Multivariate Analysis

Assignment 6

1) The probability density functions for G_1 & G_2 are given by $f(y|G_1)$, $f(y|G_2)$ where G_1, G_2 are multivariate normals of dimension $= p$

$$\cdot f(y|G_1) = \frac{1}{(\sqrt{2\pi})^p |\Sigma|^{1/2}} \exp\left\{-\frac{(y-\mu_1)^T \Sigma^{-1} (y-\mu_1)}{2}\right\}$$

$$\cdot f(y|G_2) = \frac{1}{(\sqrt{2\pi})^p |\Sigma|^{1/2}} \exp\left\{-\frac{(y-\mu_2)^T \Sigma^{-1} (y-\mu_2)}{2}\right\}$$

Now, $$\frac{f(y|G_1)}{f(y|G_2)} = \exp\left\{\frac{(y-\mu_2)^T\Sigma^{-1}(y-\mu_2) - (y-\mu_1)^T \Sigma^{-1}(y-\mu_1)}{2}\right\}$$

$$= \exp\left\{[(\mu_1-\mu_2)^T\Sigma^{-1}y + y^T\Sigma^{-1}(\mu_1-\mu_2) + \mu_2^T\Sigma^{-1}\mu_2 - \mu_1^T\Sigma^{-1}\mu_1]/2\right\}$$

We know that $(\mu_1-\mu_2)^T\Sigma^{-1}y$ is a scalar quantity.

$$\therefore \{(\mu_1-\mu_2)^T\Sigma^{-1}y\}^T = (\mu_1-\mu_2)^T\Sigma^{-1}y$$

$$\Rightarrow (\mu_1-\mu_2)\Sigma^{-1}y = y^T\Sigma^{-1}(\mu_1-\mu_2) \qquad \{\text{As } \Sigma^{-1}, \Sigma \text{ are symmetric}\}$$

$$\therefore \frac{f(y|G_1)}{f(y|G_2)} = \exp\left\{(\mu_1-\mu_2)^T\Sigma^{-1}y + \frac{(\mu_2^T\Sigma^{-1}\mu_2 - \mu_1^T\Sigma^{-1}\mu_1)}{2}\right\}$$

Now, $$\mu_2^T\Sigma^{-1}\mu_2 - \mu_1^T\Sigma^{-1}\mu_1 = \mu_2^T\Sigma^{-1}\mu_2 - \mu_2^T\Sigma^{-1}\mu_1 + \mu_2^T\Sigma^{-1}\mu_1 - \mu_1^T\Sigma^{-1}\mu_1$$

$$= \mu_2^T\Sigma^{-1}(\mu_1+\mu_2) - (\mu_1+\mu_2)^T\Sigma^{-1}\mu_1$$

[but $(\mu_1+\mu_2)^T\Sigma^{-1}\mu_1$ is a scalar $\rightarrow (\mu_1+\mu_2)^T\Sigma^{-1}\mu_1 = \mu_1^T\Sigma^{-1}(\mu_1+\mu_2)$]

$$= \mu_2^T\Sigma^{-1}(\mu_1+\mu_2) - \mu_1^T\Sigma^{-1}(\mu_1+\mu_2)$$

$$= (\mu_2-\mu_1)^T\Sigma^{-1}(\mu_1+\mu_2)$$

$$\therefore \frac{f(y|G_1)}{f(y|G_2)} = \exp\left\{(\mu_1-\mu_2)^T\Sigma^{-1}y + (\mu_2-\mu_1)^T\Sigma^{-1}(\mu_1+\mu_2)/2\right\}$$

$$= \exp\left\{(\mu_1-\mu_2)^T\Sigma^{-1}y - \frac{(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1+\mu_2)}{2}\right\}$$

[Proved]

2) We classify y as G_2 iff:

$$(\mu_1-\mu_2)^T \Sigma^{-1} y \leq \frac{1}{2}(\mu_1-\mu_2)^T \Sigma^{-1}(\mu_1+\mu_2) + \ln\left(\frac{P_2}{P_1}\right)$$

Also, if $y \in G_1$, then

$$a^T y \sim N(a^T \mu_1, a^T \Sigma a)$$

if $a = \{(\mu_1-\mu_2)^T \Sigma^{-1}\}^T \rightarrow a^T \Sigma a = (\mu_1-\mu_2)^T \Sigma^{-1} \Sigma \Sigma^{-1}(\mu_1-\mu_2)$

$$= (\mu_1-\mu_2)^T \Sigma^{-1}(\mu_1-\mu_2) = \Delta^2$$

Now, $a^T y \sim N(a^T \mu_1, \Delta^2)$ with $a = \Sigma^{-1}(\mu_1-\mu_2)$

$$P[\text{classify as } G_2 \mid G_1] = P\left[(\mu_1-\mu_2)^T \Sigma^{-1} y \leq \frac{1}{2}(\mu_1-\mu_2)^T \Sigma^{-1}(\mu_1+\mu_2) + \ln(P_2/P_1)\right]$$

$$= P\left[\frac{(\mu_1-\mu_2)^T \Sigma^{-1} y - (\mu_1-\mu_2)^T \Sigma^{-1}\mu_1}{\Delta} \leq \frac{\frac{(\mu_1-\mu_2)^T \Sigma^{-1}(\mu_1+\mu_2)}{2} + \ln(P_2/P_1) - (\mu_1-\mu_2)^T \Sigma^{-1}\mu_1}{\Delta}\right]$$

Now, $\dfrac{(\mu_1-\mu_2)^T \Sigma^{-1} y - (\mu_1-\mu_2)^T \Sigma^{-1}\mu_1}{\Delta} = \omega \sim N(0, I)$

$\therefore P[\text{classify as } G_2 \mid G_1]$

$$= P\left[w \leq \frac{-\frac{(\mu_1-\mu_2)^T \Sigma^{-1}(\mu_1-\mu_2)}{2} + \ln(P_2/P_1)}{\Delta}\right]$$

$$= P\left[\omega \leq \frac{-\frac{\Delta^2}{2} + \ln(P_2/P_1)}{\Delta}\right]$$

$$= \phi\left[\frac{-\frac{1}{2}\Delta^2 + \ln(P_2/P_1)}{\Delta}\right]$$

{Proved}

Problems 3 and 4

April 4, 2021

```
In [1]: # Problems on Classification Analysis
```

```
In [2]: import numpy as np
        import scipy as sc
        from scipy.stats import f
```

1 *Problem* 3

```
In [3]: set_A = [[189, 245, 137, 163],
                [192, 260, 132, 217],
                [217, 276, 141, 192],
                [221, 299, 142, 213],
                [171, 239, 128, 158],
                [192, 262, 147, 173],
                [213, 278, 136, 201],
                [192, 255, 128, 185],
                [170, 244, 128, 192],
                [201, 276, 146, 186],
                [195, 242, 128, 192],
                [205, 263, 147, 192],
                [180, 252, 121, 167],
                [192, 283, 138, 183],
                [200, 287, 136, 173],
                [192, 277, 150, 177],
                [200, 287, 136, 173],
                [181, 255, 146, 183],
                [192, 287, 141, 198]]

        set_B = [[181, 305, 184, 209],
                [158, 237, 133, 188],
                [184, 300, 166, 231],
                [171, 273, 162, 213],
                [181, 297, 163, 224],
                [181, 308, 160, 223],
                [177, 301, 166, 221],
                [198, 308, 141, 197],
                [180, 286, 146, 214],
```

```python
                [177, 299, 171, 192],
                [176, 317, 166, 213],
                [192, 312, 166, 209],
                [176, 285, 141, 200],
                [169, 287, 162, 214],
                [164, 265, 147, 192],
                [181, 308, 157, 204],
                [192, 276, 154, 209],
                [181, 278, 149, 235],
                [175, 271, 140, 192],
                [197, 303, 170, 205]]

        n_A = len(set_A)
        n_B = len(set_B)
        set_A = np.asmatrix(set_A)
        set_B = np.asmatrix(set_B)

        mean_A = np.mean(set_A, axis = 0)
        mean_B = np.mean(set_B, axis = 0)

        cov_A = np.cov(set_A.T)
        cov_B = np.cov(set_B.T)

        cov_pooled = ((n_A - 1) * cov_A + (n_B - 1) * cov_B) / (n_A + n_B - 2)

        coefficient = np.matmul((mean_A - mean_B), np.linalg.inv(cov_pooled))
        coefficient = np.ravel(coefficient)

        print('Multiplicative Factor of the Classification Function: ')
        print(str(np.ndarray.tolist(np.round(coefficient, 4))))
        print()

        z_mean_A = np.ravel(np.dot(coefficient, mean_A.T))[0]
        z_mean_B = np.ravel(np.dot(coefficient, mean_B.T))[0]

        print('z_mean_A = ' + str(np.round(z_mean_A, 4)))
        print('z_mean_B = ' + str(np.round(z_mean_B, 4)))

        cutoff_point = (z_mean_A + z_mean_B) / 2

        print('Required Cutoff Point = ' +
              str(np.round(cutoff_point, 4)))
```

```
Multiplicative Factor of the Classification Function:
[0.3539, -0.1445, -0.0957, -0.1443]

z_mean_A = -9.5486
z_mean_B = -23.6979
```

```
Required Cutoff Point = -16.6232
```

1.1 (*a*)

$$Classification\ Function\ z = [0.3539, -0.1445, -0.0957, -0.1443].y$$
$$= [0.3422, -0.1451, -0.1021, -0.1253].[y_1, y_2, y_3, y_4]$$
$$Cutoff\ Point = -16.6232$$

In [4]:
```
set_A_true = 0
set_B_true = 0

for sample in set_A:
  score = np.dot(coefficient, np.asarray(sample).T)
  if score < cutoff_point:
    set_A_true += 1

for sample in set_B:
  score = np.dot(coefficient, np.asarray(sample).T)
  if score < cutoff_point:
    set_B_true += 1

print('Set_A')
print('# Observations = ' + str(n_A))
print('# Samples classified into set_A = ' + str(set_A_true))
print('# Samples classified into set_B = ' + str(n_A - set_A_true))
print()

print('Set_B')
print('# Observations = ' + str(n_B))
print('# Samples classified into set_A = ' + str(n_B - set_B_true))
print('# Samples classified into set_B = ' + str(set_B_true))
```

```
Set_A
# Observations = 19
# Samples classified into set_A = 0
# Samples classified into set_B = 19

Set_B
# Observations = 20
# Samples classified into set_A = 1
# Samples classified into set_B = 19
```

True Class	True Class Frequency	Predicted as 1	Predicted as 2
1	19	0	19
2	20	1	19

2 *Problem* 4

In [5]:
```
method_1 = [[5.4, 6.0, 6.3, 6.7],
            [5.2, 6.2, 6.0, 5.8],
            [6.1, 5.9, 6.0, 7.0],
            [4.8, 5.0, 4.9, 5.0],
            [5.0, 5.7, 5.0, 6.5],
            [5.7, 6.1, 6.0, 6.6],
            [6.0, 6.0, 5.8, 6.0],
            [4.0, 5.0, 4.0, 5.0],
            [5.7, 5.4, 4.9, 5.0],
            [5.6, 5.2, 5.4, 5.8],
            [5.8, 6.1, 5.2, 6.4],
            [5.3, 5.9, 5.8, 6.0]]

method_2 = [[5.0, 5.3, 5.3, 6.5],
            [4.8, 4.9, 4.2, 5.6],
            [3.9, 4.0, 4.4, 5.0],
            [4.0, 5.1, 4.8, 5.8],
            [5.6, 5.4, 5.1, 6.2],
            [6.0, 5.5, 5.7, 6.0],
            [5.2, 4.8, 5.4, 6.0],
            [5.3, 5.1, 5.8, 6.4],
            [5.9, 6.1, 5.7, 6.0],
            [6.1, 6.0, 6.1, 6.2],
            [6.2, 5.7, 5.9, 6.0],
            [5.1, 4.9, 5.3, 4.8]]

method_3 = [[4.8, 5.0, 6.5, 7.0],
            [5.4, 5.0, 6.0, 6.4],
            [4.9, 5.1, 5.9, 6.5],
            [5.7, 5.2, 6.4, 6.4],
            [4.2, 4.6, 5.3, 6.3],
            [6.0, 5.3, 5.8, 5.4],
            [5.1, 5.2, 6.2, 6.5],
            [4.8, 4.6, 5.7, 5.7],
            [5.3, 5.4, 6.8, 6.6],
            [4.6, 4.4, 5.7, 5.6],
            [4.5, 4.0, 5.0, 5.9],
            [4.4, 4.2, 5.6, 5.5]]

method_1 = np.asarray(method_1)
method_2 = np.asarray(method_2)
method_3 = np.asarray(method_3)
methods = [method_1, method_2, method_3]

means = []
covariances = []
```

```
    sizes = []

    for method in methods:
      means.append(np.mean(method, axis = 0))
      covariances.append(np.cov(method.T))
      sizes.append(np.shape(method)[0])

    S_pooled = np.zeros(np.shape(covariances[0]))
    total_size = 0

    for index, S in enumerate(covariances):
      S_pooled += S * (sizes[index] - 1)
      total_size += sizes[index]

    S_pooled = S_pooled / (total_size - len(sizes))
```

```
In [6]: # Computing the classification functions

        multipliers = []
        constants = []

        for index in range(len(methods)):
          multiplier = np.matmul(means[index], np.linalg.inv(S_pooled))
          constant = np.matmul(multiplier, means[index].T) / 2

          multipliers.append(multiplier)
          constants.append(constant)
```

2.1 *(a)*

```
In [7]: # Displaying the various Classification Scoring functions:

        for index in range(len(methods)):
          print('For Method ' + str(index + 1) + ': ')
          print(np.ndarray.tolist(np.round(multipliers[index], 4)), end = '')
          print('.y - ' + str(np.round(constants[index], 4)))
          print()
```

```
For Method 1:
[1.3141, 17.2144, -2.4152, 9.7297].y - 75.2062

For Method 2:
[2.9798, 11.9992, -1.3346, 10.7072].y - 67.1426

For Method 3:
[1.8675, 4.9536, 5.1875, 11.4901].y - 67.2733
```

2.2 (b)

```python
In [8]: # Performing Classifications

        def compute(index, sample):
          global multipliers, constants
          multiplier = multipliers[index]
          constant = constants[index]
          return np.dot(multiplier, sample.T) - constant

        for index in range(len(methods)):
          method = methods[index]
          print('For Method ' + str(index + 1) + ': ')
          print('# Observations = ' + str(sizes[index]))

          counts = [0, 0, 0]
          for sample_index in range(sizes[index]):
            sample = method[sample_index, :]
            scores = [compute(unit, sample) for unit in range(len(methods))]
            assignment = np.argmax(scores)
            counts[assignment] += 1

          print('# Samples classified into Methods 1, 2, 3 respectively:')
          print(counts)

          print()
```

```
For Method 1:
# Observations = 12
# Samples classified into Methods 1, 2, 3 respectively:
[9, 3, 0]

For Method 2:
# Observations = 12
# Samples classified into Methods 1, 2, 3 respectively:
[3, 7, 2]

For Method 3:
# Observations = 12
# Samples classified into Methods 1, 2, 3 respectively:
[0, 1, 11]
```

True Class	True Class Frequency	Predicted as 1	Predicted as 2	Predicted as 3
1	12	9	3	0
2	12	3	7	2
3	12	0	1	11

2.3 (*c*)

In [9]:
```python
# Implementing the hold-out method

def get_all_scores(unit, sample_index):
  global methods

  means = []
  covariances = []
  sizes = []

  for index, use_method in enumerate(methods):
    method = []
    for row in range(np.shape(use_method)[0]):
      if row == sample_index and unit == index:
        continue
      method.append(np.ndarray.tolist(use_method[row, :]))

    method = np.asmatrix(method)
    means.append(np.mean(method, axis = 0))
    covariances.append(np.cov(method.T))
    sizes.append(np.shape(method)[0])

  S_pooled = np.zeros(np.shape(covariances[0]))
  total_size = 0

  for index, S in enumerate(covariances):
    S_pooled += S * (sizes[index] - 1)
    total_size += sizes[index]

  S_pooled = S_pooled / (total_size - len(sizes))

  multipliers = []
  constants = []

  for index in range(len(methods)):
    multiplier = np.matmul(means[index], np.linalg.inv(S_pooled))
    constant = np.matmul(multiplier, means[index].T) / 2

    multipliers.append(multiplier)
    constants.append(constant)

  scores = []
  for index in range(len(methods)):
    multiplier = multipliers[index]
    constant = constants[index]
    scores.append(np.dot(multiplier, sample.T) - constant)
```

```
        return scores

    print('Performing all classifiations using hold-out method')
    print()
    for index in range(len(methods)):
        method = methods[index]
        print('For Method ' + str(index + 1) + ': ')
        print('# Observations = ' + str(sizes[index]))

        counts = [0, 0, 0]
        for sample_index in range(sizes[index]):
            sample = method[sample_index, :]
            scores = get_all_scores(index, sample_index)
            assignment = np.argmax(scores)
            counts[assignment] += 1

        print('# Samples classified into Methods 1, 2, 3 respectively:')
        print(counts)

        print()
```

```
Performing all classifiations using hold-out method

For Method 1:
# Observations = 12
# Samples classified into Methods 1, 2, 3 respectively:
[8, 4, 0]

For Method 2:
# Observations = 12
# Samples classified into Methods 1, 2, 3 respectively:
[4, 5, 3]

For Method 3:
# Observations = 12
# Samples classified into Methods 1, 2, 3 respectively:
[0, 1, 11]
```

True Class	True Class Frequency	Predicted as 1	Predicted as 2	Predicted as 3
1	12	8	4	0
2	12	4	5	3
3	12	0	1	11

```
In [10]: # ^_^ Thank You
```