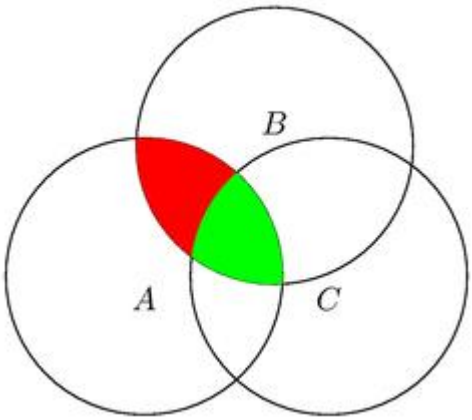
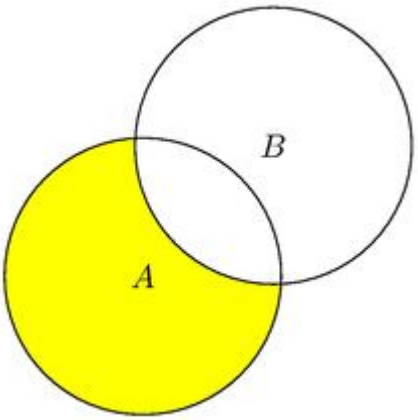
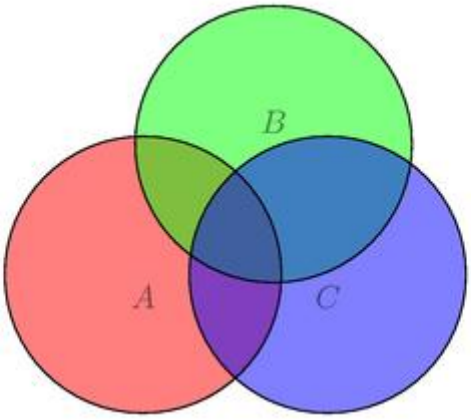
Εργασία 1 στην Αναγνώριση προτύπων







Χατζησάββας Χρήστος

Ακαδημαϊκό έτος 2023-2024

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΤΜΗΜΑ ΗΜ & ΜΥ

Άσκηση 1:

Α) Από την εκφώνηση προκύπτει πως P(n) = 0.6 αφού πρέπει P(s) + P(m) + P(n) = 1. Προκειμένου να βρούμε την απόφαση που θα πάρει το σύστημα θα αξιοποιήσουμε τον κανόνα απόφασης του Bayes.

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

Για D = 1:

p(D=1) = p(D=1|norm) * P(norm) + p(D=1|spam) * P(spam) + p(D=1|mal) * P(mal) =

0.5*0.6 + 0.05*0.3 + 0.02*0.1 = 0.317

P(norm)|D=1) = p(D=1|norm) * P(norm) / p(D=1) = 0.5 * 0.6 / 0.317 = 0.94637

P(spam)|D=1) = p(D=1|spam) * P(spam) / p(D=1) = 0.05 * 0.3 / 0.317 = 0.04731

P(mal)|D=1) = p(D=1|mal) * P(mal) / p(D=1) = 0.02 * 0.1 / 0.317 = 0.0063

Για D = 2:

p(D=2) = p(D=2|norm) * P(norm) + p(D=2|spam) * P(spam) + p(D=2|mal) * P(mal) =

0.23*0.6 + 0.15*0.3 + 0.13*0.1 = 0.196

P(norm)|D=2) = p(D=2|norm) * P(norm) / p(D=2) = 0.23 * 0.6 / 0.196 = 0.70408

P(spam)|D=2) = p(D=2|spam) * P(spam) / p(D=2) = 0.15 * 0.3 / 0.196 = 0.22959

P(mal)|D=2) = p(D=2|mal) * P(mal) / p(D=2) = 0.13 * 0.1 / 0.196 = 0.06632

Για D = 3:

p(D=3) = p(D=3|norm) * P(norm) + p(D=3|spam) * P(spam) + p(D=3|mal) * P(mal) =

0.16*0.6 + 0.4*0.3 + 0.15*0.1 = 0.231

P(norm)|D=3) = p(D=3|norm) * P(norm) / p(D=3) = 0.16 * 0.6 / 0.231 = 0.41558

P(spam)|D=3) = p(D=3|spam) * P(spam) / p(D=3) = 0.4 * 0.3 / 0.231 = 0.51948

P(mal)|D=3) = p(D=3|mal) * P(mal) / p(D=3) = 0.15 * 0.1 / 0.231 = 0.06493

Για D = 4:

p(D=4) = p(D=4|norm) * P(norm) + p(D=4|spam) * P(spam) + p(D=4|mal) * P(mal) =

0.1*0.6 + 0.3*0.3 + 0.3*0.1 = 0.18

P(norm)|D=4) = p(D=4|norm) * P(norm) / p(D=4) = 0.1 * 0.6 / 0.18 = 0.33333

P(spam)|D=4) = p(D=4|spam) * P(spam) / p(D=4) = 0.3 * 0.3 / 0.18 = 0.5

P(mal)|D=4) = p(D=4|mal) * P(mal) / p(D=4) = 0.3 * 0.1 / 0.18 = 0.1667

Για D = 5:

p(D=5) = p(D=5|norm) * P(norm) + p(D=5|spam) * P(spam) + p(D=5|mal) * P(mal) =

0.01*0.6 + 0.1*0.3 + 0.4*0.1 = 0.076

P(norm)|D=5) = p(D=5|norm) * P(norm) / p(D=5) = 0.01 * 0.6 / 0.076 = 0.07894

P(spam)|D=5) = p(D=5|spam) * P(spam) / p(D=5) = 0.1 * 0.3 / 0.076 = 0.39473

P(mal)|D=5) = p(D=5|mal) * P(mal) / p(D=5) = 0.4 * 0.1 / 0.076 = 0.52631

Γενικά, αν έχουμε δύο κατηγορίες ω1 και ω2 επιλέγουμε την ω1 εάν $P(\omega_1|x) \geq P(\omega_2|x)$.

Με βάση τα παραπάνω δεδομένα το σύστημα θα πάρει τις εξής αποφάσεις **βασιζόμενο στην μεγαλύτερη εκ των υστέρων πιθανότητα (posterior)** για κάθε περίπτωση:

D = 1 → Normal

D = 2 → Normal

D = 3 → Spam

D = 4 → Spam

D = 5 → Malicious

B) Για το ολικό σφάλμα ταξινόμησης έχουμε:

$$\boldsymbol{P_{e,total} = 1 - \sum_{i} (\max_k(P(\omega_k, x_i)))}$$

Και ισχύει ότι: $p(x,\omega_j) = p(x|\omega_j)P(\omega_j)$

Έτσι, η εξίσωση γίνεται:

$$\boldsymbol{P_{e,total} = 1 - \sum_{i} (\max_k \left(p(x|\omega_j)P(\omega_j)\right))}$$

Κάνοντας αντικατάσταση με τα δεδομένα της άσκησης:

$\mathbf{P_{e,total}}$ = 1 - (0.3 + 0.138 + 0.12 + 0.09 + 0.04) = **0.312**

Γ) Προκειμένου να υπολογίσουμε το ολικό σφάλμα ταξινόμησης για ισοπίθανες a priori πιθανότητες (1/3) θα βρούμε τα γινόμενα p(D|class)P(class) για κάθε περίπτωση.

Για D = 1:

p(D=1|norm) * P(norm) = 0.5 * 1/3 = 0.166

p(D=1|spam) * P(spam) = 0.05 * 1/3 = 0.0166

p(D=1|mal) * P(mal) = 0.02 * 1/3 = 0.0066

Για D = 2:

p(D=2|norm) * P(norm) = 0.23 * 1/3 = 0.0766

p(D=2|spam) * P(spam) = 0.15 * 1/3 = 0.05

p(D=2|mal) * P(mal) = 0.13 * 1/3 = 0.0433

Για D = 3:

p(D=3|norm) * P(norm) = 0.16 * 1/3 = 0.0533

p(D=3|spam) * P(spam) = 0.4 * 1/3 = 0.1333

p(D=3|mal) * P(mal) = 0.15 * 1/3 = 0.05

Για D = 4:

p(D=4|norm) * P(norm) = 0.1 * 1/3 = 0.03333

p(D=4|spam) * P(spam) = 0.3 * 1/3 = 0.1

p(D=4|mal) * P(mal) = 0.3 * 1/3 = 0.1

Για D = 5:

p(D=5|norm) * P(norm) = 0.01 * 1/3 = 0.00333

p(D=5|spam) * P(spam) = 0.1 * 1/3 = 0.0333

p(D=5|mal) * P(mal) = 0.4 * 1/3 = 0.1333

Συνεπώς, με τον τύπο που δείξαμε και στο ερώτημα β) αντικαθιστώντας προκύπτει:

$\mathbf{P_{e,total}}$ = 1 - (0.166 + 0.0766 + 0.1333 + 0.1 + 0.1333) = **0.39**

Άσκηση 2:

Α) Γνωρίζουμε πως προκειμένου να ταξινομήσουμε σε μία από τις δύο κλάσεις έστω στη ω1 ισχύει ότι:

$$\frac{p(x|\omega_1)}{p(x|\omega_2)} \geq \frac{\lambda_{12}-\lambda_{22}}{\lambda_{21}-\lambda_{11}} * \frac{P(\omega_2)}{P(\omega_1)}$$

Από την εκφώνηση γνωρίζουμε τα λ_{ij} καθώς και για τα $p(x|\omega_1)$ και $p(x|\omega_2)$. Πιο αναλυτικά η παραπάνω εξίσωση γίνεται:

$$\frac{p(x|\omega_1)}{p(x|\omega_2)} \geq \frac{1}{3} * \frac{\frac{3}{4}}{\frac{1}{4}}, \qquad \alpha\varphi o\acute{\upsilon}\ P(\omega_1)+P(\omega_2)=1$$

Άρα έχουμε ότι $\frac{p(x|\omega_1)}{p(x|\omega_2)} \geq 1$

Έχουμε επίσης ότι $p(x|\omega_1) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{(\chi-\mu_1)^2}{2{\sigma_1}^2}},\ \acute{o}\pi o\upsilon\ \mu_1 = 2\ \kappa\alpha\iota\ {\sigma_1}^2 = 0.5$

Αντίστοιχα, $p(x|\omega_2) = \frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{(\chi-\mu_2)^2}{2{\sigma_2}^2}},\ \acute{o}\pi o\upsilon\ \mu_2 = 1.8\ \kappa\alpha\iota\ {\sigma_2}^2 = 0.2$

Αντικαθιστώντας προκύπτει,

$$\frac{\frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{(\chi-\mu_1)^2}{2{\sigma_1}^2}}}{\frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{(\chi-\mu_2)^2}{2{\sigma_2}^2}}} \geq 1 \Rightarrow \frac{\sigma_2 e^{-\frac{(\chi-\mu_1)^2}{2{\sigma_1}^2}+\frac{(\chi-\mu_2)^2}{2{\sigma_2}^2}}}{\sigma_1} \geq 1$$

Προκειμένου να απλοποιήσουμε το κλάσμα λογαριθμίζουμε και τα δύο μέλη

$$\ln\left(\frac{\sigma_2 e^{-\frac{(\chi-\mu_1)^2}{2{\sigma_1}^2}+\frac{(\chi-\mu_2)^2}{2{\sigma_2}^2}}}{\sigma_1}\right) \geq 1 \Rightarrow \ln\left(\frac{\sigma_2}{\sigma_1}\right) - \frac{(\chi-\mu_1)^2}{2{\sigma_1}^2} + \frac{(\chi-\mu_2)^2}{2{\sigma_2}^2} \geq 0$$

$$\ln\left(\frac{\sqrt{0.2}}{\sqrt{0.5}}\right) - (\chi-2)^2 + \frac{(\chi-1.8)^2}{0.4} \geq 0 \Longrightarrow -(\chi-2)^2 + \frac{(\chi-1.8)^2}{0.4} \geq 0.45814536593$$

Η παραπάνω ανίσωση γίνεται $1.5x^2 - 5\text{x} + 3.6419 \geq 0$ και έχει ρίζες τις x1 = 2.258 και x2 = 1.075. Η συνάρτηση αυτή παίρνει θετικές τιμές στις περιοχές αριστερά και δεξιά των ρίζων ενώ μεταξύ αυτών παίρνει αρνητικές τιμές. Επομένως γίνεται επιλογή του ω1 όταν:

$$\boldsymbol{x\epsilon(-\infty, 1.075) \cup (2.258, +\infty)}$$

Ενώ επιλέγουμε το ω2 όταν:

$$\boldsymbol{x\epsilon(1.075, 2.258)}$$

Για τον προσδιορισμό του ολικού κόστους θα χρησιμοποιήσουμε τον τύπο που δίνεται στις διαφάνειες:

$$C = P_1\left[\lambda_{11}\int_{R_1} f_{x_1}(x_1/H_1)dx + \lambda_{01}\int_{R_0} f_{x_1}(x_1/H_1)dx\right] + (1-P_1)\left[\lambda_{00}\int_{R_0} f_{x_1}(x_1/H_0)dx + \lambda_{10}\int_{R_1} f_{x_1}(x_1/H_0)dx\right]$$

Τα ολοκληρώματα που βρίσκονται μέσα στις παρενθέσεις αντιπροσωπεύουν την αθροιστική συνάρτηση πιθανότητας αφού γνωρίζουμε ότι f(x) = $\frac{1}{\sigma_1\sqrt{2\pi}}e^{-\frac{(\chi-\mu_1)^2}{2{\sigma_1}^2}}$ και ότι $f(x) = \frac{dF_x(x)}{dx}$

Για την CDF ισχύει ότι:

$$P[a < X \leq b] = F_x(b) - F_x(a)$$

Δεδομένου ότι έχουμε κανονική κατανομή ισχύει: $F_x(x) = P(X \leq x) = \frac{1}{2}[1 + \text{erf}\left(\frac{\chi-\mu}{\sigma\sqrt{2}}\right)]$

Θα πρέπει να υπολογίσουμε την πιθανότητα το χ να βρίσκεται στον χώρο του ω1 ενώ ανήκει στο ω1, αλλά και την πιθανότητα να βρίσκεται στον χώρο του ω2 και να ανήκει στο ω1. Αντίστοιχα, πρέπει να υπολογιστούν και οι πιθανότητες για το χ να βρίσκεται στον χώρο του ω2 ενώ ανήκει στο ω2 αλλά και την πιθανότητα να βρίσκεται στον χώρο του ω1 και να ανήκει στο ω2.

Έχουμε,

$$P(1.075 < x < 2.258|\omega 1) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{2.258-2}{\sqrt{0.5}\sqrt{2}}\right)\right] - \frac{1}{2}\left[1 + \text{erf}\left(\frac{1.075-2}{\sqrt{0.5}\sqrt{2}}\right)\right] = 0.547$$

$$P(1.075 < x < 2.258|\omega 2) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{2.258-1.8}{\sqrt{0.2}\sqrt{2}}\right)\right] - \frac{1}{2}\left[1 + \text{erf}\left(\frac{1.075-1.8}{\sqrt{0.2}\sqrt{2}}\right)\right]$$
$$= 0.7946$$

Έτσι, $P(x\epsilon(-\infty, 1.075) \cup (2.258, +\infty)|\omega 1) = 1 - P(1.075 < x < 2.258|\omega 1) = 0.453$

και $P(x\epsilon(-\infty, 1.075) \cup (2.258, +\infty)|\omega 2) = 1 - P(1.075 < x < 2.258|\omega 2) = 0.2054$

Αντικαθιστώντας στον τύπο του κόστους έχουμε:

C = 0.25(0 * 0.453 + 3 * 0.547) + 0.75(1 * 0.2054 + 0 * 0.7946) = 0.5643

B) Ακολουθεί η διαδικασία υπολογιστικά:

```
import numpy as np

#define a priori probabilities as given
apriori = [1/4,3/4]

#roots found in the first question
roots = [1.075,2.258]

#define number of samples
num_of_samples = 10000

#split and create based on apriori probabilities samples for w1 and w2
w1_samples_num = apriori[0] * num_of_samples #w1 samples
w2_samples_num = apriori[1] * num_of_samples #w2 samples

#create random samples using normal distribution
w1 = np.random.normal(2,np.sqrt(0.5),int(w1_samples_num))
w2 = np.random.normal(1.8,np.sqrt(0.2),int(w2_samples_num))

#counters initialization to count correct and wrong classifications
recognised_wrong_w1 = 0
recognised_wrong_w2 = 0
recognised_correct_w1 = 0
recognised_correct_w2 = 0

#check for w1 how many samples recognised correctly
for i in w1:
  if i > roots[0] and i < roots[1]:
    recognised_wrong_w1 = recognised_wrong_w1 + 1
  else:
    recognised_correct_w1 = recognised_correct_w1 + 1

#check for w2 how many samples recognised correctly
for j in w2:
  if j > roots[0] and j < roots[1]:
    recognised_correct_w2 = recognised_correct_w2  + 1
  else:
    recognised_wrong_w2 = recognised_wrong_w2 + 1

#check in given samples the probabilities
correct_w1 = recognised_correct_w1 / w1_samples_num
correct_w2 = recognised_correct_w2 / w2_samples_num
wrong_w1 = recognised_wrong_w1 / w1_samples_num
wrong_w2 = recognised_wrong_w2 / w2_samples_num

#define cost as described in the report
cost = apriori[0] * ((0 * correct_w1) + (3 * wrong_w1)) + \
apriori[1] * ((1 * wrong_w2) + (0* correct_w2))

print(f"For w1 successfully classified {round(correct_w1*100,2)} %")
print(f"For w2 successfully classified {round(correct_w2*100,2)} %")
print(f"The cost is {cost}")
```

Εκτελώντας τον παραπάνω κώδικα έχουμε:

```
For w1 successfully classified 45.76 %
For w2 successfully classified 79.45 %
The cost is 0.5609
```

Αρχικά, καθορίζουμε τον αριθμό των δειγμάτων που θα δημιουργηθούν για κάθε κλάση βασιζόμενοι στις δοθείσες a priori πιθανότητες. Έπειτα, δημιουργούμε τυχαία δείγματα που ακολουθούν την κανονική κατανομή με την εντολή np.random.normal η οποία παίρνει ως ορίσματα την μέση τιμή, την διακύμανση και τον αριθμό των δειγμάτων. Χρησιμοποιώντας τα διαστήματα που βρήκαμε στο πρώτο ερώτημα, διατρέχοντας όλα τα τυχαία δείγματα μπορούμε να τα κατηγοριοποιήσουμε στην κλάση ω1 ή ω2 ανάλογα με την θέση που βρίσκονται. Εν τέλει, βλέπουμε πως το αποτέλεσμα για το κόστος που βρήκαμε με την μαθηματική ανάλυση είναι πολύ κοντά με αυτό που βρέθηκε υπολογιστικά (0.5643 και 0.5609). Αξίζει να σημειωθεί πως επειδή για κάθε εκτέλεση τα τυχαία δείγματα αλλάζουν, η τιμή του κόστους με την υπολογιστική προσέγγιση αλλάζει, παρόλα αυτά είναι πολύ κοντά στην θεωρητική.

Άσκηση 3:

Α)

```python
import math
import numpy as np

def discriminant_func(x,m,dim,covariance_matrix,apriori):
  var1 = x-m
  if dim != 1:
    det_cov_mat = np.linalg.det(covariance_matrix)
    return -0.5*np.matmul(np.matmul(np.transpose(var1),np.linalg.inv(covariance_matrix)),var1) \
    - (dim/2)*math.log(2*math.pi) - 0.5*math.log(det_cov_mat) + math.log(apriori)
  else:
    det_cov_mat = abs(covariance_matrix)
    return -0.5*(var1)*(covariance_matrix)**(-1)*var1 \
    - (dim/2)*math.log(2*math.pi) - 0.5*math.log(det_cov_mat) + math.log(apriori)

def euclidean_distance(x,y,dim):
  if dim != 1:
    return np.sqrt(np.matmul(np.transpose(x-y),(x-y)))
  else:
    return abs(x-y)

def mahalanobis_distance(x,m,covariance_matrix,dim):
  var1 = x-m
  if dim != 1:
    return np.sqrt(np.matmul(np.matmul(np.transpose(var1),np.linalg.inv(covariance_matrix)),var1))
  else:
    return np.sqrt(var1*(covariance_matrix**(-1))*var1)
```

Για την Ευκλείδεια απόσταση χρησιμοποιήθηκε ο παρακάτω τύπος από την θεωρία:

$$d_\varepsilon(x,y) = ((x-y)^T(x-y))^{1/2}$$

Για την Mahalanobis distance αντίστοιχα έχουμε:

$$d_m = ((x - \mu_i)^T \Sigma^{-1} (x - \mu_i))^{1/2}$$

Για την υλοποίηση των συναρτήσεων χρειάστηκε να διαχωρίσουμε την περίπτωση για την οποία έχουμε μία διάσταση (dim = 1). Αυτό συμβαίνει επειδή οι συναρτήσεις np.linalg.det(), και np.linalg.inv() δεν μπορούν να δεχθούν ως ορίσματα μονοδιάστατους πίνακες. Οι πολλαπλασιασμοί πινάκων γίνονται με την συνάρτηση np.matmul() η οποία λαμβάνει ως ορίσματα τους πίνακες με την σειρά την οποία φαίνονται και στην εκάστοτε εξίσωση.

Β)

```
import numpy as np

# Specify the file path
file_path = '/content/data.csv'

#import data in an array skiping the first line
total_data = np.genfromtxt(file_path, delimiter = ',', skip_header=1)

#split data for each class
data_w1 = total_data[0:39,0:3]
data_w2 = total_data[39:79,0:3]
data_w3 = total_data[79:,0:3]
class_data = total_data[:,3]

#for x1
x1_mean_w1 = np.mean(data_w1[:,0])
x1_cov_w1 = np.cov(data_w1[:,0],rowvar = False)

x1_mean_w2 = np.mean(data_w2[:,0])
x1_cov_w2 = np.cov(data_w2[:,0],rowvar = False)

x1_mean_w3 = np.mean(data_w3[:,0])
x1_cov_w3 = np.cov(data_w3[:,0],rowvar = False)
#for x1,x2
x1x2_mean_w1 = np.mean(data_w1[:,0:2],axis = 0)
x1x2_cov_w1 = np.cov(data_w1[:,0:2],rowvar = False)

x1x2_mean_w2 = np.mean(data_w2[:,0:2],axis = 0)
x1x2_cov_w2 = np.cov(data_w2[:,0:2],rowvar = False)

x1x2_mean_w3 = np.mean(data_w3[:,0:2],axis = 0)
x1x2_cov_w3 = np.cov(data_w3[:,0:2],rowvar = False)

#for x1,x2,x3
x1x2x3_mean_w1 = np.mean(data_w1[:,0:3],axis = 0)
x1x2x3_cov_w1 = np.cov(data_w1[:,0:3],rowvar = False)

x1x2x3_mean_w2 = np.mean(data_w2[:,0:3],axis = 0)
x1x2x3_cov_w2 = np.cov(data_w2[:,0:3],rowvar = False)

x1x2x3_mean_w3 = np.mean(data_w3[:,0:3],axis = 0)
x1x2x3_cov_w3 = np.cov(data_w3[:,0:3],rowvar = False)
```

Το ερώτημα αυτό δεδομένου ότι γίνεται αποκλειστικά με την χρήση κώδικα αξίζει να περιγράψουμε το πως λειτουργεί. Αρχικά, διαβάζουμε το αρχείο με τα δεδομένα και το εισάγουμε σε έναν NumPy array. Με το όρισμα skip_header = 1 αγνοούμε την πρώτη γραμμή του αρχείου των δεδομένων μιας και δείχνει απλά τι αντιπροσωπεύει κάθε στήλη. Στη συνέχεια

βρίσκουμε για τα διαθέσιμα κάθε φορά χαρακτηριστικά την μέση τιμή και την διακύμανση αφού πρώτα έχουμε χωρίσει σε 3 υποπίνακες τα δείγματα των 3 κλάσεων Για τον υπολογισμό της μέσης τιμής και της συνδιακύμανσης χρησιμοποιούνται οι συναρτήσεις της βιβλιοθήκης NumPy, np.mean() και np.cov() αντίστοιχα.

Οι αντίστοιχοι τύποι δεδομένου ότι έχουμε κανονική κατανομή από την θεωρία είναι:

$$\mu = \frac{1}{n}\sum_{k=1}^{n} x_k \ \ \kappa\alpha\iota \ \ \sigma^2 = \frac{1}{n}\sum_{k=1}^{n}(x_k - \mu)^2$$

Γ)

```
#store row size for each class
data_w1_rows = data_w1.shape[0]
data_w2_rows = data_w2.shape[0]
data_w3_rows = data_w3.shape[0]

#for class1
g1 = np.zeros((data_w1_rows,1))
g2 = np.zeros((data_w1_rows,1))
wrong_pred = 0
for i in range(0,data_w1_rows):
  g1[i,0] = discriminant_func(data_w1[i,0],x1_mean_w1,1,x1_cov_w1,0.5)
  g2[i,0] = discriminant_func(data_w1[i,0],x1_mean_w2,1,x1_cov_w2,0.5)
  if(g1[i,0] < g2[i,0]):
    wrong_pred = wrong_pred + 1

#for class2
g3 = np.zeros((data_w2_rows,1))
g4 = np.zeros((data_w2_rows,1))
for i in range(0,data_w2_rows):
  g3[i,0] = discriminant_func(data_w2[i,0],x1_mean_w1,1,x1_cov_w1,0.5)
  g4[i,0] = discriminant_func(data_w2[i,0],x1_mean_w2,1,x1_cov_w2,0.5)
  if(g3[i,0] > g4[i,0]):
    wrong_pred = wrong_pred + 1

tot_class_error = wrong_pred/(data_w1_rows + data_w2_rows)
print(f"Classification Error using only x1 is {tot_class_error}")
```

Γνωρίζοντας τις a priori πιθανότητες και χρησιμοποιώντας την συνάρτηση διάκρισης **μόνο για το πρώτο χαρακτηριστικό** βρίσκουμε τα g1 και g2 για όλα τα δείγματα της πρώτης κλάσης. Πιο συγκεκριμένα, στο g1 χρησιμοποιούμε τα δείγματα της πρώτης κλάσης με μέση τιμή και συνδιακύμανση από την κλάση 1. Στο g2 χρησιμοποιούμε πάλι τα δείγματα της πρώτης κλάσης όμως βάζουμε την μέση τιμή και την συνδιακύμανση της δεύτερης κλάσης. Προκειμένου να κατηγοριοποιήσουμε με βάση τον ταξινομητή που επιλέξαμε, για κάθε στοιχείο που πραγματικά ανήκει στην κλάση 1, συγκρίνουμε τα g1 και g2. Κανονικά, έτσι όπως τα έχουμε ορίσει πρέπει το g1 > g2 για κάθε στοιχείο της κλάσης ω1. Όμως σε περίπτωση που το g2 > g1 τότε έχουμε λάθος πρόβλεψη. Γι' αυτό τον λόγο έχουμε έναν μετρητή που μετράει τις φορές που κάποιο στοιχείο ταξινομήθηκε λανθασμένα. Αντίστοιχα, βρίσκουμε τα g3,g4 για τα

στοιχεία της κλάσης 2 και για την μέση τιμή και την συνδιακύμανση κάνουμε ότι και πριν. Για την ταξινόμηση συγκρίνουμε τα g3 και g4.

Εκτελώντας τον κώδικα βρίσκουμε ότι:

```
Classification Error using only x1 is 0.4050632911392405
```

Ως ταξινομητή επιλέγουμε την συνάρτηση διάκρισης διότι αξιοποιεί όσα δεδομένα διαθέτουμε. Πιο συγκεκριμένα, χρησιμοποιεί τους πίνακες συνδιακύμανσης και την μέση τιμή που υπολογίστηκαν στο ερώτημα β καθώς και τις δοθείσες a priori πιθανότητες για τις κλάσεις. Οι άλλοι 2 ταξινομητές χρησιμοποιούν μερικά από αυτά τα στοιχεία.

Δ) Χρησιμοποιώντας τώρα τα στοιχεία x1,x2 και έπειτα τα x1,x2,x3 έχουμε:

```
#for x1,x2

#for class1
g1 = np.zeros((data_w1_rows,1))
g2 = np.zeros((data_w1_rows,1))
wrong_pred = 0
for i in range(0,data_w1_rows):
  g1[i,0] = discriminant_func(data_w1[i,0:2],x1x2_mean_w1,2,x1x2_cov_w1,0.5)
  g2[i,0] = discriminant_func(data_w1[i,0:2],x1x2_mean_w2,2,x1x2_cov_w2,0.5)
  if(g1[i,0] < g2[i,0]):
    wrong_pred = wrong_pred + 1

#for class2
g3 = np.zeros((data_w2_rows,1))
g4 = np.zeros((data_w2_rows,1))
for i in range(0,data_w2_rows):
  g3[i,0] = discriminant_func(data_w2[i,0:2],x1x2_mean_w1,2,x1x2_cov_w1,0.5)
  g4[i,0] = discriminant_func(data_w2[i,0:2],x1x2_mean_w2,2,x1x2_cov_w2,0.5)
  if(g3[i,0] > g4[i,0]):
    wrong_pred = wrong_pred + 1

tot_class_error = wrong_pred/(data_w1_rows + data_w2_rows)
print(f"Classification Error using x1,x2 is {tot_class_error}")

#for x1,x2,x3

#for class1
g1 = np.zeros((data_w1_rows,1))
g2 = np.zeros((data_w1_rows,1))
wrong_pred = 0
for i in range(0,data_w1_rows):
  g1[i,0] = discriminant_func(data_w1[i,0:3],x1x2x3_mean_w1,3,x1x2x3_cov_w1,0.5)
  g2[i,0] = discriminant_func(data_w1[i,0:3],x1x2x3_mean_w2,3,x1x2x3_cov_w2,0.5)
  if(g1[i,0] < g2[i,0]):
    wrong_pred = wrong_pred + 1

#for class2
g3 = np.zeros((data_w2_rows,1))
g4 = np.zeros((data_w2_rows,1))
for i in range(0,data_w2_rows):
  g3[i,0] = discriminant_func(data_w2[i,0:3],x1x2x3_mean_w1,3,x1x2x3_cov_w1,0.5)
  g4[i,0] = discriminant_func(data_w2[i,0:3],x1x2x3_mean_w2,3,x1x2x3_cov_w2,0.5)
  if(g3[i,0] > g4[i,0]):
    wrong_pred = wrong_pred + 1

tot_class_error = wrong_pred/(data_w1_rows + data_w2_rows)
print(f"Classification Error using x1,x2,x3 is {tot_class_error}")
```

Στο ερώτημα αυτό όπως και πριν, ανάλογα με τον αριθμό των χαρακτηριστικών και βασιζόμενοι στις δοθείσες a priori πιθανότητες υπολογίζουμε την τιμή της συνάρτησης διάκρισης για όλα τα στοιχεία των κλάσεων ω1 και ω2. Αξίζει να σημειωθεί πως στο συγκεκριμένο ερώτημα πρέπει να αλλάξουμε την τιμή της διάστασης που υπάρχει ως όρισμα στην discriminant_func αφού ανάλογα με των αριθμό των χαρακτηριστικών αλλάζει και η διάσταση. Για την ταξινόμηση ισχύει ακριβώς ότι και στο προηγούμενο ερώτημα.

Ακολουθούν τα αποτελέσματα από την εκτέλεση του κώδικα.

```
Classification Error using x1,x2 is 0.379746835443038
Classification Error using x1,x2,x3 is 0.21518987341772153
```

Για το συγκεκριμένο σετ δεδομένων παρατηρούμε πως όσο αυξάνουμε το μέγεθος του προβλήματος οδηγούμαστε σε μικρότερο σφάλμα ταξινόμησης. Αυτό είναι δείγμα του ότι τα επιπλέον στοιχεία σε αυτήν την περίπτωση δεν εισάγουν θόρυβο ώστε να προκληθεί μεγαλύτερο σφάλμα ταξινόμησης.(x1 → 0.405 | x1, x2 → 0.3797 | x1, x2, x3 → 0.2151)

E) Λαμβάνοντας υπόψη την αλλαγή στις a priori πιθανότητες, πλέον εισάγονται και τα δείγματα της κλάσης ω3. Ο κώδικας ακολουθεί.

```
#for x1,x2,x3

#for class1
g1 = np.zeros((data_w1_rows,1))
g2 = np.zeros((data_w1_rows,1))
g3 = np.zeros((data_w1_rows,1))
wrong_pred = 0
for i in range(0,data_w1_rows):
  g1[i,0] = discriminant_func(data_w1[i,0:3],x1x2x3_mean_w1,3,x1x2x3_cov_w1,0.8)
  g2[i,0] = discriminant_func(data_w1[i,0:3],x1x2x3_mean_w2,3,x1x2x3_cov_w2,0.1)
  g3[i,0] = discriminant_func(data_w1[i,0:3],x1x2x3_mean_w3,3,x1x2x3_cov_w3,0.1)
  if(g1[i,0] < g2[i,0]) or (g1[i,0] < g3[i,0]):
    wrong_pred = wrong_pred + 1

#for class2
g4 = np.zeros((data_w2_rows,1))
g5 = np.zeros((data_w2_rows,1))
g6 = np.zeros((data_w2_rows,1))
for i in range(0,data_w2_rows):
  g4[i,0] = discriminant_func(data_w2[i,0:3],x1x2x3_mean_w1,3,x1x2x3_cov_w1,0.8)
  g5[i,0] = discriminant_func(data_w2[i,0:3],x1x2x3_mean_w2,3,x1x2x3_cov_w2,0.1)
  g6[i,0] = discriminant_func(data_w2[i,0:3],x1x2x3_mean_w3,3,x1x2x3_cov_w3,0.1)
  if(g5[i,0] < g4[i,0]) or (g5[i,0] < g6[i,0]):
    wrong_pred = wrong_pred + 1
```

```
#for class3
g7 = np.zeros((data_w3_rows,1))
g8 = np.zeros((data_w3_rows,1))
g9 = np.zeros((data_w3_rows,1))
for i in range(0,data_w3_rows):
  g7[i,0] = discriminant_func(data_w3[i,0:3],x1x2x3_mean_w1,3,x1x2x3_cov_w1,0.8)
  g8[i,0] = discriminant_func(data_w3[i,0:3],x1x2x3_mean_w2,3,x1x2x3_cov_w2,0.1)
  g9[i,0] = discriminant_func(data_w3[i,0:3],x1x2x3_mean_w3,3,x1x2x3_cov_w3,0.1)
  if(g9[i,0] < g8[i,0]) or (g9[i,0] < g7[i,0]):
    wrong_pred = wrong_pred + 1

tot_class_error = wrong_pred/(data_w1_rows + data_w2_rows + data_w3_rows)
print(f"Classification Error using x1,x2,x3 is {tot_class_error}")
```

Έπειτα από την εκτέλεση για το σφάλμα έχουμε ότι:

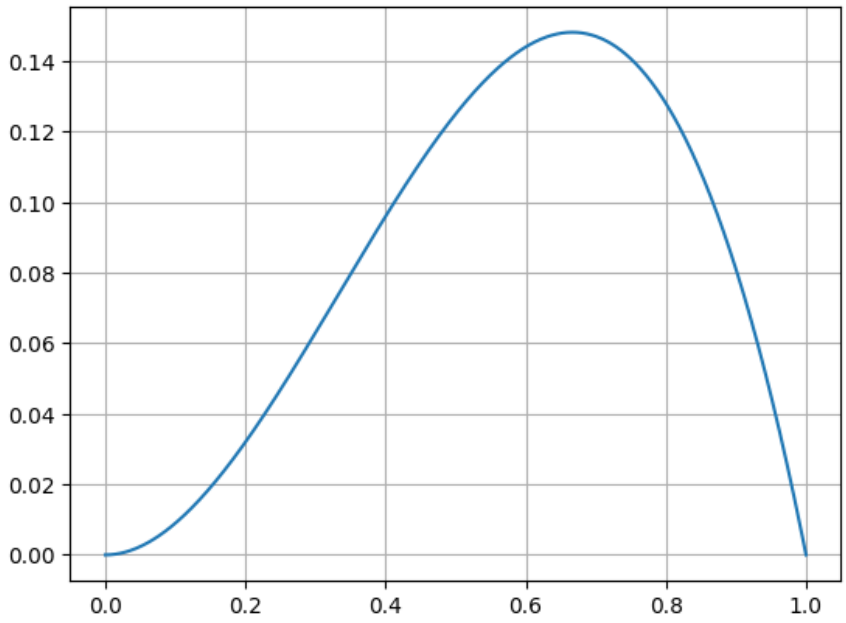
```
Classification Error using x1,x2,x3 is 0.55
```

Στο ερώτημα αυτό, δεδομένου ότι υπάρχει και 3η κλάση το κριτήριο για την κατηγοριοποίηση διαφοροποιείται ελαφρώς. Πιο συγκεκριμένα για τα δείγματα της κάθε κλάσης υπολογίζουμε 3 τιμές της συνάρτησης διάκρισης. Συγκρίνουμε την τιμή του g που έχει ως μέση τιμή και διασπορά αυτό της κλάσης στην οποία ανήκει πραγματικά το δείγμα με τις άλλες 2 τιμές για την discriminant_func στις οποίες έχουμε εισάγει ως ορίσματα την μέση τιμή και την διασπορά από τις άλλες κλάσεις όπως υπολογίστηκαν στο ερώτημα β. Όπως φαίνεται και στον κώδικα η συνθήκη πλέον αποτελείται από 2 επιμέρους ελέγχους που οδηγούν σε λάθος ταξινόμηση εάν πληρείται τουλάχιστον ο ένας από τους δύο.

Η χρήση της Ευκλείδειας απόστασης σύμφωνα με την θεωρία γίνεται όταν τα δεδομένα σε όλες τις κλάσεις ακολουθούν Γκαουσιανή κατανομή, οι κλάσεις είναι ισοπίθανες, ο πίνακας συνδιακύμανσης είναι ο ίδιος για όλες τις κλάσεις αλλά και είναι διαγώνιος με όλα τα στοιχεία στην διαγώνιο να είναι ίσα. Η διαφοροποίηση που υπάρχει για την Mahalanobis απόσταση είναι ότι δεν απαιτεί να είναι ο πίνακας συνδιακύμανσης διαγώνιος με όλα τα στοιχεία της διαγωνίου ίσα. Τέλος, η χρήση της συνάρτησης διάκρισης είναι εφικτή και για διαφορετικές a priori πιθανότητες στις κλάσεις ενώ και οι πίνακες συνδιακύμανσης μπορεί να είναι οποιοιδήποτε.

Άσκηση 4:

Α) Αρχικά σχεδιάζουμε την συνάρτηση της πιθανότητας του σφάλματος που εξαρτάται από την a priori πιθανότητα της κλάσης ω_1, P_1.



Δεδομένου ότι το σύστημα πρέπει να σχεδιαστεί για την χειρότερη περίπτωση σφάλματος αυτή θα είναι εκεί που η συνάρτηση αυτή παρουσιάζει ολικό μέγιστο. Υπολογίζοντας με την χρήση κώδικα, βρίσκουμε πως παρουσιάζει μέγιστο για $P_1 = 0.667$ το 0.148148. Επομένως για $P_1 = 0.667$ έχουμε πιθανότητα σφάλματος $P_{error} = 0.148148$.

```
F function has maximum value: 0.14814814814814817 at x: 0.666
```

Διαφορετικά, θα μπορούσαμε να βρούμε που παρουσιάζει μέγιστο η συνάρτηση βρίσκοντας την παράγωγο της και κοιτώντας για ποια τιμή μηδενίζει.

Β) Σύμφωνα με την θεωρία, για κάθε τιμή των a priori πιθανοτήτων (π.χ. $P(\omega1) = 0.66$) υπάρχει ένα αντίστοιχο βέλτιστο όριο απόφασης. Για κάθε τέτοιο όριο, αν οι a priori πιθανότητες αλλαχθούν, η πιθανότητα σφάλματος θα αλλάξει σαν γραμμική συνάρτηση του $P(\omega1)$. Με λίγα λόγια, η πιθανότητα σφάλματος θα ακολουθεί την πορεία της εφαπτομένης που διέρχεται από το σημείο το οποίο αντιπροσωπεύει την a priori πιθανότητα για την οποία σχεδιάστηκε.

Επειδή σύμφωνα με την εκφώνηση ρυθμίσαμε το σύστημα υποθέτοντας πιθανότητα $P_1 = 0.3$, πρέπει να βρούμε την εφαπτόμενη στο σημείο αυτό ευθεία ώστε να προσδιορίσουμε την πιθανότητα σφάλματος σε περίπτωση που αλλάξει η a priori πιθανότητα.

Η συνάρτηση είναι $F(x) = x^2(1 - x)$ την οποία αν την παραγωγίσουμε δίνει:

$$F'(x) = 2x - 3x^2$$

Η τιμή της συνάρτησης F στο σημείο x = 0.3 είναι:

$$F(0.3) = 0.3^2 * (1 - 0.3) = \frac{63}{1000}$$

Η παράγωγος στο σημείο x = 0.3 είναι:

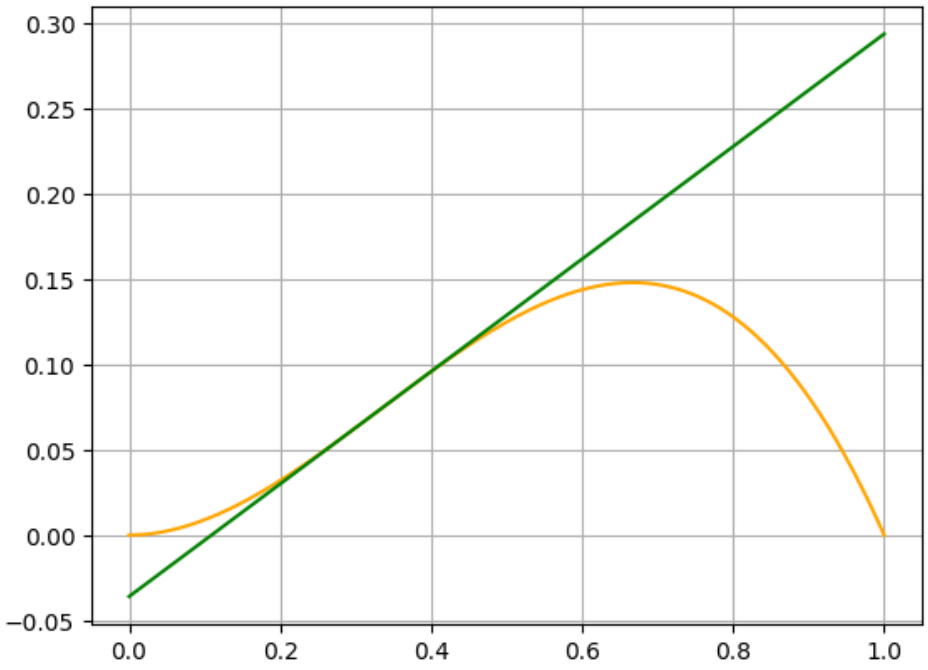
$$F'(0.3) = 2 * 0.3 - 3 * 0.3^2 = \frac{33}{100}$$

Άρα η εξίσωση της ζητούμενης ευθείας είναι:

$$y - \frac{63}{1000} = \frac{33}{100}(x - 0.3) \Leftrightarrow y = 0.33x - \frac{9}{250}$$

Κάνοντας πλέον plot και τις δύο εξισώσεις, βρίσκουμε με την χρήση κώδικα ότι η πιθανότητα σφάλματος δεδομένης της a priori πιθανότητας $P_1 = 0.7$ είναι 0.195.

```
The error probability for P1 = 0.7 is 0.19523123123123123
```



Παρατηρούμε πως για τιμές του P_1 μικρότερες του 0.1, η εφαπτομένη παίρνει αρνητικές τιμές. Αυτό οφείλεται στο γεγονός ότι η συνάρτηση σφάλματος δεν είναι κοίλη σε όλο το πεδίο ορισμού της. Παρόλα αυτά, δεν επηρεάζει το αποτέλεσμα που ζητάμε για $P_1 = 0.7$

Ακολουθεί ο κώδικας που χρησιμοποιήθηκε για την δημιουργία των διαγραμμάτων.

```python
import numpy as np
import matplotlib.pyplot as plt

#create x values
x = np.linspace(0,1,1000)

# Calculate y values using the function y = x^2 * (1-x)
y = x**2 * (1 - x)
new_line = 0.33*x - 9/250

x_max = np.argmax(y)
max = np.max(y)

# Create the plot
fig, ax = plt.subplots()
ax.plot(x, y, 'orange')
ax.plot(x, new_line, 'green')
plt.grid(True)
plt.show()

print(f"F function has maximum value: {max} at x: {x_max/1000}")
print(f"The error probability for P1 = 0.7 is {new_line[700]}")
```

Άσκηση 5:

Σύμφωνα με την θεωρία από τις διαφάνειες του μαθήματος, η πιθανότητα θ να φέρουμε κεφάλι όταν έχουμε πετάξει το νόμισμα N = 10 φορές και έχουμε φέρει {κ,γ,κ,κ,κ,γ,κ,κ,γ,κ} δηλαδή κ = 7 φορές κεφάλι είναι:

$$\hat{\theta} = \frac{k}{N} = \frac{7}{10} = 0.7$$

Άρα η πιθανότητα θ να φέρουμε κεφάλι είναι 0.7

Α) Δεδομένης της κατανομής του θ:

$$p(\theta|D^0) = A * \theta(1-\theta)^4 \, ,για \; 0 \leq \theta \leq 1$$

και γνωρίζοντας ότι για την συνάρτηση πυκνότητας πιθανότητας ισχύει:

$$\int_{-\infty}^{\infty} p(\theta|D^0)d\theta = 1$$

Αντικαθιστώντας προκύπτει:

$$\int_{-\infty}^{\infty} p(\theta|D^0)d\theta = 1 \Longrightarrow \int_{0}^{1} p(\theta|D^0)d\theta = 1 \Longrightarrow \int_{0}^{1} A * \theta(1-\theta)^4 d\theta = 1 \Longrightarrow$$

$$A * \int_0^1 (\theta^5 - 4\theta^4 + 6\theta^3 - 4\theta^2 + \theta) d\theta = 1 \Longrightarrow$$

$$A * \left[\frac{\theta^6}{6} - 4\frac{\theta^5}{5} + 6\frac{\theta^4}{4} - 4\frac{\theta^3}{3} + \frac{\theta^2}{2}\right]_0^1 = 1 \Longrightarrow A * \left[\frac{1}{6} - \frac{4}{5} + \frac{6}{4} - \frac{4}{3} + \frac{1}{2}\right] = 1 \Longrightarrow \boldsymbol{A = 30}$$

Διαφορετικά, για να βρούμε το Α μπορούμε να ανατρέξουμε στις ιδιότητες της beta distribution όπου βλέπουμε πως το Α εξαρτάται από τιμές της συνάρτησης γάμμα.

Η συνάρτηση πυκνότητας πιθανότητας (PDF) της κατανομής β είναι:

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha - 1}(1 - \chi)^{\beta - 1}$$

Για τιμές της συνάρτησης γάμμα έχουμε ότι:

$$\Gamma(n) = (n - 1)!\,, \textit{για κάθε θετικό ακέραιο } n$$

Συγκρίνοντας τον τύπο που μας δίνεται με αυτόν της θεωρίας για την κατανομή β προκύπτει ότι α = 2 και β = 5. Άρα, Γ(α + β) = Γ(7) = 6! = 720. Επιπλέον ισχύει ότι Γ(α) = Γ(2) = 1! = 1 και Γ(β) = Γ(5) = 4! = 24.

Άρα αφού $A = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \overset{\alpha=2,\beta=5}{\Longrightarrow} \boldsymbol{A = \frac{720}{24} = 30}$

Β) Από την θεωρία γνωρίζουμε για την αναδρομική εκτίμηση ότι:

$$p(\theta|D^n) = \frac{p(x_n|\theta) * p(\theta|D^{n-1})}{\int p(x_n|\theta) * p(\theta|D^{n-1}) d\theta}$$

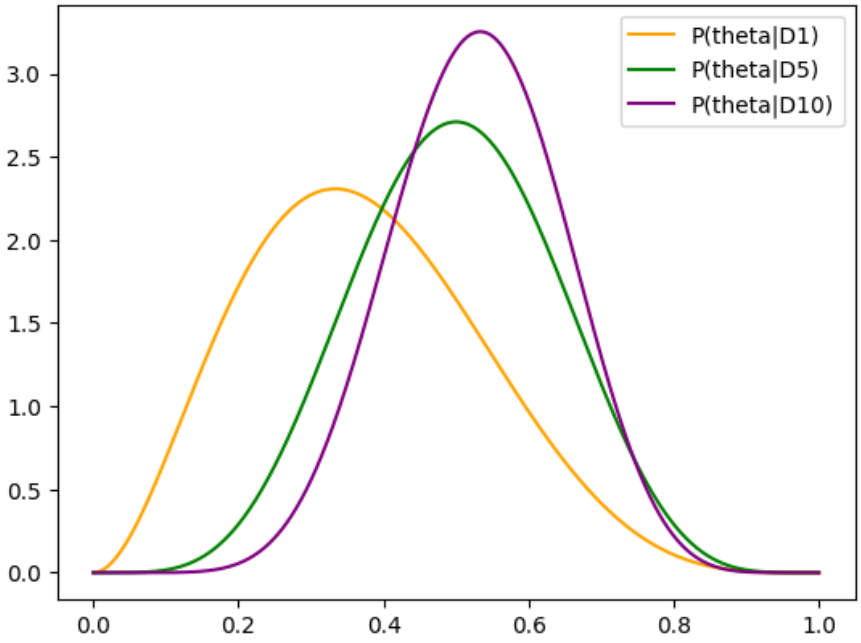
Όπου για τις ρίψεις ισχύει ότι:

$$p(x_n|\theta) = \begin{cases} \theta, & \textit{για το κεφάλι} \\ 1 - \theta, & \textit{για τα γράμματα} \end{cases}$$

Επομένως για Ν ρίψεις και κάνοντας την αντικατάσταση έχουμε:

$$p(\theta|D^N) = \frac{\theta^\kappa * (1 - \theta)^{N-k} * 30 * \theta(1 - \theta)^4}{\int \theta^\kappa * (1 - \theta)^{N-k} * 30 * \theta(1 - \theta)^4 \, d\theta}$$

Προκειμένου να σχεδιαστούν σε κοινό διάγραμμα τα $p(\theta|D^1), p(\theta|D^5), p(\theta|D^{10})$ θα χρησιμοποιήσουμε την βιβλιοθήκη matplotlib.



Για τις παραπάνω γραφικές παραστάσεις, προκειμένου να ελέγξουμε ότι είναι σωστές πρέπει το ολοκλήρωμα τους να είναι ίσο με το 1. Αυτό μπορούμε να το ελέγξουμε εύκολα μέσω του κώδικα. Πράγματι, παρατηρούμε ότι κάθε ένα από τα ολοκληρώματα είναι ίσο με το 1.

```
Integral for p_th[0] = 1.0
Integral for p_th[1] = 1.0
Integral for p_th[2] = 1.0
```

Γ) Αφού έχουμε την γραφική παράσταση της $p(\text{theta}|D^{10})$ και αναζητούμε το $p(\chi = \gamma|D^{10})$ μπορούμε εύκολα να το βρούμε κοιτώντας σε ποια τιμή η γραφική παίρνει την μέγιστη τιμή (το υλοποιούμε με την χρήση κώδικα για μεγαλύτερη ακρίβεια).

```
The maximum value is 3.251919687635525 for x = 0.533
```

Έπειτα το ζητούμενο είναι: $1 - p(\text{theta}|D^{10}) = 1 - 0.533 = 0.467$

Ακολουθεί ο κώδικας για την συγκεκριμένη άσκηση:

```python
import numpy as np
import matplotlib.pyplot as plt

#define theta distribution function as given
def theta_distr(theta1):
  return 30*theta1*(1-theta1)**4

#define p(theta|D^N) numerator
def numerator(theta1,k):
  return (theta1**k)*((1-theta1)**(N-k))*theta_distr(theta1)

#define p(theta|D^N) denominator
def denominator(theta1,k):
  return np.trapz(numerator(theta1,k),dx = 1/1000)

#define flip results given (1-> heads 0-> tails)
flips = np.array([1,0,1,1,1,0,1,1,0,1])

#create an array to store all samples for D^1,D^5,D^10
p_th = np.zeros((3, 1000))

#calculate for N = 1,5,10 p(theta|D^N)
theta = np.linspace(0,1,1000)
temp1 = 0 #temporary variable to help indicate the right array line
for N in (1,5,10):
  k = flips[0:N].sum()
  p_th[temp1, :] = numerator(theta, k) / denominator(theta, k)
  temp1 = temp1 + 1

#plot the result
fig, ax = plt.subplots()
ax.plot(theta, p_th[0], 'orange', label="P(theta|D1)")
ax.plot(theta, p_th[1], 'green', label="P(theta|D5)")
ax.plot(theta, p_th[2], 'purple', label="P(theta|D10)")
ax.legend()
plt.show()

#check if the integral is equal to 1
for t in (0,1,2):
  print(f"Integral for p_th[{t}] = {round(np.trapz(p_th[t,:],dx = 1/1000),6)}")

#find where the maximum value for P(theta|D^10) appears
print(f"The maximum value is {np.max(p_th[2,:])} for x = {np.argmax(p_th[2,:])/1000}")
```

Για τον υπολογισμό του ολοκληρώματος που βρίσκεται στον παρανομαστή του κλάσματος που υπολογίζει το $p(\theta|D^N)$, χρησιμοποιούμε την συνάρτηση np.trapz() της βιβλιοθήκης NumPy. Λαμβάνει ως ορίσματα τις τιμές του άξονα y (τις τιμές της εκάστοτε συνάρτησης) και το διάστημα μεταξύ των δειγμάτων dx (στην περίπτωση του παραπάνω κώδικα dx = 1/1000 αφού έχουμε 1000 δείγματα μεταξύ 0 και 1). Η συνάρτηση αυτή λαμβάνει προαιρετικά και άλλα ορίσματα όμως στην προκειμένη περίπτωση δεν χρειάζονται. Εναλλακτικά, μπορεί να χρησιμοποιηθεί και η συνάρτηση quad.