Αναγνώριση Προτύπων

2021-2022

Εργασία 3η

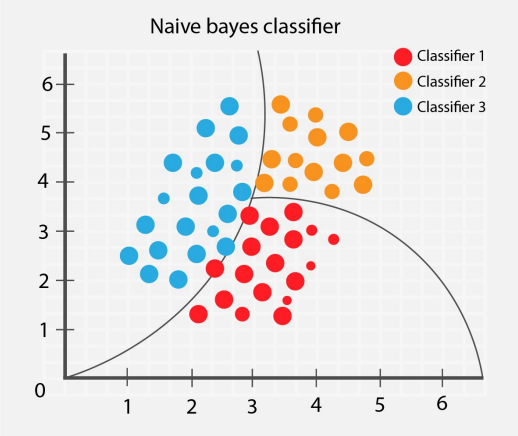
ΛΗΨΗ ΑΠΟΦΑΣΗΣ ΚΑΤΑ BAYES - ΕΚΤΙΜΗΤΕΣ BAYES

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$



Καρυπίδης Ευστάθιος 57556

30/11/21, Ξάνθη

Για την υλοποίηση των ερωτημάτων της εργασίας έγινε χρήση γλώσσας python και των βιβλιοθηκών numpy,scipy και matplotlib.

Άσκηση 1

Οι συναρτήσεις της άσκησης 1 βρίσκονται στο αρχείο **HW3_Exercise1.py** .

Ερώτημα α

Ζητείται να γραφεί ενα πρόγραμμα για υπολογισμό της συνάρτησης διάκρισης της μορφής:

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

για μια δεδομένη κανονική κατανομή d διαστάσεων και εκ των προτέρων πιθανότητα $P(\omega_i)$.

Για το λόγο αυτό υλοποίησα την παρακάτω συνάρτηση:

```
def discriminant_function(x, m, s, prior, d):
    c = x - m
    if d > 1:
        S_inv = np.linalg.inv(s)
        det_S = round(abs(np.linalg.det(s)), 5)
        return -0.5*np.linalg.multi_dot([c.T, S_inv, c]) - (d/2)*math.log(2*math.pi) -
0.5*math.log(det_S) + math.log(prior)
    else:
        S_inv = s ** (-1)
        det_S = abs(s)
        return -0.5 * c ** 2 * S_inv - (d / 2) * math.log(2*math.pi) - 0.5 *
math.log(det_S) + math.log(prior)
```

Συγκεκριμένα δέχεται ως ορίσματα:

- x: Το διάνυσμα του δείγματος
- m: Η μέση τιμή ή ο πίνακας μέσων τιμών όταν d > 1
- s: Η μεταβλητότητα ή ο πίνακας συμμεταβλητότητας όταν d > 1
- prior: Η a priori πιθανότητα.
- d: Ο αριθμός των διαστάσεων της κατανομής. Θα μπορούσαμε να πάρουμε αυτή τη πληροφορία και από το μήκος του διανύσματος x αλλά για ευκολία το βάζουμε ως όρισμα.

Όπως παρατηρούμε υπάρχει μία συνθήκη if η οποία ελέγχει αν ο αριθμός των διαστάσεων είναι μεγαλύτερος από 1. Αυτό συμβαίνει καθώς η βιβλιοθήκη numpy και ειδικότερα οι συναρτήσεις(`np.linalg.inv`, `np.linalg.multi_dot`) περιμένουν πίνακες ως όρισμα αλλά όταν είμαστε σε 1 διάσταση τότε έχουμε αριθμούς και συνεπώς υπήρχε πρόβλημα. Συνεπώς διαχωρίζουμε τις 2 περιπτώσεις. Ειδικότερα όταν έχουμε 1 διάσταση ο τύπος απλοποιείται όπως βλέπουμε. Η συνάρτηση `np.linalg.inv` υπολογίζει τον αντίστροφο ενός πίνακα, η `np.linalg.multi_dot` υπολογίζει εσωτερικά γινόμενα πινάκων στη σειρά και τέλος η `math.log` υπολογίζει τον φυσικό λογάριθμο(βάση το e).

Ερώτημα β

Στο ερώτημα αυτό ζητείται να γραφτεί ένα πρόγραμμα για τον υπολογισμό της Ευκλείδιας απόστασης μεταξύ 2 αυθαίρετων σημείων x_1 και x_2 στις d διαστάσεις. Αυτή, μαθηματικά ορίζεται ως εξής:

$$D_e(x,y) = \left(\sum_{i=1}^{d}(x_i - y_i)^2\right)^{\frac{1}{2}} = ((x-y)^T(x-y))^{\frac{1}{2}}$$

Η συνάρτηση που υλοποίησα είναι η εξής:

```
def euclidian_distance(x1, x2, d):
    if d > 1:
        return np.sqrt(np.sum(np.dot((x1-x2).T, x1-x2)))
        # return math.sqrt(np.sum(np.square(x1 - x2)))
    else:
        return abs(x1 - x2)
```

Στη συγκεκριμένη συνάρτηση δίνουμε ως όρισμα τα σημεία x_1 και x_2 και τις διαστάσεις τους. Αντιστοιχα για τον ίδιο λόγο με πρίν ξεχωρίζουμε τις περιπτώσεις αν d>1 ή όχι. Προτιμάω τον 1ο τύπο από επάνω καθώς ο 2ος θα βόλευε καλύτερα σε περίπτωση που θέλουμε να υπολογίσουμε αποστάσεις πολλών σημείων (περισσοτέρων από 2).

Αντίστοιχα, στην numpy υπάρχει η συνάρτηση `np.linalg.norm(x1-x2)` με την οποία μπορούμε επίσης να υπολογίσουμε την ευκλείδια απόσταση.

Ερώτημα γ

Η απόσταση Mahalanobis είναι ένα μετρικό της απόστασης σε πολλές διαστάσεις μεταξύ ενός σημείου x(διάνυσμα) και μιας κατανομής. Ουσιαστικά μετράει πόσες τυπικές αποκλίσεις της κατανομής απέχει το σημείο x από την μέση τιμή (και συνεπώς απο τη κατανομή). Αυτό μπορούμε να το δούμε και απο τον ορισμό της:

$$D_m^2 = (x-\mu)^T\Sigma^{-1}(x-\mu)$$

Σε περίπτωση που ο πίνακας συμμεταβλητότητας είναι ο μοναδιαίος τότε η απόσταση Mahalanobis ανάγεται στην ευκλείδια απόσταση. Η συνάρτηση που υλοποίησα είναι η εξής:

```
def mahalanobis_distance(x, m, s, d):
    c = x - m
    if d > 1:
        S_inv = np.linalg.inv(s)
        return np.sqrt(np.linalg.multi_dot([c.T, S_inv, c]))
    else:
        S_inv = s ** (-1)
        return math.sqrt(c*S_inv*c)
```

Συγκεκριμένα δέχεται ως ορίσματα:

- d: Ο αριθμός των διαστάσεων της κατανομής. Θα μπορούσαμε να πάρουμε αυτή τη πληροφορία και από το μήκος του διανύσματος x αλλά για ευκολία το βάζουμε ως όρισμα.
- x: Το διάνυσμα του δείγματος
- m: Η μέση τιμή ή ο πίνακας μέσων τιμών όταν d > 1
- s: Η μεταβλητότητα ή ο πίνακας συμμεταβλητότητας όταν d > 1

Άσκηση 2

Αρχικά, έγραψα τα δεδομένα τις άσκησης σε έναν numpy array πίνακα και τον αποθήκευσα σε ένα αρχείο το οποίο το διαβάζω κατα την αρχή του προγράμματος. Ειδικότερα μέσω της παρακάτω εντολής έχουμε:

```
my_data = np.load("my_data.npy")
```


my_data

	0	1	2	3	4	5	6	7	8
0	-5.01000	-8.12000	-3.68000	-0.91000	-0.18000	-0.05000	5.35000	2.26000	8.13000
1	-5.43000	-3.48000	-3.54000	1.30000	-2.06000	-3.53000	5.12000	3.22000	-2.66000
2	1.08000	-5.52000	1.66000	-7.75000	-4.54000	-0.95000	-1.34000	-5.31000	-9.87000
3	0.86000	-3.78000	-4.11000	-5.47000	0.50000	3.92000	4.48000	3.42000	5.19000
4	-2.67000	0.63000	7.39000	6.14000	5.72000	-4.85000	7.11000	2.39000	9.21000
5	4.94000	3.29000	2.08000	3.60000	1.26000	4.36000	7.17000	4.33000	-0.98000
6	-2.51000	2.09000	-2.59000	5.37000	-4.63000	-3.65000	5.75000	3.97000	6.65000
7	-2.25000	-2.13000	-6.94000	7.18000	1.46000	-6.66000	0.77000	0.27000	2.41000
8	5.56000	2.86000	-2.26000	-7.39000	1.17000	6.30000	0.90000	-0.43000	-8.71000
9	1.03000	-3.33000	4.33000	-7.50000	-6.32000	-0.31000	3.52000	-0.36000	6.43000

Ερώτημα 1

Γνωρίζουμε ότι:

$$p(\omega_1) = p(\omega_2) = \frac{1}{2} \text{ και } p(\omega_3) = 0$$

Άρα στη περίπτωση αυτή δε μας ενδιαφέρει η κλάση 3. Εφόσον δεν έχουμε τα πραγματικά στατιστικά των κατανομών από τις οποίες προέρχονται τα δεδομένα αλλά γνωρίζουμε ότι οι κατανομές είναι Gaussian καταφεύγουμε στην εκτίμηση παραμέτρων με χρήση της τεχνικής Maximum Likelihood Estimation. Ειδικότερα, γνωρίζουμε ότι:

$$\hat{\mu} = \frac{1}{n}\sum_{k=1}^{n} x_k \quad \text{και} \quad \hat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{n}\left(x_k - \hat{\mu}\right)^2$$

Αφού χωρίσουμε λοιπόν τα δεδομένα της κάθε κλάσης προχωράμε σε εκτίμηση των παραμέτρων:

```
# Assign data for each class
w1_data = my_data[:, 0:3]
w2_data = my_data[:, 3:6]
w3_data = my_data[:, 6:]

# Estimating parameters/statistics of Gaussian's
w1_means = np.mean(w1_data, axis=0)
w2_means = np.mean(w2_data, axis=0)
w3_means = np.mean(w3_data, axis=0)
w1_covariance = np.cov(w1_data, rowvar=False)
w2_covariance = np.cov(w2_data, rowvar=False)
w3_covariance = np.cov(w3_data, rowvar=False)
```

Αξίζει να σημειωθεί ότι στη συνάρτηση `np.cov` της numpy δίνουμε όρισμα `rowvar = False` έτσι ώστε να αναγνωρίζει τις γραμμές σαν παρατηρήσεις και τις στείλες ως τα γνωρίσματα/χαρακτηριστικά και να έχουμε σωστά αποτελέσματα.

Το σημείο διαχωρισμού μπορεί να βρεθεί από την παρακάτω εξίσωση: $g(x) = g_1(x) - g_2(x) = 0$ δηλαδή αν εξισώσουμε την διαφορά των 2 συναρτήσεων διάκρισης. Εδώ υπάρχουν 2 τρόποι εφόσον θέλουμε να το λύσουμε σε κώδικα. Αρχικά μπορούμε να βρούμε με ένα διακριτό μικρό βήμα(πχ 10^{-4}) τις τιμές της $g(x)$ και να ελέγχουμε πότε αυτή θα γίνει σχεδόν μηδέν(τουλάχιστον μικρότερη από μια ανοχή της τάξης tol = 10^{-5}) . Ένας άλλος τρόπος είναι να χρησιμοποιήσουμε την βιβλιοθήκη sympy και να ορίσουμε μια συμβολική μεταβλητή(εδώ την z1) και με τη χρήση της συνάρτησης `sympy.solve` να βρούμε τις αντίστοιχες λύσεις. Δοκιμάστηκαν και τα 2 στα πλαίσια επιβεβαίωσης των αποτελεσμάτων. Τέλος όπως αναφέρεται και στην εκφώνηση χρησιμοποιείται μόνο το χαρακτηριστικό x1.

```
# ---------- Exercise 2.1 ----------
# Mean and covariance estimation using only feature x1 (First column or column at index 0)
one_dim_w1_mean = w1_means[0]  # mean of class 1 for feature x1
one_dim_w2_mean = w2_means[0]  # mean of class 2 for feature x1
one_dim_w1_covariance = w1_covariance[0, 0]  # covariance of class 1 for feature x1
one_dim_w2_covariance = w2_covariance[0, 0]  # covariance of class 1 for feature x1
# Solving equation with discrete steps
tol = 10 ** (-5)  # tolerance
x = np.arange(-10, 10, tol*10)  # Defining the space where we seek for the solution
g = Ex1.discriminant_function(x, one_dim_w1_mean, one_dim_w1_covariance, 0.5, 1) -
Ex1.discriminant_function(x, one_dim_w2_mean, one_dim_w2_covariance, 0.5, 1)
index = list(zip(*np.where(abs(g) < tol)))  # Finding the solutions with tolerance
for i in range(len(list(index))):
    print("g1(x)-g2(x)=0 for x: ", x[index[i]], "", "with error tolerance: ", g[index[i]])
# Solving equation using symbolic variables
z1 = sym.Symbol("z1")
eq = Ex1.discriminant_function(z1, one_dim_w1_mean, one_dim_w1_covariance, 0.5, 1) -
Ex1.discriminant_function(z1, one_dim_w2_mean, one_dim_w2_covariance, 0.5, 1)
sol = sympy.solve(eq)
print("Solutions using sympy: ", sol[0], " and ", sol[1])
```

Τα αποτελέσματα είναι τα εξής:

```
g1(x)-g2(x)=0 for x:  -5.085500000011454  with error tolerance:  1.5986443457904898e-07
g1(x)-g2(x)=0 for x:  4.337499999966585  with error tolerance:  -7.4852035387706906e-06
Solutions using sympy:  -5.08550080052558  and  4.33746251829143
```

Όπως παρατηρούμε οι λύσεις και με τους 2 τρόπους είναι σχεδόν ίδιες με απόκλιση μετά το 7 δεκαδικό.

Ερώτημα 2

Στη συνέχεια ζητείται να προσδιοριστεί το εμπειρικό σφάλμα κατάρτισης σχετικά με τα δείγματα, δηλαδή το ποσοστό των σημείων που ταξινομείται εσφαλμένα. Για να ταξινομήσουμε τα σημεία μπορούμε να χρησιμοποιήσουμε την discriminant_function που ορίσαμε στην άσκηση 1. Με βάση τη θεωρία ένα δείχμα ταξινομείται σε μία κλάση αν το αποτέλεσμα ή αλλιώς η τιμή της συνάρτησης διάκρισης είναι μεγαλύτερη από τις αντίστοιχες τιμές των συναρτήσεων διάκρισης των υπόλοιπων κλάσεων. Εδώ εφόσον έχουμε ταξινόμηση 2 κλάσεων:

- Ταξινομώ ένα δείγμα στην κλάση 1 εάν $\large g_1(x)>g_2(x) $
- Ταξινομώ ένα δείγμα στην κλάση 2 εάν $\large g_1(x)<g_2(x) $

Ετσι, για να υπολογίσουμε το σφάλμα μπορούμε να ακολουθήσουμε την εξής μεθοδολογία.

- Υπολογίζω τις τιμές των $g_1(x)$ και $g_2(x)$ για τα δεδομένα της κλάσης 1 και αυξάνω τον αριθμό των δειγμάτων που ταξινομήθηκε λανθασμένα κατα 1 κάθε φορά όπου $g_1(x) < g_2(x) \Rightarrow g_a(x) = g_1(x) - g_2(x) < 0$.
- Αντίστοιχα υπολογίζω τις τιμές των $g_1(x)$ και $g_2(x)$ για τα δεδομένα της κλάσης 2 και αυξάνω τον αριθμό των δειγμάτων που ταξινομήθηκε λανθασμένα κατα 1 κάθε φορά όπου $\large g_1(x)>g_2(x) \Rightarrow g_a(x)=g_1(x)-g_2(x) >0 $.

Τέλος, σε κάθε περίπτωση για να βρώ το ποσοστό σφάλματος διαιρώ με τον αριθμώ των δειγμάτων. Στη δική μας περίπτωση έχουμε 10 δειγματα από την πρώτη κλάση και 10 από τη δεύτερη άρα στο σύνολο 20.

```
# Calculating classification error
g_a = Ex1.discriminant_function(w1_data[:, 0], one_dim_w1_mean, one_dim_w1_covariance, 0.5,
1) - Ex1.discriminant_function(w1_data[:, 0], one_dim_w2_mean, one_dim_w2_covariance, 0.5,
1)
g2 = Ex1.discriminant_function(w2_data[:, 0], one_dim_w1_mean, one_dim_w1_covariance, 0.5,
1) - Ex1.discriminant_function(w2_data[:, 0], one_dim_w2_mean, one_dim_w2_covariance, 0.5,
1)
wrong = (g1 < 0).sum() + (g2 > 0).sum()
print(wrong, "/", 20, " points misclassified using only feature x1. Error is : ", wrong/20
* 100, "%")
```

Τα αποτελέσματα είναι τα εξής:

```
6 / 20  points misclassified using only feature x1. Error is :  30.0 %
```

Ερώτημα 3

Στη περίπτωση αυτή ζητείται να επαναληφθεί η παραπάνω διαδικασία αλλά προσθέτοντας ένα ακόμη χαρακτηριστικό. Συνεπώς, από τις εκτιμήσεις των στατιστικών για 3 χαρακτηριστικά κρατάω εγώ για τα 2 και αντίστοιχα δοκιμάζω με τον παρακάτω κώδικα την συνάρτηση discriminant_function σε 2 διαστάσεις.

```
# ---------- Exercise 2.3 ----------
# Mean and covariance estimation using features x1,x2 (First and second column or column at
index 0 and 1)
two_dim_w1_mean = w1_means[0:2]  # means of class 1 for features x1,x2
two_dim_w2_mean = w2_means[0:2]  # means of class 2 for features x1,x2
two_dim_w1_covariance = w1_covariance[0:2, 0:2]  # covariance matrix of class 1 for feature
x1,x2
two_dim_w2_covariance = w2_covariance[0:2, 0:2]  # covariance matrix of class 2 for feature
x1,x2

wrong = 0
for i in range(10):
    g_a = Ex1.discriminant_function(w1_data[i, 0:2], two_dim_w1_mean,
two_dim_w1_covariance, 0.5, 2) - Ex1.discriminant_function(w1_data[i, 0:2],
two_dim_w2_mean, two_dim_w2_covariance, 0.5, 2)
    g_b = Ex1.discriminant_function(w2_data[i, 0:2], two_dim_w1_mean,
two_dim_w1_covariance, 0.5, 2) - Ex1.discriminant_function(w2_data[i, 0:2],
two_dim_w2_mean, two_dim_w2_covariance, 0.5, 2)
    if g_a < 0 and g_b > 0:
        wrong += 2
    elif g_a < 0 or g_b > 0:
        wrong += 1
print(wrong, "/", 20, " points misclassified using only feature x1. Error is : ", wrong/20
* 100, "%")
```

Τα αποτελέσματα είναι τα εξής:

```
9 / 20  points misclassified using features x1 and x2. Error is :  45.0 %
```

Ερώτημα 4

Αντίστοιχα, όπως τα προηγούμενα 2 ερωτήματα αλλά σε αυτή τη περίπτωση χρησιμοποιώ και τα 3 χαρακτηριστικά.

```
# ---------- Exercise 2.4 ----------
wrong = 0
for i in range(10):
    g_a = Ex1.discriminant_function(w1_data[i, :], w1_means, w1_covariance, 0.5, 3) -
Ex1.discriminant_function(w1_data[i, :], w2_means, w2_covariance, 0.5, 3)
    g_b = Ex1.discriminant_function(w2_data[i, :], w1_means, w1_covariance, 0.5, 3) -
Ex1.discriminant_function(w2_data[i, :], w2_means, w2_covariance, 0.5, 3)
    if g_a < 0 and g_b > 0:
        wrong += 2
    elif g_a < 0 or g_b > 0:
        wrong += 1
print(wrong, "/", 20, " points misclassified using only feature x1. Error is : ", wrong/20
* 100, "%")
```

Τα αποτελέσματα είναι τα εξής:

```
3 / 20  points misclassified using all features x1, x2, x3. Error is :  15.0 %
```

Ερώτημα 5

Αρχικά, παραθέτω συγκεντρωτικά τα αποτελέσματα:

```
6 / 20  points misclassified using only feature x1. Error is :  30.0 %
9 / 20  points misclassified using features x1 and x2. Error is :  45.0 %
3 / 20  points misclassified using all features x1, x2, x3. Error is :  15.0 %
```

Παρατηρούμε εδώ ότι αρχικά αυξάνοντας τον αριθμό των χαρακτηριστικών από 1 σε 2 το σφάλμα αυξάνεται. Αυτό, μπορεί να συμβαίνει καθώς το χαρακτηριστικό x2 αντί να εισάγει περισσότερη πληροφορία μπορεί να εισάγει θόρυβο. Επίσης, όπως είδαμε στη θεωρία η χρήση περισσότερων παραμέτρων μπορεί να έχει αρνητικό αποτέλεσμα εαν δεν γίνει και αύξηση των αντίστοιχων δεδομένων εκπαίδευσης(Φαινόμενο Curse of dimensionality). Τέλος, η προσθήκη και του τρίτου χαρακτηριστικού φαίνεται να συνέβαλε καθοριστικά στη βελτίωση της ταξινόμησης καθώς το σφάλμα μειώθηκε στο 1/3 σε σχέση με τη χρήση 2 χαρακτηριστικών και στο 1/2 σε σχέση με τη χρήση 1 χαρακτηριστικού. Αυτό, σημαίνει πως το συγκεκριμένο χαρακτηριστικό είναι πλούσιο σε πληροφορία.

Ερώτημα 6

Στο ερώτημα αυτό ζητείται να βρεθούν οι συνατήσεις διάκρισης $g_1(x), g_2(x), g_3(x)$ για $p(\omega_1) = 0.8$ και $p(\omega_2) = p(\omega_3) = 0.1$. Η υλοποίηση σε κώδικα έγινε με τη χρήση αντιστοιχων συμβολικών μεταβλητών για τα γνωρίσαμτα x_1, x_2, x_3.

```
print("\n Symbolic Equations for g1,g2,g3: \n")
x1, x2, x3 = sym.symbols("x1 x2 x3")
matrix = sym.Matrix([x1, x2, x3])
# Original shape was (3,)
g_1 = sym.simplify(Ex1.discriminant_function(matrix, w1_means.reshape((3, 1)),
w1_covariance, 0.8, 3))
g_2 = sym.simplify(Ex1.discriminant_function(matrix, w2_means.reshape((3, 1)),
w2_covariance, 0.1, 3))
g_3 = sym.simplify(Ex1.discriminant_function(matrix, w3_means.reshape((3, 1)),
w3_covariance, 0.1, 3))
print(" g1(x) = ", g_1, "\n", "g2(x) = ", g_2, "\n", "g3(x) = ", g_3, "\n")
```

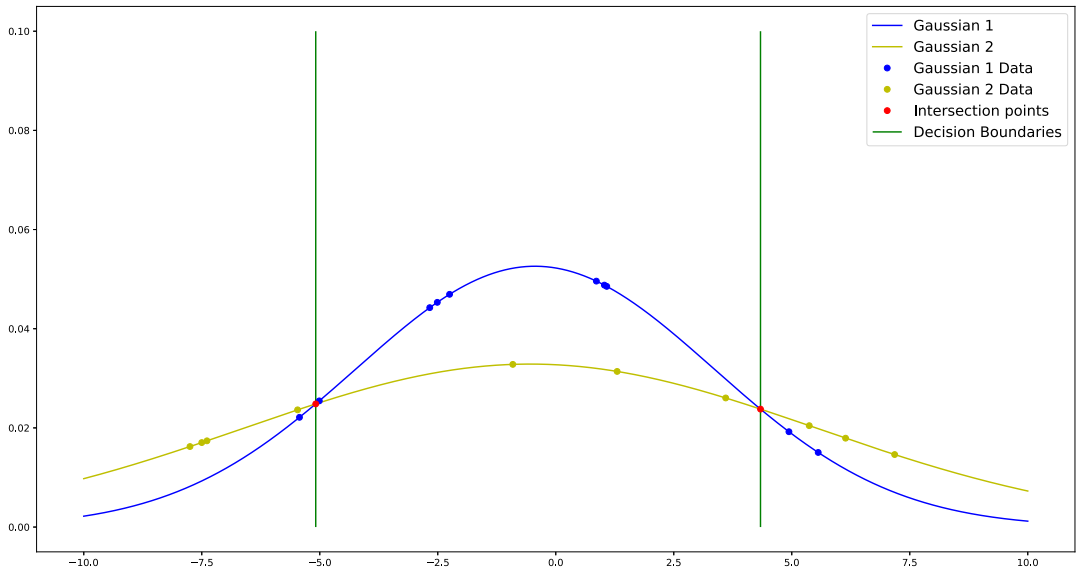
Το αποτέλεσμα είναι το εξής:

```
Symbolic Equations for g1,g2,g3:

g1(x) =  [[-0.0494938232782041*x1**2 + 0.0491666304006326*x1*x2 + 0.0109494488156713*x1*x3 + 0.050825149878691*x1 - 0.0483133257071814*x2**2 + 0.00886328367685942*x2*x3 - 0.140577420650968*x2 -
0.0273710998842904*x3**2 - 0.0216128843930103*x3 - 7.06141923156482]]
g2(x) =  [[-0.0353195699215583*x1**2 + 0.0555278158282839*x1*x2 - 0.063993974204001*x1*x3 - 0.0307295912922285*x1 - 0.0598200315235911*x2**2 + 0.051991613892289*x2*x3 - 0.0328346693175741*x2 -
0.0561483149700974*x3**2 - 0.0559958916344339*x3 - 9.16583643962454]]
g3(x) =  [[-0.325871490005846*x1**2 + 0.481259214036771*x1*x2 + 0.0630908224491925*x1*x3 + 1.76882181340108*x1 - 0.265241303448903*x2**2 - 0.00359904713428515*x2*x3 - 1.13309896654123*x2 - 0
.0189039991540648*x3**2 - 0.180292737386593*x3 - 10.5849034871073]]
```

Οπτικοποίηση των αποτέλεσμάτων

Στη περίπτωση χρήσης ενός χαρακτηριστικού μπορούμε να οπτικοποιήσουμε όλα τα αποτελέσματα δηλαδή τις pdf, τα σημεία, τα σημεία τομής και τέλος τις συναρτήσεις διάκρισης.



Όπως παρατηρούμε και εδώ υπάρχουν 6 σημεία που θα ταξινομηθούν λάθος.

Άσκηση 3

Ερώτημα 1

Δίνεται η κατανομή του θ:

$$p\left(\theta \mid D^0\right) = \begin{cases} A\sin\left(\pi\theta\right) & \text{για } 0 \leq \theta \leq 1 \\ 0 & \text{αλλού} \end{cases}$$

Για κάθε pdf ισχυεί:

$$\int_{-\inf}^{+\inf} p\left(\theta/D^0\right)d\theta = 1 \Rightarrow \int_0^1 p\left(\theta/D^0\right)d\theta = 1 \Rightarrow \int_0^1 \mathrm{A}\sin(\pi\theta)d\theta = 1 \Rightarrow$$

$$A\left[\frac{-\cos\left(\pi\theta\right)}{\pi}\right]_0^1 = 1 \Rightarrow \mathrm{A}\left[-\frac{-1}{\pi} + \frac{1}{\pi}\right] = 1 \Longrightarrow \mathrm{A} = \frac{\pi}{2}$$

Αντίστοιχα αυτό το αποτέλεσμα μπορούμε να το παράξουμε και μέσω κώδικα με χρήση της συνάρτησης `quad` από τη βιβλιοθήκη `numpy.linalg`

```
def p_theta_0(theta2):
    return np.sin(math.pi*theta2)


A = 1/quad(p_theta_0, 0, 1)[0]
print("A = ", A)
```

Το αποτέλεσμα είναι το εξής:

```
A =  1.5707963267948966
```

Όπως βλέπουμε επιβεβαιώνεται και το θεωρητικό αποτέλεσμα(π/2 = 1.57...)

Ερώτημα 2

Η γραφική απεικόνιση των $p\left(\theta \mid D^1\right), p\left(\theta \mid D^5\right), p\left(\theta \mid D^{10}\right)$ γίνεται με χρήση του κώδικα. Ωστόσο, ας δούμε θεωρητικά πως θα παραχτούν τα αποτελέσματα:

Από τη θεωρία γνωρίζουμε το παρακάτω αναδρομικό τύπο.

$$p\left(\boldsymbol{\theta} \mid D^n\right) = \frac{p\left(\mathbf{x}_n \mid \boldsymbol{\theta}\right)p\left(\boldsymbol{\theta} \mid D^{n-1}\right)}{\int p\left(\mathbf{x}_n \mid \boldsymbol{\theta}\right)p\left(\boldsymbol{\theta} \mid D^{n-1}\right)d\boldsymbol{\theta}}$$

Ακόμη ισχυεί ότι:

$$p\left(x_n \mid \boldsymbol{\theta}\right) = \begin{cases} \boldsymbol{\theta} & \text{εάν Κεφάλι} \\ 1-\boldsymbol{\theta} & \text{εάν Γράμματα} \end{cases}$$

Άρα προκύπτει ότι:

$$p\left(\boldsymbol{\theta} \mid D^n\right) = \frac{\boldsymbol{\theta}^k(1-\boldsymbol{\theta})^{n-k}p\left(\boldsymbol{\theta} \mid D^0\right)}{\int \boldsymbol{\theta}^k(1-\boldsymbol{\theta})^{n-k}p\left(\boldsymbol{\theta} \mid D^0\right)d\boldsymbol{\theta}} = \frac{\boldsymbol{\theta}^k(1-\boldsymbol{\theta})^{n-k}\frac{\pi}{2}sin(\pi\boldsymbol{\theta})}{\int \boldsymbol{\theta}^k(1-\boldsymbol{\theta})^{n-k}\frac{\pi}{2}sin(\pi\boldsymbol{\theta})d\boldsymbol{\theta}}$$

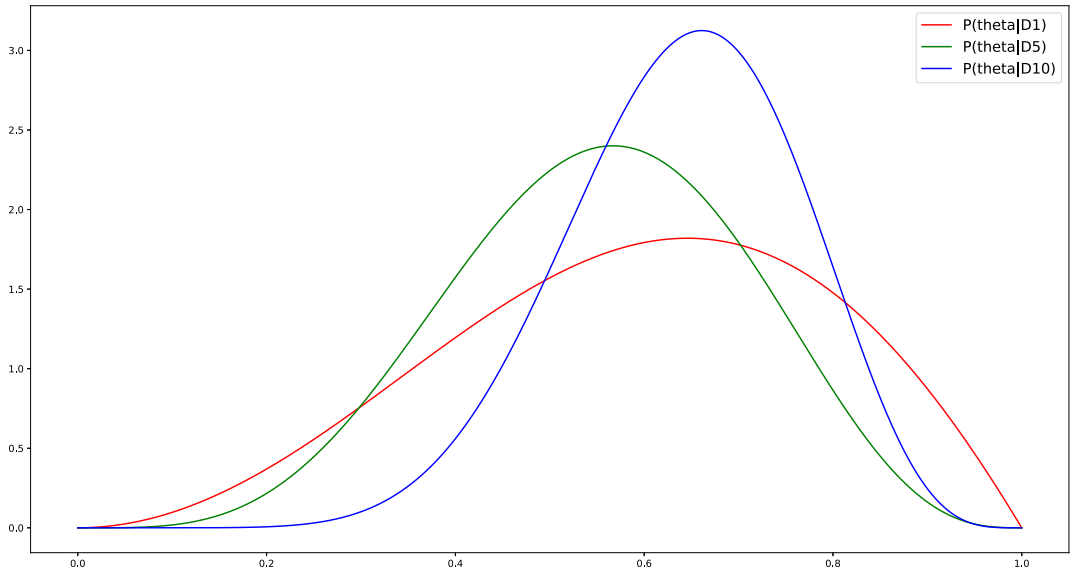
Ορίζουμε μια συνάρτηση αρχικά του θ και και κ όπου θα υπολογίζει τον αριθμητή ο οποίος όπως παρατηρούμε μπορούμε να τον χρησιμοποιήσουμε για τον υπολογισμό του παρανομαστή όπου είναι απλά το ολοκλήρωμα του αριθμητή.

```
def numerator(theta1, k):
    return (theta1 ** k)*(1-theta1)**(i-k)*A*p_theta_0(theta1)

d = np.array([1, 1, 0, 1, 0, 1, 1, 1, 0, 1])  # Results of flip
Num_Of_Points = 1000
y = []
for i in [1, 5, 10]:
    K = (d[0:i] == 1).sum()
    denominator = quad(numerator, 0, 1, args=K)[0]
    theta = np.linspace(0, 1, Num_Of_Points)
    y.append(numerator(theta, K)/denominator)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
plt.plot(theta, y[0], 'r')
plt.plot(theta, y[1], 'g')
plt.plot(theta, y[2], 'b')
plt.legend(["P(theta|D1)", "P(theta|D5)", "P(theta|D10)"], prop={'size': 15})
plt.show()
```

Τα αποτελέσματα είναι τα εξής:



Ερώτημα γ

Η τιμή $p(x = k|\boldsymbol{\theta})$ αντιστοιχεί στην μέγιστη τιμή της $p\left(\theta \mid D^{10}\right)$. Άρα την παίρνουμε από το αντίστοιχο διάνυσμα που ήδη έχει δημιουργηθεί για το προηγούμενο ερώτημα και την τυπώνουμε.

```
print("Mmax value of P(θ|D10) is:", max(y[2]), "for x ",(np.argmax(y[2]))/Num_Of_Points)
```

```
The max value of P(θ|D10) is: 3.1244687598953202 for x  0.66
```

Άσκηση 4

Ερώτημα α

Για την παραγωγή των τυχαίων δειγμάτων έφτιαξα τη παρακάτω συνάρτηση

```
def create_data(num_of_classes, size, priors_list, list_of_means, list_of_covariance_m):
    rng = np.random.default_rng(seed=0)
    Data = []
    available_num_of_samples = size
    for j in range(num_of_classes-1):
        classSize = math.floor(size*priors_list[j])
        Data.append(rng.multivariate_normal(list_of_means[j], list_of_covariance_m[j],
size=classSize))
        available_num_of_samples -= classSize
    # for last class we need to take the remaining samples cause for sum of p we might not
have round number of data
    Data.append(rng.multivariate_normal(list_of_means[-1], list_of_covariance_m[-1],
size=available_num_of_samples))
    return Data
```

Η συνάρτηση αυτη είναι γενικής χρήσης και παίρνει τα εξής ορίσματα:

- num_of_classes => Ο αριθμός των κλάσεων
- size => Ο αριθμός των δειγμάτων που θέλουμε να παράγουμε:
- priors_list => Λίστα με τις a priori πιθανότητες των κλάσεων.
- list_of_means => Λίστα με τους μέσους όρους για κάθε κλάση
- list_of_covariance_m => Λίστα με τους πίνακες συμμεταβλητότητας κάθε κλάσης.

Αξίζει να σημειωθεί ότι γίνεται μέριμνα ώστε στο τέλος ο αριθμός των δεδομένων που παράγονται να είναι ίσο με αυτό που θέλουμε(πολλές φορές λόγω των apriori μπορεί να μην προκύπτουν ακέραιοι αριθμοί η το συνολικό άθροισμα να μην ειναι το ζητούμενο πχ 1/3 * 1000 = 333 επι 3 φορές είναι 999 και όχι 1000.)

Στη συνέχεια ορίζουμε όλες αυτές τις παραμέτρους στο κώδικα και παράγουμε τα δεδομένα.

```
training_size = 10000
testing_size = 1000

prior_1 = prior_2 = prior_3 = 1/3
priors = [prior_1, prior_2, prior_3]

mean1 = np.array([0, 0, 0])
mean2 = np.array([1, 2, 2])
mean3 = np.array([3, 3, 4])
means = [mean1, mean2, mean3]

covariance_m1 = covariance_m2 = covariance_m3 = np.array([[0.8, 0.2, 0.1],
                                                          [0.2, 0.8, 0.2],
                                                          [0.1, 0.2, 0.8]])
covariance_ms = [covariance_m1, covariance_m2, covariance_m3]


training_data = create_data(3, training_size, priors, means, covariance_ms)
testing_data = create_data(3, testing_size, priors, means, covariance_ms)
```

Ερώτημα β

Στη συγκεκριμένη περίπτωση δοκιμάστηκαν 3 ταξινομητές:

- Euclidian Minimum Distance Classifier βασισμένος στην ευκλείδια απόσταση από την άσκηση 1
- Mahalanobis Minimum Distance Classifier βασισμένος στην απόσταση Mahalanobis από την άσκηση 1
- Bayesian Classifier βασισμένος στη συνάρτηση διάκρισης από την άσκηση 1.

Ακόμη, ζητείται να χρησιμοποιηθούν οι παραπάνω παράμετροι δηλαδή τα θεωρητικά στατιστικά των κατανομών. Για τους ταξινομητές ελάχιστης απόστασης(Euclidian και Mahalanobis) η λογική είναι η ίδια. Υπολογίζουμε τις αποστάσεις χρησιμοποιώντας τις αντίστιχες συναρτήσεις απο κάθε κλάση. Κάθε δείγμα ταξινομείται στη κλάση με τη μικρότερη απόσταση. Η μεθοδολογία που θα ακολουθηθεί είναι:

- Διατρέχω τα δεδομένα μιας κλάσης, έστω τις κλάσης 1
- Υπολογίζω τις αποστάσεις προς όλες τις κλάσεις.
- Αυξάνω τον αριθμό των σημειών που ταξινομήθηκαν λάθος κατα έαν, εάν η απόσταση από την κλάση την οποία ανήκει πχ της 1ης, δεν είναι η μικρότερη από τις άλλες 2 αποστάσεις.
- Αντίστοιχα δουλεύω και για τις υπόλοιπες κλάσεις.

Επειδή η χρήση των ταξινομητών θα γίνει πολλές φορές(και στα επόμενα ερωτήματα) τους υλοποίησα το κάθε έναν σε μία συνάρτηση.

```
def euclidian_classifier(test_data, test_size, list_of_means):
    # ---------- Euclidian Classifier ----------
    wrong_euclidian = 0
    # Class 1 Classification with Euclidian Classifier
    for i in range(test_data[0].shape[0]):
        # test of class_1 - euclidian distance
        distance_to_c1 = Ex1.euclidian_distance(test_data[0][i], list_of_means[0], 3)
        distance_to_c2 = Ex1.euclidian_distance(test_data[0][i], list_of_means[1], 3)
        distance_to_c3 = Ex1.euclidian_distance(test_data[0][i], list_of_means[2], 3)
        # A point is misclassified if distance from class_1 is not the smallest
        if distance_to_c1 != min([distance_to_c1, distance_to_c2, distance_to_c3]):
            wrong_euclidian += 1
    # Class 2 Classification with Euclidian Classifier
    for i in range(test_data[1].shape[0]):
        # test of class_2 - euclidian distance
        distance_to_c1 = Ex1.euclidian_distance(test_data[1][i], list_of_means[0], 3)
        distance_to_c2 = Ex1.euclidian_distance(test_data[1][i], list_of_means[1], 3)
        distance_to_c3 = Ex1.euclidian_distance(test_data[1][i], list_of_means[2], 3)
        # A point is misclassified if distance from class_2 is not the smallest
        if distance_to_c2 != min([distance_to_c1, distance_to_c2, distance_to_c3]):
            wrong_euclidian += 1
    # Class 3 Classification with Euclidian Classifier
    for i in range(test_data[2].shape[0]):
        # test of class_3 - euclidian distance
        distance_to_c1 = Ex1.euclidian_distance(test_data[2][i], list_of_means[0], 3)
        distance_to_c2 = Ex1.euclidian_distance(test_data[2][i], list_of_means[1], 3)
        distance_to_c3 = Ex1.euclidian_distance(test_data[2][i], list_of_means[2], 3)
        # A point is misclassified if distance from class_2 is not the smallest
        if distance_to_c3 != min([distance_to_c1, distance_to_c2, distance_to_c3]):
            wrong_euclidian += 1
    print("Euclidian Minimum Distance Classifier misclassified ", wrong_euclidian)
    print("The error is:", wrong_euclidian/test_size)
```

```
def mahalanobis_classifier(test_data, test_size, list_of_means, list_of_covariance_m):
    # ---------- Mahalanobis Classifier ----------
    wrong_Mahalanobis = 0
    # Class 1 Classification with Mahalanobis Classifier
    for i in range(test_data[0].shape[0]):
        # test of class_1 - Mahalanobis distance
        distance_to_c1 = Ex1.mahalanobis_distance(test_data[0][i], list_of_means[0],
list_of_covariance_m[0], 3)
        distance_to_c2 = Ex1.mahalanobis_distance(test_data[0][i], list_of_means[1],
list_of_covariance_m[1], 3)
        distance_to_c3 = Ex1.mahalanobis_distance(test_data[0][i], list_of_means[2],
list_of_covariance_m[2], 3)
        # A point is misclassified if distance from class_1 is not the smallest
        if distance_to_c1 != min([distance_to_c1, distance_to_c2, distance_to_c3]):
            wrong_Mahalanobis += 1
    # Class 2 Classification with Mahalanobis Classifier
    for i in range(test_data[1].shape[0]):
        # test of class_2 - Mahalanobis distance
        distance_to_c1 = Ex1.mahalanobis_distance(test_data[1][i], list_of_means[0],
list_of_covariance_m[0], 3)
        distance_to_c2 = Ex1.mahalanobis_distance(test_data[1][i], list_of_means[1],
list_of_covariance_m[1], 3)
        distance_to_c3 = Ex1.mahalanobis_distance(test_data[1][i], list_of_means[2],
list_of_covariance_m[2], 3)
        # A point is misclassified if distance from class_2 is not the smallest
        if distance_to_c2 != min([distance_to_c1, distance_to_c2, distance_to_c3]):
            wrong_Mahalanobis += 1
    # Class 3 Classification with Mahalanobis Classifier
    for i in range(test_data[2].shape[0]):
        # test of class_3 - Mahalanobis distance
        distance_to_c1 = Ex1.mahalanobis_distance(test_data[2][i], list_of_means[0],
list_of_covariance_m[0], 3)
        distance_to_c2 = Ex1.mahalanobis_distance(test_data[2][i], list_of_means[1],
list_of_covariance_m[1], 3)
        distance_to_c3 = Ex1.mahalanobis_distance(test_data[2][i], list_of_means[2],
list_of_covariance_m[2], 3)
        # A point is misclassified if distance from class_2 is not the smallest
        if distance_to_c3 != min([distance_to_c1, distance_to_c2, distance_to_c3]):
            wrong_Mahalanobis += 1
    print("Mahalanobis Minimum Distance Classifier misclassified ", wrong_Mahalanobis)
    print("The error is:", wrong_Mahalanobis/test_size)
```

Τέλος μένει ο ταξινομητής Bayes. Στη περίπτωση αυτή όμως χρησιμοποιούμε την συνάρτηση διάκρισης που υλοποιήθηκε στο ερώτημα ένα. Η λογική για την ταξινόμηση και τα λάθη ταξινόμηση είναι γνωστή, δηλαδή:

- Διατρέχω τα δεδομένα μιας κλάσης, έστω τις κλάσης 1
- Υπολογίζω τις συναρτήσεις διάκρισης για όλες τις κλάσεις
- Αυξάνω τον αριθμό των σημειών που ταξινομήθηκαν λάθος κατα έαν, εάν η τιμή της συνάρτησης διάκρισης για τη κλάση που ανηκει πχ της 1ης, δεν είναι η μεγαλύτερη από τις άλλες 2 τιμές των συναρτήσεων διάκρισης.
- Αντίστοιχα δουλεύω και για τις υπόλοιπες κλάσεις.

Στην επόμενη σελίδα παρουσιάζεται και ο κώδικας του Bayesian classifier.

```
def bayes_classifier(test_data, test_size, list_of_means, list_of_covariance_m,
priors_list):
    # ---------- Bayes Classifier ----------
    wrong_Bayes = 0
    # Class 1 Classification with Bayes Classifier
    for i in range(test_data[0].shape[0]):
        # test of class_1 - Bayes distance
        g1 = Ex1.discriminant_function(test_data[0][i], list_of_means[0],
list_of_covariance_m[0], priors_list[0], 3)
        g2 = Ex1.discriminant_function(test_data[0][i], list_of_means[1],
list_of_covariance_m[1], priors_list[1], 3)
        g3 = Ex1.discriminant_function(test_data[0][i], list_of_means[2],
list_of_covariance_m[2], priors_list[2], 3)
        # A point is misclassified if value from class_1 is not the biggest
        if g1 != max([g1, g2, g3]):
            wrong_Bayes += 1
    # Class 2 Classification with Bayes Classifier
    for i in range(test_data[1].shape[0]):
        # test of class_1 - Bayes distance
        g1 = Ex1.discriminant_function(test_data[1][i], list_of_means[0],
list_of_covariance_m[0], priors_list[0], 3)
        g2 = Ex1.discriminant_function(test_data[1][i], list_of_means[1],
list_of_covariance_m[1], priors_list[1], 3)
        g3 = Ex1.discriminant_function(test_data[1][i], list_of_means[2],
list_of_covariance_m[2], priors_list[2], 3)
        # A point is misclassified if value from class_2 is not the smallest
        if g2 != max([g1, g2, g3]):
            wrong_Bayes += 1
    # Class 3 Classification with Euclidian Classifier
    for i in range(test_data[2].shape[0]):
        # test of class_3 - Bayes distance
        g1 = Ex1.discriminant_function(test_data[2][i], list_of_means[0],
list_of_covariance_m[0], priors_list[0], 3)
        g2 = Ex1.discriminant_function(test_data[2][i], list_of_means[1],
list_of_covariance_m[1], priors_list[1], 3)
        g3 = Ex1.discriminant_function(test_data[2][i], list_of_means[2],
list_of_covariance_m[2], priors_list[2], 3)
        # A point is misclassified if value from class_2 is not the biggest
        if g3 != max([g1, g2, g3]):
            wrong_Bayes += 1
    print("Bayes Classifier misclassified ", wrong_Bayes)
    print("The error is:", wrong_Bayes/test_size)
```

Τώρα το μόνο που απέμεινε είναι να καλέσω τις συναρτήσεις με τα κατάλληλα δεδομένα.

```
print("\nClassification Errors results using real means and Covariance matrices \n")
euclidian_classifier(testing_data, testing_size, means)
mahalanobis_classifier(testing_data, testing_size, means, covariance_ms)
bayes_classifier(testing_data, testing_size, means, covariance_ms, priors)
```

Τα αποτελέσματα είναι τα εξής:

```
----- Exercise 3.4 b -----

Classification Errors results using real means and Covariance matrices

Euclidian Minimum Distance Classifier misclassified  99
The error is: 0.099
Mahalanobis Minimum Distance Classifier misclassified  102
The error is: 0.102
Bayes Classifier misclassified  102
The error is: 0.102
```

Όπως βλέπουμε τα αποτελέσματα είναι πολύ καλά και έχουν ελάχιστη διαφορά. Συγκεκριμένα ο Euclidian Classifier βρίσκεται στο 9.9% σφάλμα ενώ οι άλλοι 2 στο 10.2%. Τα αποτελέσματα είναι τα αναμενόμενα δλδ παρόμοια γενικά ίδια των Mahalanobis και Bayesian καθώς ισχύουν οι παρακάτω συνθήκες(εκτός τις τελευταίας που δεν ισχυεί απόλυτα)

1.4 MINIMUM DISTANCE CLASSIFIERS

1.4.1 The Euclidean Distance Classifier

The optimal Bayesian classifier is significantly simplified under the following assumptions:

- The classes are equiprobable.
- The data in *all* classes follow Gaussian distributions.
- The covariance matrix is the *same* for all classes.
- The covariance matrix is diagonal and *all* elements across the diagonal are *equal*. That is, $S = \sigma^2 I$, where I is the identity matrix.

Under these assumptions, it turns out that the optimal Bayesian classifier is equivalent to the minimum Euclidean distance classifier. That is, given an unknown x, assign it to class ω_i if

$$||x - m_i|| \equiv \sqrt{(x - m_i)^T (x - m_i)} < ||x - m_j||, \quad \forall i \neq j$$

It must be stated that the Euclidean classifier is often used, even if we know that the previously stated assumptions are not valid, because of its simplicity. It assigns a pattern to the class whose mean is closest to it with respect to the Euclidean norm.

1.4.2 The Mahalanobis Distance Classifier

If one relaxes the assumptions required by the Euclidean classifier and removes the last one, the one requiring the covariance matrix to be diagonal and with equal elements, the optimal Bayesian classifier becomes equivalent to the minimum Mahalanobis distance classifier. That is, given an unknown x, it is assigned to class ω_i if

$$\sqrt{(x - m_i)^T S^{-1} (x - m_i)} < \sqrt{(x - m_j)^T S^{-1} (x - m_j)}, \quad \forall j \neq i$$

where S is the common covariance matrix. The presence of the covariance matrix accounts for the shape of the Gaussians [Theo 09, Section 2.4.2].

Ειδικότερα όλες οι apriori πιθανότητες είναι ίδιες, τα δεδομενα προέρχονται απο κατανομές Gauss, ο πίνακας συμμεταβλητότητας είναι ίδιος για όλες τις κλάσεις. Ωστόσο στη περίπτωση μας δεν ισχυεί ότι $S = \sigma^2 I$ καθώς τα στοιχεία εκτός της διαγωνίου είναι μη μιδενικά. Σε περίπτωση που ίσχυε και αυτό περιμέναμε και οι 3 ταξινομητές να έχουν ιδιο αποτέλεσμα. Για το λόγο ότι δεν ισχύει η τελευταία συνθήκη, ίδιο αποτέλεσμα έχουν μόνο οι Mahalanobis και Bayes.

Ερώτημα γ

Στο συγκεκριμένο ερώτημα ζητείται να γίνει εκτίμηση παραμέτρων με χρήση με χρήση της τεχνικής Maximum Likelihood Estimation. Όπως είδαμε και παραμάνω ισχυεί:

$$\hat{\mu} = \frac{1}{n}\sum_{k=1}^{n} x_k \qquad \text{και} \qquad \hat{\sigma^2} = \frac{1}{n}\sum_{k=1}^{n}\left(x_k - \hat{\mu}\right)^2$$

Η συνάρτηση που υλοποίησα είναι η εξής:

```
def maximum_likelihood_estimation(train_data):
    # Estimate means and covariance matrices from our training data
    class_1_mean = np.mean(train_data[0], axis=0)
    class_2_mean = np.mean(train_data[1], axis=0)
    class_3_mean = np.mean(train_data[2], axis=0)
    class_1_cov = np.cov(train_data[0], rowvar=False)
    class_2_cov = np.cov(train_data[1], rowvar=False)
    class_3_cov = np.cov(train_data[2], rowvar=False)
    return [class_1_mean, class_2_mean, class_3_mean], [class_1_cov, class_2_cov,
class_3_cov]
```

Ειδικότερα παίρνει ως όρισμα τα δεδομένα εκπαίδευσης και επιστρέφει σε 2 λίστες τα στατιστικά κάθε κλάσης. Τώρα μένει να την καλέσουμε και στη συνέχεια να ξανα-υπολογίσουμε τα σφάλματα ταξινόμησης:

```
means_estimates, covariance_ms_estimate = maximum_likelihood_estimation(training_data)

print("\nClassification Errors results using estimated means and Covariance matrices  \n")
euclidian_classifier(testing_data, testing_size, means_estimates)
mahalanobis_classifier(testing_data, testing_size, means_estimates, covariance_ms_estimate)
bayes_classifier(testing_data, testing_size, means_estimates, covariance_ms_estimate,
priors)
```

Τα αποτελέσματα είναι τα εξής:

```
----- Exercise 3.4 c -----

Classification Errors results using estimated means and Covariance matrices

Euclidian Minimum Distance Classifier misclassified  101
The error is: 0.101
Mahalanobis Minimum Distance Classifier misclassified  99
The error is: 0.099
Bayes Classifier misclassified  99
The error is: 0.099
```

Όπως βλέπουμε τα αποτελέσματα είναι σχεδόν ίδια με πρίν (διαφορές τις τάξης 0.2%). Ακόμη, παρατηρούμε και πάλι πως οι ταξινομητές Mahalanobis και Bayes παρουσιάζουν ιδια αποτελέσματα. Γενικά, τα αποτελέσματα είναι σωστά και λογικά καθώς δεδομένα εκπαίδευσης ακολουθούν κανονική κατανομή και είναι 10 φορές περισσότερα από τα δεδομένα ελέγχου με αποτέλεσμα οι εκτίμηση των στατιστικών να είναι πολύ καλή.

Ερώτημα 4

Στο συγκεκριμένω ερώτημα δίνονται καινούργια θεωρητικά στατιστικά για τις κατανομές των δεδομένων εκπαίδευσης και ζητείται να επαναλάβουμε τα πειράματα, δηλαδή αρχικά ταξινόμηση με χρήση των θεωρητικών στατιστικών και στη συνέχεια με εκτίμηση παραμέτρων. Αρχικά λοιπόν ορίζουμε τα στατιστικά των κατανομών και παράγουμε τα δεδομένα μας.

```
prior_1, prior_2, prior_3 = 1/6, 1/6, 2/3
priors = [prior_1, prior_2, prior_3]

mean1 = np.array([0, 0, 0])
mean2 = np.array([1, 2, 2])
mean3 = np.array([3, 3, 4])
means = [mean1, mean2, mean3]

covariance_m1 = np.array([[0.8, 0.2, 0.1],
                          [0.2, 0.8, 0.2],
                          [0.1, 0.2, 0.8]])
covariance_m2 = np.array([[0.6, 0.2, 0.01],
                          [0.2, 0.8, 0.01],
                          [0.01, 0.01, 0.6]])
covariance_m3 = np.array([[0.6, 0.1, 0.1],
                          [0.1, 0.6, 0.1],
                          [0.1, 0.1, 0.6]])

covariance_ms = [covariance_m1, covariance_m2, covariance_m3]

training_data = create_data(3, training_size, priors, means, covariance_ms)
testing_data = create_data(3, testing_size, priors, means, covariance_ms)
```

Στη συνέχεια αυτό που απομένει είναι αρχικά ο υπολογισμός του σφάλματος για τους 3 ταξινομητές με χρήση των θεωρητικών μεσων τιμών και πινάκων συνδιακύμανσης και στη συνέχεια με χρήση αυτών που προκύπτουν από τη εκτίμηση παραμέτρων με μέθοδο μέγιστης πιθανοφάνειας.

```
print("\nClassification Errors results using real means and Covariance matrices \n")
euclidian_classifier(testing_data, testing_size, means)
mahalanobis_classifier(testing_data, testing_size, means, covariance_ms)
bayes_classifier(testing_data, testing_size, means, covariance_ms, priors)

means_estimates, covariance_ms_estimate = maximum_likelihood_estimation(training_data)

print("\nClassification Errors results using estimated means and Covariance matrices  \n")
euclidian_classifier(testing_data, testing_size, means_estimates)
mahalanobis_classifier(testing_data, testing_size, means_estimates, covariance_ms_estimate)
bayes_classifier(testing_data, testing_size, means_estimates, covariance_ms_estimate,
priors)
```

Τα αποτελέσματα παρουσιάζονται και σχολιάζονται στην επόμενη σελίδα.

Τα αποτελέσματα είναι τα εξής:

```
Classification Errors results using real means and Covariance matrices

Euclidian Minimum Distance Classifier misclassified  49
The error is: 0.049
Mahalanobis Minimum Distance Classifier misclassified  43
The error is: 0.043
Bayes Classifier misclassified  37
The error is: 0.037

Classification Errors results using estimated means and Covariance matrices

Euclidian Minimum Distance Classifier misclassified  47
The error is: 0.047
Mahalanobis Minimum Distance Classifier misclassified  45
The error is: 0.045
Bayes Classifier misclassified  38
The error is: 0.038
```

Όπως παρατηρούμε, και πάλι δεν παίζει πολυ ρόλο αν χρησιμοποιούμε τα πραγματικά/θεωρητικά στατιστικά η αυτά που προέρχονται απο την εκτίμηση καθώς και στις 2 περιπτώσεις τα αποτελέσματα είναι πολύ κοντα. Ακόμη, παρατηρούμε ότι και στη περίπτωση των θεωρητικών και στη περίπτωση όπου χρησιμοποιούμε τις εκτιμήσεις των στατιστικών, ο ταξινομητής Bayes δίνει τα καλύτερα αποτελέσματα, ακολουθεί ο ταξινομητής Mahalanobis και τέλος έρχεται ο Euclidian ταξινομητής. Αυτό δικαιολογείται απο το γεγονός ότι:

- Ο Euclidian ταξινομητής δεν λαμβάνει υπ όψην τους πίνακες συνδιακύμανσης
- Ο Mahalanobis ταξινομητής δεν λαμβάνει υπ όψην τις a priori πιθανοτητες
- Ο Bayesian ταξινομητής λαμβάνει υπ όψην όλα τα παραπάνω

Συνεπώς η κατάταξη αυτή των αποτελεσμάτων είναι λογική.