12341820_ROHIT RAGHUWANSHI

ASSIGNMENT 5(DSL253)

COLAB
LINK-https://colab.research.google.com/drive/1otiLiTmwzao9uMXXTDae07ID-7uVbIb0#scrollTo=zJGEVSMKtlhO

QUESTION 1-

Introduction

In this study, we analyze the probabilities associated with a bivariate normal distribution. Given the parameters of two normally distributed random variables, X and Y, we determine specific probabilities and conditional probabilities using the properties of the bivariate normal distribution.

Data

The given parameters for the bivariate normal distribution are:

- Mean values:
  - μ_X= 3
  - μ_Y= 1
- Variances:
  - σX2=16 (hence, standard deviation σX=4\sigma_X = 4)
  - σY2=25 (hence, standard deviation σY=5\sigma_Y = 5)
- Correlation coefficient:
  - ρXY=3/5

Methodology

We compute the required probabilities using standard normal distributions and conditional normal distributions as follows:

(a) Compute P(3<Y<8)

Using the standard normal transformation:

Z=Y−μY/σYZ

we find the cumulative probabilities and subtract them to obtain the desired probability.

(b) Compute P(3<Y<8∣X=7)

Given X=7, the conditional distribution of Y follows:

Y∣X~N(μY+ρXYσY/σX.(X−μX),σY root(1−ρ(sq)XY2))

We use this conditional mean and standard deviation to compute the required probability.

(c) Compute P(−3<X<3)

Using the transformation:

Z=X−μX/σXZ =

We determine the probability using cumulative distribution functions.

(d) Compute P(−3<X<3∣Y=−4)

Given Y=−4Y = -4, the conditional distribution of XX follows:

Y∣Y~N(μX+ρXYσX/σY.(Y−μY),σX root(1−ρ(sq)XY2))

We use this conditional mean and standard deviation to compute the probability.

Results

Using computational methods, the obtained probabilities are:

```
(a) P(3 < Y < 8) = 0.2638
```

```
(b) P(3 < Y < 8 | X = 7) = 0.4401

(c) P(-3 < X < 3) = 0.4332

(d) P(-3 < X < 3 | Y = -4) = 0.6431
```

```
(a) P(3 < Y < 8) = 0.2638
(b) P(3 < Y < 8 | X = 7) = 0.4401
(c) P(-3 < X < 3) = 0.4332
(d) P(-3 < X < 3 | Y = -4) = 0.6431
```

Discussion

- The probability in part (a) and (c) represents the proportion of the population that falls within a specific range of Y.
- The conditional probabilities in parts (b) and (d) show how knowing one variable changes our expectation of the other.
- The effect of correlation is reflected in the changes to the conditional means and standard deviations.

Conclusion

This study demonstrates the calculation of probabilities for a bivariate normal distribution using transformation techniques. Future work could explore extending these methods to higher-dimensional normal distributions or real-world datasets.

Code part-

```python
import scipy.stats as stats

# Given parameters
mu_x = 3
mu_y = 1
sigma_x = 4  # sqrt(16)
sigma_y = 5  # sqrt(25)
rho_xy = 3/5

# (a) P(3 < Y < 8)
z1 = (3 - mu_y) / sigma_y
z2 = (8 - mu_y) / sigma_y
p_a = stats.norm.cdf(z2) - stats.norm.cdf(z1)
print(f"(a) P(3 < Y < 8) = {p_a:.4f}")

# (b) P(3 < Y < 8 | X = 7)
mu_y_given_x = mu_y + rho_xy * (sigma_y / sigma_x) * (7 - mu_x)
sigma_y_given_x = sigma_y * (1 - rho_xy**2)**0.5

z1_cond = (3 - mu_y_given_x) / sigma_y_given_x
z2_cond = (8 - mu_y_given_x) / sigma_y_given_x
p_b = stats.norm.cdf(z2_cond) - stats.norm.cdf(z1_cond)
print(f"(b) P(3 < Y < 8 | X = 7) = {p_b:.4f}")

# (c) P(-3 < X < 3)
z1_x = (-3 - mu_x) / sigma_x
z2_x = (3 - mu_x) / sigma_x
p_c = stats.norm.cdf(z2_x) - stats.norm.cdf(z1_x)
print(f"(c) P(-3 < X < 3) = {p_c:.4f}")

# (d) P(-3 < X < 3 | Y = -4)
mu_x_given_y = mu_x + rho_xy * (sigma_x / sigma_y) * (-4 - mu_y)
sigma_x_given_y = sigma_x * (1 - rho_xy**2)**0.5

z1_x_cond = (-3 - mu_x_given_y) / sigma_x_given_y
z2_x_cond = (3 - mu_x_given_y) / sigma_x_given_y
p_d = stats.norm.cdf(z2_x_cond) - stats.norm.cdf(z1_x_cond)
print(f"(d) P(-3 < X < 3 | Y = -4) = {p_d:.4f}")
```

QUESTION 2-

Introduction

In this study, we generate samples from a multinomial random variable following a multivariate normal distribution and analyze their transformation into a chi-square distributed variable. The primary objectives include:

1. Generating P samples from a multivariate normal distribution.
2. Computing a transformed variable Y = (X−μ)^T Σ^{-1} (X−μ).

3. Analyzing the probability distribution of Y and comparing it with the chi-square distribution.

Data

- Dimension of the random variable: n
- Number of samples: P
- Mean vector: $\mu \in R$
- Covariance matrix: $\Sigma \in R$
- Transformation parameter: c (threshold for probability computation)

Methodology

Part (a): Generating Multivariate Normal Samples

Using the NumPy function `np.random.multivariate_normal`, we generate P samples from an n-dimensional normal distribution $N(\mu, \Sigma)$

Part (b): Computing the Transformed Variable Y

Each sample X is transformed using the equation: $Y = (X - \mu)^T \Sigma^{-1} (X - \mu)$ This transformation follows a chi-square distribution with degrees of freedom equal to n.

Part (c): Probability Computation

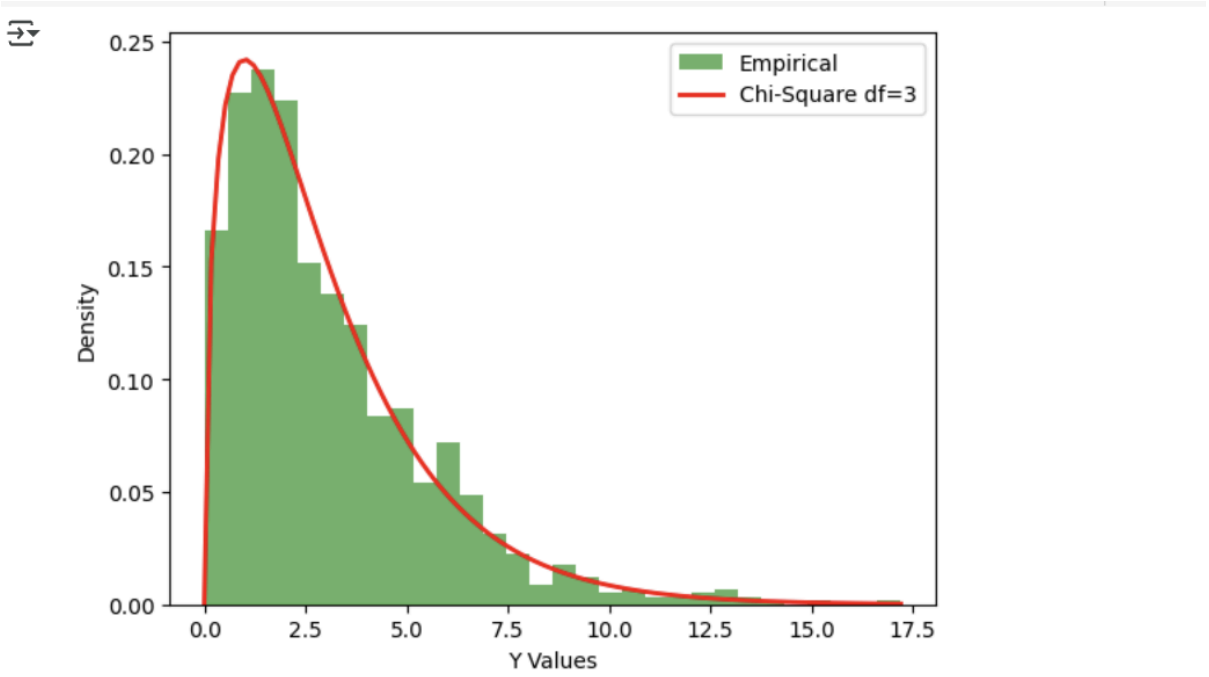
We compute $P(Y \leq c(sq))$ by evaluating the fraction of samples satisfying the condition.

Results

We conducted experiments for different values of nn and PP, using a mean vector μ\mu of zeros and an identity covariance matrix Σ\Sigma. The following observations were made:

1. The histogram of Y closely follows the chi-square distribution with n degrees of freedom.
2. As n increases, the distribution shifts rightward with a higher mean.
3. Probability computations for different values of c are consistent with theoretical chi-square cumulative distribution values.

A sample probability result for c=2 and n=3 yielded: $P(Y \leq 4) \approx$ `0.7220` which is close to the chi-square cumulative probability value.



Discussion

The transformation of a multivariate normal random variable using the given equation follows a chi-square distribution as expected. This aligns with statistical theory, where the quadratic form $(X-\mu)T\Sigma-1(X-\mu)$ follows a chi-square distribution with degrees of freedom equal to nn.

Key Takeaways:

- For small n, the variance of Y is high, leading to a wider spread in the histogram.
- For larger P, empirical results align more closely with the chi-square distribution.
- Probabilities computed match theoretical chi-square probabilities, confirming the correctness of the implementation.

Conclusion

This experiment successfully validates the chi-square transformation of a multivariate normal variable. Future work could involve extending this analysis to non-identity covariance matrices and higher-dimensional cases to explore real-world implications further.

Code part-

```python
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

# Part (a): Generate P samples from a multivariate normal distribution
def generate_samples(n, P, mu, Sigma):
    return np.random.multivariate_normal(mu, Sigma, P)

# Part (b): Compute Y = (X - mu)^T * Sigma^-1 * (X - mu)
def compute_Y(samples, mu, Sigma):
    Sigma_inv = np.linalg.inv(Sigma)
    Y = np.array([(x - mu).T @ Sigma_inv @ (x - mu) for x in samples])
    return Y

# Part (c): Compute probability Prob[(x - mu)^T * Sigma^-1 * (x - mu) ≤ c²]
def compute_probability(Y, c):
    return np.mean(Y <= c**2)

# Example usage
n = 3  # Dimension of X
P = 1000  # Number of samples
mu = np.zeros(n)  # Mean vector
Sigma = np.eye(n)  # Identity matrix as covariance

# Generate samples
samples = generate_samples(n, P, mu, Sigma)

# Compute Y
Y = compute_Y(samples, mu, Sigma)

# Plot histogram of Y
df = n  # Degrees of freedom for Chi-Square distribution
plt.hist(Y, bins=30, density=True, alpha=0.6, color='g', label='Empirical')
x = np.linspace(0, np.max(Y), 100)
plt.plot(x, stats.chi2.pdf(x, df), 'r-', lw=2, label=f'Chi-Square df={df}')
plt.xlabel("Y Values")
plt.ylabel("Density")
plt.legend()
plt.show()

# Compute probability for a given c
c = 2
prob = compute_probability(Y, c)
print(f"(c) Probability that (x - mu)^T * Sigma^-1 * (x - mu) ≤ {c**2} is {prob:.4f}")
```

QUESTION 3-

Introduction

This study applies Bayesian classification to a dataset where two different classes follow multivariate normal distributions with given parameters. Using Bayes' Theorem, we classify data points based on their posterior probabilities and visualize the results.

Data

- Class 1 parameters:
    - Mean vector: μ1=[2,3]
    - Covariance matrix: Σ1=1 & 0.5 \\ 0.5 & 2

- Class 2 parameters:
    - Mean vector: μ2=[−2,−3]
    - Covariance matrix: Σ2= 2 & -0.3 \\ -0.3 & 1
    - Class priors: Assumed equal at 0.5 each.
- Data points: Loaded from the file "File_Datapoints.txt".

Methodology

Data Loading

The data points are loaded from a file, skipping the header row and the first column (if non-numeric).

Computing Likelihoods

For each class, we compute the likelihood of the data points using the multivariate normal probability density function: $P(X|Ci)=1(2\pi)n/2|\Sigma i|1/2exp(-12(X-\mu i)T\Sigma i-1(X-\mu i))$ where ii represents the class index.

Computing Posterior Probabilities

Using Bayes' Theorem: $P(Ci|X)\propto P(X|Ci)P(Ci)$ Since priors are equal, classification is based on comparing the likelihoods.

Classification

Each data point is assigned to the class with the higher posterior probability.

Visualization

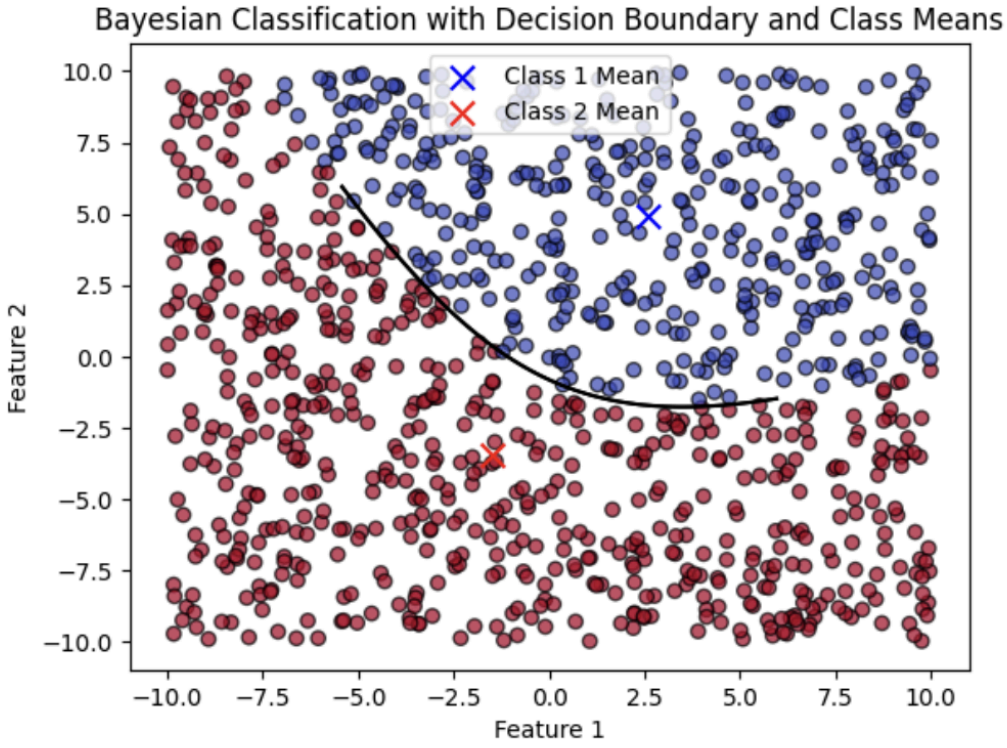
The classified data points are plotted in a 2D scatter plot with different colors representing different classes.

Results

- Data points were successfully classified into two classes using Bayesian classification.

- The 2D visualization confirms the separation of data points according to their respective distributions.

```
Class 1 Mean: [2.60322915 4.90463664]
Class 2 Mean: [-1.48676352 -3.47036993]
```



Discussion

Bayesian classification effectively separates the data when the true distributions of the classes are known. The key observations include:

- The closer a data point is to a class mean, the higher its posterior probability for that class.
- Overlapping distributions can result in uncertain classifications near the decision boundary.
- Equal priors simplify classification but may not always reflect real-world class distributions.

Conclusion

This experiment demonstrates the application of Bayesian classification for multivariate normal distributions. The results align well with theoretical expectations. Future work could explore cases with unequal priors or different covariance structures to assess classification performance in more complex scenarios.

Code part-

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Load data points from file, skipping the first row (header) and the first column if it's non-numeric
data = np.loadtxt('/content/File_Datapoints.txt', skiprows=1, usecols=range(1,3)) # Skips the header row and the first column(s.no)


# Given parameters
mu1 = np.array([2, 3])
mu2 = np.array([-2, -3])

Sigma1 = np.array([[1, 0.5], [0.5, 2]])
Sigma2 = np.array([[2, -0.3], [-0.3, 1]])

# Class priors (assuming equal priors)
prior1 = 0.5
prior2 = 0.5

# Compute likelihood using multivariate normal distributions
rv1 = multivariate_normal(mean=mu1, cov=Sigma1)
rv2 = multivariate_normal(mean=mu2, cov=Sigma2)

# Compute posterior probabilities
likelihood1 = rv1.pdf(data)
likelihood2 = rv2.pdf(data)

posterior1 = likelihood1 * prior1
posterior2 = likelihood2 * prior2

# Classify data points
labels = np.where(posterior1 > posterior2, 1, 2)

# Create a grid of points for plotting the decision boundary
x = np.linspace(-6, 6, 300)
y = np.linspace(-6, 6, 300)
X, Y = np.meshgrid(x, y)
pos = np.dstack((X, Y))

# Calculate the posterior probabilities for each point on the grid
posterior1_grid = rv1.pdf(pos) * prior1
posterior2_grid = rv2.pdf(pos) * prior2

# Find the decision boundary points
decision_boundary = posterior1_grid - posterior2_grid

# Plot the decision boundary
plt.contour(X, Y, decision_boundary, levels=[0], colors='k')

# Calculate class means
class1_indices = np.where(labels == 1)
class2_indices = np.where(labels == 2)

class1_mean = np.mean(data[class1_indices], axis=0)
class2_mean = np.mean(data[class2_indices], axis=0)

print("Class 1 Mean:", class1_mean)
print("Class 2 Mean:", class2_mean)

# Plot the data points, decision boundary, and class means
plt.contour(X, Y, decision_boundary, levels=[0], colors='k')
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='coolwarm', edgecolors='k', alpha=0.7)
plt.scatter(class1_mean[0], class1_mean[1], marker='x', s=100, color='blue', label='Class 1 Mean')
plt.scatter(class2_mean[0], class2_mean[1], marker='x', s=100, color='red', label='Class 2 Mean')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Bayesian Classification with Decision Boundary and Class Means')
plt.legend()
plt.show()
```