**Assignment 1**

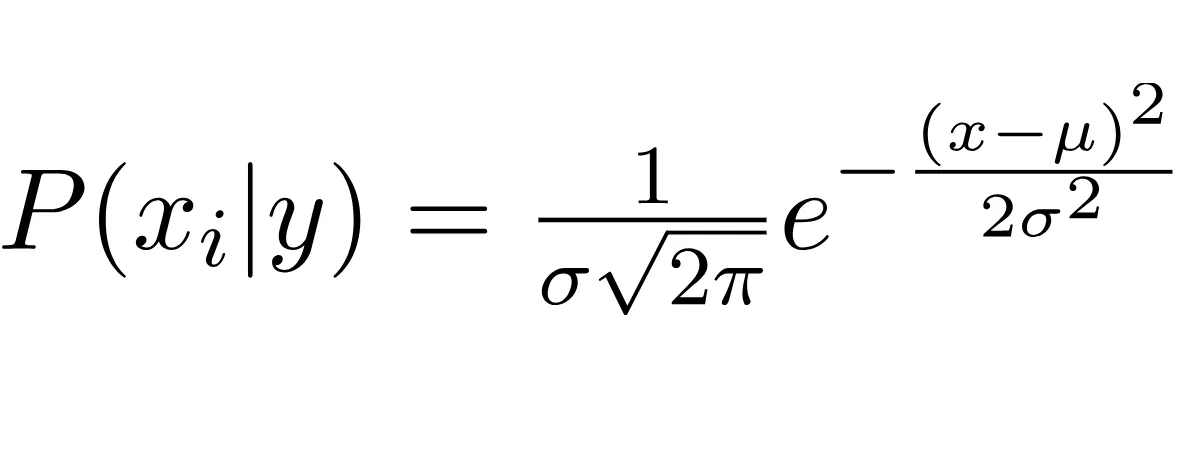
Team Name : BHCV

Team Members :

Mohamed Ayman Bahy , 52-1096 , T-2 mohamed.elgamal@student.guc.edu.eg

**Task 2: Naïve Bayes Classifier**

The Gaussian Naïve Bayes algorithm assumes that the features in the dataset are normally (Gaussian) distributed. This means that the values of each feature follow a bell-shaped curve when plotted on a graph , and under the naive assumption that the features are independent . The probability of feature 𝑥ᵢ belonging to class y is calculated using the following formula :

 With σ and μ being the standard deviation and the mean of feature 𝑥ᵢ in class y respectively .

The probability of x as a whole belonging to class y is :

**P(x | Y) = log(P(Y)) + Σ [ log(P(𝑥ᵢ | Y)) ]**

The logs are being used to avoid the numerical underflow caused by multiplying many small probabilities. With the final prediction being the maximum probability between the 2 classes (0 & 1) in case or RGB and Grayscale images and (1-7 excluding 6) in case of multi-spectrum images .

Two datasets were used in this task the first one is the [BSDS Dataset](https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/) providing the RGB images which was later converted into Grayscale to provide the dataset for the Grayscale images and a threshold of 128 was used to create the ground truth with the grayscale values above 128 being set to 255 and the others are set to 0. The other dataset used was the [Statlog Dataset](https://archive.ics.uci.edu/dataset/146/statlog+landsat+satellite) providing images with 36 bands of colors and 6 classes to predict that being from 1 to 7 excluding 6 which had no datapoints .

**train\_test\_split(images\_directory, labels\_directory)**

This method is given the directory of the data using a mode selection variable, reads the images and labels from the given directories, shuffles the data and makes the train-test split, saves a random test image to be used later for visualization and finally return the x\_train, y\_train, x\_test, and y\_test arrays .

**BayesModel(data, truth)**

This method is given the x\_train and the y\_train as an array of arrays, each sub-array has length 1 (grayscale) or 3 (RGB) and the truth which is an array of class labels (0 or 255) and returns the model which is a dictionary containing class probabilities and channel statistics . There is another version of this method called **BM\_multi\_spectral(data, truth)** for multi-spectrum images . The code figures out which BM to use based on the number of color channels (the length of the sub-arrays) which can be 1 , 3 or 36 .

**BayesPredict(model, test\_data)**

Given the model and the test\_data , this method produces a label for each datapoint in the array called test\_data using the equation described above and returns the labels , and similarly there is another version of this method called **BP\_multi\_spectral(model, test\_data)** for multi-spectrum data with the same concepts except that this predicts over 6 classes instead of 7 .

**ConfMtrx(y\_test, lbl)**

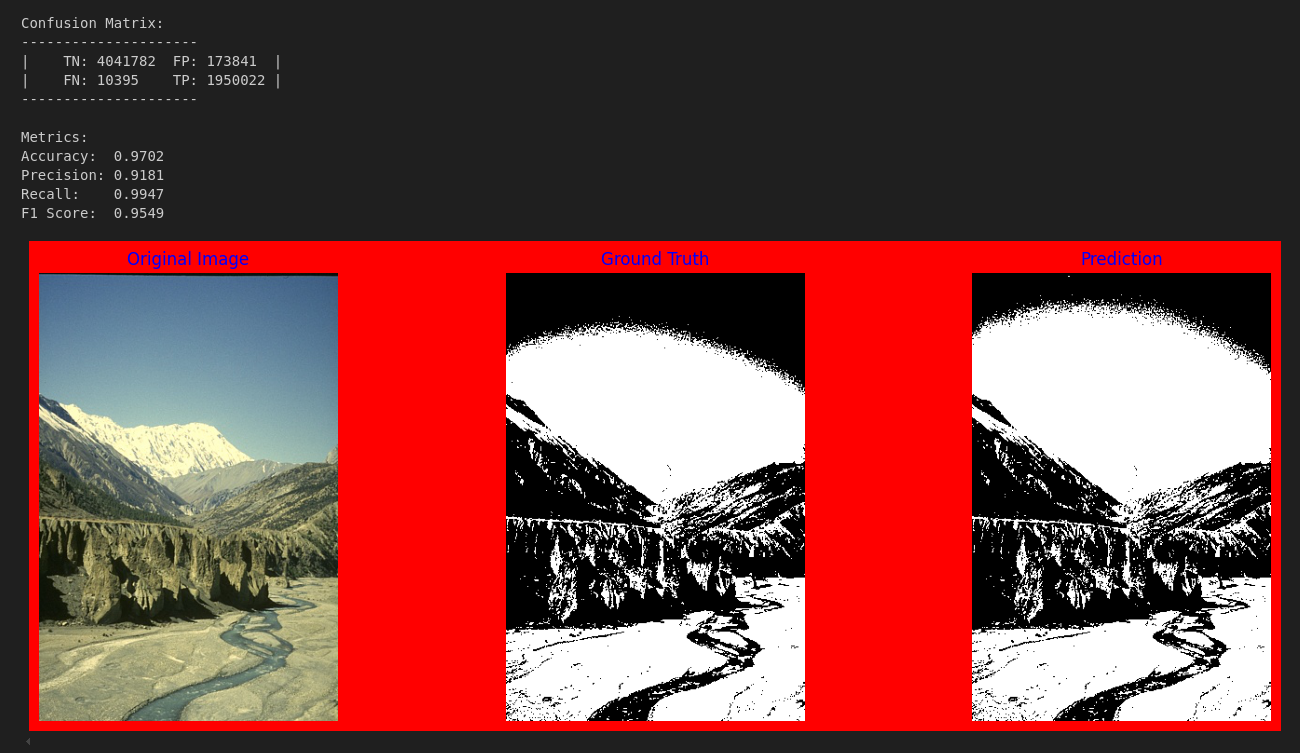
Simple method to print out the confusion matrix containing (True Positives, False Positives, True Negatives, and False Negatives) given the predicted labels and the actual labels. The multi-spectrum version of this method is called **ConfMtrx\_multi\_spectral(actual, predicted)**

**visualize(model)**

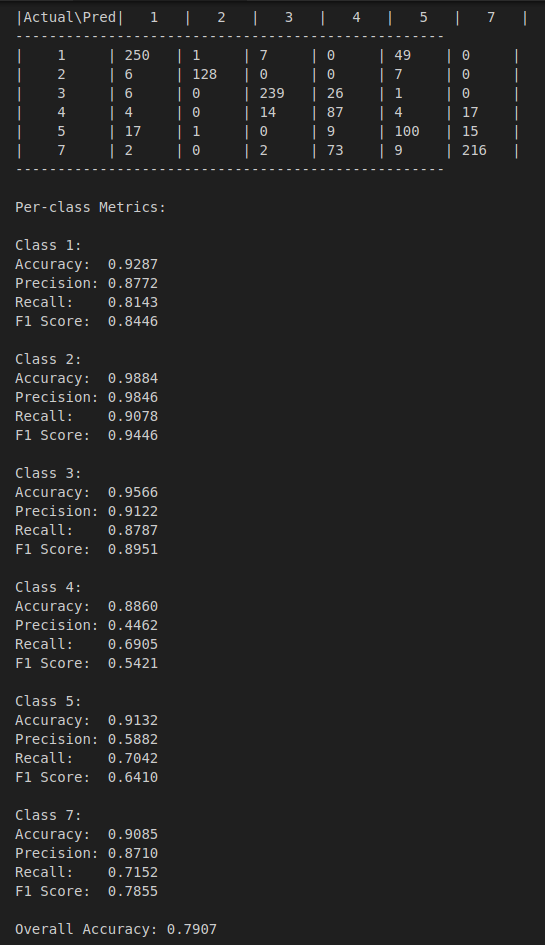
This method draws a comparison between the original test image saved during the train test split and the ground truth image obtained by binarizing the image on the spot and the predicted image outputted by running the given model on the original image . Similarly there is a method called **viz\_multi\_spectral(actual, predicted)** for multi-spectrum images, this one draws a comparison between the actual classes vs the predicted classes .

**Results**

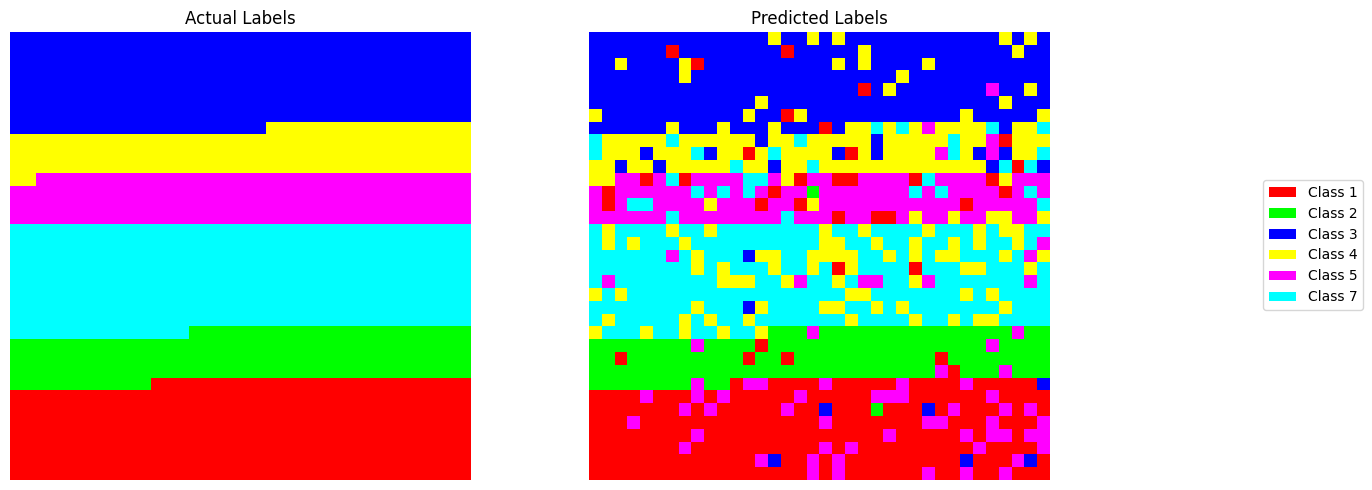
Grayscale

RGB

multi-spectrum



multi-spectrum



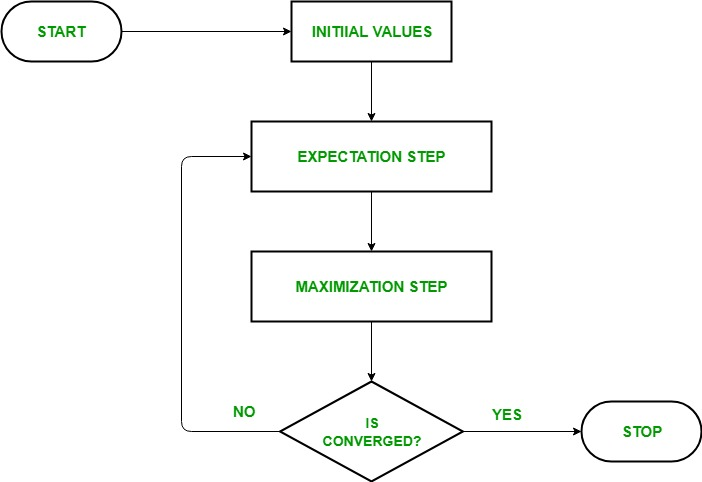
**Task 3: Expectation Maximization algorithm**

EM is an iterative algorithm that initially guesses some parameters those being the means for each feature in each class , the covariance matrix (initially is the identity matrix) and the mixing coefficients . Using those parameters it assign each pixel to a class by the following equation :

**[ 1/sqrt((2π)^d|Σ|) \* exp(-0.5(x-μ)ᵀΣ⁻¹(x-μ)) ]**

**X**

**[ mixing coefficients ]**

And this is calculated for each class , then normalized and the pixel is assigned to the class with the higher probability. Then the parameters of each class is updated using the members in the class, this process is repeated a 100 times. EM can be visualized as follows :

**GaussianMixtureModel\_ByHand**

This class implements the Expectation-Maximization (EM) algorithm with the parameters : n\_components (int): Number of classes (default=2 for foreground / background ) and max\_iter (int): Maximum number of iterations (default=100) . This class contains the following methods :

**initialize\_parameters(self, X)** : Initialize model parameters randomly from the data.

**gaussian\_pdf(self, X, mean, cov)** : Compute multivariate Gaussian probability density function by applying the equation described above .

**e\_step(self, X)** : Expectation step, compute probability of each pixel belonging to each cluster.

**m\_step(self, X, resp)** : Maximization step, update model parameters based on current responsibilities.

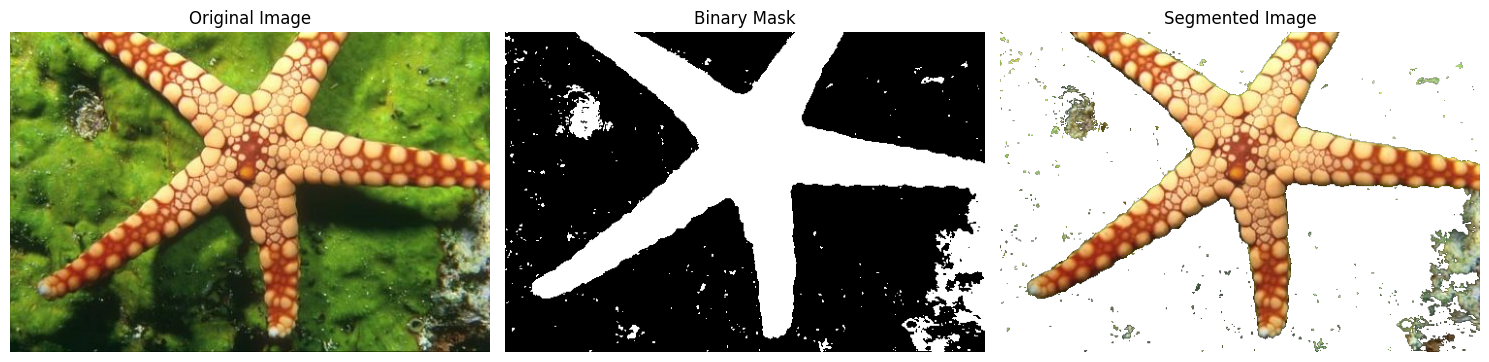
**fit(self, X)** : Fit the GMM to the pixel data using EM algorithm.

**predict(self, X)** : Predict cluster assignments for pixels.

**GMM (original\_image\_path)**

This is the only function used in the task , it reads the image and shape it into a 2D array with the length being the total number of pixels and the width being the color channels in the image . It chooses wether to fit using the per-defined model from sklearn or the custom implemented model based on a variable set by the user . It then fits the model on the pixels and then predict them to produce the labels (0 or 1) for each pixel. The class with the lower number of pixels is considered the object class and the other is considered the background class . This labels array is used to create the binary mask which is later multiplied by the original image to produce the segmented image . Finally the method draws the original image vs the binary mask vs the segmented image

**Results**

****

**Naïve Bayes vs Expectation Maximization**

|  |  |  |
| --- | --- | --- |
|  | **NB** | **EM** |
| **How it works** | * Assumes all the features are independent * Uses Bayes Theorem to calculate the probabilities | * Iterative approach to improve the parameters * Data is modeled as a mixture of Gaussians |
| **Type** | * Supervised | * Unsupervised |
| **Time taken** | * Slow as there is a training step on all the dataset before testing your image | * Fast as you train on your test image directly |
| **Limitations** | * Needs labeled data * Performs well only if trained on similar images | * Dependent on the start point (random initialization) |
| **Clustering** | * Hard | * Soft |
| **Pros** | * Easier to understand | * More flexible as it learns from the image itself. |