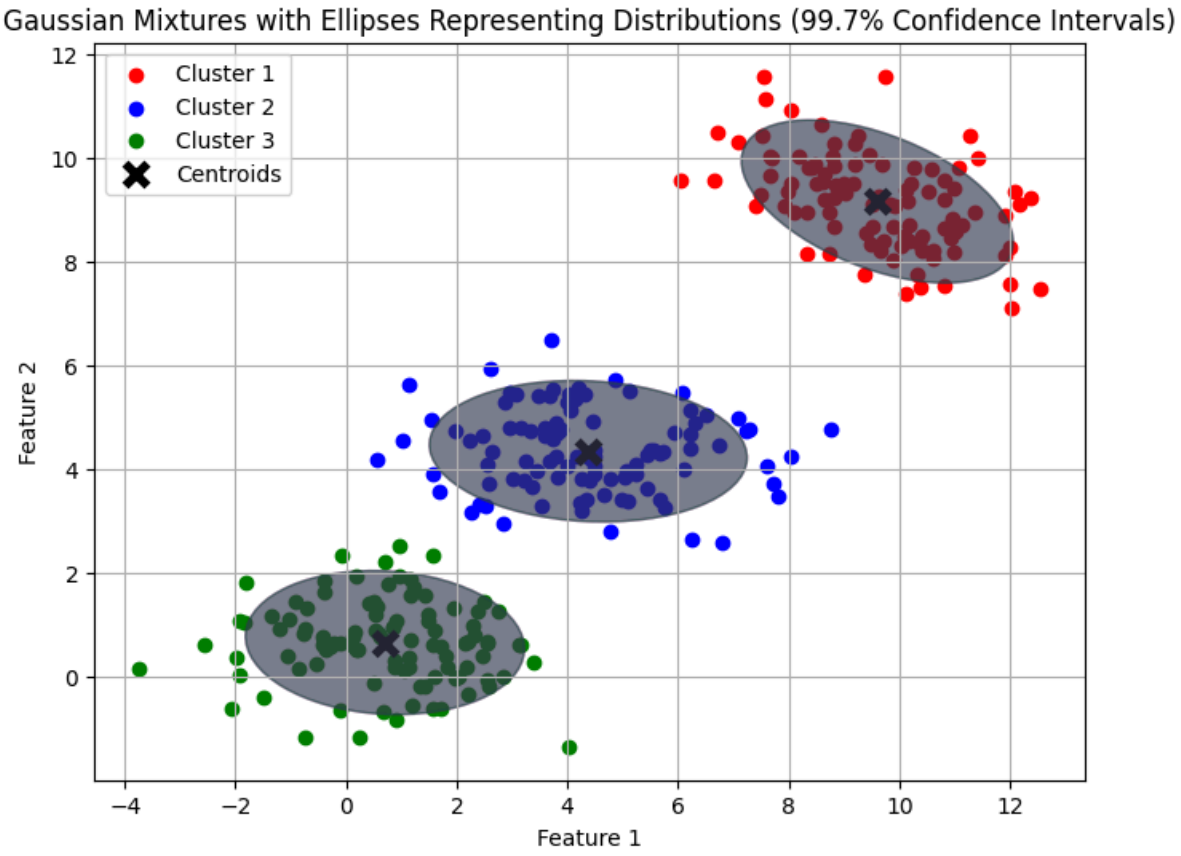
μ = [0.70, 0.66]

Σ = [(2.12, -0.10), (-0.10, 0.64)]

Below is the visualization:



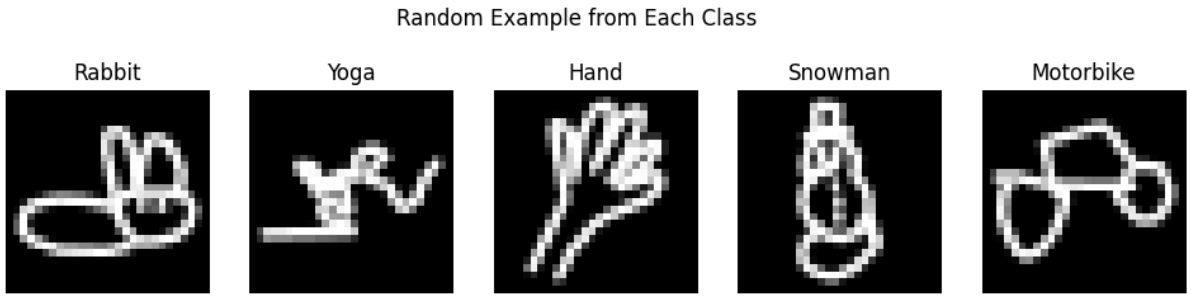
Below is the 99.7% of confidence interval of the gaussians:



Quick Draw Classification

Part 1. Exploration and Preprocessing

We are given a dataset of quick hand drawings where 28x28 pixel images depict a rabbit, yoga, a hand, a snowman or a motorbike. We are provided 20.000 images for training and 5.000 for testing. Below is an example demonstration of how our data look like:



The data is evenly distributed, meaning that both our training and testing sets include an equal number of examples from each class. Therefore, we do not need additional preprocessing steps to make sure that they are evenly distributed.

Feature Extraction

Working with image data can be quite challenging in terms of extracting features. Unlike tabular data, images contain spatial relationships, therefore, we cannot directly feed the raw pixel values into our classifiers. Hence, we need different methods to extract features.

Nowadays, most popular and effective methods usually use Convolutional Neural Networks to extract features from image data. It is possible to train a model from scratch or to use the first few layers of a pre-trained deep CNN model. However, for the scope of this assignment, we are not allowed to use these methods.

Another method that's known to work well for extracting features from simple images and sketches is Histogram of Oriented Gradients (HOG) [2]. HOG captures the direction of edges and shapes within small regions of the image, rather than relying on exact pixel values. For our case of simple hand drawn sketches, it is one of the simplest and most suitable methods to extract features.

Even though we have not implemented HOG from scratch within the scope of this assignment, the key idea of it is to:

1. Calculate differences of color densities (gradients) from each pixel to its neighbors.

2. Divide the image into a certain number of blocks, and for each block, create a histogram that counts how often certain edge directions occur.

3. Normalize the values across blocks.

4. Combine the blocks into a single feature vector.

In order to balance the accuracy and training speed below configuration for HOG was decided for feature extraction after a number of experiments:

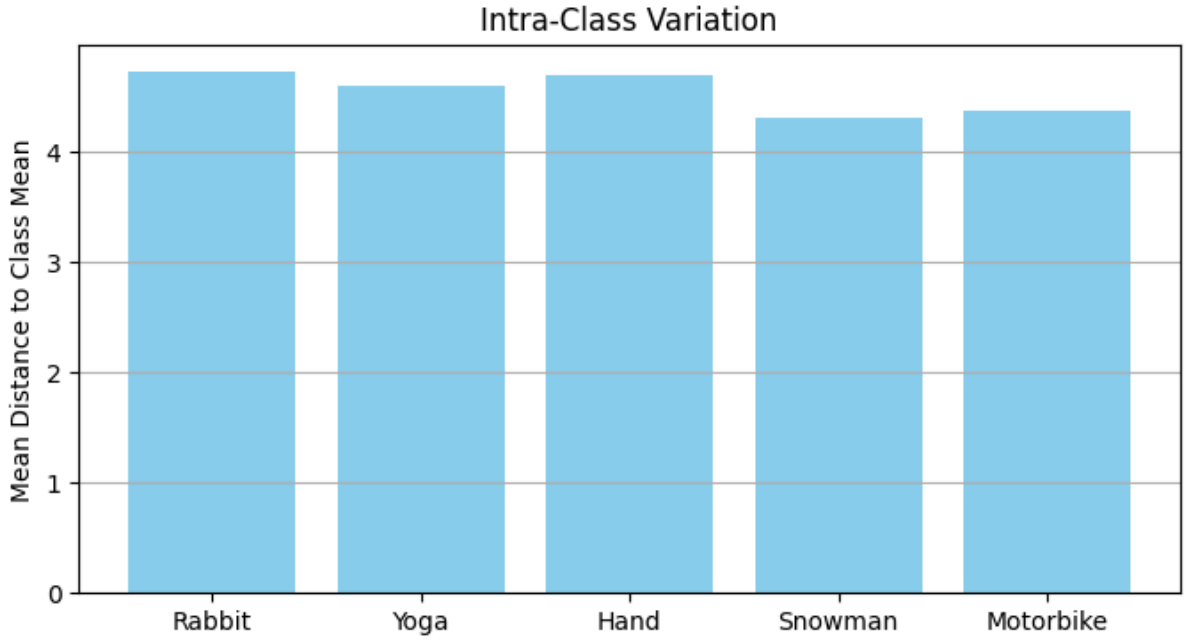
Orientations: 12
Pixels per cell: (4, 4)
Cells per block: (1, 1)

This configuration means that 28x28 images will be divided into 7x7 blocks. For each pixel, one of the directions from orientations will be assigned. This means that for each pixel the algorithm will be able to differentiate between a 180 degrees / 12 = 15 degrees of difference. The resulting feature vector will have 49 x 12 = 588 features per image.

Between and within class variations

After extracting the features, between and within class variations are studied, and the results are given below:

Intra Class Variations



Inter Class Variations

```
Inter-Class Distances (Centroid-to-Centroid):

Rabbit     vs Yoga       -> Distance: 1.4509
Rabbit     vs Hand       -> Distance: 1.5256
Rabbit     vs Snowman    -> Distance: 1.6532
Rabbit     vs Motorbike  -> Distance: 2.3065
Yoga       vs Hand       -> Distance: 1.5787
Yoga       vs Snowman    -> Distance: 2.0242
Yoga       vs Motorbike  -> Distance: 1.6974
Hand       vs Snowman    -> Distance: 2.2801
Hand       vs Motorbike  -> Distance: 2.2967
Snowman    vs Motorbike  -> Distance: 3.0103
```

As can be seen from the provided graphs, intra class variations are evenly distributed, meaning that each class has approximately the same amount of within class variance. For inter-class variance however, snowman vs. motorbike variances are especially higher than others, suggesting that they are most probably more differentiable from each other. We can also observe that rabbit and yoga generally have lower inter-class variance suggesting that

these two classes are more similar to each other in terms of their feature distribution, making them harder to differentiate. This could be attributed to potential similarities in the overall shape or structure of the images within these two categories, which results in a smaller distance between their centroids.

Part 2. k-NN

k-NN is a simple algorithm that works by assigning the most probable estimate to a test sample by visiting each data point in the training set and calculating a distance value based on a distance metric of our choice. For each test sample, the algorithm computes a distance metric (such as Euclidean distance) to measure the similarity between the test sample and every point in the training data. The k-NN algorithm then identifies the k closest training samples and assigns the most frequent class label among these neighbors as the predicted class for the test sample. The key idea is that closer samples tend to be from the same classes.

There are 2 hyperparameters to decide on in k-NN: the k value and the distance metric. For the training we decided to evaluate the performance of the model using k values 1, 3, 5, 7, 9 using the euclidean distance since it is the most commonly used distance metric in k-NN, especially for continuous data like pixel values in images. Below are the results, the best test performance is around 0.85-0.86 accuracy.

k	**Accuracy**
1	0.8376
3	0.8510
5	0.8546
7	0.8574
9	0.8602

Part 3. Gaussian Naive Bayes

The Gaussian Naive Bayes classifier is based on the Naive Bayes algorithm where we assume that the features, conditional on the class, follow a Gaussian distribution. What makes this classifier is Naive is the fact that we assume all the features are independent of each other and therefore:

$$P(X|y) = P(X_1|y) \, . \, P(X_2|y) \ldots\ldots\ldots\ldots P(X_n|y)$$

What makes it Gaussian is the fact that we assume that each $P(X_i|y)$ has a Gaussian density.

Our implementation has 2 main functions:

> **1. fit:** Inside the fit function, we take the features and labels, and for each class, we calculate their mean and variance for their feature vectors. We also calculate the prior for this class by dividing the number of data points that belong to that class to the total number of data points. In our case we have an even distribution therefore we expect the priors to be the same for each class and equal 1/5.
>
> **2. predict:** After fitting the training data, test data can be given to the model to make predictions. For each data point, the model computes the posterior probability for each class using Bayes' Theorem. We then select the class with the highest posterior probability as our prediction.

Abovementioned Gaussian Naive Bayes implementation works significantly faster than k-NN and logistic regression, however, it provides lower test accuracy equal to **0.6292**. The reason lies in the assumption, our features are not independent of each other since we are working with image data. The images contain spatial information, meaning that our features (shapes, directions, edges) highly affect each other to construct an object.

Part 4. Logistic Regression

In this part we use the Multinomial version of the logistic regression model. The aim is to minimize the cross entropy loss function using gradient descent. The non-linear function that maps regression output to probabilities is softmax, where the probabilities for each class add up to 1 and indicate the probability of that data point to belong to that class.

Our implementation also incorporates L2 regularization to prevent overfitting by penalizing large weight values. During training, we tuned the following hyperparameters to optimize model performance:

- **Learning rate:** controls the step size of gradient updates
- **Number of epochs:** determines how many times the model iterates over the training data
- **Regularization strength (lambda):** balances the trade-off between fitting the training data and keeping the model weights small.

The most difficult part of the training was the speed of training and the need to optimize the hyperparameters. After many experiments below hyperparameter configuration was used:

Learning rate = 0.001
Epochs = 50.000

Lambda = 0.001

The test accuracy with this configuration was **0.8308** with a training and test loss as below:

It should be noted that training took around 1.5 hours, suggesting that SGD or mini-batch approaches can be more suitable for this task.

Part 5. Comparison of Models

Below is the summary of all the models:

Model	**Speed**	**Test Accuracy**
k-NN	~2 minutes	0.8602
Gaussian Naive Bayes	<1 second	0.6292
Logistic Regression	~1.5 hours	0.8308

All three models offer different advantages and disadvantages in terms of accuracy, scalability, and efficiency.

- **k-NN** seems to achieve the best test accuracy, however, it struggles to scale as the number of test samples increase. This limits its practicality in real-time or large-scale scenarios.

- **Logistic Regression** performs nearly as well, with a test accuracy of 0.8308. While the training process is time-consuming (taking ~1.5 hours), once trained, the model can make predictions very quickly. This makes it well-suited for deployment where fast inference is critical, and training can be done offline.

- **Gaussian Naive Bayes**, despite having the lowest test accuracy (0.6292), offers significant advantages in terms of training speed—it trains in under a second. This makes it an attractive choice in situations where quick model updates or rapid prototyping is needed. Its lower performance is likely due to the "naive" assumption of feature independence, which often doesn't hold for image-based or spatial data where pixel values tend to be correlated.

Finally, based on the results above, logistic regression, a linear discriminant function classifier performed comparably well, suggesting that the features are at least approximately linearly separable, as a linear model was sufficient to capture the decision boundaries between classes.

References

[1] https://moodle.bogazici.edu.tr/pluginfile.php/1790531/mod_resource/content/1/EM.pdf

[2] Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee, 2005.

[3] https://scikit-image.org/docs/stable/api/skimage.feature.html