# CSE 353 Assignment 2
# Generative Discriminative Models Report

Chengzhi Dong
ID 112890166

Due October 4, 2021

## 1 Introduction

In this assignment, I will implement the Generative model for face classification. I am given a training data set and a testing data set of faces and non-face background images. I will design an algorithm that can determine whether an image belongs to the face-class or the non-face-background-class. I use the Generative model on the training data-set by choosing multivariate normal distribution for $Pr(x)$ with parameter $\theta = (\mu, \Sigma)$. Then, I use maximum likelihood estimation (MLE) to calculate the parameter $\theta = (\mu, \Sigma)$ and the likelihood $Pr(x|y) = Norm_x[\mu_y, \Sigma_y]$. Then, I use the likelihood $Pr(x|y)$ and the prior $P(y)$ to compute the posterior $Pr(y|x)$ by Bayes's rule. Last, the algorithm can classify images in the testing data-set by comparing two posteriors.

## 2 Method

First, I need to model the likelihood $Pr(x|y)$, where $x$ represents the set of images matrix: $x = [x_1, x_2, x_3, ..., x_I] \quad s.t. \quad x_i \in R^{D\times 1}$, and $y$ represents the pixel is a face or background: $y = \{1, 0\}$ where 1 represents face, 0 represents background.

Choose $Pr(x)$ to be in the form of multivariate normal distribution:

$$Pr(x) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} exp[-0.5(x-\mu)^T\Sigma^{-1}(x-\mu)]$$
$$Pr(x|y) = Norm_x[\mu_y, \Sigma_y]$$

The multivariate normal distribution takes takes two parameters:
1. a vector containing mean position, $\mu$.
2. a symmetric "positve definite" covariance matrix $\Sigma$.

Calculate the parameter $\theta = (\mu, \Sigma)$ by applying maximum likelihood estimation on the logarithm of $\prod_{i=1}^{I} Pr(x_i|\theta)$:

$$Pr(x_i) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} exp[-0.5(x-\mu)^T\Sigma^{-1}(x-\mu)] \qquad s.t. \quad x_i \in R^{D\times 1}$$

$$\hat{\theta}_{ML} = \arg\max_{\theta} \prod_{i=1}^{I} Pr(x_i|\theta) \qquad s.t. \quad \theta = \{\mu, \Sigma\}$$

$$= \arg\max_{\theta} \prod_{i=1}^{I} \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \quad ^{-0.5(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

$$= \arg\max_{\theta} \sum_{i=1}^{I} [-lg(2\pi)^{\frac{D}{2}} - lg|\Sigma|^{\frac{1}{2}} - \frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}]$$

$$L = \sum_{i=1}^{I} [-lg(2\pi)^{\frac{D}{2}} - lg|\Sigma|^{\frac{1}{2}} - \frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}]$$

Finding $\mu$ by taking the partial derivative of $L$ over $\mu$ and set it to zero:
$\frac{\partial L}{\partial \mu} = \Sigma^{-1}\sum_{i=1}^{I}(x_i - \mu) = 0 => \mu = \frac{\sum_{i=1}^{I} x_i}{I}$

Finding $\Sigma$ by taking the partial derivative of $L$ over $\Sigma$ and set it to zero:
$\frac{\partial L}{\partial \Sigma^{-1}} = \sum_{i=1}^{I} \frac{1}{2}\Sigma = \sum_{i=1}^{I} \frac{1}{2}(x_i - \mu)(x_i - \mu)^T = 0 => \Sigma = \frac{\sum_{i=1}^{I}(x_i - \hat{\mu})(x_i - \hat{\mu})^T}{I}$

In training phase, my algorithm, which is coded in Python, calculate the $\mu_{face}$, $\Sigma_{face}$, $\mu_{background}$, $\Sigma_{background}$.
For $\mu_{face}$, the algorithm initialized a $D \times 1$ array for $\mu_{face}$. In this case, $D = 900$ because the size of each image is $20 \times 15$ and each pixel has 3 values for RGB ($20 \times 15 \times 3 = 900$). The first 300 cells of the array will contain the red values for each image, the second 300 cells will contain the green values, and the last 300 cells will contain the blue values. Then, the algorithm loop over all the face images in the training face folder and also counting the number of images. For each pixel of each image, the algorithm sum of the RGB values of each image to the corresponding cell in the $\mu_{face}$ array. For example, the first cells will contain the sum of red values of the first pixel of all the training face images. After that, the $\mu_{face}$ contains values of $\sum_{i=1}^{I} x_i$. So it loops over the $\mu_{face}$ array and divides the value in each cell of $\mu_{face}$ array by $I$ the number of images in the training face folder.
For $\Sigma_{face}$, assume each pixel in an image is independent, which means the covariance matrix will be a diagonal matrix because the covariance matrix may be singular causing the determinant to be zero and a divide by zero problem in $Norm_x[\mu_y, \Sigma_y]$. The algorithm initialized a $900 \times 900$ ($D \times D$) array for $\Sigma_{face}$. Then, the algorithm loop over all the face images in the training face folder and also counting the number of the image in the training face folder. For each pixel of each image, the algorithm store $(x_i - \hat{\mu})$ the difference of the RGB values

of the current image and the RGB values of the $\mu_{face}$ array that it calculated earlier in a temporary $900 \times 1$ array. Then, the algorithm loop the $(x_i - \hat{\mu})$ array and square each element and add to the corresponding diagonal position in the $\Sigma_{face}$ array. For example, the first element in $(x_i - \hat{\mu})$ array will be add to position $\sigma_{11}$. After that, the $\Sigma_{face}$ array contains values of variance of the same pixels among all the face images. Then, the algorithm loop over the $\Sigma_{face}$ and divides the value on the diagonal by $I$ the number of images in the training face folder.
For $\mu_{background}$ and $\Sigma_{background}$, the algorithm did the same thing as $\mu_{face}$, $\Sigma_{face}$, but using the image in the training background folder.

For the testing phase, assume the prior $P(y)$ is uniform. According to the Bayes' rule:

$$Pr(y|x^*) = \frac{Pr(x^*|y)Pr(y)}{Pr(x^*)}$$

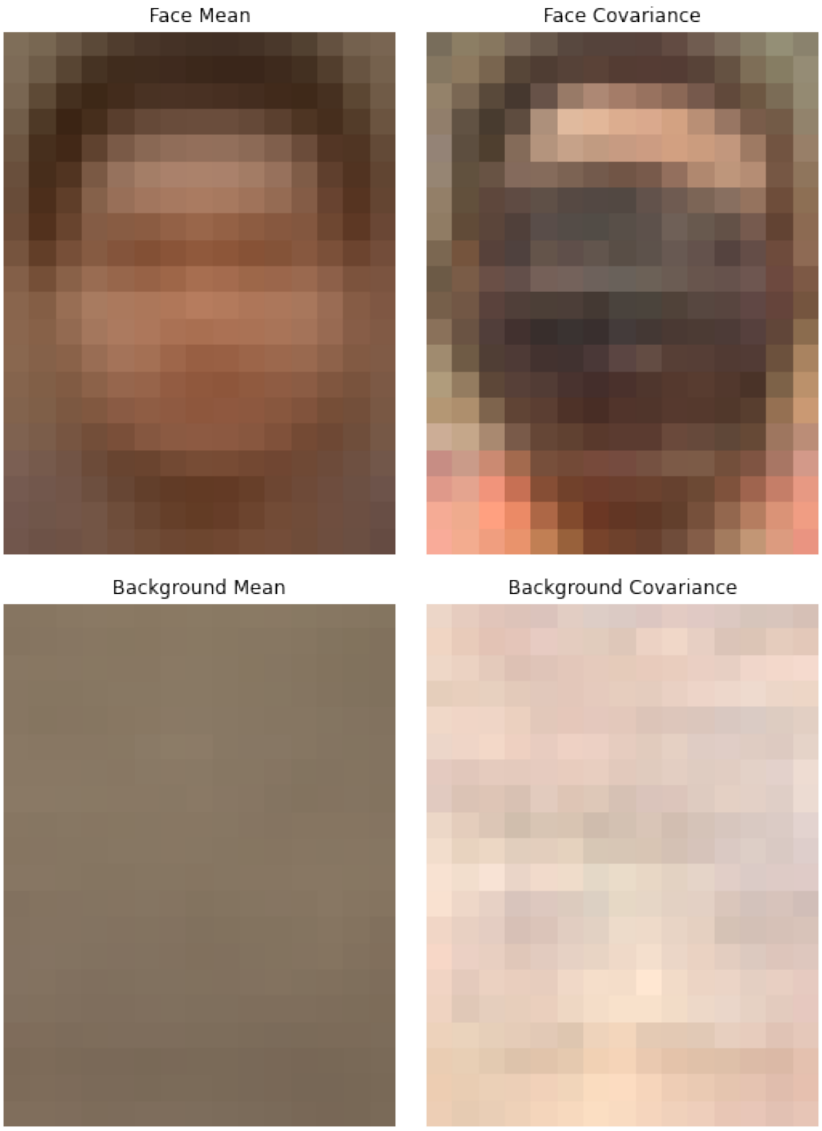
Since $P(y)$ and $Pr(x^*)$ are the same for $y = 1$ and $y = 0$, comparing the posterior $Pr(y = 1|x^*)$ and $Pr(y = 0|x^*)$ will yield the same result as comparing the likelihood $Pr(x^*|y = 1)$ and $Pr(x^*|y = 0)$ because the prior $P(y)$ and the evidence $Pr(x^*)$ would cancelled each other out. Therefore, the algorithm compares the likelihoods that calculated in training phase. However, straight comparing the likelihood also have difficulties because the $D = 900$ is a large number. For example, $(2\pi)^{D/2} -> \infty => \frac{1}{(2\pi)^{D/2}} -> 0$, and if $(\sigma_{dd})^2 < 1$, $\frac{1}{(\prod_{d=1}^{D} \sigma_{dd})^{1/2}} -> \infty$, and if $(\sigma_{dd})^2 > 1$, $\frac{1}{(\prod_{d=1}^{D} \sigma_{dd})^{1/2}} -> 0$. It need to be optimize more. First, like-terms can be cancelled, e.g. $\frac{1}{(2\pi)^{D/2}}$. Second, take the logarithm so the product becomes summation. Therefore, the algorithm actually compares

$$-\frac{1}{2}\sum_{d=1}^{D} lg((\sigma_{dd}^2)_{y=1}) - \sum_{d=1}^{D} \frac{((x_d - \mu_d)_{y=1})^2}{2(\sigma_{dd}^2)_{y=1}} \quad \text{and} \quad -\frac{1}{2}\sum_{d=1}^{D} lg((\sigma_{dd}^2)_{y=0}) - \sum_{d=1}^{D} \frac{((x_d - \mu_d)_{y=0})^2}{2(\sigma_{dd}^2)_{y=0}}$$

The algorithm loop all the images in the testing face folder and the testing background folder. For each image, it compare the logarithm of the likelihood $Pr(x^*|y = 1)$ and $Pr(x^*|y = 0)$. If $Pr(x^*|y = 1)$ is greater than $Pr(x^*|y = 0)$, the image is classified as a face image. Else if $Pr(x^*|y = 0)$ is greater than $Pr(x^*|y = 1)$, the image is classified as a background image. The algorithm records the number of correct face and background classification for testing face folder and the testing background folder. Then it calculate the classification accuracy for face and background separately by $\frac{\#\text{correctly classified face images}}{\text{total \# of face images}}$ and $\frac{\#\text{correctly classified background images}}{\text{total \# of background images}}$.

# 3 Experiment

I ran my algorithm and calculated the $\mu_{face}$, $\Sigma_{face}$, $\mu_{background}$, $\Sigma_{background}$ in training phase. Following images are the visualizations of these four parameters:




Although these images are blurred, we are still able to see the human face in the Face Mean Image and the Face Covariance Image. As we can observe, the center part of the Face Co-variance Image is much darker than the surrounding, which means the center part has relatively low covariance compared to the outer part of the image. This is reasonable because parts of the human face, such as chin, cheek, and nose, have very similar RGB values, while the outer part, such as background, have various RGB values.
On the other hand, the colors in the Background Mean Image are more uni-

form, and the colors in the Background Covariance Image are much lighter everywhere. The reason for this is that there are various backgrounds and they have different colors. There is no specific pattern for the background, so the mean seems uniform and the covariance is high everywhere.

During the testing phase, my algorithm classified the face and background with a decent accuracy. Here are my results:

```
Number of Correctly Classified Faces Images: 191
Total Number of Face Images: 232
Accuracy of Face Classification: 0.8232758620689655

Number of Correctly Classified Background Images: 421
Total Number of Background Images: 564
Accuracy of Background Classification: 0.7464539007092199
```

For the face test, there are a total of 232 face images in the test folder, and my algorithm classified 191 of them correctly. The accuracy of face classification is approximately 82.33 %.
For the background test, there are a total of 564 background images in the test folder, and my algorithm classified 421 of them correctly. The accuracy of face classification is approximately 74.65 %.
The accuracy of the face and background classification meets my expectation. With RGB values as the only variable, the results are fairly good. The algorithm did a great job on faces and non-face background classification.

# 4 Discussion

One problem I encounter is that when comparing the likelihoods $Norm_{x^*}[\mu_{y=1}, \Sigma_{y=1}]$ and $Norm_{x^*}[\mu_{y=0}, \Sigma_{y=0}]$, the product $\prod$ makes it difficult to calculate the actual values because the actual value either approaches 0 or $\infty$. One way to resolve this problem is to compare the logarithm of the likelihoods which would convert the product $\prod$ into a summation $\sum$. We could also cancel out the like-terms to simply the expression.

Another problem I encounter is that when I try to visualize the covariance of faces and backgrounds, most of the values of the covariance exceeded 1000 which is much greater than the max RGB value 255. This caused the output visualization image with random colors. I had to find the max value in the matrix, and convert all the values of the matrix into a scale of 255. Then the outcome seems reasonable.

One thing I found interesting is that when we modeling the problem, the theory and the math equations seem perfect and will solve the problem. However, when we actually apply them in the real world, many problems occur, and we had to make some assumptions and modifications to bypass the problem. For example, if we don't assume the pixels to be independent, the covariance may become a singular matrix with zero determinants. The divide by zero error will occur.

My algorithm can be improved by introducing more variables, such as gradient, magnitude, orientation, and other color spaces. With more input variables being introduced, the run time of my algorithm will be longer, but the accuracy of classification may reach close to 100%.