# Machine Vision

# ASSIGNMENT 1


## Kamakshi Bansal

## 16114486


# Part 1- Mixtures of Gaussians

**Folder**: Part I
**Code**: practicalMixGaussA

**To Do (a)** : mean and covariance of the Gaussian for RGBSkin and RGBNonSkin pixels is calculated here in the fitGaussianModel function.

$$\text{Mean} = \mu = \frac{\sum_1^I x_i}{I}$$

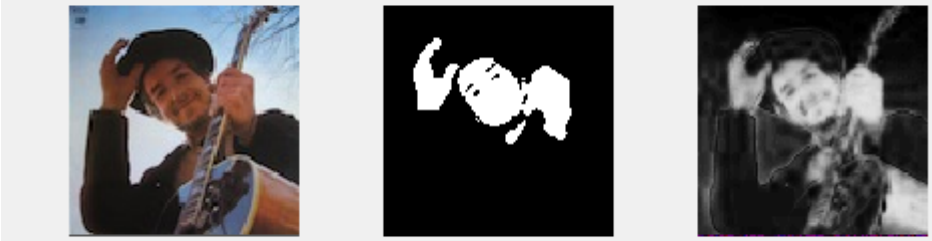
$$\text{Covariance} = \frac{(X-\mu)(X-\mu)'}{I-1}$$

**To Do (b)** : Likelihood (Gaussian probability) for each pixel data is calculated in the function calcGaussianProbability.

$$Gaussian\ Probability = \left(\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}\right) * \exp[-0.5 * (x-\mu)^T \Sigma^{-1}(x-\mu)]$$

**To Do (c)** : Posterior probability for each RGBskin pixel using the learned parameters and priors is calculated here using the Baye's Rule which results in the third image in the output below :

$$\Pr(w=1|x) = \frac{\Pr(x|w=1)\Pr(w=1)}{\sum_{k=0}^{1}\Pr(x|w=k)\Pr(w=k)}$$ (Generative Model Approach)



**Output:**

1. The first image is used as an input whose skin is to be detected.
2. The second image represents the ground truth image of the detected skin from the first image.
3. The third image is the output of the posterior probability of RGBskin (To do (C)), using the generative model approach, skin pixels can be seen clearly .

## **Code**: practicalMixGaussB

**To do (d)**: For a Gaussian with known mean μ and variance $\sigma^2$ , we can generate the sample data for this normal distribution by using : $data = \sigma x + \mu$ where $x$ is the sample from standard normal distribution with mean 0 and variance 1. Since, here we are using multivariate Gaussians, hence data is generated by randomly selecting mean and covariance (randomly choosing the value of h i.e. either h=1 or h=2 ) out of the two Gaussians by using:

$$data = mixGauss.cov(h) * randn(1) + misGauss.mean(h)$$

Where $randn(1)$ generates random numbers from standard normal distribution.

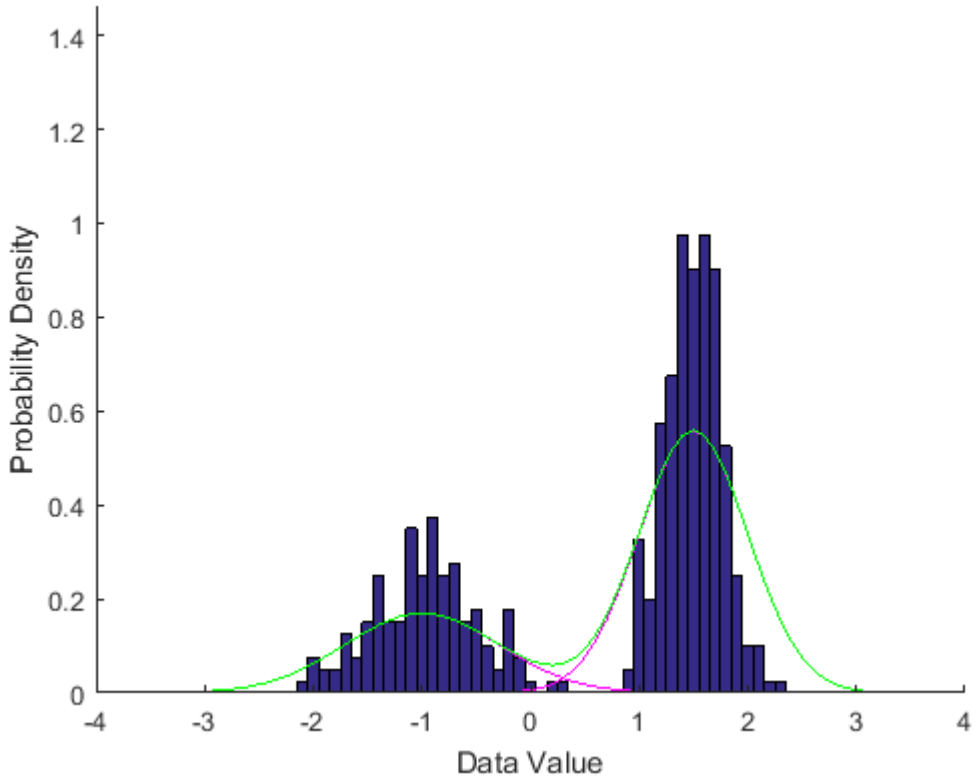
**To do (e)** : Here log of likelihood is calculated using :

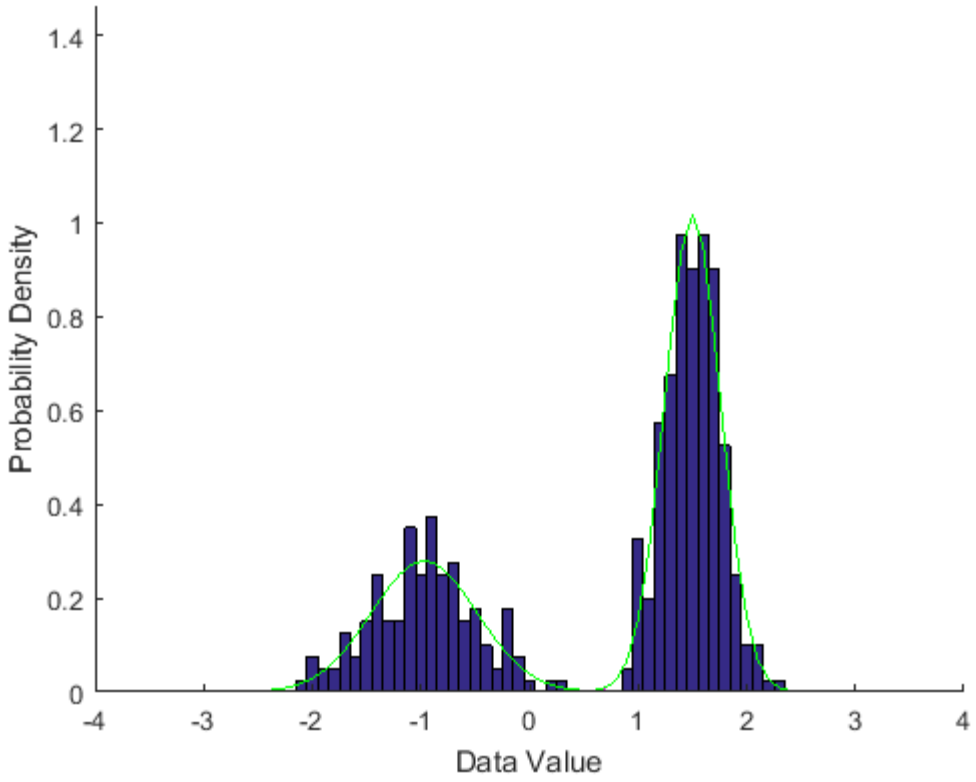
$$L = \sum_{i=1}^{I} \log[\sum_{k=1}^{K} \lambda_k Norm_{x_i}[\mu_k, \Sigma_k]]$$

Log of likelihood is taken because it allows to avoid the computation of exponentials in Gaussians and help representing small numbers efficiently in matlab, infact log x is a monotonically increasing function and hence will yield the same results as likelihood.

**Outputs:**

1. The first image is the ground truth.
2. The second image is the output from the program which we got after doing the expectation step where the likelihood is calculated and the maximization step where the Gaussian parameters are updated for the next iteration till the Gaussian fits the data completely as can be seen from the output below.






**Code**: practicalMixGaussC

To do (f) : The sampling of data is done here by choosing randomly (using h) the Gaussian parameters (mean (μ) and covariance (∑) ) for any one of the Gaussian out of the three Gaussians using :

$$data = chol(mixGauss.cov(h) * randn(1)) + misGauss.mean(h)$$

Where $randn(1)$ generates random numbers from standard normal distribution and cholesky decomposition.

http://stats.stackexchange.com/questions/32169/how-can-i-generate-data-with-a-prespecified-correlation-matrix

**To do (g)** : This is the Expectation step where the likelihood and responsibilities are calculated i.e. posterior probability of each data point which defines for which hidden variable(i.e. for which of the three Gaussians), the Gaussian probability is higher.

$$Responsibilities = r_{ik} = l_{ik}/(\sum_{k=1}^{K} l_{ik}$$

**To do (h):** This is the maximization step, here the Gaussian parameter weight ( $\lambda$ ) is updated for each gaussian ready to be used for the next iteration.

$$\lambda_k^{[t+1]} = \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{k=1}^{K}\sum_{i=1}^{I} r_{ik}}$$

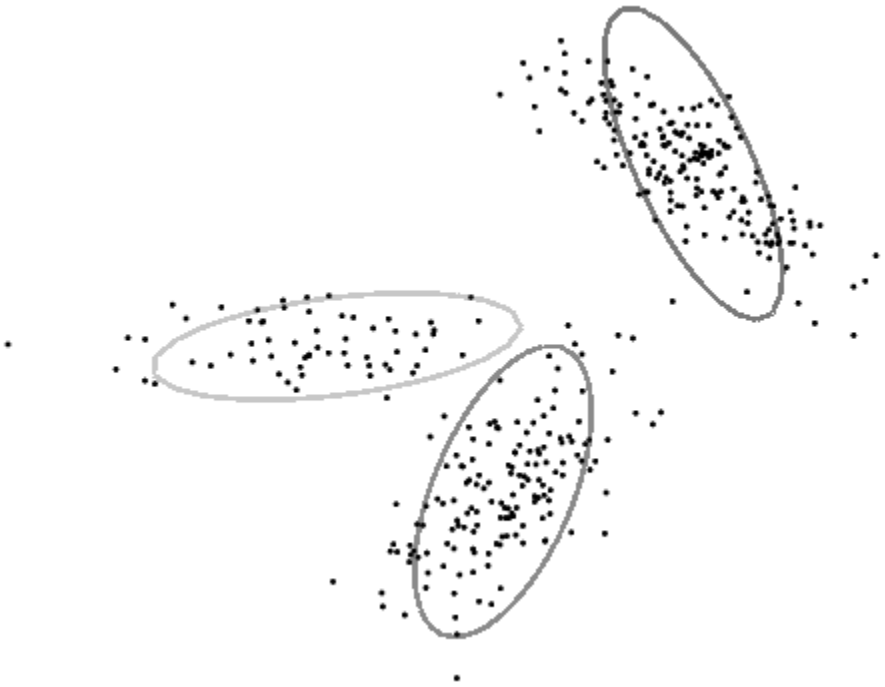
**To do (i):** This is also part of the maximization step, here the Gaussian parameter i.e mean ($\mu$ ) is updated for each Gaussian ready to be used for the next iteration.

$$\mu_k^{[t+1]} = \frac{\sum_{i=1}^{I} r_{ik} x_i}{\sum_{i=1}^{I} r_{ik}}$$

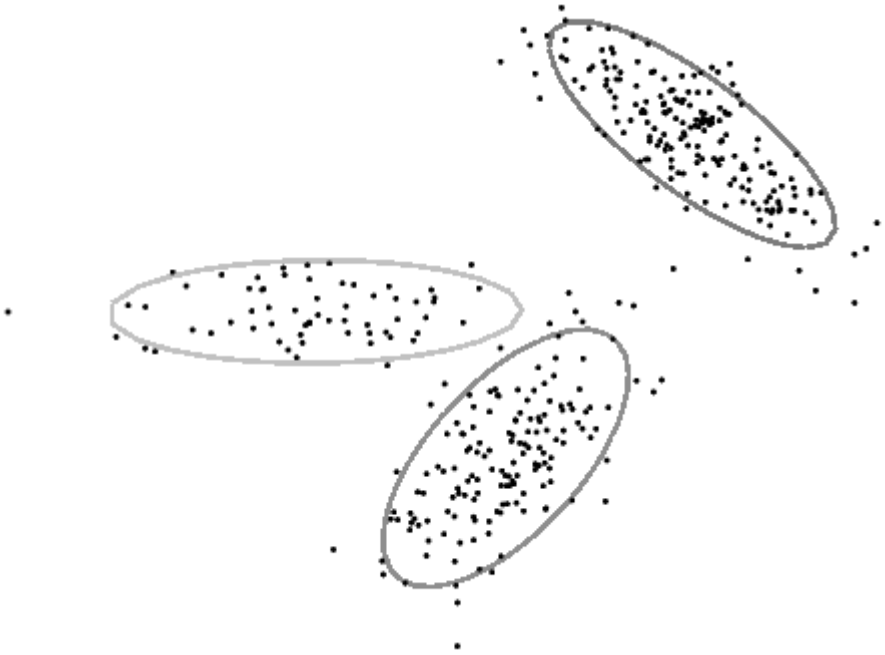
**To do (j)** : This is also part of the maximization step, here the Gaussian parameter i.e covariance ( $\sum$ ) is updated for each Gaussian ready to be used for the next iteration.

$$\Sigma_k^{[t+1]} = \frac{\sum_{i=1}^{I} r_{ik} x_i \left(x_i - \mu_k^{[t+1]}\right)\left(x_i - \mu_k^{[t+1]}\right)^T}{\sum_{i=1}^{I} r_{ik}}$$

Ground Truth :



The  Final Output :



# ASSIGNMENT 2

# Part 2 - Apples

Goal : To classify apples and non apples i.e our code should be able to detect apples from the images.

- MainCode – for training the model with Mixture of Gaussian and testing at the bottom part of the code
- TestCode.m – for Testing the image with RGB normalization
- crossValidation.m – for creating training set using two image and validating on third image.

**MainCode.m** :

Trainning Data generation :

1. Images are loaded from the folder apples where curI is the original image and curmask is the masked image where the apple skin is white (1) and non apple skin is black (represented as 0 pixels).
2. Normalizing the images : Earlier each image was divided by 255 but was not giving a good output, instead a Normalization function is created to process the images i.e make each image pixel of same intensity by dividing each Red, Green and Blue component with $\sqrt{Red^2 + Green^2 + Blue^2}$

3. Apple data and non apple data is generated for the training of the Gaussians ie to get the Gaussians parameters which can be used to test the other images.
4. By multiplying the pixels of the image with the masked image, we get the apple pixels and by multiplying the image with the inverted masked image, we get the non apple pixels.
5. We could see, that there are many pixels which have all the three components as zero ie RGB pixel= 0, we need to remove these 0 pixels which is done in the function RemoveRedundantPixels( ).

Parameters Estimation :

6. Then the generated apple data and non apple data is sent once at a time to the function fitMixGauss (data, k) which takes the data and the no. of hidden variables (for the

Gaussians) as input and does the Expectation and Maximization steps to return the Gaussian parameters (weight, mean and covariance) for both apple data and non apple data respectively .

Posterior Calcultaion for Apple data :

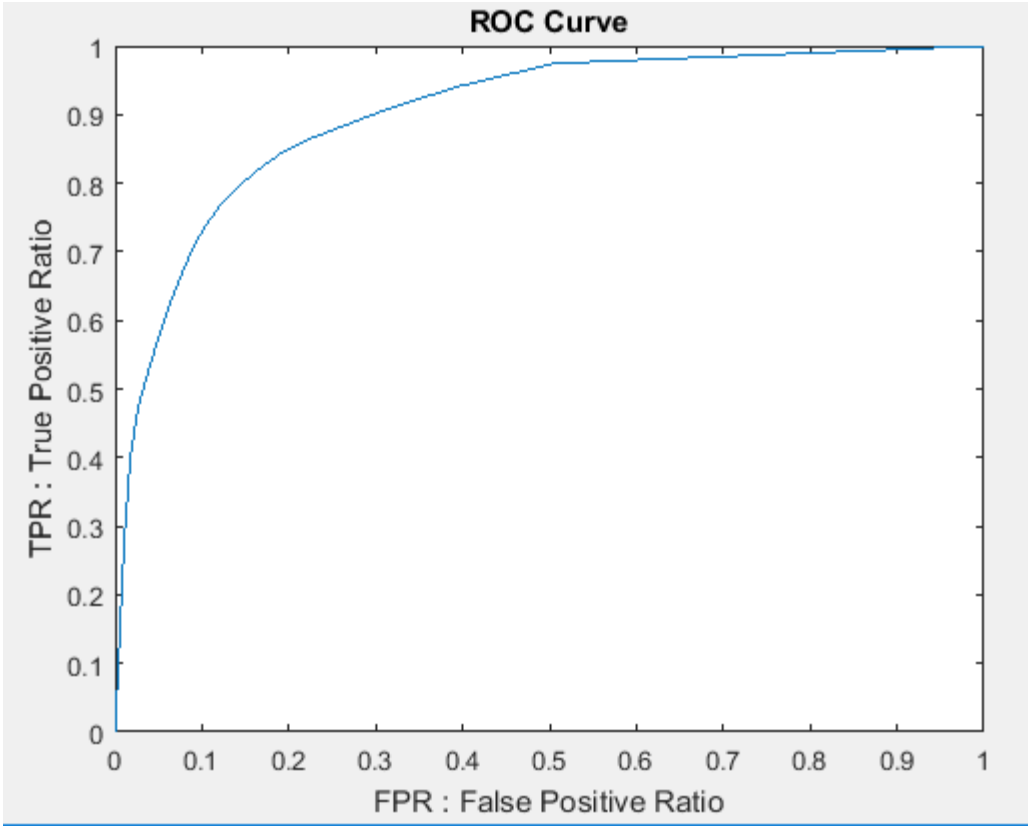
7. After we get the trained parameters, we send these to the function calculatePosteriorApple( Apple_Data_Parameters, Test_image, Non_Apple_Data_Parameters) where posterior is calculated to detect the apples.
8. ROC (Receiver Operator Characteristic)
    i) This is done in the function RocCurve( )
    ii) It takes the two parameters as inputs : the test image's posterior probability and the ground thruth mask of the test image.
    iii) For different threshold values of the posterior probability, True positive, false negative, true negative and false positive are counted using the and relationship between the first two columns.

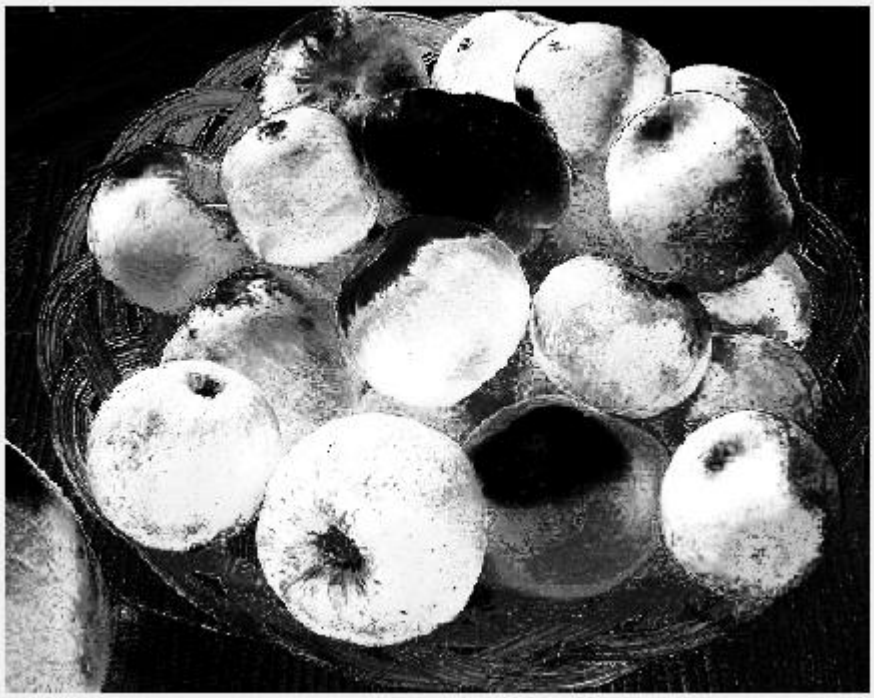
| Test Image | Ground Truth Mask | output |
|---|---|---|
| Apple Pixels | Apple Pixels | True Positive (TP) |
| Apple Pixels | Non Apple Pixels | False negative (FN) |
| Non Apple Pixels | Non Apple Pixels | True negative (TN) |
| Non Apple Pixels | Apple Pixels | False Positive (FP) |

    iv) Then using the formula below, the TPR(True positive ratio) and FPR (False positive ratio) are calculated to plot the ROC curve.

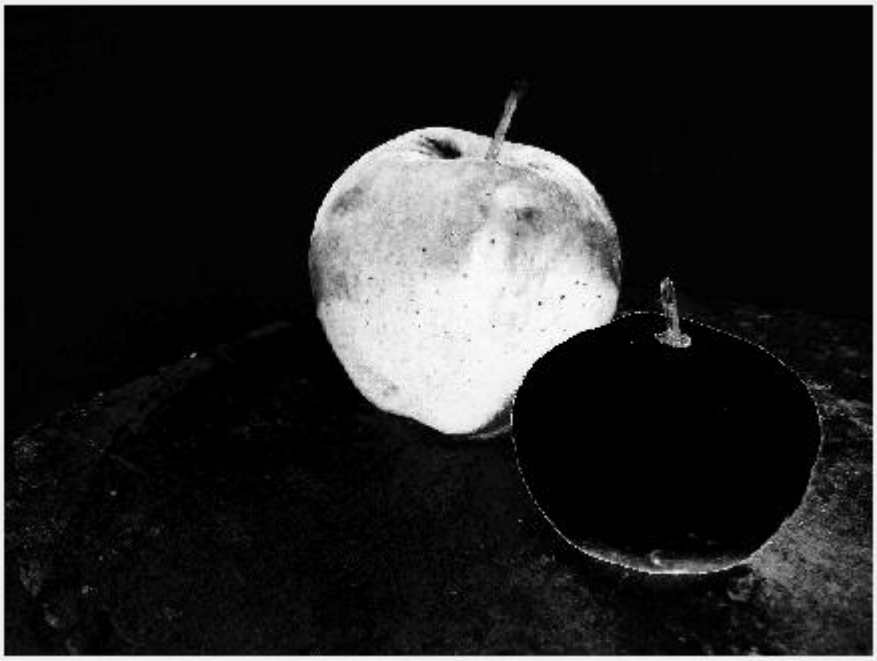
$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

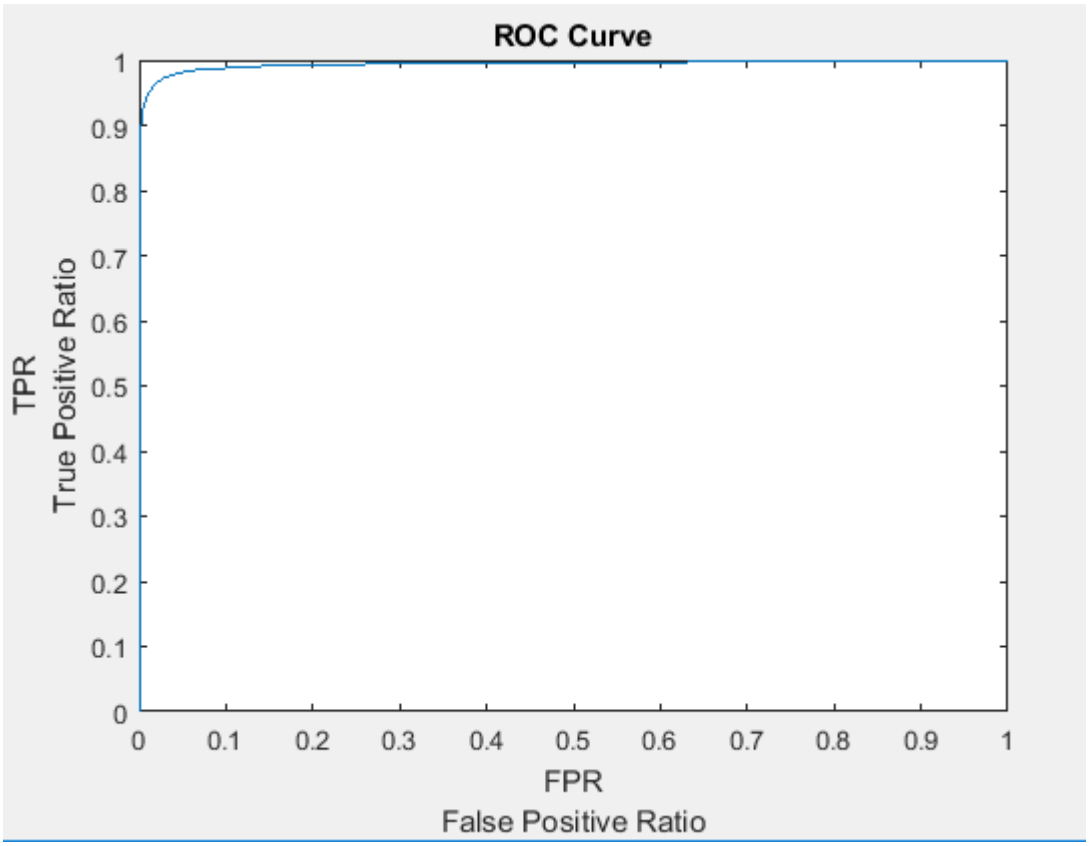
    v) Roc Curve plots the performance of the binary classifier with respect to the threshold values.


ROC Curve
TPR : True Positive Ratio
FPR : False Positive Ratio
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



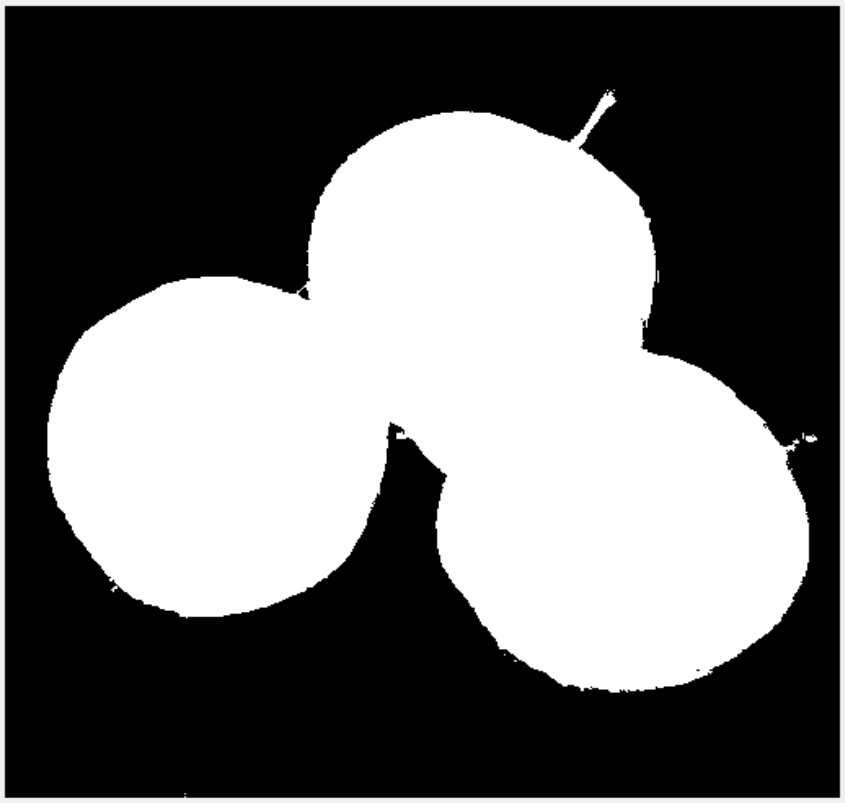
For Testing :

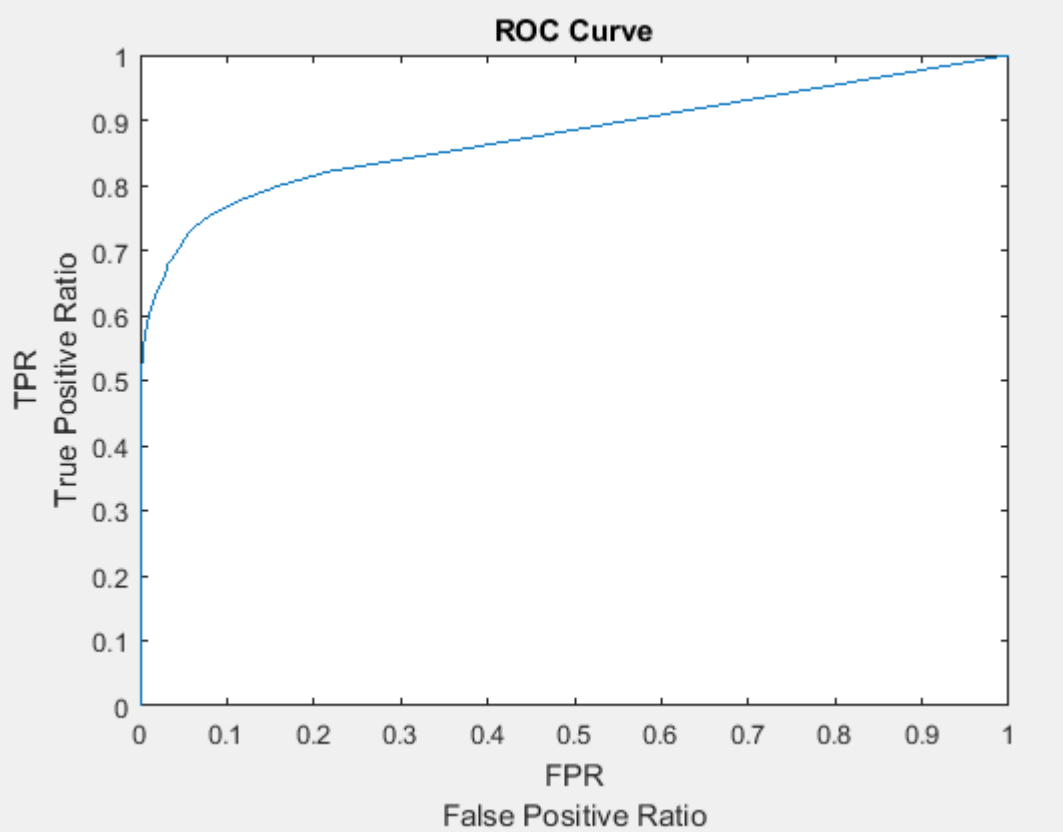



ROC Curve
TPR
True Positive Ratio
FPR
False Positive Ratio
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

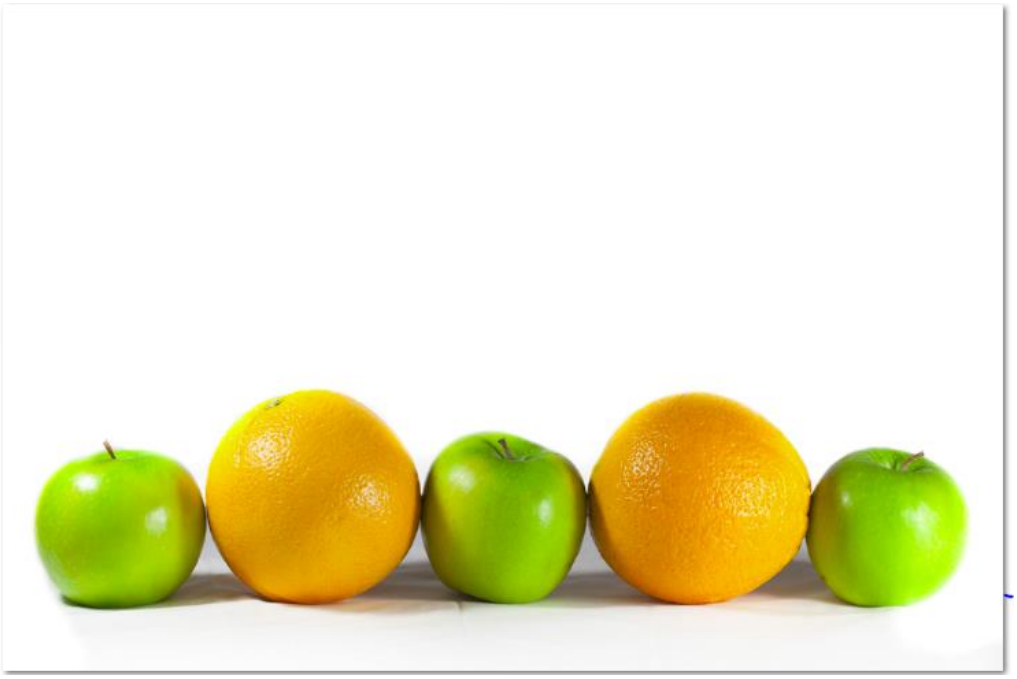
Part E :

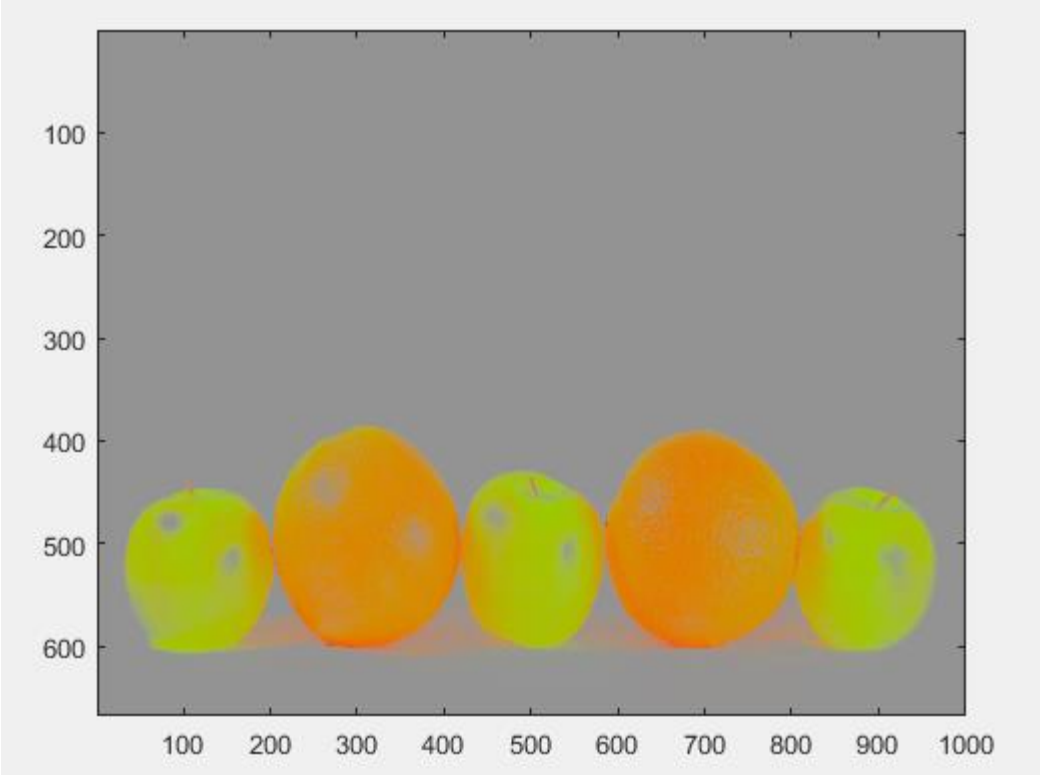

50
100
150
200
250
300
350
400
450
500

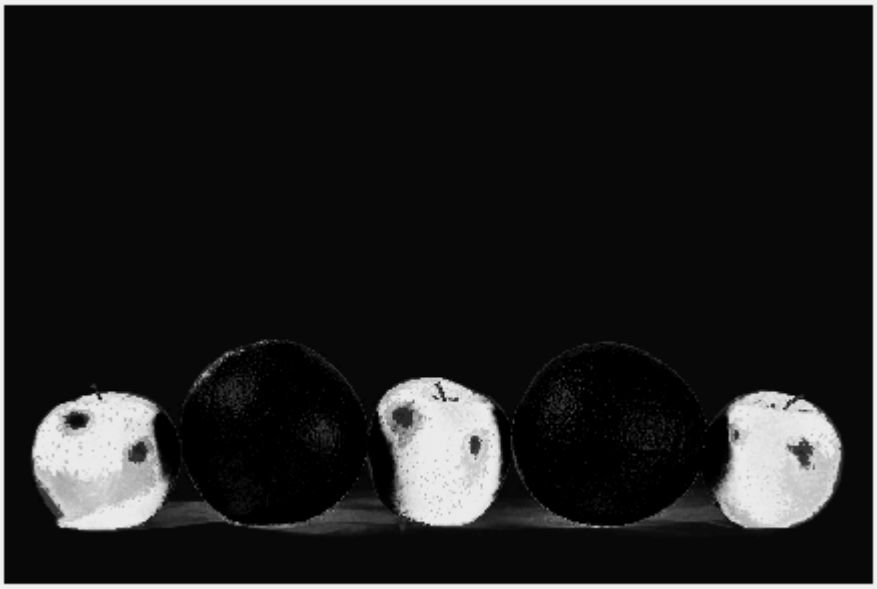


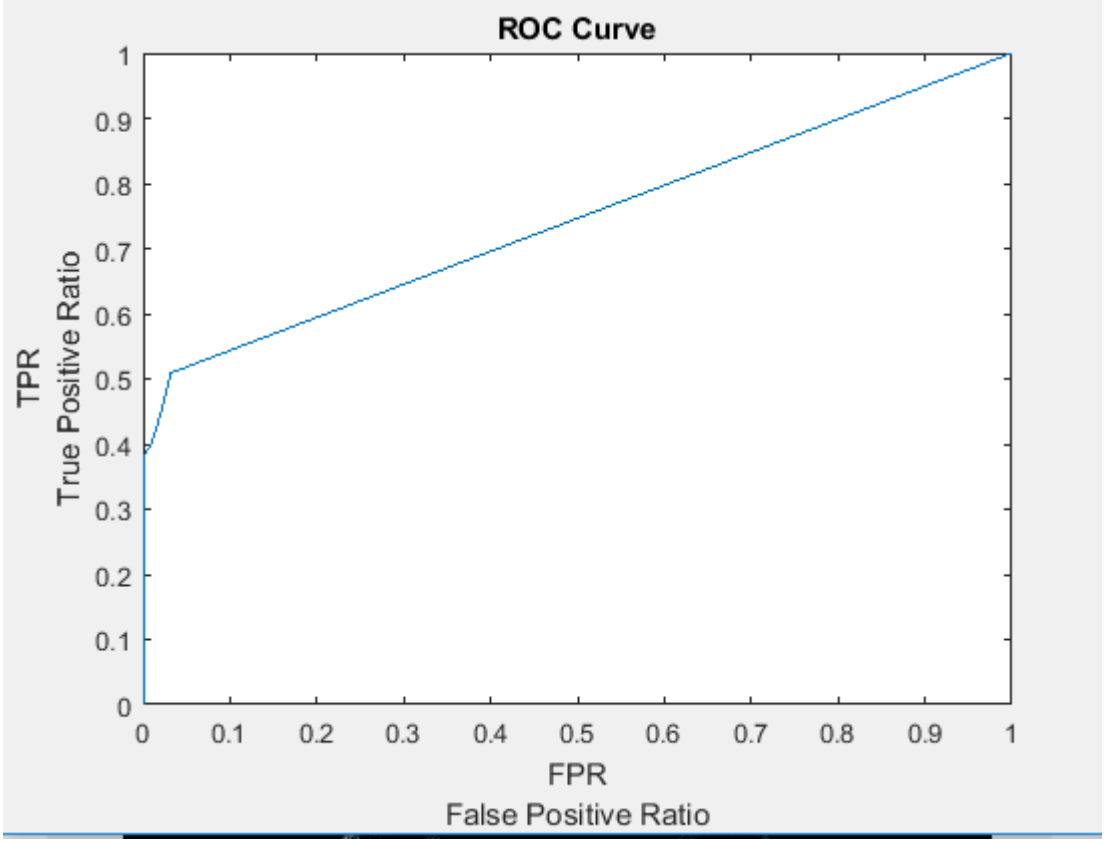



Test Image 2 (with RGB normalization) :




100
200
300
400
500
600
100
200
300
400
500
600
700
800
900
1000






Roc Curve plots the performance of the binary classifier with respect to the threshold values.

Part F

- Always we train our model using the training data (like here we gave few apple images to get the learned Gaussian parameters) then use that model on different images called as test set, to check whether our model is able to generalize the trend or not.
- If we train the model with our test images, it might lead to over fitting problem (where our model tends to learn the parameters instead of generalizing the trend) and might not be able to predict the correct output if given the new test images.
- Sometimes when we do not have the test set, we train our model by dividing the training data into two sets, training set and validation set. We first train our model using the training set and test our model using validation set like in the code, training is done on the two images 2 and 3 and a validation set is prepared on the third image.