Batch - T7
Assignment No. - 1
Title – A univariate classifier from first principles
Student Name - Sharaneshwar Bharat Punjal
Student PRN - 23520011

a. Generate distributions (gaussian to start with) for male and female heights (1000 samples each). Fix the mean of female heights to 152 cm and male mean height to 166 cm. label the appropriate gender for samples in each of the distribution on (M or F)

b. Fix the sd of both the distributions to 5

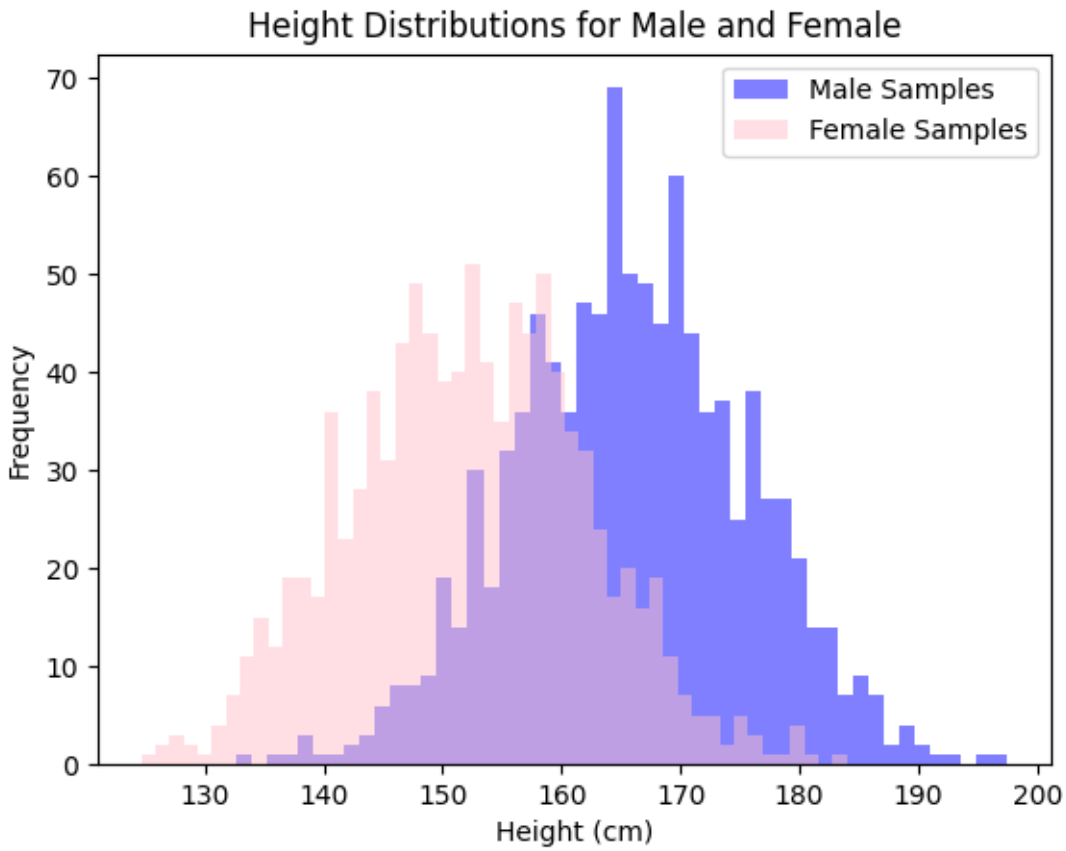
c. Try classification of gender using following approaches with aim to minimise misclassification
i. Assign gender based on likelihood calculated from distributions (empirically estimated mean and sd and calculate probability assuming gaussian distributions)
ii. Derive a threshold height to separate male female
iii. Quantize the data at scale of 0.5 cm and empirically estimate the likelihood of male female in each segment based on majority
iv. In each of the above cases output a confusion matrix for classification

d. Try following values of sd (eg. 2.5, 7.5 and 10) repeat 3.a, 3.b, 3.c, 3,d observe impact of change in sd on classification accuracy

e. Change the quantization interval length (say 0.001, 0.05, 0.1, 0.3, 1, 2, 5,10 cm etc) repeat 3.a, 3.b, 3.c, 3,d observe impact of change in sd on classification accuracy

Observations:

a.



The male and female distributions overlap slightly due to the same SD, causing some inherent misclassification.

Verify that the distributions align with the specified means and SDs.

b. Steps –

1. Calculate likelihoods for each height based on Gaussian probability density functions (PDFs).
2. Assign gender labels by comparing probabilities:
3. A sample is male if P(Male | height) > P(Female | height), otherwise female.
4. Use statistics.NormalDist or scipy.stats.norm for likelihood calculations.
5. Compute classification accuracy and confusion matrix.

c. Steps –

1. Test thresholds across the range of heights of 135 to 180 cm.
2. Compute misclassification error for each threshold and choose the one with the lowest error.
3. Output the confusion matrix for the optimal threshold.

The optimal threshold will depend on the overlap between distributions. A threshold closer to 159 (midpoint of male and female means) minimizes error for the given SD.

d. Steps –

1. Divide height range into intervals (e.g., 0.5 cm bins).
2. Count male and female samples in each bin.
3. Assign gender for each bin based on the majority.
4. Classify samples based on bin assignment and compute confusion matrix.

Smaller bin sizes (e.g., 0.001) yield more precise classifications but increase computational cost. Larger bins reduce resolution, leading to higher error.

e. Steps -

1. Repeat the above steps for different SDs (2.5, 7.5, 10) and quantization intervals (0.001, 0.05, ..., 10).
2. Observe the impact on classification accuracy and confusion matrices.

- **Varying SD**:
  - Higher SD increases overlap, worsening accuracy.
  - Lower SD sharpens distributions, improving separability.
- **Varying Quantization**:
  - Very fine intervals (e.g., 0.001) result in better accuracy but higher computation.
  - Coarse intervals (e.g., 10) lose detail, increasing errors.

Code:

// main.py

```
import matplotlib.pyplot as plt
from data_generation import generateSamples
from threshold_classification import findOptimalThreshold
from likelihood_classification import classifyByLikelihood
from quantization_classification import classifyByQuantization

samplesCount = 1000
femaleMean = 152
maleMean = 166
sdInitial = 5
sdList = [2.5, 5, 7.5, 10]
quantIntervals = [0.001, 0.05, 0.1, 0.3, 1, 2, 5, 10]

for sd in sdList:
    maleSamples = generateSamples(maleMean, sd, samplesCount)
    femaleSamples = generateSamples(femaleMean, sd, samplesCount)
```

```
    print(f"\n-----------------------------------\nSD = {sd}\n-------------------
---------------\n")

    # Threshold Classification
    bestThreshold, thresholdError, thresholdCM = findOptimalThreshold(maleSamples,
femaleSamples, samplesCount)
    print(f"Optimal Threshold: {bestThreshold}\nThreshold Error:
{thresholdError:.2f}%\nConfusion Matrix: {thresholdCM}\n")

    # Likelihood Classification
    likelihood_error, likelihood_cm = classifyByLikelihood(maleSamples,
femaleSamples, samplesCount)
    print(f"Likelihood Classification Error: {likelihood_error:.2f}%\nConfusion
Matrix: {likelihood_cm}\n")

    # Quantization Classification
    for interval in quantIntervals:
        quant_error, quant_cm = classifyByQuantization(maleSamples, femaleSamples,
interval, samplesCount)
        print(f"Quantization Interval = {interval}, Quantization Error:
{quant_error:.2f}%, Confusion Matrix: {quant_cm}")

plt.hist(maleSamples, bins=50, alpha=0.5, label='Male Samples', color='blue')
plt.hist(femaleSamples, bins=50, alpha=0.5, label='Female Samples', color='pink')
plt.xlabel('Height (cm)')
plt.ylabel('Frequency')
plt.legend()
plt.title('Height Distributions for Male and Female')
plt.show()
```

// data_generation.py

```
import numpy as np

def generateSamples(mean, sd, size):
    return np.random.normal(loc=mean, scale=sd, size=size)
```

// likelihood_classification.py

```
import statistics

def classifyByLikelihood(maleSamples, femaleSamples, samplesCount):
    maleMeanActual = statistics.mean(maleSamples)
    femaleMeanActual = statistics.mean(femaleSamples)
    maleSDActual = statistics.stdev(maleSamples)
    femaleSDActual = statistics.stdev(femaleSamples)
```

```
    maleDist = statistics.NormalDist(mu=maleMeanActual, sigma=maleSDActual)
    femaleDist = statistics.NormalDist(mu=femaleMeanActual, sigma=femaleSDActual)

    tp = 0
    fn = 0
    fp = 0
    tn = 0

    for height in maleSamples:
        probMale = maleDist.pdf(height)
        probFemale = femaleDist.pdf(height)
        if probFemale > probMale:
            fn += 1
        else:
            tp += 1

    for height in femaleSamples:
        probMale = maleDist.pdf(height)
        probFemale = femaleDist.pdf(height)
        if probMale > probFemale:
            fp += 1
        else:
            tn += 1

    error = ((fn + fp) * 100) / (samplesCount * 2)
    confusion_matrix = [[tp, fn], [fp, tn]]
    return error, confusion_matrix


// threshold_classification.py
def findOptimalThreshold(maleSamples, femaleSamples, samplesCount):
    minError = 100
    bestThreshold = 0
    bestConfusionMatrix = None

    for threshold in range(135, 180):
        tp = fn = fp = tn = 0
        for height in maleSamples:
            if height >= threshold:
                tp += 1
            else:
                fn += 1

        for height in femaleSamples:
            if height >= threshold:
                fp += 1
            else:
                tn += 1
```

```
        mismatchCount = fn + fp
        errorPer = (mismatchCount * 100) / (samplesCount * 2)

        if errorPer < minError:
            minError = errorPer
            bestThreshold = threshold
            bestConfusionMatrix = [[tp, fn], [fp, tn]]

    return bestThreshold, minError, bestConfusionMatrix


// quantization_classification.py
import numpy as np

def classifyByQuantization(maleSamples, femaleSamples, interval_length,
sample_count):
    bins = np.arange(min(femaleSamples), max(maleSamples), interval_length)
    maleHist, _ = np.histogram(maleSamples, bins)
    femaleHist, _ = np.histogram(femaleSamples, bins)

    tp = fn = fp = tn = 0

    for height in maleSamples:
        binIndex = np.digitize(height, bins) - 1
        if binIndex < len(femaleHist) and femaleHist[binIndex] >
maleHist[binIndex]:
            fn += 1
        else:
            tp += 1

    for height in femaleSamples:
        binIndex = np.digitize(height, bins) - 1
        if binIndex < len(maleHist) and maleHist[binIndex] > femaleHist[binIndex]:
            fp += 1
        else:
            tn += 1

    errorRate = ((fn + fp) * 100) / (2 * sample_count)
    confusionMatrix = [[tp, fn], [fp, tn]]
    return errorRate, confusionMatrix
```