1η Εργασία
Αναγνώριση Προτύπων
Μπούρχα Ιωάννα 58019

Επιβλέπων καθηγητής: Ηλίας Θεοδωρακόπουλος

Ακαδημαϊκό έτος: 2023 - 2024

ΠΕΡΙΕΧΟΜΕΝΑ

ΑΣΚΗΣΗ 1

Ερώτημα Α:
Υπολογίστε την απόφαση που θα πάρει το σύστημα για κάθε πιθανή τιμή D με βάση τον κανόνα του Bayes.

Εφόσον ένα email μπορεί να βρίσκεται μονάχα σε μία κατηγορία Normal, Malicious, Spam, το άθροισμα των a priori πιθανοτήτων αυτών τω κατηγοριών πρέπει να ισούται με μονάδα. Έτσι, υπολογίζω την a priori πιθανότητα της κλάσης Normal.

$$P_{Normal} = 1 - P_{Malicius} - P_{Spam} = 1 - 0.1 - 0.3 = 0.6$$

Προκειμένου να κατατάξω κάθε email D σε μία από τις τρεις κατηγορίες εφαρμόζω τον κανόνα απόφασης Bayes. Σύμφωνα με αυτόν τον κανόνα, κάθε email τοποθετείται στην κατηγορία για την οποία παρουσιάζει μεγαλύτερη εκ των υστέρων (posterior) πιθανότητα. Ο τύπος υπολογισμού για την εκ των υστέρων πιθανότητα σύμφωνα με τις διαφάνειες του μαθήματος είναι:

$$P(\omega_j|x) = \frac{p(x|\omega_j)\,P(\omega_j)}{\sum_i^N p(x|\omega_i)\,P(\omega_i)}$$

Το email τελικώς κατατάσσεται στην κατηγορία για την οποία παρουσίασε την μέγιστη εκ των υστέρων πιθανότητα.

⇨ Για D = 1

$$P(\omega_{Normal}|D1) = \frac{p(D1|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.5 * 0.6}{0.5 * 0.6 + 0.05 * 0.3 + 0.02 * 0.1} = \frac{0.3}{0.317} = 0.9467$$

$$P(\omega_{Spam}|D1) = \frac{p(D1|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.05 * 0.3}{0.5 * 0.6 + 0.05 * 0.3 + 0.02 * 0.1} = \frac{0.015}{0.317} = 0.0473$$

$$P(\omega_{Malicious}|D1) = \frac{p(D1|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.02 * 0.1}{0.5 * 0.6 + 0.05 * 0.3 + 0.02 * 0.1} = \frac{0.002}{0.317} = 0.0063$$

Επομένως, το email D1 ταξινομείται στην κλάση Normal.

⇨ Για D = 2

$$P(\omega_{Normal}|D2) = \frac{p(D2|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.23 * 0.6}{0.23 * 0.6 + 0.15 * 0.3 + 0.13 * 0.1} = \frac{0.138}{0.196} = 0.7041$$

$$P(\omega_{Spam}|D2) = \frac{p(D2|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.15 * 0.3}{0.23 * 0.6 + 0.15 * 0.3 + 0.13 * 0.1} = \frac{0.045}{0.196} = 0.2296$$

$$P(\omega_{Malicious}|D2) = \frac{p(D2|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.13 * 0.1}{0.23 * 0.6 + 0.15 * 0.3 + 0.13 * 0.1} = \frac{0.013}{0.196} = 0.0063$$

Επομένως, το email D2 ταξινομείται στην κλάση Normal.

⇨ Για D = 3

$$P(\omega_{Normal}|D3) = \frac{p(D3|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.16 * 0.6}{0.16 * 0.6 + 0.4 * 0.3 + 0.15 * 0.1} = \frac{0.096}{0.231} = 0.4156$$

$$P(\omega_{Spam}|D3) = \frac{p(D3|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.4 * 0.3}{0.16 * 0.6 + 0.4 * 0.3 + 0.15 * 0.1} = \frac{0.12}{0.231} = 0.5195$$

$$P(\omega_{Malicious}|D3) = \frac{p(D3|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.15 * 0.1}{0.16 * 0.6 + 0.4 * 0.3 + 0.15 * 0.1} = \frac{0.015}{0.231} = 0.0649$$

Επομένως, το email D3 ταξινομείται στην κλάση Spam.

⇨ Για D = 4

$$P(\omega_{Normal}|D4) = \frac{p(D4|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.1 * 0.6}{0.1 * 0.6 + 0.3 * 0.3 + 0.3 * 0.1} = \frac{0.06}{0.18} = 0.3333$$

$$P(\omega_{Spam}|D4) = \frac{p(D4|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.3 * 0.3}{0.1 * 0.6 + 0.3 * 0.3 + 0.3 * 0.1} = \frac{0.09}{0.18} = 0.5$$

$$P(\omega_{Malicious}|D4) = \frac{p(D4|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.3 * 0.1}{0.1 * 0.6 + 0.3 * 0.3 + 0.3 * 0.1} = \frac{0.03}{0.18} = 0.1667$$

Επομένως, το email D4 ταξινομείται στην κλάση Spam.

⇨ Για D = 5

$$P(\omega_{Normal}|D5) = \frac{p(D5|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.01 * 0.6}{0.01 * 0.6 + 0.1 * 0.3 + 0.4 * 0.1} = \frac{0.006}{0.076} = 0.0789$$

$$P(\omega_{Spam}|D5) = \frac{p(D5|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.1 * 0.3}{0.01 * 0.6 + 0.1 * 0.3 + 0.4 * 0.1} = \frac{0.03}{0.076} = 0.3947$$

$$P(\omega_{Malicious}|D5) = \frac{p(D5|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.4 * 0.1}{0.01 * 0.6 + 0.1 * 0.3 + 0.4 * 0.1} = \frac{0.04}{0.076} = 0.5263$$

Επομένως, το email D5 ταξινομείται στην κλάση Malicious.

Ο κώδικας που υλοποιεί την αντίστοιχη ταξινόμηση ο ακόλουθος. Αρχικά, δηλώνω τα δεδομένα μου, δηλαδή τις a priori πιθανότητες κάθε κλάσης και την κατανομή της επικινδυνότητας για κάθε email.

```
D = [[0.5, 0.05, 0.02],
     [0.23, 0.15, 0.13],
     [0.16, 0.4, 0.15],
     [0.1, 0.3, 0.3],
     [0.01, 0.1, 0.4]
    ]

ps = 0.3
pm = 0.1
pn = 1 - ps - pm  # το άθροισμα των πιθανοτήτων είναι 1
P = [pn, ps, pm]
```

Για κάθε email D υπολογίζω το evidence (παρονομαστής) και τους αριθμητές. Προκειμένου να μην υπολογίζω για κάθε κατηγορία ξεχωριστά το evidence, καθώς αυτό αποτελεί το άθροισμα των αριθμητών για όλες τις κατηγορίες, χρησιμοποιώ μία δομή επανάληψης για τον υπολογισμό τόσο του evidence όσο και των αριθμητών. Οι τιμές των αριθμητών αποθηκεύονται σε μία λίστα, τα περιεχόμενα της οποίας στην συνέχεια διαιρούνται με το evidence.

```
classes = []
for d in D:
    evidence = 0
    posteriors = []
    numberators = []

    # Calculating posteriors of each d
```

```
    for i in range(0, len(d)):
        evidence = evidence + d[i]*P[i]
        numberators.append(d[i]*P[i])

    for val in numberators:
        posteriors.append(val/evidence)

    # Determining the class of each d according to the maximum posterior
    val = max(posteriors)
    val_index = list.index(posteriors, val)
    classes.append(val_index)
    # 0 --> Normal      1 --> Spam      2 --> Malicious
```

Στην συνέχεια, αποτυπώνω το αποτέλεσμα, δηλαδή σε ποια κλάση ανήκει κάθε email.

```
# Printing the results
counter = 1
for c in classes:
    match c:
        case 0:
            print("D" + str(counter) + " email belongs to category NORMAL.")
        case 1:
            print("D" + str(counter) + " email belongs to category SPAM.")
        case 2:
            print("D" + str(counter) + " email belongs to category MALICIUS.")

    counter += 1
```

```
D1 email belongs to category NORMAL.
D2 email belongs to category NORMAL.
D3 email belongs to category SPAM.
D4 email belongs to category SPAM.
D5 email belongs to category MALICIUS.
```

Τα αποτελέσματα του κώδικα συμβαδίζουν με αυτά στα οποία κατέληξα και εγώ προηγουμένως. Όπως ακριβώς αναμενόταν.

Ερώτημα Β
Να υπολογιστεί το ολικό σφάλμα ταξινόμησης.

Το λάθος ταξινόμησης αντιστοιχεί στην πιθανότητα να μην κατατάξω κάποιο email στην κατηγορία στην οποία ανήκει. Άρα,:

$$P_{error_D} = evidence * \left(1 - \max\left(P(\omega_j|D)\right)\right)$$

Το ολικό σφάλμα προκύπτει από το άθροισμα της πιθανότητα λάθους για όλα τα emails.

Το ολικό σφάλμα ταξινόμησης ορίζεται ως:

$$P_{error_{total}} = \sum_{D=1}^{N=5} P_{error_D} = \sum_{j}^{N} evidence_j * \left(1 - \max\left(P(\omega|Dj)\right)\right)$$

Στο συγκεκριμένο πρόβλημα έχω:

$$P_{error_{total}} = 0.316(1 - 0.9467) + 0.196(1 - 0.7041) + 0.231(1 - 0.5195) + 0.18(1 - 0.5) + 0.076(1 - 0.5263) = 0.312$$

Αναφορικά με την υλοποίηση του αντίστοιχου υπολογισμού σε κώδικα, είναι φανερό από τον τύπο πως εξυπηρετεί να υπολογίζω το σφάλμα για κάθε D ξεχωριστά και στο τέλος να τα προσθέσω. Συνεπώς, το δεύτερο τμήμα κώδικα παίρνει την ακόλουθη μορφή:

```
classes = []
error = []
```

```
for d in D:
    evidence = 0
    posteriors = []
    numberators = []

    # Calculating posteriors of each d
    for i in range(0, len(d)):
        evidence = evidence + d[i]*P[i]
        numberators.append(d[i]*P[i])

    for val in numberators:
        posteriors.append(val/evidence)

    # Determining the class of each d according to the maximum posterior
    val = max(posteriors)
    val_index = list.index(posteriors, val)
    classes.append(val_index)
    # 0 --> Normal      1 --> Spam      2 --> Malicious

    # Calculating the error
    error.append(evidence*(1-val))
```

Οι διαφορές εντοπίζονται στην ύπαρξη της λίστας error και στην τελευταία εντολή όπου υπολογίζεται και αποθηκεύεται στην προαναφερόμενη λίστα το πιθανό σφάλμα ταξινόμησης για κάθε email. Η εμφάνιση του αποτελέσματος γίνεται με την εντολή

```
print("Total error is: " + str(sum(error)))
```

το οποίο είναι:

```
Total error is: 0.31200000000000006
```

Ερώτημα Γ:
Να υπολογιστεί το ολικό σφάλμα ταξινόμησης στην περίπτωση που δεν γνωρίζατε τίποτα για τις a priori πιθανότητες των κλάσεων.

Εφόσον δεν γνωρίζω τις a priori πιθανότητες των κλάσεων, θεωρώ ότι αυτές είναι ισοπίθανες, δηλαδή η a priori πιθανότητα έκαστης είναι $^1/_3$. Με αυτήν την υπόθεση επαναλαμβάνω την προαναφερόμενη διαδικασία.

⇨ Για D = 1

$$P(\omega_{Normal}|D1) = \frac{p(D1|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.5 * {}^1/_3}{0.5 * {}^1/_3 + 0.05 * {}^1/_3 + 0.02 * {}^1/_3} = \frac{0.1667}{0.19} = 0.8772$$

$$P(\omega_{Spam}|D1) = \frac{p(D1|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.05 * {}^1/_3}{0.5 * {}^1/_3 + 0.05 * {}^1/_3 + 0.02 * {}^1/_3} = \frac{0.0167}{0.19} = 0.0872$$

$$P(\omega_{Malicious}|D1) = \frac{p(D1|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.02 * {}^1/_3}{0.5 * {}^1/_3 + 0.05 * {}^1/_3 + 0.02 * {}^1/_3} = \frac{0.0067}{0.19} = 0.0351$$

Επομένως, το email D1 ταξινομείται στην κλάση Normal.

⇨ Για D = 2

$$P(\omega_{Normal}|D2) = \frac{p(D2|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.23 * {}^1/_3}{0.23 * {}^1/_3 + 0.15 * {}^1/_3 + 0.13 * {}^1/_3} = \frac{0.0767}{0.17} = 0.451$$

$$P(\omega_{Spam}|D2) = \frac{p(D2|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.15 * {}^1/_3}{0.23 * {}^1/_3 + 0.15 * {}^1/_3 + 0.13 * {}^1/_3} = \frac{0.049}{0.17} = 0.2941$$

$$P(\omega_{Malicious}|D2) = \frac{p(D2|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.13 * {}^1/_3}{0.23 * {}^1/_3 + 0.15 * {}^1/_3 + 0.13 * {}^1/_3} = \frac{0.043}{0.17} = 0.2549$$

Επομένως, το email D2 ταξινομείται στην κλάση Normal.

⇨ Για D = 3

$$P(\omega_{Normal}|D3) = \frac{p(D3|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.16 * {}^1/_3}{0.16 * {}^1/_3 + 0.4 * {}^1/_3 + 0.15 * {}^1/_3} = \frac{0.053}{0.236} = 0.2254$$

$$P(\omega_{Spam}|D3) = \frac{p(D3|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.4 * {}^1/_3}{0.16 * {}^1/_3 + 0.4 * {}^1/_3 + 0.15 * {}^1/_3} = \frac{0.13}{0.236} = 0.5634$$

$$P(\omega_{Malicious}|D3) = \frac{p(D3|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.15 * {}^1/_3}{0.16 * {}^1/_3 + 0.4 * {}^1/_3 + 0.15 * {}^1/_3} = \frac{0.05}{0.236} = 0.2113$$

Επομένως, το email D3 ταξινομείται στην κλάση Spam.

⇨ Για D = 4

$$P(\omega_{Normal}|D4) = \frac{p(D4|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.1 * {}^1/_3}{0.1 * {}^1/_3 + 0.3 * {}^1/_3 + 0.3 * {}^1/_3} = \frac{0.033}{0.233} = 0.1429$$

$$P(\omega_{Spam}|D4) = \frac{p(D4|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.3 * {}^1/_3}{0.1 * {}^1/_3 + 0.3 * {}^1/_3 + 0.3 * {}^1/_3} = \frac{0.1}{0.233} = 0.4286$$

$$P(\omega_{Malicious}|D4) = \frac{p(D4|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.3 * {}^1/_3}{0.1 * {}^1/_3 + 0.3 * {}^1/_3 + 0.3 * {}^1/_3} = \frac{0.1}{0.233} = 0.4286$$

Παρατηρώ ότι η πιθανότητα για την κατηγορία Spam και Malicious είναι ίδιες οπότε αυθαίρετα αποφασίζω να το κατατάξω στην κατηγορία Spam καθώς αυτή εξετάστηκε πρώτη, θεωρώντας ότι το σύστημά μου δεν είναι ιδιαίτερα αυστηρό. Το πρόβλημα αυτό εντοπίζεται διότι οι κατηγορίες θεωρήθηκαν ισοπίθανες, καθώς δεν γνωρίζω την a priori πιθανότητα των κλάσεων, και ότι η εκτιμηθείσα επικινδυνότητα για τις δύο κλάσεις είναι ίδιες.

⇨ Για D = 5

$$P(\omega_{Normal}|D5) = \frac{p(D5|\omega_{Normal})\,P(\omega_{Normal})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.01 * {}^1/_3}{0.01 * {}^1/_3 + 0.1 * {}^1/_3 + 0.4 * {}^1/_3} = \frac{0.003}{0.7} = 0.0196$$

$$P(\omega_{Spam}|D5) = \frac{p(D5|\omega_{Spam})\,P(\omega_{Spam})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.1 * {}^1/_3}{0.01 * {}^1/_3 + 0.1 * {}^1/_3 + 0.4 * {}^1/_3} = \frac{0.033}{0.17} = 0.1967$$

$$P(\omega_{Malicious}|D5) = \frac{p(D5|\omega_{Malicious})\,P(\omega_{Malicious})}{\sum_i^N p(D1|\omega_i)\,P(\omega_i)} = \frac{0.4 * {}^1/_3}{0.01 * {}^1/_3 + 0.1 * {}^1/_3 + 0.4 * {}^1/_3} = \frac{0.13}{0.17} = 0.7843$$

Επομένως, το email D5 ταξινομείται στην κλάση Malicious.

Παρατηρώ ότι και πάλι τα email κατατάσσονται στην ίδια κατηγορία, παρόλο που άλλαξαν οι a priori πιθανότητες των κλάσεών μου. Αυτό οφείλεται στο γεγονός ότι εφόσον οι καταστάσεις της φύσης είναι ισοπίθανες, η απόφαση βασίζεται εξ ολοκλήρου στην posterior πιθανότητα.

Γενικά, τόσο οι a priori όσο και οι posteriors πιθανότητες είναι σημαντικοί παράγοντες για την λήψη μίας απόφασης. Ο κανόνας απόφασης κατά Bayes τους συνδυάζει αποσκοπώντας στην ελάχιστη πιθανότητα σφάλματος.

Ακολουθώντας την ίδια μεθοδολογία του προηγούμενου ερωτήματος, το ολικό σφάλμα ταξινόμησης είναι:

$$P_{error_{total}} = 0{,}19(1-0{,}8772) + 0{,}17(1-0{,}451) + 0{,}236(1-0{,}5634) + 0{,}233(1-0{,}4286) + 0{,}17(1-0{,}7843) = 0.39$$

Ο κώδικας που υλοποιεί τις παραπάνω πράξεις είναι ο ίδιος με εκείνον των ερωτημάτων Α και Β με διαφορετικές τιμές στον πίνακα p των a priori πιθανοτήτων.

```
classes = []
error_c = []

for d in D:
    evidence = 0
    posteriors = []
    numberators = []

    # Calculating posteriors of each d
    for i in range(0, len(d)):
        evidence = evidence + d[i]*P[i]
        numberators.append(d[i]*P[i])   # γινόμενο της πιθανοφάνειας με την a priori πιθανότητα

    for val in numberators:
        posteriors.append(val/evidence)

    # Determining the class of each d according to the index of the maximum posterior
    classes.append(list.index(posteriors, max(posteriors)))
    # 0 --> Normal    1 --> Spam    2 --> Malicious

    # Calculating the error
    error_c.append(evidence*(1-max(posteriors)))


# Printing the results    !!! python version >= 3.8.10 !!!
counter = 1
for c in classes:
    match c:
        case 0:
            print("D" + str(counter) + " email belongs to category NORMAL.")
        case 1:
            print("D" + str(counter) + " email belongs to category SPAM.")
        case 2:
            print("D" + str(counter) + " email belongs to category MALICIUS.")
    counter += 1

# Calculating the total error
print("\nNew total error is: " + str(sum(error_c)))
```

Παρατηρώ ότι τα δύο σφάλματα των ερωτημάτων Β και Γ διαφέρουν κατά 0,078 με το σφάλμα στο ερώτημα Γ να είναι μεγαλύτερο. Το αποτέλεσμα αυτό είναι λογικό και αναμενόμενο καθώς δεν γνωρίζω έναν από τους παράγοντες που συμμετέχουν στην λήψη απόφασης κατά Bayes (a priori πιθανότητα).

ΑΣΚΗΣΗ 2

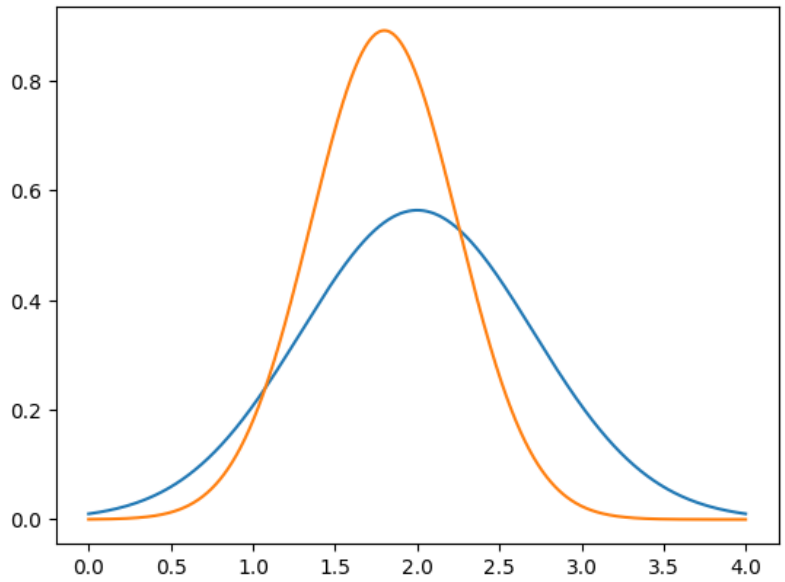
Για ένα πρόβλημα ταξινόμησης δύο κλάσεων ω₁ και ω₂ όπου $p(x|\omega_1) = N(2, 0.5)$ ***και*** $p(x|\omega_2) = N(1.8, 0.2)$ ***με εκ των προτέρων πιθανότητα*** $P(\omega_1) = \frac{1}{4}$ ***όπου το κόστος έχει οριστεί ως*** $\lambda = \begin{bmatrix} 0 & 1 \\ 3 & 0 \end{bmatrix}$ ***να βρεθεί η βέλτιστη λύση (κανόνας απόφασης) και να υπολογιστεί το ολικό κόστος.***

Να προσομοιωθεί η διαδικασία υπολογιστικά δημιουργώντας τυχαία δείγματα που ακολουθούν την κανονική κατανομή.

Σύμφωνα με τον τύπο της Γκαουσσιανής κατανομής $N(\mu, \sigma^2) = p(x) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}$, οι πυκνότητες πιθανότας για τις δύο κλάσεις είναι:

Κλάση ω₁	N(2 , 0.5)	$N(\mu, \sigma^2) = p(x\|\omega_1) = \frac{e^{-\frac{(x-2)^2}{2\ 0.5}}}{\sqrt{2\pi\ 0.5}} = \frac{e^{-(x-2)^2}}{\sqrt{\pi}}$
Κλάση ω₂	N(1.8 , 0,2)	$N(\mu, \sigma^2) = p(x\|\omega_2) = \frac{e^{-\frac{(x-1.8)^2}{2\ 0.2}}}{\sqrt{2\pi\ 0.2}} = \frac{e^{-\frac{(x-1.8)^2}{0.4}}}{\sqrt{0.4\pi}}$

Η γραφική παράσταση των παραπάνω κατανομών και ο αντίστοιχος κώδικας είναι:



Εικόνα 1: Γραφική αναπαράσταση των Γκαουσσιανών κατανομών N(2, 0.5) με το μπλε χρώμα και N(1.8,0.2) με το πορτοκαλί χρώμα.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from math import sqrt

global start
global step
global end

# Plot between 0 and 4 with .001 step
start = 0
step = 0.001
end = 4
x_axis = np.arange(start, end, step)

series_parameters = [[2, sqrt(0.5)],   # p(x|ω1)
                     [1.8, sqrt(0.2)] # p(x|ω2)
                    ]
```

```
px1 = norm.pdf(x_axis,series_parameters[0][0], series_parameters[0][1])
px2 = norm.pdf(x_axis,series_parameters[1][0], series_parameters[1][1])

plt.plot(x_axis, px1)
plt.plot(x_axis, px2)
plt.show()
```

Η απόφαση ταξινόμησης κάθε x στην κλάση ω_1 ή ω_2 προκύπτει σύμφωνα με τον κανόνα του Bayes.

$$P(\omega_j|x) = \frac{p(x|\omega_j)\ \ P(\omega_j)}{\sum_i p(x|\omega_i)\ \ P(\omega_i)} = \frac{p(x|\omega_j)\ \ P(\omega_j)}{p(x)}$$

Ο τύπος $\lambda = \begin{bmatrix}\lambda_{11} & \lambda_{12}\\ \lambda_{21} & \lambda_{22}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 3 & 0\end{bmatrix}$ όπου $\lambda_{ij} = \lambda(a_1|\omega_j)$ ονομάζεται συνάρτηση απωλειών και περιγράφει την απώλεια που προκύπτει όταν επιλέγουμε την ενέργεια α_i ενώ η κατάσταση της φύσης είναι ω_j. Εφόσον $P(\omega_j|x)$ η πιθανότητα η σωστή κατάσταση της φύσης να είναι η ω_j, η αναμενόμενη απώλεια, ή αλλιώς δεσμευμένο ρίσκο, στην περίπτωση που επιλέξουμε την ενέργεια α_i είναι:

$$R(a_i|x) = \sum_{j=1}^{c} \lambda(a_\iota|\omega_j)\ P(\omega_j|x)$$

Επομένως, για την βέλτιστη λύση, θα πρέπει να ελαχιστοποιήσω την αναμενόμενη απώλεια. Αυτό επιτυγχάνεται επιλέγοντας την ενέργεια που αποφέρει το μικρότερο δεσμευμένο ρίσκο.

Έστω α_1 η ενέργεια που αντιστοιχεί στην απόφαση ότι η σωστή κατάσταση της φύσης είναι η ω_1 και η ενέργεια α_2 η ενέργεια που αντιστοιχεί στην απόφαση ότι η σωστή κατάσταση είναι η ω_2. Με βάση την προηγούμενη εξίσωση, το δεσμευμένο ρίσκο για κάθε επιλογή είναι:

$$R(a_1|x) = \lambda_{11}\ P(\omega_1|x) + \lambda_{12}\ P(\omega_2|x)$$
$$R(a_2|x) = \lambda_{21}\ P(\omega_1|x) + \lambda_{22}\ P(\omega_2|x)$$

Ο θεμελιώδης κανόνας που εκφράζει τον κανόνα απόφασης ελαχίστου ρίσκου είναι πως επιλέγω την κατάσταση ω_1 αν $R(a_1|x) < R(a_2|x)$. Διαφορετικά, επιλέγω την κατάσταση ω_2.

$$R(a_1|x)R(a_1|x) < R(a_2|x)R(a_2|x)$$
$$\Leftrightarrow (\lambda_{21} - \lambda_{11})\ P(\omega_1|x) > (\lambda_{12} - \lambda_{22})\ P(\omega_2|x)$$
$$\Leftrightarrow (\lambda_{21} - \lambda_{11})\ p(x|\omega_1)\ P(\omega_1) > (\lambda_{12} - \lambda_{22})\ p(x|\omega_2)\ P(\omega_2)$$
$$\Leftrightarrow \frac{p(x|\omega_1)}{p(x|\omega_2)} > \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}} \frac{P(\omega_2)}{P(\omega_1)}$$

Δεδομένου ότι έχω μονάχα δύο κλάσεις, το άθροισμα των a priori πιθανοτήτων τους ισούται με την μονάδα. Εφόσον, $P(\omega_1) = \frac{1}{4}$, εύκολα προκύπτει ότι $P(\omega_2) = \frac{3}{4}$. Αντικαθιστώντας και τα υπόλοιπα μεγέθη με τις αριθμητικές τους τιμές προκύπτει ότι:

$$\frac{\frac{e^{-(x-2)^2}}{\sqrt{\pi}}}{\frac{e^{-\frac{(x-1.8)^2}{0.4}}}{\sqrt{0.4\pi}}} > \frac{1}{3}\frac{3/4}{1/4} \Leftrightarrow \frac{e^{-(x-2)^2}\sqrt{0.4}}{e^{-\frac{(x-1.8)^2}{0.4}}} > 1 \Leftrightarrow \ln\frac{e^{-(x-2)^2}\sqrt{0.4}}{e^{-\frac{(x-1.8)^2}{0.4}}} > ln(1) \Leftrightarrow 1.5x^2 - 5x + 4.1 + 0.5 + \ln\frac{2}{5} > 0$$

$$\Leftrightarrow 1.5x^2 - 5x + 3.64 > 0$$

Το τριώνυμο $1.5x^2 - 5x + 3.64$ έχει ρίζες $\rho_1 = 1{,}76$ και $\rho_2 = 2{,}258$. Δεδομένου ότι διατηρεί το πρόσημο σε κάθε διάστημα, με δοκιμή προκύπτει ότι:

$-\infty$	$\rho_1 = 1{,}076$	$\rho_2 = 2{,}258$	$+\infty$
+	-	+	

Συνεπώς, στην κατηγορία ω_1 ανήκουν τα x που βρίσκονται στο διάστημα $(-\infty, 1.076) \cup (2.258, -\infty)$, ενώ τα υπόλοιπα ανήκουν στην κατηγορία ω_2. Για τις οριακές περιπτώσεις όπου τα x ταυτίζονται με τις τιμές των

ριζών, όπως έχει οριστεί αυθαίρετα από τον κανόνα, αυτά κατατάσσονται στην κατηγορία ω_2. Τα προαναφερόμενα σύνολα προσδιορίζονται αντίστοιχα ως R1 και R2 αντίστοιχα και φαίνονται στην εικόνα 2.

Η υλοποίηση σε κώδικα της παραπάνω διαδικασίας είναι η εξής:

```
#  Define the number of the classes
num_classes = 2

# Define a priori probabilities
P1 = 1/4       # P(ω1)
P2 = 1 - P1    # P(ω2)
p = [P1, P2]

# Define cost table
L = [[0, 1], [3, 0] ]

classes=[[] for i in range(num_classes)]
for x in range(0, len(x_axis)):
  R1 = L[0][0] * px1[x]*P1  +  L[0][1] * px2[x]*P2
  R2 = L[1][0] * px1[x]*P1  +  L[1][1] * px2[x]*P2

  if R1 < R2 :
    classes[0].append(x_axis[x])
  else:
    classes[1].append(x_axis[x])
```

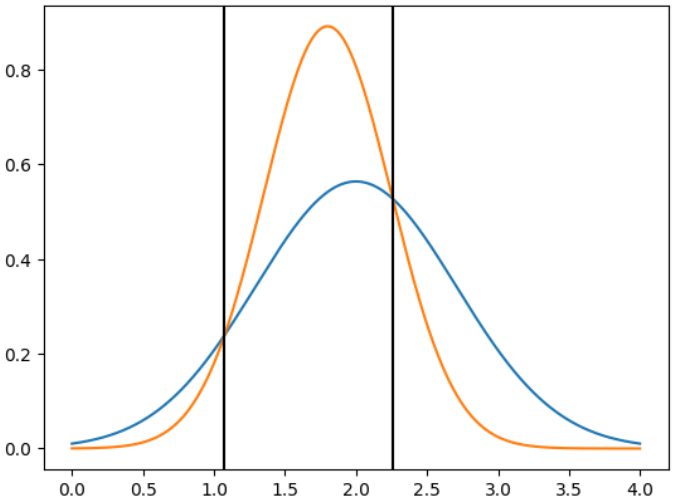
Σημειώνω ότι ο κώδικας αυτός τρέχει και για περισσότερες κλάσεις, αρκεί να δηλωθεί κατάλληλα ο πίνακας κόστους και η λίστα που θα περιέχει τα περιεχόμενα κάθε κλάσης.

Για την ευανάγνωστη παρουσίαση των αποτελεσμάτων του κώδικα έγραψα το ακόλουθο τμήμα. Ελέγχει για το εάν οι τιμές των x που περιέχονται στην λίστα κάθε κλάσης είναι συνεχόμενες, δηλαδή αν η διαφορά μεταξύ τους ισούται με το βήμα που έχω ορίσει (εδώ 0,001). Στο σημείο αυτό τονίζεται ότι εφόσον πρόκειται για δεκαδικούς αριθμούς, προκειμένου να έχω σωστό αποτέλεσμα στην λογική σύγκριση, στρογγυλοποιώ σε κάποιο δεκαδικό ψηφίο. Αποφάσισα να στρογγυλοποιήσω στο τέταρτο δεκαδικό καθώς η ακρίβεια για το βήμα αντιστοιχεί σε τρία δεκαδικά. Η ανάγκη για στρογγυλοποίηση προκύπτει από τον τρόπο που αναπαρίστανται οι αριθμοί κινητής υποδιαστολής στο δυαδικό σύστημα.

```
saved4plot = []
limits = []
for c in range(0, len(classes)):
  # c == 0 --> ω1
  # c == 1 --> ω2

  a = []  # λίστα που αποθηκεύω τα όρια των διαστημάτων των x που ανήκουν σε κάθε κλάση
  x_previous = classes[c][0]

  for x in classes[c]:
    if round(x,4) == round(x_previous+step,4) :
      x_previous = x
    else:
      a.append(round(x_previous, 4))
      a.append(round(x,4))
      x_previous = x

  a.pop(0)  # Deleting the first element because it is added two times, one at the be-
ginning and one at the end of the first interval
  a.append(classes[c][-1]) # Adding the last element to determing the end of the last
interval
  limits.append(a)   # saving the limits of each class in order to calculate the total
cost

  print("Στην κλάση ω" + str(c+1) + " ανήκουν τα x που βρίσκονται στο διάστημα:")

  if len(a) > 2:
    for i in range(0, len(a), 2):
```

```
      print("[" + str(a[i]) + "," + str(a[i + 1]) + "]")
      saved4plot.append(a[i])
      saved4plot.append(a[i+1])
  else:
    print("[" + str(a[0]) + "," + str(a[1]) + "]")

print("\n")
print(saved4plot)
print(limits)

plt.plot(x_axis, px1)
plt.plot(x_axis, px2)
for x in range(1, len(saved4plot)-1):
  plt.axvline(x = saved4plot[x], color = 'black')
plt.show()
```

Αναγνωρίζω ότι ο ανωτέρω κώδικας ίσως να μην είναι βέλτιστος, αλλά λόγω απουσίας επιπλέον χρόνο, δεν πρόλαβα να τον βελτιστοποιήσω. Το αποτέλεσμά του φαίνεται στην εικόνα 2.



Εικόνα 2: Γραφική αναπαράσταση των κατανομών με τα όρια των περιοχών απόφασης.

Όπως προαναφέρθηκε, τα δείγματα που βρίσκονται στην περιοχή $(-\infty, 1.75) \cup (2.26, -\infty)$ ταξινομούνται στην περιοχή ω_1, αλλά μπορεί να ανήκουν και στην κλάση ω_2. Με άλλα λόγια, το δείγμα μπορεί να ταξινομηθεί στην λάθος κλάση, μία απόφαση που παρουσιάζει κάποιο κόστος. Το ολικό κόστος του ταξινομητή μας είναι το άθροισμα της πιθανότητας να έχουμε επιλέξει την σωστή κλάση επί το αντίστοιχο κόστος συν την πιθανότητα να έχουμε επιλέξει την σωστή κλάση επί το αντίστοιχο κόστος για κάθε κλάση του προβλήματός μας. Η αντίστοιχη μαθηματική περιγραφή είναι:

$$C = P(\omega_2)\left[\lambda_{22}\int_{R2} p(x|\omega_2)dx + \lambda_{12}\int_{R1} p(x|\omega_2)dx\right] + P(\omega_1)\left[\lambda_{11}\int_{R1} p(x|\omega_1)dx + \lambda_{21}\int_{R2} p(x|\omega_1)dx\right]$$

Ο οποίος για τις μη μηδενικές τιμές λ_{ij} της εκφώνησης παίρνει την μορφή:

$$C = P(\omega_2)\left[\lambda_{12}\int_{R1} p(x|\omega_2)dx\right] + P(\omega_1)\left[\lambda_{21}\int_{R2} p(x|\omega_1)dx\right]$$

Παρατηρώ, λοιπόν, ότι το ολικό κόστος εξαρτάται μονάχα από τις λάθος αποφάσεις. Με άλλα λόγια, η σωστή απόφαση δεν επιβραβεύεται. Αυτό φαίνεται και από τον πίνακα κόστους καθώς $\lambda_{11} = \lambda_{22} = 0$. Η παρατήρηση αυτή βοήθησε ιδιαίτερα στην υλοποίηση του κώδικα για τον υπολογισμό του ολικού κόστους.

Αντικαθιστώντας τις τιμές των a priori πιθανοτήτων και των λ_{ij}:

$$C = \frac{3}{4}\left[\int_{R1} p(x|\omega_2)dx\right] + \frac{1}{4}\left[3\int_{R2} p(x|\omega_1)dx\right]$$

Εφόσον η περιοχή R1 δεν είναι συνεχής, σπάω το πρώτο ολοκλήρωμα στα αντίστοιχα διαστήματα.

$$C = \frac{3}{4}\left[\int_{-\infty}^{1,076} \frac{e^{-\frac{(x-1.8)^2}{0.4}}}{\sqrt{0.4\pi}}\,dx + \int_{2,258}^{+\infty} \frac{e^{-\frac{(x-1.8)^2}{0.4}}}{\sqrt{0.4\pi}}\,dx\right] + \frac{1}{4}\left[3\int_{1,076}^{2,258} \frac{e^{-(x-2)^2}}{\sqrt{\pi}}\,dx\right]$$

Για τον υπολογισμό ενός ολοκληρώματος της μορφής $\int_{\alpha}^{\beta} \mathrm{f}(x)\,dx$ όπου f μία συνάρτηση που ακολουθεί την κανονική κατανομή χρησιμοποιώ την αθροιστική συνάρτηση κατανομής Φ. Έτσι, ο υπολογισμός γίνεται:

$$\int_{\alpha}^{\beta} \mathrm{f}(x)\,dx = \Phi(\beta) - \Phi(a) = \frac{1}{2}\left[\mathrm{erf}\left(\frac{\beta-\mu}{\sigma\sqrt{2}}\right) - \mathrm{erf}\left(\frac{\alpha-\mu}{\sigma\sqrt{2}}\right)\right]$$

Επομένως, τα ολοκληρώματα του κόστους ισούνται με:

$$\int_{-\infty}^{1,076} \frac{e^{-\frac{(x-1.8)^2}{0.4}}}{\sqrt{0.4\pi}}\,dx = \Phi(\beta) - \Phi(a) = \frac{1}{2}\left[\mathrm{erf}\left(\frac{1,076-1,8}{\sqrt{2}\,0,2}\right) - \lim_{x\to-\infty}\left\{\mathrm{erf}\left(\frac{x-1,8}{\sqrt{2}\,0,2}\right)\right\}\right] = \frac{-0.8946+1}{2} = 0.0527$$

$$\int_{2,25}^{+\infty} \frac{e^{-\frac{(x-1.8)^2}{0.4}}}{\sqrt{0.4\pi}}\,dx = \Phi(\beta) - \Phi(a) = \frac{1}{2}\left[\lim_{x\to+\infty}\left\{\mathrm{erf}\left(\frac{x-1,8}{\sqrt{2}\,0,2}\right)\right\} - \mathrm{erf}\left(\frac{2,258-1,8}{\sqrt{2}\,0,2}\right)\right] = \frac{1-0,6952}{2} = 0,1524$$

$$\int_{1,076}^{2,258} \frac{e^{-(x-2)^2}}{\sqrt{\pi}}\,dx == \Phi(\beta) - \Phi(a) = \frac{1}{2}\left[\mathrm{erf}\left(\frac{2,25-2}{\sqrt{2}*0,5}\right) - \mathrm{erf}\left(\frac{1,076-2}{\sqrt{2}*0,5}\right)\right] = \frac{0,2847+0,8087}{2} = 0,5467$$

Άρα,:

$$C = \frac{3}{4}[0.0527 + 0,1524] + \frac{1}{4}[3 * 0,5467] = 0,56385$$

Ο κώδικας που υπολογίζει το ολικό κόστος φαίνεται παρακάτω. Για τον υπολογισμό του ολοκληρώματος, όπως φάνηκε και παραπάνω, χρειάζομαι την συνάρτηση erf η οποία βρίσκεται στην βιβλιοθήκη `scipy.special`. Αποφάσισα να δημιουργήσω μία συνάρτηση για τον υπολογισμό του ολοκληρώματος ξεχωριστά ώστε ο κώδικας να είναι ευανάγνωστος. Όπως προαναφέρθηκε, το κόστος προκύπτει μονάχα από τα δείγματα που ταξινομήθηκαν στην λάθος κλάση (η σωστή ταξινόμηση δεν επιβραβεύεται), στην συνάρτηση `cost_wrong` κράτησα μονάχα τις μη μηδενικές τιμές του λ με την σειρά που με βόλευε για την υλοποίηση του κώδικα.

```
import scipy.special as sp

def SumOfSeries(m, s, lower_limit, upper_limit):

  if upper_limit + step == end+step:     # inf
    f_upper = sp.erf(float("inf"))
  else:
    f_upper = sp.erf( (round(upper_limit,4)-m)/(sqrt(2)*s) )

  if lower_limit +step == start+step:     # - inf
    f_lower = - sp.erf(float("inf"))
  else:
    f_lower = sp.erf( (lower_limit-m)/(sqrt(2)*s) )

  return 0.5 * ( f_upper - f_lower )

def cost_wrong(series_parameter,P, L, limits):
  # Προκύπτει από τα δεδομένα της κατανομής που βρίσκονται στην άλλη περιοχή απόφασης.
Για αυτό αντιστρέφω την limits

  cost = 0
  L_wrong = [ L[1][0], L[0][1] ] # κρατάω μονάχα τα μη μηδενικά λ με την σειρά που με
βολεύει

  limits.reverse()
  for p in range(0, len(P)):
```

```
    if len(limits[p]) > 2:
      a = 0
      for i in range(0, len(limits[p]), 2):
        aa = SumOfSeries(series_parameter[p][0], series_parameter[p][1], limits[p][i],
limits[p][i+1])
        a += aa
    else:
      a = SumOfSeries(series_parameter[p][0], series_parameter[p][1], limits[p][0],
limits[p][1])

    b = L_wrong[p]
    cost += P[p] * b * a

  return cost
```

Το αποτέλεσμα του παραπάνω κώδικα είναι 0.5636981344722247, πολύ κοντά σε αυτό που υπολόγισα και εγώ.

Ερώτημα Β:
Να προσομοιωθεί η διαδικασία υπολογιστικά, δημιουργώντας τυχαία δείγματα που ακολουθούν την κανονική κατανομή, και να εκτιμηθεί αριθμητικά το κόστος από την λύση του ερωτήματος Α.

Για την δημιουργία των τυχαίων δειγμάτων που ακολουθούν τια κανονικές κατανομές της εκφώνησης χρησιμοποίησα την συνάρτηση `random.normal` της βιβλιοθήκης `numpy`.

Με βάση τα γνωστά όρια κάθε κλάσης εξετάζω σε ποιά κλάση ταξινομούνται τα δεδομένα και αυξάνω τον αντίστοιχο μετρητή για την σωστή και την λάθος απόφαση. Έπειτα, διαιρώ την τιμή των μετρητών με το συνολικό πλήθος των δειγμάτων προκειμένου να πάρω το ποσοστό της σωστής και της λάθος ταξινόμησης. Στην συνέχεια υπολογίζω το κόστος κάθε κλάσης με βάση τον τύπο της θεωρίας. Τέλος, προσθέτω τα κόστη των κλάσεων προκειμένου να υπολογίσω το ολικό κόστος.

```
samples = 1000
w1 = np.random.normal(series_parameters[0][0], series_parameters[0][1],samples)
w2 = np.random.normal(series_parameters[1][0], series_parameters[1][1], samples)

w1_wrong = 0
w1_right = 0
w2_wrong = 0
w2_right = 0

for point in w1:
  if point >= 1.076 and point <= 2.258:
    w1_wrong += 1
  else:
    w1_right += 1

w1_right = w1_right / samples
w1_wrong = w1_wrong / samples
costw1 = p[0] * ( L[0][0]*w1_right + L[1][0]*w1_wrong )

print("\nData from ω1 that classified in ω1: " + str(w1_right) )
print("Data from ω1 that classified in ω2: " + str(w1_wrong) )
print("Total cost in class ω1: " + str(costw1) )

for point in w2:
  if point >= 1.076 and point <= 2.258:
    w2_right += 1
  else:
    w2_wrong += 1

w2_right = w2_right / samples
w2_wrong = w2_wrong / samples
costw2 = p[1] * ( L[1][1]*w2_right + L[0][1]*w2_wrong )
```

```
print("\nData from ω2 that classified in ω1: " + str(w2_wrong) )
print("Data from ω2 that classified in ω2: " + str(w2_right) )
print("Total cost in class ω2: " + str(costw2) )

print("\nTotal cost is: " + str(costw1+costw2) )
```

Παρατηρώ ότι το κόστος αυτό δεν διαφέρει σημαντικά ούτε από εκείνο που υπολόγισα νωρίτερα αλλά ούτε και από το κόστος που υπολόγισα θεωρητικά. Η παρατήρηση αυτή ήταν αναμενόμενη.

ΑΣΚΗΣΗ 3

Ερώτημα Α
Να υλοποιήσετε σε κώδικα συναρτήσεις που θα δίνουν την τιμή της

- ***Συνάρτησης διάκρισης για δεδομένη κατανομή N(μ,Σ) d διαστάσεων και εκ των προτέρων πιθανότητας P(ωi)***
- ***Ευκλείδειας απόστασης μεταξύ δύο αυθαίρετων σημείων x1 και x2 στον χώρο των d διαστάσεων***
- ***Απόστασης Mahalanobis μεταξύ του μέσου μ και ενός αυθαίρετου σημείου x στον χώρο των d διαστάσεων δεδομένου του πίνακα συνδυασποράς Σ***

Το παρόν ερώτημα υλοποιείται αποκλειστικά με κώδικα. Συγκεκριμένα επιλέχτηκε η γλώσσα Python και οι κώδικες βρίσκονται σε αρχείο με αντίστοιχο όνομα.

Έστω ότι οι περιοχές απόφασης είναι παρακείμενες. Τότε αυτές χωρίζονται από μία υπερεπιφάνεια απόφασης στον πολυδιάστατο χώρο των στατιστικών. Μερικές φορές, αντί να δουλεύουμε με πιθανότητες είναι πιο βολικό να εργαζόμαστε με ισοδύναμη συνάρτηση η οποία καλείται συνάρτηση διάκρισης (discrimination function). Σύμφωνα με την εκφώνηση, ο τύπος της συνάρτησης αυτής είναι:

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma_i|) + \ln(P(\omega_i))$$

Η υλοποίησή της γίνεται με το ακόλουθο κομμάτι κώδικα:

```
def DiscriminationFunction(d, x, m, sigma, prior):
    x_m = x - m

    if d>1:
        Det_sigma = abs(np.linalg.det(sigma))
        inv_sigma = np.linalg.inv(sigma)
        g = -0.5 * (x_m.T).dot(inv_sigma).dot(x_m) - (d/2) * np.log(2*pi) - 0.5 *
np.log(Det_sigma) + np.log(prior)
    else:
        inv_sigma = sigma**-1
        g = -0.5 * x_m * inv_sigma * x_m - (d/2)*log(2*pi) - 0.5*log(abs(sigma)) +
np.log(prior)
    return g
```

d	Πλήθος διευθύνσεων	int
x	Διάνυσμα χαρακτηριστικών	numpy_array
m	Διάνυσμα μέσης τιμής	numpy_array
sigma	Πίνακας συνδιασποράς Σ	numpy_array
prior	Εκ των προτέρων πιθανότητα	float

Εφόσον η NumPy υπολογίζει αντίστροφους και ανάστροφους πίνακες για τουλάχιστον διδιάστατους πίνακες, γίνεται διάκριση ανάλογα την τιμή της μεταβλητής d. Το πρόβλημα εντοπίζεται στις συναρτήσεις `np.det()` και `np.linalg.inv()`.

Η Ευκλείδεια απόσταση 2 τυχαίων διανυσμάτων δίνεται από τον τύπο:

$$d_\varepsilon(\mathbf{x},\mathbf{y}) = ((\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y}))^{1/2} = \left[\sum_{i=1}^{n}(x_i-y_i)^2\right]^{1/2}$$

και το αντίστοιχο κομμάτι κώδικα είναι:

```
def EucleidianDistance(d, x1, x2):
    if d == 1:
        distanace = np.sqrt(np.sum((np.square(x1-x2))))
    else:
        distanace = np.sqrt(np.sum((np.square(x1-x2).T.dot(x1-x2) )))
    return distanace
```

x1	Διάνυσμα χαρακτηριστικών	numpy_array
x2	Διάνυσμα χαρακτηριστικών	numpy_array

Η Mahalanobis απόσταση 2 τυχαίων διανυσμάτων δίνεται από τον τύπο:

$$d_m = ((\underline{x} - \underline{\mu}_i)^T \Sigma^{-1} (\underline{x} - \underline{\mu}_i))^{\frac{1}{2}}$$

και το αντίστοιχο κομμάτι κώδικα είναι:

```
def MahalanobisDistanace(d, m, sigma, x):
    x_m = x - m
    if d>1:
        inv_sigma = np.linalg.inv(sigma)
        distance = np.sqrt(x_m.T.dot(inv_sigma).dot(x_m))
    else:
        inv_sigma = sigma**-1
        distance = np.sqrt(x_m*inv_sigma*x_m)
    return distance
```

Ερώτημα Β:

Υποθέτοντας πως οι υποκείμενες κατανομές των κλάσεων είναι Gaussian, να εκτιμήσετε τις παραμέτρους τους με την μέθοδο της μέγιστης πιθανοφάνειας για τις περιπτώσεις που:

Β1: χρησιμοποιείται μονάχα το πρώτο χαρακτηριστικό (x1)

Β2: χρησιμοποιείται το πρώτο και το δεύτερο χαρακτηριστικό (x1 και x2)

Β3: χρησιμοποιούνται όλα τα χαρακτηριστικά (x1, x2 και x3)

Αρχικά, εισάγω το αρχείο data.csv και το επεξεργάζομαι ώστε να πάρω τις τιμές.

```
import pandas as pd
data = pd.read_csv('data.csv', usecols=[0,1,2])
c = np.transpose(pd.read_csv('data.csv', usecols=[3]).values)
df = np.transpose(data.values)
```

Όπως εδώ, έτσι και σε πολλά προβλήματα οι συναρτήσεις πυκνότητας πιθανότητας δεν είναι γνωστές, άρα πρέπει να εκτιμηθούν από τα διαθέσιμα δεδομένα. Ανάλογα με τη διαθέσιμη πληροφορία υιοθετείται η κατάλληλη μέθοδος για τον προσδιορισμό των άγνωστων παραμέτρων. Μία από τις σημαντικότερες είναι η μέθοδος της μέγιστης πιθανοφάνειας. Γνωρίζοντας τα χαρακτηριστικά κάθε κλάσης, επιδιώκουμε να εκτιμήσουμε τις άγνωστες παραμέτρους. Θεωρούμε ότι οι κατανομές των κλάσεων είναι ανεξάρτητες, δηλαδή τα δεδομένα της μίας κλάσης δεν επηρεάζουν τα δεδομένα της άλλης. Έτσι, η μέθοδος αυτή εκτιμά τις παραμέτρους των κατανομών ώστε η συνάρτηση πιθανοφάνειας να λάβει την μέγιστη τιμή. Ο υπολογισμός των παραμέτρων της συνάρτησης πιθανοφάνειας γίνεται με τον ακόλουθο κώδικα.

```
def maximum_likelihood_estimation(data):
    mean = np.mean(data, axis=0)
    covariance_matrix = np.cov(data, rowvar=False)
    return mean, covariance_matrix
```

Σε πρώτης φάση χωρίζω τα δεδομένα σε κλάσεις σύμφωνα με την τελευταία στήλη του αρχείου. Εφόσον γνωρίζω ότι έχω τρεις κλάσεις, δημιουργώ μία λίστα η οποία περιέχει τρεις λίστες, σε κάθε μία θα αποθηκεύονται και τα αντίστοιχα δεδομένα του προβλήματος.

```
classes = [ [], [], []]
```

Στο πρώτο ερώτημα ο υπολογισμός των παραμέτρων γίνεται μονάχα με βάση το πρώτο χαρακτηριστικό, οπότε αυτό μονάχα αποθηκεύω και στην εκάστοτε λίστα. Στην συνέχεια, περνάω τα δεδομένα κάθε κλάσης ως ορίσματα στην επάνω συνάρτηση ώστε να υπολογιστούν οι παράμετροι των εκάστοτε κατανομών.

```
classes_x1 = [ [], [], []]
for i in range(len(df[0])):
  if c[0][i] == 1:
    classes_x1[0].append([df[0][i]])
  elif c[0][i] == 2:
    classes_x1[1].append([df[0][i]])
  else:
    classes_x1[2].append([df[0][i]])

for i in range (0,len(classes_x1)):
  m, s = maximum_likelihood_estimation(classes_x1[i])
  print("Κλάση "+str(i+1)+": μ = "+str(round(m[0],4))+" \tκαι\tΣ = "+str(s))
```

Τα αποτελέσματα μου είναι:

```
Κλάση 1: μ = 0.9064    και   Σ = 26.65228151147099
Κλάση 2: μ = -1.4842   και   Σ = 38.952850705128206
Κλάση 3: μ = 3.6802    και   Σ = 9.45306743902439
```

Επαναλαμβάνω την ίδια διαδικασία για το ερώτημα Β2 όμως αυτήν την φορά αποθηκεύω σε κάθε κλάση τα δύο πρώτα χαρακτηριστικά x1 και x2.

```
classes_x1x2 = [ [], [], []]
for i in range(len(df[0])):
  if c[0][i] == 1:
    classes_x1x2[0].append([df[0][i], df[1][i]])
  elif c[0][i] == 2:
    classes_x1x2[1].append([df[0][i], df[1][i]])
  else:
    classes_x1x2[2].append([df[0][i], df[1][i]])


for i in range (0,len(classes_x1x2)):
  m, s = maximum_likelihood_estimation(classes_x1x2[i])
  print("Κλάση "+str(i+1)+": μ = "+str(np.round(m,4))+" \tκαι\tΣ =
"+str(np.round(s[0],4)))
  print("\t\t\t\t\t    "+str(np.round(s[1],4))+'\n')
```

Τα αποτελέσματα μου είναι:

```
Κλάση 1: μ = [ 0.9064 -1.0449]     και   Σ = [26.6523 15.8237]
                                             [15.8237 18.3887]
Κλάση 2: μ = [-1.4842 -1.1483]     και   Σ = [38.9529  7.97  ]
                                             [ 7.97   11.408]
Κλάση 3: μ = [3.6802 1.2444]       και   Σ = [9.4531 6.3758]
                                             [6.3758 6.8551]
```

Επαναλαμβάνω την ίδια διαδικασία για το ερώτημα Β3 όμως αυτήν την φορά αποθηκεύω σε κάθε κλάση όλα τα χαρακτηριστικά x1, x2 και x3.

```
classes_x1x2x3 = [ [], [], []]
for i in range(len(df[0])):
  if c[0][i] == 1:
    classes_x1x2x3[0].append([df[0][i], df[1][i], df[2][i]])
  elif c[0][i] == 2:
    classes_x1x2x3[1].append([df[0][i], df[1][i], df[2][i]])
  else:
    classes_x1x2x3[2].append([df[0][i], df[1][i], df[2][i]])

for i in range (0,len(classes_x1x2x3)):
```

```
  m, s = maximum_likelihood_estimation(classes_x1x2x3[i])
  print("Κλάση "+str(i+1)+": μ = "+str(np.round(m,4))+" \tκαι\tΣ =
"+str(np.round(s[0],4)))
  print("\t\t\t\t\t\t    "+str(np.round(s[1],4)))
  print("\t\t\t\t\t\t    "+str(np.round(s[2],4))+"\n")
```

Τα αποτελέσματά μου είναι:

```
Κλάση 1: μ = [ 0.9064 -1.0449 -0.3618]  και   Σ = [26.6523 15.8237  4.6556]
                                                  [15.8237 18.3887 -0.8336]
                                                  [ 4.6556 -0.8336 20.7802]

Κλάση 2: μ = [-1.4842 -1.1483 -1.0092]  και   Σ = [ 38.9529   7.97   -18.8435]
                                                  [ 7.97    11.408    0.4259]
                                                  [-18.8435   0.4259  19.7095]

Κλάση 3: μ = [3.6802 1.2444 1.0561]     και   Σ = [9.4531 6.3758 9.4663]
                                                  [6.3758 6.8551 5.7652]
                                                  [ 9.4663  5.7652 40.6077]
```

Ερώτημα Γ: Επιλέξτε τον κατάλληλο ταξινομητή αιτιολογώντας την επιλογή σας.

Ας αναλύσουμε τον κάθε ταξινομητή ξεχωριστά:

- **<u>Ταξινομητής με Ευκλείδεια απόσταση</u>**

Ο ταξινομητής αυτός ενδείκνυται για περιπτώσεις που πληρούνται τα ακόλουθα κριτήρια:

1. Οι κλάσεις είναι ισοπίθανες
2. Τα δεδομένα όλων των κλάσεων ακολουθούν γκαουσσιανή κατανομή
3. Ο πίνακας συνδιασποράς Σ είναι:
   - ίδιος για όλες τις κλάσεις,
   - διαγώνιος
   - τα απέναντι στοιχεία της διαγωνίου είναι ίσα.

Από αυτά, δεν ικανοποιείται η τρίτη συνθήκη, οπότε δεν είναι ο κατάλληλος ταξινομητής μου.

Η ταξινόμηση βασίζεται στην απόσταση του διανύσματος του χαρακτηριστικού από εκείνου του μέσου μ.

- **<u>Ταξινομητής με Mahalanobis Απόσταση</u>**

Ο ταξινομητής αυτός ενδείκνυται για περιπτώσεις που πληρούνται τα ακόλουθα κριτήρια:

1. Οι κλάσεις είναι ισοπίθανες
2. Τα δεδομένα όλων των κλάσεων ακολουθούν γκαουσιανή κατανομή
3. Ο πίνακας συνδιασποράς Σ είναι ίδιος για όλες τις κλάσεις

Από αυτά, δεν ικανοποιείται πάλι η τρίτη συνθήκη, οπότε δεν είναι ο κατάλληλος ταξινομητής μου.

Η ταξινόμηση βασίζεται στην απόσταση του διανύσματος του χαρακτηριστικού από εκείνου του μέσου, καθώς στον πίνακα συνδυασποράς.

- **<u>Ταξινομητής με την δοθείσα συνάρτηση διάκρισης</u>**

Για τον ταξινομητή αυτόν δεν δίνονται ενδεικτικές περιπτώσεις χρήσης. Η ταξινόμηση βασίζεται στην απόσταση του διανύσματος του χαρακτηριστικού από εκείνου του μέσου, στον πίνακα συνδυασποράς και την a priori πιθανότητα της εκάστοτε κλάσης.

Συνεπώς, ο ταξινομητής αυτός θεωρείται ο πλέον κατάλληλος, παρόλο που το υπολογιστικό κόστος είναι σαφώς μεγαλύτρο.

Σημειώνεται ότι η συνθήκη των ισοπίθανων κλάσεων ικανοποιείται σαφώς μονάχα για τα ερωτήματα που δεν λαμβάνω υπόψη την τρίτη κλάση, δηλαδή για τα ερωτήματα Γ και Δ.

Να προσδιοριστεί υπολογιστικά το εμπειρικό σφάλμα ταξινόμησης, δηλαδή το ποσοστό των σημείων που ταξινομούνται εσφαλμένα, υποθέτοντας ότι οι a priori πιθανότητες είναι P(ω1) = P(ω2) = 0.5 κάνοντας χρήση μονάχα του πρώτου χαρακτηριστικού.

Εφόσον P(ω1) = P(ω2) = 0.5, θεωρώ ότι δεν υπάρχει η τρίτη κλάση καθώς, σύμφωνα με θεωρία, το άθροισμα των a priori πιθανοτήτων των συνολικών καταστάσεων της φύσης για ένα πρόβλημα ισούται με την μονάδα. Για αυτό, αφαιρώ από την μεταβλητή `classes`, το τελευταίο στοιχείο.

```
priors = [0.5 , 0.5]
classes_x1c = classes_x1.copy()
classes_x1c.pop()  # Deleting third class ω3
```

Ένα δείγμα ταξινομείται στην κλάση για την οποία προκύπτει η μεγαλύτερη τιμή της συνάρτησης διάκρισης. Το σφάλμα προκύπτει αν ένα δείγμα δεν ταξινομηθεί στην κλάση στην οποία ανήκει. Για παράδειγμα, εάν ένα δείγμα x παρουσιάσει μέγιστη τιμή της συνάρτησης ταξινόμησης g για την κλάση ω_2, θα ταξινομηθεί σε αυτήν. Εάν όμως ανήκει στην ω_1 θεωρείται σφάλμα.

Ο ταξινομητής μου υλοποιείται στην συνάρτηση `classifier`. Με βάση τα δεδομένα κάθε κλάσης εκτιμώνται οι παράμετροι της. Στην συνέχεια, υπολογίζεται η τιμή της συνάρτησης g για κάθε κλάση ξεχωριστά και αποθηκεύονται. Από αυτές αναζητείται η μέγιστη τιμή. Εάν αυτή δεν προέκυψε από την κλάση στην οποία ανήκει κανονικά το δείγμα, τότε αυξάνω τον μετρητή λάθους κατά μία μονάδα. Τέλος, προκειμένου να υπολογίσω το ποσοστό, διαιρώ την τιμή του μετρητή λαθών με το πλήθος των στοιχείων για τις πρώτες δύο κλάσεις.

```
def classifier(d, classes, priors):
    m_x = []
    s_x = []
    wrong = 0

    for y in range(0, len(classes)):
        m, s = maximum_likelihood_estimation(classes[y])
        m_x.append(m)  # Need to keep these values in order to calculate the discrimi-
nation function below
        s_x.append(s.tolist())


    for cc in range(0, len(classes)):
        for i in range(len(classes[cc])):
            # Wrong classification for features belonging to classI --> Occurs when gI
is not minimum
            g = []
            for y in range(0, len(classes)):
                gi = DiscriminationFunction(d, classes[cc][i], m_x[y], s_x[y], pri-
ors[y])
                g.append(gi)

            if max(g) != g[cc]:
                wrong += 1

    total = sum(len(v) for v in classes)
    return wrong / total
```

```
error_1 = classifier(1, classes_x1c, priors)
print("Classification error using only feature x1: "+str(error_1))
```

Προκειμένου να ελέγχω την εξέλιξη του προγράμματος εκτυπώνεται και το πλήθος των λαθών σε κάθε κατηγορία καθώς και το πλήθος των δειγμάτων. Τα αποτελέσματά μου είναι:

```
    Classification errors in class ω1: 10
    Classification errors in class ω2: 22
    Total samples are:79
```

```
Classification error using only feature x1: 0.4050632911392405
```

Ερώτημα Δ1

Επαναλάβετε το προηγούμενο βήμα χρησιμοποιώντας τα πρώτα δύο χαρακτηριστικά (x1 και x2).

```
classes_x1x2d = classes_x1x2.copy()
classes_x1x2d.pop() # Deleting third class ω3

error_2 = classifier(2, classes_x1x2d, priors)
print("Classification error using only features x1 and x2: "+str(error_2))
```

```
      Classification errors in class ω1: 8
      Classification errors in class ω2: 22
      Total samples are:79
Classification error using only features x1 and x2: 0.379746835443038
```

Ερώτημα Δ2:

Επαναλάβετε το προηγούμενο βήμα χρησιμοποιώντας όλα τα χαρακτηριστικά (x1, x2, x3).

```
classes_x1x2x3d = classes_x1x2x3.copy()
classes_x1x2x3d.pop()   # Deleting third class ω3

error_3 = classifier(3, classes_x1x2x3d, priors)
print("Classification error using all features x1, x2 and x3: " + str(error_3))
```

```
      Classification errors in class ω1: 11
      Classification errors in class ω2: 6
      Total samples are:79
Classification error using all features x1, x2 and x3: 0.21518987341772153
```

Ερώτημα Δ3:

Σχολιάστε τα αποτελέσματά σας και ιδιαίτερα την σχέση του εμπειρικού σφάλματος με τις διαστάσεις του προβλήματος.

Σύμφωνα με τα παραπάνω αποτελέσματα, είναι φανερό πως η αύξηση του πλήθους των χαρακτηριστικών οδηγεί σε μικρότερο σφάλμα ταξινόμησης. Αυξάνοντας τις διαστάσεις παίρνουμε επιπλέον πληροφορία για κάθε κλάση η οποία είναι δυνατό να μας βοηθήσει στην ταξινόμηση. Ωστόσο, αν η πληροφορία αυτή δεν βοηθάει στην ταξινόμηση, δηλαδή λειτουργεί σαν θόρυβος, είναι πιθανό να αυξηθεί το σφάλμα μας. Η αύξηση των διαστάσεων δεν ισοδυναμεί πάντα με μείωση του ολικού σφάλματος, αυτό εξαρτάται από το εάν τα επιπρόσθετα χαρακτηριστικά εξυπηρετούν την ταξινόμηση.

Επιπλέον, σημειώνεται ότι η αύξηση των διαστάσεων ενδέχεται να χειροτερέψει τα αποτελέσματα της εκτίμησης των παραμέτρων, εάν παράλληλα δεν αυξηθεί και το πλήθος των δειγμάτων. Το πρόβλημα αυτό είναι γνωστό ως Curse of Dimentionality: εάν για μία μονοδιάσταση (d = 1) pdf απαιτούνται N δείγματα για να εκτιμηθούν καλά οι παράμετροι, τότε για περισσότερες διαστάσεις (d>1) απαιτούνται N^d δείγματα.

Ερώτημα Ε:

Επαναλάβετε την εκτίμηση του σφάλματος χρησιμοποιώντας όλα τα χαρακτηριστικά για $P(\omega_1) = 0.8$ και $P(\omega_2) = P(\omega_3) = 0.1$

Πλέον έχω και τις τρεις κλάσεις οπότε δεν χρειάζεται να αφαιρώ το τελευταίο στοιχείο.

```
priors = [0.8, 0.1, 0.1]

error_4 = classifier(3, classes_x1x2x3, priors)
print("Classification error using all features x1, x2 and x3: " + str(error_4))
```

```
      Classification errors in class ω1: 0
      Classification errors in class ω2: 30
      Classification errors in class ω3: 36
      Total samples are:120
Classification error using all features x1, x2 and x3: 0.55
```

ΑΣΚΗΣΗ 4

Έστω σύστημα ταξινόμησης Bayesian σε δύο κλάσεις ω_1 και ω_2 και καταλήγουμε πως η πιθανότητα σφάλματος εξαρτάται από την a priori πιθανότητα $P(\omega_1)$.

$$P_{error} = P(\omega_1)^2(1 - P(\omega_1))$$

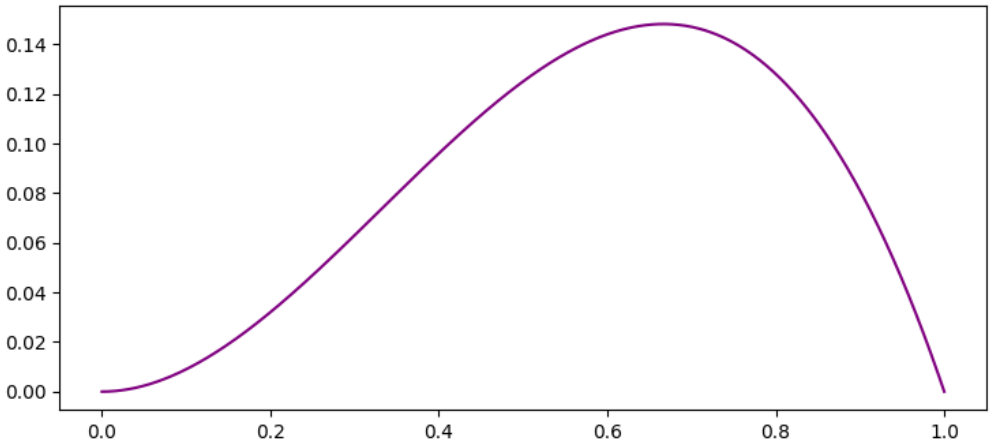
Ερώτημα Α

Εάν δεν γνωρίζετε την $P(\omega_1)$, για ποια τιμή της θα πρέπει να σχεδιάσετε τα όρια απόφασης του συστήματος ώστε να ελαχιστοποιήσετε το μέγιστο πιθανό σφάλμα. Ποια θα είναι η τιμή της πιθανότητας σφάλματος τότε;

Η τιμή της $P(\omega_1)$ για την οποία θα σχεδιαστεί το σύστημά μας καθορίζεται από το κριτήριο του μέγιστου ελαχίστου. Σύμφωνα με το κριτήριο ο ταξινομητής μας σχεδιάζεται έτσι ώστε το χειρότερο ολικό σφάλμα για οποιαδήποτε τιμή των εκ των προτέρων πιθανοτήτων να είναι το μικρότερο δυνατό (ελαχιστοποιούμε το μέγιστο πιθανό σφάλμα).

Προκειμένου να βρω για ποια τιμή της $P(\omega_1)$ το σφάλμα μεγιστοποιείται, σχεδιάζω την γραφική παράσταση. Έπειτα, υπολογίζω το μέγιστο σφάλμα και την τιμή για την οποία προκύπτει.

```
import numpy as np
from matplotlib import pyplot as plt
import sympy as sp

def f(x):
  return x**2 * (1-x)


plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True

x = np.linspace(0, 1, 1000)

plt.plot(x, f(x), color='purple')
plt.show()
```



Εικόνα 3: Συνάρτηση σφάλματος άσκηση 4

```
def P_error_ideal():
  x = np.linspace(0, 1, 1000)
  max_error = max(f(x))
  index = np.argmax(f(x))

  print("The maximum error is " + str(max_error) + " which occurs for x = " + str(x[in-
dex]))
  return x[index], max_error
```

```
The maximum error is 0.14814814814814817 which occurs for x = 0.6666666666666666
```

Παρατηρώ ότι η P_{error} γίνεται μέγιστη για $P(\omega_1) = 0{,}6667$. Συνεπώς, αυτό είναι το όριο της απόφασης για την οποία σχεδιάζω τον ταξινομητή μου.

Εάν επιλέξω $P(\omega_1)$ ώστε να ελαχιστοποιείται το μέγιστο δυνατό σφάλμα, τότε για κάθε άλλη τιμή $P(\omega_1)$ το σφάλμα θα είναι μία ευθεία εφαπτομένη της καμπύλης στο $P(\omega_1)$ που επέλεξα. Για οποιαδήποτε άλλη τιμή ρύθμισης του συστήματος $P(\omega_1) \neq 0.667$, το σφάλμα θα λαμβάνει μεγαλύτερες τιμές για κάποια $P(\omega_1)$.

Ερώτημα Β

Εάν ρυθμίσατε το σύστημά σας υποθέτοντας πιθανότητα $P(\omega_1) = 0.3$, ποια θα είναι η πιθανότητα σφάλματος εάν η πραγματική a priori πιθανότητα της της κλάσης ω_1 εάν $P(\omega_1) = 0.7$;

Εάν ρύθμισα το σύστημα υποθέτοντας πως $P(\omega_1) = 0.3$, τότε θεωρήθηκε πως το μέγιστο δυνατό σφάλμα είναι 0.063. Προκειμένου να βρω την σφάλμα για $P(\omega_1) = 0.7$ υπολογίζω αρχικά την εφαπτομένη στο $P(\omega_1) = 0.3$. Η εξίσωση της εφαπτομένης είναι

$$y_{εφαπτομένη} = ax + b, \text{ όπου}$$

$$a = \frac{df}{d\mathrm{p}}\Big|_{p=0.3} = 2p - 3p^2|_{p=0.3} = 2 * 0.3 - 3 * 0.3^2 = 0.6 - 0.27 = 0.33$$

$$b = f(0.3) - a * 0.3 = 0.063 - 0.099 = -0.036$$

Το σφάλμα ισούται με το σημείο της εφαπτομένης για $P(\omega_1) = 0.7$: $P_{error} = a * 0.7 + b = 0.195$

Ο κώδικας που υλοποιεί την παραπάνω διαδικασία είναι:

```
def P_error(p, p_true):
  p_ideal, max_error = P_error_ideal()

  if round(p_ideal, 4) == p:
    print("Ideal")
    return max_error
  else:
    x = sp.symbols('x')
    df = sp.diff(f(x), x)
    a = df.subs(x, p).evalf()
    b = f(p) - a * p

    print("The tangent's equation is: y = " +str(round(a,4)) + "x" + str(round(b,4)))
    return a * p_true + b

p_estimated = 0.3
p_true = 0.7
error = P_error(p_estimated, p_true)
print("Error for estimated priority " + str(p_estimated)+ " when the truth is " +
str(p_true) + ": " + str(error))
```

```
The maximum error is 0.14814814814814817 which occurs for x = 0.6666666666666666
The tangent's equation is: y = 0.3300x-0.0360
Error for estimated priority 0.3 when the truth is 0.7: 0.1950000000000000
```

Τα αποτελέσματα συμβαδίζουν με εκείνα που υπολόγισα.

ΑΣΚΗΣΗ 5

Πετάμε ένα νόμισμα N=10 φορές και φέρνουμε κατά σειρά {κ,γ,κ,κ,κ,γ,κ,κ,γ,κ} (κ=κεφάλι, γ=γράμματα). Ποια είναι η πιθανότητα θ να φέρουμε κεφάλι; Η αρχική κατανομή του θ είναι

$$\boldsymbol{p(\theta|D^0) = A \cdot \theta \cdot (1-\theta)^4}\text{, για } 0 \le \theta \le 1 \text{ (beta distribution).}$$

Ερώτημα Α: Να υπολογισθεί το Α

Η $\mathrm{p}(\theta|\mathrm{D}^0)$ είναι συνάρτηση πυκνότητας πιθανότητας οπότε το ολοκλήρωμά της για τις τιμές του θ ισούται με μονάδα.

$$\int_0^1 \mathrm{p}(\theta|\mathrm{D}^0)\,d\theta = 1 \Leftrightarrow \int_0^1 A \cdot \theta \cdot (1-\theta)^4\,d\theta = 1 \Leftrightarrow \frac{A}{30} = 1 \Leftrightarrow A = 30$$

Ερώτημα Β: Να σχεδιασθούν σε κοινό διάγραμμα τα $p(\theta|D^1)$, $p(\theta|D^5)$ και $p(\theta|D^{10})$.

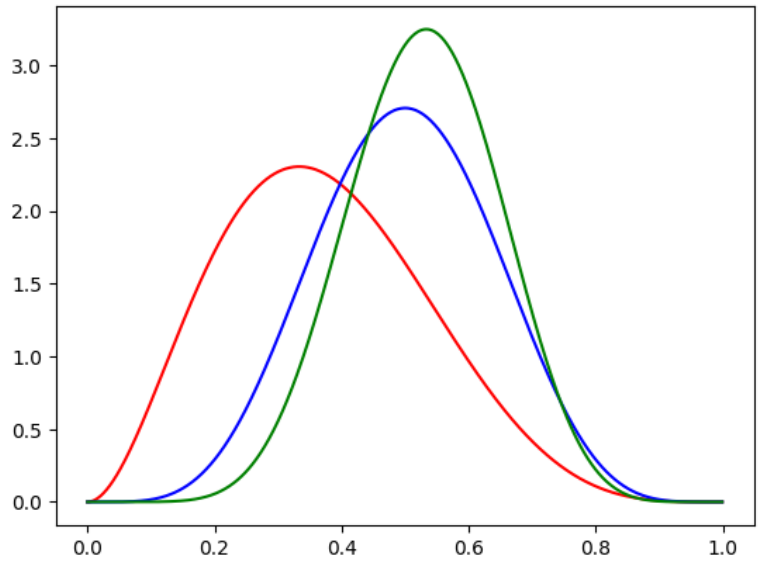
Σύμφωνα με τον τύπο $\mathrm{p}(\theta|\mathrm{D}^{\mathrm{N}}) = \frac{p(x|\theta)\cdot p(\theta|D^{N-1})}{\int_0^1 p(x|\theta)\cdot p(\theta|D^{N-1})d\theta} = \frac{\theta^k(1-\theta)^{N-k}\cdot p(\theta|D^0)}{\int_0^1 p(x|\theta)\cdot p(\theta|D^0)d\theta}$, αφού $p(x|\theta) = \begin{cases} \theta & \text{για κεφάλι} \\ 1-\theta & \text{για γράμματα} \end{cases}$ όπου N το πλήθος των ρίψεων και k το πλήθος των ρίψεων που έχουν φέρει κεφάλι μέχρι τότε, μπορώ να βρω τις εξισώσεις.

$$\mathrm{p}(\theta|\mathrm{D}^1) = \frac{\theta^1(1-\theta)^0 \cdot 30\theta(1-\theta)^4}{\int_0^1 \theta^1(1-\theta)^0 \cdot 30\theta(1-\theta)^4\,d\theta} = \frac{30\theta^2(1-\theta)^4}{2/7} = 105\theta^2(1-\theta)^4$$

$$\mathrm{p}(\theta|\mathrm{D}^5) = \frac{\theta^4(1-\theta)^1 \cdot 30\theta(1-\theta)^4}{\int_0^1 \theta^4(1-\theta)^1 \cdot 30\theta(1-\theta)^4\,d\theta} = \frac{30\theta^2(1-\theta)^4}{5/462} = 2772\theta^5(1-\theta)^5$$

$$\mathrm{p}(\theta|\mathrm{D}^{10}) = \frac{\theta^7(1-\theta)^3 \cdot 30\theta(1-\theta)^4}{\int_0^1 \theta^7(1-\theta)^3 \cdot 30\theta(1-\theta)^4\,d\theta} == 102960\theta^8(1-\theta)^7$$

Οι γραφικές τους παραστάσεις σε κοινό διάγραμμα είναι:



Εικόνα 4: Γραφικές παραστάσεις άσκηση 5Β

Ερώτημα Γ: Να βρεθεί (αριθμητικά) το $p(x = \gamma|D^{10})$ μετά τη 10η ρίψη.

$$\int_0^1 (1-\theta)\mathrm{p}(\theta|\mathrm{D}^{10})\,d\theta = \frac{8}{17} \approx 0{,}47$$