CS4023D Artificial Intelligence Assignment 2

By Dev Sony, B180297CS

The question, report and source code can be found here.

Github Repo

Solution 1

Based on the formula given:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i).$$

The function has been defined:

```python
# Discriminant function as defined in the question
def discriminant_function(x, mean, cov, d, P):
    # Checking if the dimensions turn out to be scalars in the case only 1 feature is being taken.
    if d == 1:
        output = -0.5*(x - mean) * (1/cov)
        output = output * (x - mean)
        output += -0.5*d*log(2*pi) - 0.5*log(cov)

    else:
        output = np.matmul(-0.5*(x - mean), np.linalg.inv(cov))
        output = np.matmul(output, (x - mean).T)
        output += -0.5*d*log(2*pi) - 0.5*log(np.linalg.det(cov))

    # Adding Prior Probability
    output += (log(P) if P != 0 else 0)
```

The variables can be configured based on the scenario. Here, it's assumed that prior probabilities are equally distributed and all features are taken:

```python
    # Arbitary values
    n = len(data)
    P = [1/n for i in range(n)]
    d = len(data[0][0])
    g = [np.array([]) for _ in range(n)]
```

The input is the sample dataset, each set separated by the class they belong to as given below:

```python
# Sample Data
data = [
    # W1
    np.array([
        [-5.01, -8.12, -3.68],
        [-5.43, -3.48, -3.54],
        [1.08, -5.52, 1.66],
        [0.86, -3.78, -4.11],
        [-2.67, 0.63, 7.39],
        [4.94, 3.29, 2.08],
        [-2.51, 2.09, -2.59],
        [-2.25, -2.13, -6.94],
        [5.56, 2.86, -2.26],
        [1.03, -3.33, 4.33]
    ]),

    # W2
    np.array([
        [-0.91, -0.18, -0.05],
        [1.30, -2.06, -3.53],
        [-7.75, -4.54, -0.95],
        [-5.47, 0.50, 3.92],
        [6.14, 5.72, -4.85],
        [3.60, 1.26, 4.36],
        [5.37, -4.63, -3.65],
        [7.18, 1.46, -6.66],
        [-7.39, 1.17, 6.30],
        [-7.50, -6.32, -0.31]

    ]),

    # W3
    np.array([
        [5.35, 2.26, 8.13],
        [5.12, 3.22, -2.66],
        [-1.34, -5.31, -9.87],
        [4.48, 3.42, 5.19],
        [7.11, 2.39, 9.21],
        [7.17, 4.33, -0.98],
        [5.75, 3.97, 6.65],
        [0.77, 0.27, 2.41],
        [0.90, -0.43, -8.71],
        [3.52, -0.36, 6.43]
    ])
]
```

In order to classify the sample data, we first run the function through our sample dataset, classwise. On each sample, we find the class which gives the maximum output from its discriminant function.

A count and total count is maintained in order to find the success and failiure rates.

```python
# Taking each dataset from the classes in sample data
for j in range(n):
    print("\nData classes should be classified as:", j+1)
    total_count, count = 0, 0

    # Taking x as dataset belonging to class j + 1
    for x in data[j]:
        g_values = [0 for g in range(n)]        # Array for all discrminant function outputs.

        # Itering through each class' discriminant function
        for i in range(n):
            g_values[i] = discriminant_function(x, means[i], cov[i], d, P[i])

        # Now to output the maximum result
        result = g_values.index(max(g_values)) + 1
        print(x, "\twas classified as", result)
        total_count, count = total_count + 1, (count + 1 if j == result - 1 else count)

    print("Success Rate:", (count/total_count)*100,"%")
    print("Fail Rate:", 100 - ((count/total_count))*100,"%")
```

Assuming that all classes have an equal prior probability (as per the configuration in the example picture), the following output is produced:

```
› Assg1 main* python Qn1/main.py

Data classes should be classified as: 1
[-5.01 -8.12 -3.68]     was classified as 1
[-5.43 -3.48 -3.54]     was classified as 1
[ 1.08 -5.52  1.66]     was classified as 1
[ 0.86 -3.78 -4.11]     was classified as 1
[-2.67  0.63  7.39]     was classified as 2
[4.94 3.29 2.08]        was classified as 3
[-2.51  2.09 -2.59]     was classified as 1
[-2.25 -2.13 -6.94]     was classified as 1
[ 5.56  2.86 -2.26]     was classified as 3
[ 1.03 -3.33  4.33]     was classified as 1
Success Rate: 70.0 %
Fail Rate: 30.0 %

Data classes should be classified as: 2
[-0.91 -0.18 -0.05]     was classified as 2
[ 1.3  -2.06 -3.53]     was classified as 3
[-7.75 -4.54 -0.95]     was classified as 2
[-5.47  0.5   3.92]     was classified as 2
[ 6.14  5.72 -4.85]     was classified as 2
[3.6  1.26 4.36]        was classified as 3
[ 5.37 -4.63 -3.65]     was classified as 2
[ 7.18  1.46 -6.66]     was classified as 2
[-7.39  1.17  6.3 ]     was classified as 2
[-7.5  -6.32 -0.31]     was classified as 2
Success Rate: 80.0 %
Fail Rate: 20.0 %

Data classes should be classified as: 3
[5.35 2.26 8.13]        was classified as 3
[ 5.12  3.22 -2.66]     was classified as 3
[-1.34 -5.31 -9.87]     was classified as 3
[4.48 3.42 5.19]        was classified as 3
[7.11 2.39 9.21]        was classified as 3
[ 7.17  4.33 -0.98]     was classified as 3
[5.75 3.97 6.65]        was classified as 3
[0.77 0.27 2.41]        was classified as 1
[ 0.9  -0.43 -8.71]     was classified as 3
[ 3.52 -0.36  6.43]     was classified as 3
Success Rate: 90.0 %
Fail Rate: 10.0 %
```

Solution 2

Part (a) and (b)

In order to match the question, the configuration variables are altered.

- (data-1) for n indicates that only 2 classes will be considered (the final class would not be considered as its Prior probability is 0, implying that it wouldn't appear.)
- The d value is changed to 1, indicating that only 1 feature will be used. (which is x_1)

```
78      # Configuration values
79      n = len(data) - 1
80      P = [0.5, 0.5, 0]
81      d = 1
```

The parameters being passed is also changed

- x[0] indicates that only x^1 will be used.

- means[i][0] indiciates that we need the mean only for x_1).
- cov[i][0][0] indicates the variance of feature x_1).