LEIC-T 2024/2025
Aprendizagem - Machine Learning
Homework I

Group 36
Francisco Uva – 106340
Pedro Pais - 107482

I) Bayesian Classifier

a)

$\mu_{A1} = (0.6 + 1 + 1.6 + 1.8)/4 = 1.25$

$\mu_{A2} = (0.4 + 1.1 + 1.5 + 1.8)/4 = 1.2$

$\sigma_{A1} = (((0.6 - 1.25)^2 + (1 - 1.25)^2 + (1.6 - 1.25)^2 + (1.8 - 1.25)^2)/3)^{1/2} = 0.550757$

$\sigma_{A2} = (((0.4 - 1.2)^2 + (1.1 - 1.2)^2 + (1.5 - 1.2)^2 + (1.8 - 1.2)^2)/3)^{1/2} = 0.60553$

$\mu_{B1} = (2 + 2 + 3 + 4)/4 = 2.75$

$\mu_{B2} = (0 + 1 + 0 + 1.2)/4 = 0.55$

$\sigma_{B1} = (((2 - 2.75)^2 + (2 - 2.75)^2 + (3 - 2.75)^2 + (4 - 2.75)^2)/3)^{1/2} = 0.957427$

$\sigma_{B1} = (((0 - 0.55)^2 + (1 - 0.55)^2 + (0 - 0.55)^2 + (1.2 - 0.55)^2)/3)^{1/2} = 0.640312$

p(A)=0.5, p(B)=0.5

p(A, xquery) = p(xquery,1|A1) · p(xquery,2|A2) · p(A) = 0.653444 * 0.275267 * 0.5 = 0.089936

p(B, xquery) = p(xquery,1|B1) · p(xquery,2|B2) · p(B) = 0.078403 * 0.047971 * 0.5 = 0.001881

p(A|xquery) = p(A, xquery)/ (p(A,xquery) +p(B,xquery)) = 0.979519

p(B|xquery) = p(A, xquery)/ (p(A,xquery) +p(B,xquery)) = 0.020481

Python Program that we use:

```
import numpy as np


def gaussian_prob(x, mu, sigma):
    return (1 / (np.sqrt(2 * np.pi * sigma2))) * np.exp(-((x - mu)2) / (2 * sigma*2))
```

Group 36
Francisco Uva – 106340
Pedro Pais - 107482

```
mu_A1, sigma_A1 = 1.25, 0.550757
mu_A2, sigma_A2 = 1.2, 0.60553
mu_B1, sigma_B1 = 2.75, 0.957427
mu_B2, sigma_B2 = 0.55, 0.640312
x1_query, x2_query = 1, 2
p_A = 0.5
p_B = 0.5
p_x1_A = gaussian_prob(x1_query, mu_A1, sigma_A1)
p_x2_A = gaussian_prob(x2_query, mu_A2, sigma_A2)
p_A_xquery = p_x1_A p_x2_A * p_A
print(p_A_xquery)

p_x1_B = gaussian_prob(x1_query, mu_B1, sigma_B1)
p_x2_B = gaussian_prob(x2_query, mu_B2, sigma_B2)
p_B_xquery = p_x1_B * p_x2_B * p_B
print(p_B_xquery)

p_A_given_xquery = p_A_xquery / (p_A_xquery + p_B_xquery)
p_B_given_xquery = p_B_xquery / (p_A_xquery + p_B_xquery)
print(p_A_given_xquery, p_B_given_xquery)
```

LEIC-T 2024/2025
Aprendizagem - Machine Learning
Homework I

Group 36
Francisco Uva – 106340
Pedro Pais - 107482

b)

$\mu_A=(1.25,1.2)^T$

ca11 = $(((0.6 - 1.25)^2 + (1 - 1.25)^2 + (1.6 - 1.25)^2 + (1.8 - 1.25)^2))/3 = 0.303333$

ca22 = $(((0.4 - 1.2)^2 + (1.1 - 1.2)^2 + (1.5 - 1.2)^2 + (1.8 - 1.2)^2))/3 = 0.366667$

ca12 = $((0.6 - 1.25) * (0.4 - 1.2) + (1 - 1.25) * (1.1 - 1.2) + (1.6 - 1.25) * (1.5 - 1.2) +(1.8 - 1.25) * (1.8 - 1.2))/3 = 0.326667$

$\Sigma A = \begin{pmatrix} 0.303333 & 0.326667 \\ 0.326667 & 0.366667 \end{pmatrix}$

$\Sigma^{-1} A = \begin{pmatrix} 81.285167 & -72.417702 \\ -72.417702 & 67.244866 \end{pmatrix}$

Det(A) = 0.004511

$\mu_B=(2.75,0.55)^T$

cb11 = $(((2 - 2.75)^2 + (2 - 2.75)^2 + (3 - 2.75)^2 + (4 - 2.75)^2))/3 = 0.916667$

cb22 = $(((0 - 0.55)^2 + (1 - 0.55)^2 + (0- 0.55)^2 + (1.2 - 0.55)^2))/3 = 0.41$

cb12 = $((2 - 2.75) * (0 - 0.55) + (2 - 2.75) * (1 - 0.55) + (3 - 2.75) * (0- 0.55) +(4 - 2.75) * (1.2 - 0.55))/3 = 0.25$

$\Sigma B = \begin{pmatrix} 0.916667 & 0.25 \\ 0.25 & 0.41 \end{pmatrix}$

$\Sigma^{-1} B = \begin{pmatrix} 1.308510 & -0.797872 \\ -0.797872 & 2.925532 \end{pmatrix}$

Det(B) = 0.313333

p(A, xquery) = p(xquery,1|A) · p(A) = $4.32594*10^{-17}$ * 0.5 = $2.162968*10^{-17}$

p(B,xquery)=p(xquery,1|B)·p(B) = 0.000233 * 0.5 = 0.000117

p(A|xquery) = p(A, xquery)/ (p(A,xquery) +p(B,xquery)) = $1.84869*10^{-13}$

Group 36
Francisco Uva – 106340
Pedro Pais - 107482

R: No, they are not the same. Using 1-dimensional Gaussians assumes feature independence and may ignore correlations, while 2-dimensional Gaussians captures feature correlations.

Pyhon code that we use:

```
import numpy as np


def multivariate_normal_pdf(x, mu, Sigma):
    D = len(mu)
    Sigma_det = np.linalg.det(Sigma)
    Sigma_inv = np.linalg.inv(Sigma)
    norm_factor = 1 / (np.power(2 * np.pi, D / 2) * np.sqrt(Sigma_det))
    diff = x - mu
    exponent = -0.5 * np.dot(np.dot(diff.T, Sigma_inv), diff)
    return norm_factor * np.exp(exponent)
muA = np.array([1.25, 1.2])
muB = np.array([2.75, 0.55])
SigmaA = np.array([[0.303333, 0.326667],
                   [0.326667, 0.366667]])
SigmaB = np.array([[0.916667, 0.25],
                   [0.25, 0.41]])
x = np.array([1, 2])


print(multivariate_normal_pdf(x, muA, SigmaA))
print(multivariate_normal_pdf(x, muB, SigmaB))
```

LEIC-T 2024/2025
Aprendizagem - Machine Learning
Homework I

Group 36
Francisco Uva – 106340
Pedro Pais - 107482

c)

X_3	CLASS
0	A
1	A
1	A
0	A
1	B
1	B
0	B
1	B

And the query vector X_3 = True = 1

p(A|1) = Card(A.1) / Card(1) = $\frac{2}{5}$

P(B|1) = Card(B.1) / Card(1) = $\frac{3}{5}$

Most probable class is B.

d)

p(A, xquery) = p((1,2)|A) ·P(1|A) ·p(A) = $4.32594*10^{-17} * \frac{2}{4} *0.5 = 1.081485\times10^{-17}$

p(B, xquery) = p((1,2)|B) ·P(1|B) ·p(B) = $0.000233 * \frac{3}{4} * 0.5 = 8.737\times10^{-5}$

p(A|xquery) = p(A, x_{query})/ (p(A,x_{query}) +p(B,x_{query})) = $\frac{1.081485\times10-17}{1.081485\times10-17+ 8.737\times10-5}$ = 1.23782×10^{-13}

p(B|xquery) =1- p(A|xquery)= 0.9999999999998762

LEIC-T 2024/2025
Aprendizagem - Machine Learning
Homework I

Group 36
Francisco Uva – 106340
Pedro Pais - 107482

III) Software Experiments

For the Digits dataset, kNN with k=3 achieves the highest accuracy (0.97), slightly better than kNN with k=30 (0.94) and Gaussian Naive Bayes (0.84). For the Wine dataset, Gaussian Naive Bayes clearly performs better (0.97), while kNN accuracy is lower with k=30 (0.72) and k=3 (0.64).

The Digits dataset likely benefits from the localized decision-making of kNN, as it involves images, where pixel values correlate strongly with neighbors. In contrast, the Wine dataset has features that better match the assumptions of Gaussian distributions, which explains why GaussNB outperforms kNN in this case.