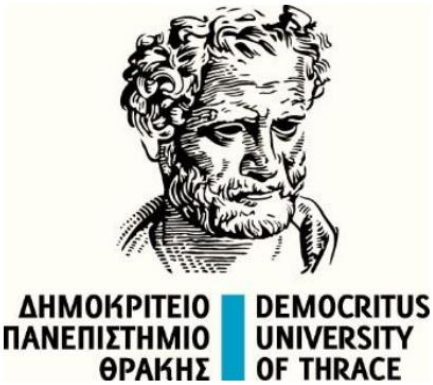



# 3Η ΕΡΓΑΣΙΑ

Στο μάθημα:
«Αναγνώριση Προτύπων»
Καθηγητής: Νικόλαος Μητιανούδης

Στέλιος Μούσλεχ
ΑΜ:57382
14/11/2020 ,Ξάνθη

# ΆΣΚΗΣΗ 3.1

**Οι συναρτήσεις του ερωτήματος βρίσκονται στο αρχείο "Ex3_1.py"**

Α) Για τον υπολογισμό της συνάρτηση διάκρισης όρισα την συνάρτηση :

```
def classifyFunc(d,x,m,s,pw):
```

όπου δέχεται ως ορίσματα τα:

d: διαστάσεις που έχουμε στην κατανομή
x: το δείγμα που έχουμε
μ: την μέση τιμή (ή πίνακα μέσων τιμών d>1)
s: το variance (ή covariance matrix d>1)
pw: την a priori πιθανότητα για την κλάση που υπολογίζουμε

Για τον υπολογισμό του τύπου:

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

με βάση της τις διαστάσεις θα πρέπει να κάνουμε μια διάκριση για d=1 και d>1 καθώς η NumPy υπολογίζει αντίστροφους πίνακας και ανάστροφους πίνακες μόνο αν βάλουμε στην συνάρτηση έναν 2D πίνακα οπότε θα υπολογίζουμε τον τύπο ξεχωριστά για αυτές τις 2 περιπτώσεις όπως βλέπουμε παρακάτω:

```
def classifyFunc(d,x,m,s,pw):
    c = x - m  # not necesseary but to have cleaner code in g

    #np.det() and np.linalg.inv()  input must be at least 2D array so we must check first for dimension of our distribution
    if d>1: #that means we have at least 2D so S is a 2D matrix
        detS=round(abs(np.linalg.det(s)),4) #roundind is needed cause linalg.det sometimes returns floating points with 1 bit rounding error
        invS=np.linalg.inv(s)
        g = -0.5 * (c.T).dot(invS).dot(c) - (d / 2) * math.log(2 * math.pi) - 0.5 * math.log(detS) + math.log(pw)
    else:
        detS=abs(s)
        invS=s**-1
        g = -0.5 * c**2*invS- (d / 2) - (d / 2) * math.log(2 * math.pi) - 0.5 * math.log(detS) + math.log(pw)
    return g
```

Β) Για τον υπολογισμό της ευκλείδειας απόστασης 2 τυχαίων σημείων N διαστάσεων με βάση τον παρκάτω τύπο:

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

Έφτιαξα μια συνάρτηση που δέχεται 2 σημεία x1,x2 και υπολογίζει την απόσταση αυτή προσέχοντας πάλι τις διαστάσεις :

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

Γ) Η απόσταση Mahalanobis είναι ένα μετρικό της απόστασης μεταξύ ενός σημείου x και μιας κατανομής. Αποτελεί την πολυδιάστατη γενίκευση της ιδέας να μετράμε πόσες τυπικές αποκλίσεις της κατανομής απέχει το σημείο x από την κατανομή.

Ο τύπος της είναι:

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

Οπότε έφτιαξα την συνάρτηση:

```
def mahalanobisDistance(x,m,s,d):
```

όπου δέχεται ως ορίσματα τα:

d: διαστάσεις που έχουμε στην κατανομή
x: το δείγμα που έχουμε
μ: την μέση τιμή (ή πίνακα μέσων τιμών d>1)
s: το variance (ή covariance matrix d>1)

και υπολογίζουμε την απόσταση με βάση τον τύπο. Εδώ η NumPy αντιμετωπίζει το ίδιο πρόβλημα με αυτό που περιγράψαμε παραπάνω οπότε ελέγχουμε πάλι για το αν έχουμε 1 διάσταση ή παραπάνω στη κατανομή μας.

```
def mahalanobisDistance(x,m,s,d):
    c=x-m
    if d>1:
        invS=np.linalg.inv(s)
        distance = np.sqrt(c.T.dot(invS).dot(c))
    else:
        invS = s ** -1
        distance = math.sqrt(c*invS*c)
    return distance
```

# ΆΣΚΗΣΗ 3.2

**Ο κώδικας της άσκησης βρίσκεται στο αρχείο "Ex3_2.py"**

Αρχικά για να φορτώσω τα δεδομένα τα αντέγραψα σε ένα αρχείο txt με όνομα data.txt (βρίσκεται στα αρχεία μαζί με τους κώδικες και το report) και με την χρήση της NumPy τα έκανα NumPy array και τα χώρισα σε 1 array ανά κλάση:

```
#get our data from txt file for ease of use
my_data = np.genfromtxt('data.txt', delimiter='')

#split our data to each class
classW1=np.zeros((10,3))
classW2=np.zeros((10,3))
classW3=np.zeros((10,3))

for i in range(10):
   classW1[i,0:3]=my_data[i,1:4]
   classW2[i, 0:3] = my_data[i, 4:7]
   classW3[i, 0:3] = my_data[i, 7:]
```

1.Αρχικά έχω P(ω1) = P(ω2) = 1/2 και P(ω3) = 0 οπότε θα ασχοληθώ με τις 2 πρώτες κλάσεις.

Θα κάνω για τα δεδομένα μου εκτίμηση των παραμέτρων μ,Σ. Έχοντας κανονικές κατανομές θα χρησιμοποιήσω τους τύπους

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

Η NumPy έχει συναρτήσεις για τον υπολογισμό αυτών των παραμέτρων με ακριβώς τον ίδιο τρόπο

```
onedimMeanW1=np.mean(classW1[:,0])
onedimMeanW2=np.mean(classW2[:,0])
onedimCovarianceW1=np.cov(classW1[:,0],rowvar=False)
onedimCovarianceW2=np.cov(classW2[:,0],rowvar=False)
```

Το σημείο διαχωρισμού είναι εκείνο για το οποίο το g1(x)-g2(x)=0 όπως είδαμε στην προηγούμενη εργασία.

Οπότε ένας πρακτικός και υπολογιστικός τρόπος για να το υπολογίσουμε χρησιμοποιώντας την συνάρτηση που φτιάξαμε στην αρχή είναι για ένα σύνολο τιμών x και με ένα πολύ μικρό βήμα να δοκιμάζουμε να υπολογίζουμε την διαφορά των 2 συναρτήσεων διάκρισης μέχρι αυτή να γίνει σχεδόν 0 (βάζουμε κάποιο tolerance πχ 0.0001) (το εύρος των τιμών που θα δοκιμάσουμε είναι λίγο μεγαλύτερο από την ελάχιστη και μέγιστη τιμή των δεδομένων μας)

Η υλοποίηση του παραπάνω σε κώδικα:

```
for n in np.arange(-9,9, 0.0001):
   y=Ex3_1.classifyFunc(1, n, onedimMeanW1, onedimCovarianceW1, 0.5) - Ex3_1.classifyFunc(1,
n, onedimMeanW2, onedimCovarianceW2,0.5)
   if abs(y)<0.00001:
      print("G1(x)-G2(x)=0 for x: ", n,"","with error tolerance: ",y)
```

και το αποτέλεσμα (τα 2 σημεία που προκύπτουν):

```
G1(x)-G2(x)=0 for x:  -5.085500000009123  with error tolerance:  1.5986490087271932e-07
G1(x)-G2(x)=0 for x:  4.337499999968916  with error tolerance:  -7.4852040041761825e-06
```

2.Έχοντας εκτιμήσει τα μ,Σ για ένα χαρακτηριστικό χ1 μπορούμε να κάνουμε την ταξινόμηση με την χρήση της συνάρτησης classifyFunc που γράψαμε στο πρώτο ερώτημα. Θα ταξινομήσουμε ένα δείγμα σε μία κλάση αν το αποτέλεσμα της συνάρτησης διάκρισης g(x) είναι το μεγαλύτερο για αυτή τη κλάση σε σχέση με την άλλη. Έτσι για τα δείγματα της κλάσης 1 αν το g1(x)<g2(x) θα έχουμε σφάλμα και αντίστοιχα το αντίθετο για την κλάση 2 . Έτσι μπορούμε να βρούμε  πόσες λάθος ταξινομήσεις κάναμε και να διαιρέσουμε το αποτέλεσμα με τον αριθμό των δειγμάτων (20) για να βρούμε το ποσοστό των σημείων που ταξινομήθηκε λάθος.

```
wrong=0
for i in range(10):
   if Ex3_1.classifyFunc(1,classW1[i,0],onedimMeanW1,onedimCovarianceW1,
0.5)<Ex3_1.classifyFunc(1,classW1[i,0],onedimMeanW2,onedimCovarianceW2,0.5):
      wrong=wrong+1

   if Ex3_1.classifyFunc(1,classW2[i,0],onedimMeanW1,onedimCovarianceW1,
0.5)>Ex3_1.classifyFunc(1,classW2[i,0],onedimMeanW2,onedimCovarianceW2,0.5):
      wrong=wrong+1
errorUsingx1=wrong/20
print("Classification error using only x1: ",errorUsingx1)
```

και έχουμε ποσοστό:

```
Classification error using only x1:  0.3
```

3. Προσθέτοντας άλλο 1 χαρακτηριστικό θα πρέπει να ξανά-υπολογίσω τα Σ,μ όπως πριν και στην συνέχεια να κάνω ακριβώς την ίδια διαδικασία για να βγάλω το σφάλμα:

```
#lets use 2 characteristics now

twodimMeanW1=np.mean(classW1[:,0:2] ,axis=0)
twodimMeanW2=np.mean(classW2[:,0:2] ,axis=0)
twodimCovarianceW1=np.cov(classW1[:,0:2],rowvar=False)
```

```
twodimCovarianceW2=np.cov(classW2[:,0:2],rowvar=False)

wrong=0
for i in range(10):
   if Ex3_1.classifyFunc(2,classW1[i,0:2],twodimMeanW1,twodimCovarianceW1,
0.5)<Ex3_1.classifyFunc(2,classW1[i,0:2],twodimMeanW2,twodimCovarianceW2,0.5):
      wrong=wrong+1

   if Ex3_1.classifyFunc(2,classW2[i,0:2],twodimMeanW1,twodimCovarianceW1,
0.5)>Ex3_1.classifyFunc(2,classW2[i,0:2],twodimMeanW2,twodimCovarianceW2,0.5):
      wrong=wrong+1
errorUsingx1x2=wrong/20
print("Classification error using only x1,x2: ",errorUsingx1x2)
```

Και έχω σφάλμα:

```
Classification error using only x1,x2:  0.45
```

4.Επαναλμβάνουμε το ίδιο βάζοντας και το x3

```
#lets use 3 characteristics now

threedimMeanW1=np.mean(classW1[:,0:3] ,axis=0)
threedimMeanW2=np.mean(classW2[:,0:3] ,axis=0)


threedimCovarianceW1=np.cov(classW1[:,0:3],rowvar=False)
threedimCovarianceW2=np.cov(classW2[:,0:3],rowvar=False)

wrong=0
for i in range(10):
   if Ex3_1.classifyFunc(3,classW1[i,0:3],threedimMeanW1,threedimCovarianceW1,
0.5)<Ex3_1.classifyFunc(3,classW1[i,0:3],threedimMeanW2,threedimCovarianceW2,0.5):
      wrong=wrong+1

   if Ex3_1.classifyFunc(3,classW2[i,0:3],threedimMeanW1,threedimCovarianceW1,
0.5)>Ex3_1.classifyFunc(3,classW2[i,0:3],threedimMeanW2,threedimCovarianceW2,0.5):
      wrong=wrong+1
errorUsingx1x2x3=wrong/20
print("Classification error using  x1,x2,x3: ",errorUsingx1x2x3)
```

Και έχω τώρα σφάλμα:

```
Classification error using  x1,x2,x3:  0.15
```

5. όπως βλέπουμε και από τα αποτελέσματα συγκεντρωτικά:

```
Classification error using only x1:  0.3
Classification error using only x1,x2:  0.45
Classification error using  x1,x2,x3:  0.15
```

Βάζοντας παραπάνω χαρακτηριστικά το σφάλμα μας σε αυτή τη περίπτωση βελτιώθηκε στα 3 ενώ χειροτέρεψε στα 2 σε σχέση με το 1. Αυτό συμβαίνει καθώς αυξάνοντας τις διαστάσεις μας μπορεί να πάρουμε παραπάνω πληροφορία για την κάθε κλάση και να βρούμε καλύτερα στατιστικά «trends» αλλά μπορεί και να εισάγουμε ουσιαστικά απλά παραπάνω θόρυβο και να χάσουμε αυτά τα «trends»

που βρήκαμε. Επίσης η εκτίμηση των παραμέτρων μας μπορεί να χειροτερέψει αν δεν προσθέσουμε και νέα δεδομένα προσθέτοντας παραπάνω διαστάσεις. Αυτό που είδαμε στις διαφάνειες ως Curse of Dimentionality δηλαδή Εάν για την καλή εκτίμηση των παραμέτρων μιας μονοδιάστατης pdf χρειάζονται N δείγματα, για την εκτίμηση της pdf σε d διαστάσεις χρειάζονται N^d δείγματα

6. Θα λύσουμε συμβολικά την συνάρτηση g ως προς x1,x2,x3 για να έχουμε τις συναρτήσεις

Επειδή αντιμετώπισα προβλήματα στο SymPy στην python αυτό το ερώτημα το έκανα σε MATLAB όπου ξαναόρισα την συνάρτηση διάκρισης στο αρχείο sunarthshDiakrisis.m και έκανα ένα script όπου παίρνοντας τις τιμές των μ1,μ2,μ3 και Σ1,Σ2,Σ3 από την python. Οπότε ο κώδικας

```
m1=[-0.44  -1.749 -0.766]
m2=[-0.543 -0.762 -0.542]
m3=[3.883 1.376 1.58 ]

s1=[14.38051111  7.69537778  4.12232222;
    7.69537778 14.62312111  3.90684   ;
    4.12232222  3.90684    19.72453778]

s2=[ 36.82933444   9.98092667 -16.36675111;
   9.98092667  13.16855111   0.40905111;
  -16.36675111   0.40905111  18.42121778]

s3=[ 8.30475667  7.44494667 13.14957778;
  7.44494667  8.56044889 11.60861111;
 13.14957778 11.60861111 47.28728889]

syms x1 x2 x3

x=[x1 x2 x3]

g1=simplify(sunarthshDiakrisis(x,m1,s1,0.8,3))
g2=simplify(sunarthshDiakrisis(x,m2,s2,0.1,3))
g3=simplify(sunarthshDiakrisis(x,m3,s3,0.1,3))
```

και έχω τα εξής αποτελέσματα

```
g1 =

((11*x1 + 25*abs(x1)^2)*((3542829096380169*x2)/72057594037927936 - (111450182070343*x1)/1125899906842624 +
+(6311927493707513*x3)/576460752303423488 + 1464935207996220428 3/28823037615171174 4000))/(50*x1) +
+((383*x3 + 500*abs(x3)^2)*((6311927493707513*x1)/576460752303423488 + (5109335178349673*x2)/576460752303423488 -
- (3944591206980533*x3)/72057594037927936 - 1245897959221182 2427/576460752303423488 000))/(1000*x3) +
+((1749*x2 + 1000*abs(x2)^2)*((3542829096380169*x1)/72057594037927936 - (6962684023450061*x2)/72057594037927936+
+ (5109335178349673*x3)/576460752303423488 - 4051868284511960 4557/28823037615171174 4000))/(2000*x2) - 9020085090936 5757/9007199254740992
```

```
g2 =

((381*x2 + 500*abs(x2)^2)*((8002401626740909*x1)/144115188075855872 - (4310487548579809*x2)/36028797018963968 +
+(7492781216806101*x3)/144115188075855872 - 4731974545242037503/144115188075855872000))/(1000*x2) - ((543*x1 + 1000*abs(x1)^2)*((5090086463784277*x1)/72057594037927936 -
- (8002401626740909*x2)/144115188075855872 + (1152812953650171*x3)/18014398509481984 + 22143004135601468 1/72057594037927936 00))/(2000*x1) -
-((271*x3 + 500*abs(x3)^2)*((1152812953650171*x1)/18014398509481984 - (7492781216806101*x2)/144115188075855872 + (8091824971079303*x3)/72057594037927936 +
+ 4034929226050029157/72057594037927936000))/(1000*x3) - 29659647426409507/2251799813685248
```

```
g3 =

((172*x2 - 125*abs(x2)^2)*((597270318396299*x2)/1125899906842624 - (8669595280751967*x1)/18014398509481984 + (4149418926090885*x3)/1152921504606846976 +
 3265935418497456056 57/2882303761517117440 00))/(250*x2) + ((79*x3 - 50*abs(x3)^2)*((4149418926090885*x2)/1152921504606846976 - (2273086437635619*x1)/36028797018963968 +
 (2724353393297103*x3)/72057594037927936 + 10149578816894514 3/5629499534213120 00))/(100*x3) - ((3883*x1 - 1000*abs(x1)^2)*((8669595280751967*x2)/18014398509481984 -
(9172467015345 7*x1)/1407374883553 28 + (2273086437635619*x3)/36028797018963968 + 1593213053315248 3183/9007199254740992000))/(2000*x1) - 26225588647232405/2251799813685248
```

# ΆΣΚΗΣΗ 3.3

**Ο κώδικας της άσκησης βρίσκεται στο αρχείο "Ex3_3.py"**

1. Ισχύει ότι : $\int_0^1 p(\theta / D^0)d\theta = 1$

   Άρα:

$$\int_0^1 p(\theta / D^0)d\theta = 1 \implies \int_0^1 A sin(\pi\theta)d\theta = 1 \implies A\left[\frac{-cos(\pi\theta)}{\pi}\right]_0^1 = 1 \implies$$
$$\implies A\left[-\frac{-1}{\pi} + \frac{1}{\pi}\right] = 1 \implies A = \frac{\pi}{2}$$

2. Για τον υπολογισμό μερικών πράξεων θα κάνω χρήση της βιβλιοθήκης **SciPy**.
   Για να υπολογίσουμε και να σχεδιάσουμε τα: $p(\theta \mid D^1),\ p(\theta \mid D^5), p(\theta \mid D^{10})$
   Θα χρησιμοποιήσουμε τον αναδρομικό τύπο όπως τον είδαμε στις διαφάνειες που προκύπτει ως εξής :

$$p(\boldsymbol{\theta} \mid D^n) = \frac{p(x_n \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid D^{n-1})}{\int p(x_n \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid D^{n-1}) d\boldsymbol{\theta}}, \quad p(\boldsymbol{\theta} \mid D^0) \text{ γνωστό (apriori)}$$

$$\text{ομως } p(x_n \mid \boldsymbol{\theta}) = \begin{cases} \boldsymbol{\theta} & \text{εάν κεφάλι} \\ (1-\boldsymbol{\theta}) & \text{εάν γράμματα} \end{cases}, \text{ και συνεπώς έχουμε}$$

$$p(\boldsymbol{\theta} \mid D^N) = \frac{\boldsymbol{\theta}^k (1-\boldsymbol{\theta})^{N-k} p(\boldsymbol{\theta} \mid D^0)}{\int \boldsymbol{\theta}^k (1-\boldsymbol{\theta})^{N-k} p(\boldsymbol{\theta} \mid D^0) d\boldsymbol{\theta}}$$

Και στη περίπτωση μας το: $p(\theta \mid D^0) = \begin{cases} A\sin(\pi\theta) & \text{για } 0 \le \theta \le 1 \\ 0 & \text{αλλού} \end{cases}$

Οπότε ξεκινάμε για να έχουμε τον τύπο υπολογιστικά ορίζοντας το $p(\theta|D^0)$ και βγάζοντας το A (το υπολογίζουμε και μεσω python για επαλήθευση):

```
ptheta0= lambda theta: np.sin(math.pi*theta)
A=1/integrate.quad(ptheta0,0,1)[0]
print(A)
```

και βλέπουμε ότι βγαίνει A= 1.5707963267948966 (=π/2)

στη συνέχεια ορίζουμε το διάνυσμα με τις ρίψεις μας

```
#kefali 1 grammata 0
d=np.array([1,1,0,1,0,1,1,1,0,1])
```

όπου κ=1 και γ=0 ώστε να μετράμε το άθροισμα των 1 ανάλογα με το μέγεθος N.

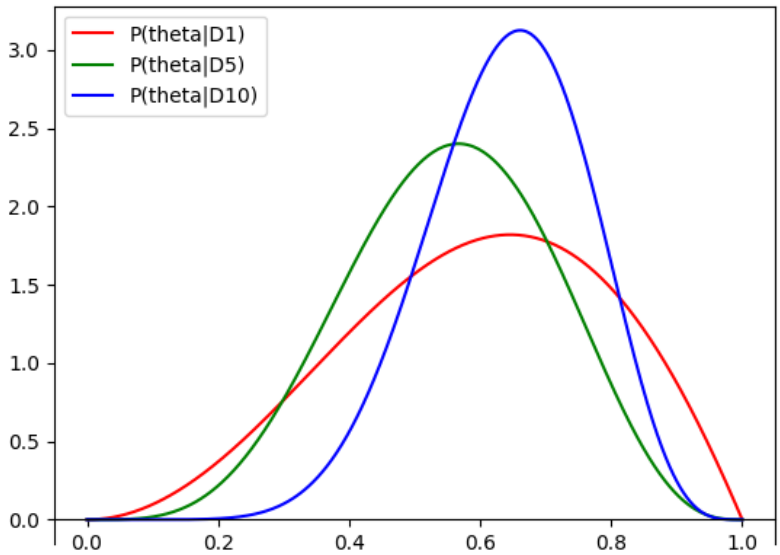
Οπότε θα κάνουμε for loop για τιμές 1,5,10 (αυτές που ζητάει η εκφώνηση) και μέσα σε κάθε loop θα μετράμε τον αριθμό των ρίψεων που είναι κεφάλι , θα υπολογίζουμε τον

αριθμητή και το παρονομαστή του τύπου ως προς μια μεταβλητή θ που θα πάρει τιμές από 0 ως 1 (1000 γραμμικά για να γίνει το plot στην python) και θα αποθηκέψουμε την κάθε κατανομή σε μια λίστα για να κάνουμε μετά τα 3 plot μαζί:

Ο κώδικας που κάνει αυτό που περιέγραψα:

```
y=[]
for i in[1,5,10]:
   k=np.count_nonzero(d[0:i])
   numerator= lambda theta: (theta**k)*(1-theta)**(i-k)*A*ptheta0(theta)
   denominator=integrate.quad(numerator,0,1)[0]
   theta=np.linspace(0,1,1000)
   y.append(numerator(theta)/denominator)
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.spines['bottom'].set_position('zero')
plt.plot(theta,y[0], 'r',label="P(theta|D1)")
plt.plot(theta,y[1], 'g',label="P(theta|D5)")
plt.plot(theta,y[2], 'b',label="P(theta|D10)")
plt.legend()
plt.show()
```

και βλέπουμε το αποτέλεσμα της σχεδίασης:




Με βάση τα αποτελέσματα των ρίψεων μας στις 1,5,10 το αποτέλεσμα της σχεδίασης μοιάζει σωστό

3. Αρκεί να βρούμε για $p(\theta|D^{10})$ τη μέγιστη τιμή που θα πάρει και για ποιο Θ είναι αυτή

```
print("the max value of P(θ|D10) is:",max(y[2]))
print("for x value",(np.argmax(y[2]))/1000)
```

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

# ΆΣΚΗΣΗ 3.4

**Ο κώδικας της άσκησης βρίσκεται στο αρχείο "Ex3_4.py" και στο "Ex3_4d.py"**

A. Για την δημιουργία δειγμάτων σε κάποιες κλάσεις έφτιαξα μια συνάρτηση με τρόπο ώστε να μπορεί να γενικευτεί για ότι αριθμό κλάσεων θέλω να δημιουργήσω:

```
def createMultivariateClassData(numberOfClasses,datasize,p,listofm,listofs):
```

Όπου
numberOfClasses: ο αριθμός των κλασεων που θέλω να δημιουργήσω.
datasize: το συνολικό μέγεθος των δεδομένων που θέλω να δημιουργήσω.
p: μια λίστα με όλες της a priori πιθανότητες
listofm: μια λίστα με όλα τα μ
listofs: μια λίστα με όλα τα Σ

Μέσα στη συνάρτηση ανάλογα με την a priori πιθανότητα για κάθε κλάση και τα συνολικά δεδομένα δημιουργώ για κάθε κλάση τα δεδομένα μέσω της συνάρτησης της NumPy προσέχοντας ώστε η τελευταία κλάση να στρογγυλοποιεί σωστά τα δεδομένα ώστε να είναι όσα ζητήθηκαν αρχικά.
Η συνάρτηση επιστρέφει μια λίστα από NumPy arrays, 1 array για κάθε κλάση
Ο κώδικας:

```
def createMultivariateClassData(numberOfClasses,datasize,p,listofm,listofs):
  listOfData=[]
  numberOfSamplesCreated=0 #check how many we have created so last class rounds to datasize!!!!
  for i in range(numberOfClasses-1):
    classSize=math.floor(datasize*p[i])
    listOfData.append(np.random.multivariate_normal(listofm[i],listofs[i],size=classSize))
    numberOfSamplesCreated=numberOfSamplesCreated+classSize
  #for last class we need to take the remaining samples cause for sum of p we might not have round number of data
  lastclassSize=datasize-numberOfSamplesCreated
  listOfData.append(np.random.multivariate_normal(listofm[-1],listofs[-1],size=lastclassSize))
  return listOfData
```

Έχοντας ορίσει αυτήν την συνάρτηση θα δημιουργήσουμε τα training και test data όπως ζητάει η εκφώνηση:

```
trainsize=10000
testsize=1000

p1=p2=p3=1/3
listofP=[p1,p2,p3]

m1=np.array([0,0,0])
m2=np.array([1,2,2])
m3=np.array([3,3,4])
listofm=[m1,m2,m3]

s1=s2=s3=np.array([[0.8,0.2,0.1],[0.2,0.8,0.2],[0.1,0.2,0.8]])
listofS=[s1,s2,s3]

train_data=createMultivariateClassData(3,trainsize,listofP,listofm,listofS)
test_data=createMultivariateClassData(3,testsize,listofP,listofm,listofS)
```

Β. Μας ζητάει να χρησιμοποιήσουμε τις παραπάνω παραμέτρους άρα με τα θεωρητικά μ και Σ θα δοκιμάσουμε να ταξινομήσουμε με βάση την Ευκλείδεια απόσταση, την απόσταση mahalanobis και και τον Bayesian Classifier και θα υπολογίσουμε τα error για κάθε περίπτωση.

Ουσιαστικά για την ευκλείδεια απόσταση και την mahalanobis θα υπολογίζουμε για κάθε τεστ δείγμα την απόσταση από την κάθε κλάση, και θα την ταξινομούμε στην κλάση με την μικρότερη απόσταση. Άρα θα έχουμε σφάλμα ταξινόμησης αν για τα test data της κλάσης 1 η απόσταση από την κλάση 1 δεν είναι η μικρότερη και αντίστοιχα το ίδιο για την 2 και την 3. Στο τέλος αθροίζουμε τα λάθη από κάθε κλάση και τα διαιρούμε με τον αριθμό των test data.

```
#test our data now with m s from ekfonisi

#EUCLIDIAN DISTANCE

euclidianFalses=0
for i in range(test_data[0].shape[0]):
   #test of class1 -eucldian distance
   distanceto1=Ex3_1.euclidianDistance(test_data[0][i],m1,3)
   distanceto2=Ex3_1.euclidianDistance(test_data[0][i],m2,3)
   distanceto3=Ex3_1.euclidianDistance(test_data[0][i],m3,3)

   #we made mistake for class1 if distanceto1 is not the smallest
   if distanceto1>distanceto2 or distanceto1>distanceto3:
      euclidianFalses= euclidianFalses + 1

#do the same for the rest of classes
for i in range(test_data[1].shape[0]):
   # test of class2 -euclidian distance
   distanceto1 = Ex3_1.euclidianDistance(test_data[1][i], m1,3)
   distanceto2 = Ex3_1.euclidianDistance(test_data[1][i], m2,3)
   distanceto3 = Ex3_1.euclidianDistance(test_data[1][i], m3, 3)

   # we made mistake for class2 if distanceto2 is not the smallest
   if distanceto2 > distanceto1 or distanceto2 > distanceto3:
      euclidianFalses = euclidianFalses + 1


for i in range(test_data[2].shape[0]):
   # test of class3 -euclidian distance
   distanceto1 = Ex3_1.euclidianDistance(test_data[2][i], m1, 3)
   distanceto2 = Ex3_1.euclidianDistance(test_data[2][i], m2, 3)
   distanceto3 = Ex3_1.euclidianDistance(test_data[2][i], m3, 3)

   # we made mistake for class2 if distanceto3 is not the smallest
   if distanceto3 > distanceto1 or distanceto3 > distanceto2:
      euclidianFalses = euclidianFalses + 1

euclidianError=euclidianFalses/testsize

print("Euclidian Minimun Distance Classifier Error:",euclidianError," Using theoretical Mean and
Covariance")


#MAHALANOBIS CLASSIFIER
```

```
mahalanobisFalses=0
for i in range(test_data[0].shape[0]):
    #test of class1 -mahalanobis distance
    distanceto1=Ex3_1.mahalanobisDistance(test_data[0][i],m1,s1,3)
    distanceto2=Ex3_1.mahalanobisDistance(test_data[0][i],m2,s2,3)
    distanceto3=Ex3_1.mahalanobisDistance(test_data[0][i],m3,s3,3)

    #we made mistake for class1 if distanceto1 is not the smallest
    if distanceto1>distanceto2 or distanceto1>distanceto3:
        mahalanobisFalses=mahalanobisFalses+1
    #do the same for the rest of classes

for i in range(test_data[1].shape[0]):
    # test of class2 -mahalanobis distance
    distanceto1 = Ex3_1.mahalanobisDistance(test_data[1][i], m1, s1, 3)
    distanceto2 = Ex3_1.mahalanobisDistance(test_data[1][i], m2, s2, 3)
    distanceto3 = Ex3_1.mahalanobisDistance(test_data[1][i], m3, s3, 3)

    # we made mistake for class2 if distanceto2 is not the smallest
    if distanceto2 > distanceto1 or distanceto2 > distanceto3:
        mahalanobisFalses = mahalanobisFalses + 1


for i in range(test_data[2].shape[0]):
    # test of class3 -mahalanobis distance
    distanceto1 = Ex3_1.mahalanobisDistance(test_data[2][i], m1, s1, 3)
    distanceto2 = Ex3_1.mahalanobisDistance(test_data[2][i], m2, s2, 3)
    distanceto3 = Ex3_1.mahalanobisDistance(test_data[2][i], m3, s3, 3)

    # we made mistake for class2 if distanceto3 is not the smallest
    if distanceto3 > distanceto1 or distanceto3 > distanceto2:
        mahalanobisFalses = mahalanobisFalses + 1

mahalanobisError=mahalanobisFalses/testsize

print("Mahalanobis Minimun Distance Classifier Error:",mahalanobisError," Using theoretical
Mean and Covariance")
```

Με την ίδια λογική θα είναι και ο Bayesian Classifier με την διαφορά ότι εδώ χρησιμοποιώντας την συνάρτηση που γράψαμε στο πρώτο ερώτημα classifyFunc θα ταξινομήσουμε ένα δείγμα σε μία κλάση αν το αποτέλεσμα της συνάρτησης διάκρισης g(x) είναι το μεγαλύτερο για αυτή τη κλάση σε σχέση με τις άλλες. Ο αντίστοιχος κώδικας για αυτόν τον εκτιμητή:

```
#Bayesian Classifier
# test of class1 - bayesian classifier
bayesianFalses=0
for i in range(test_data[0].shape[0]):
    d1=Ex3_1.classifyFunc(3,test_data[0][i],m1,s1,p1)
    d2=Ex3_1.classifyFunc(3,test_data[0][i],m2,s2,p2)
    d3=Ex3_1.classifyFunc(3,test_data[0][i],m3,s3,p3)

    #we made mistake for class1 if d1 not the Biggest
    if d1<d2 or d1<d3:
        bayesianFalses=bayesianFalses+1
#do the same for the rest of classes
for i in range(test_data[1].shape[0]):
```

```
    d1=Ex3_1.classifyFunc(3,test_data[1][i],m1,s1,p1)
    d2=Ex3_1.classifyFunc(3,test_data[1][i],m2,s2,p2)
    d3=Ex3_1.classifyFunc(3,test_data[1][i],m3,s3,p3)

    #we made mistake for class1 if d2 not the Biggest
    if d2<d1 or d2<d3:
        bayesianFalses=bayesianFalses+1

#do the same for the rest of classes
for i in range(test_data[2].shape[0]):
    d1=Ex3_1.classifyFunc(3,test_data[2][i],m1,s1,p1)
    d2=Ex3_1.classifyFunc(3,test_data[2][i],m2,s2,p2)
    d3=Ex3_1.classifyFunc(3,test_data[2][i],m3,s3,p3)

    #we made mistake for class1 if d3 not the Biggest
    if d3<d1 or d3<d2:
        bayesianFalses=bayesianFalses+1

bayesianError=bayesianFalses/testsize
print("Bayesian Classifier Error:",bayesianError," Using theoretical Mean and Covariance")
```

Τρέχοντας τον κώδικα έχω τα παρακάτω αποτελέσματα:

```
Euclidian Minimun Distance Classifier Error: 0.088  Using theoretical Mean and Covariance
Mahalanobis Minimun Distance Classifier Error: 0.082  Using theoretical Mean and Covariance
Bayesian Classifier Error: 0.082  Using theoretical Mean and Covariance
```

Εδώ παρατηρούμε ότι τα αποτελέσματα είναι πολύ καλά και κοντινά μεταξύ τους. Όλα βρίσκονται στο ~8% με τον ευκλείδειο να είναι κατά 0.6% χειρότερος. Θεωρώ λογικό το αποτέλεσμα καθώς έχουμε τον ίδιο πίνακα διασποράς Σ, και οι κλάσεις μας έχουν ίδια a priori πιθανότητα

Γ. Τώρα ξανακάνω την παραπάνω μέτρηση αλλά υπολογίζω τα μ και Σ με βάση τον τύπο

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

Δηλαδή κάνω ML εκτίμηση των παραμέτρων θεωρώντας ότι δεν τις ξέρω από πριν και

Μέσω των συναρτήσεων της NumPy μπορούμε να τα υπολογίσουμε ακριβώς έτσι από τα δεδομένα εκπαίδευσης:

```
#calculate means and covariance matrices from our training data
class1mean=np.mean(train_data[0],0)
class2mean=np.mean(train_data[1],0)
class3mean=np.mean(train_data[2],0)
class1cov=np.cov(train_data[0],rowvar=False) #rowvar=False means that rows are observations
and cols are variables
class2cov=np.cov(train_data[1],rowvar=False)
class3cov=np.cov(train_data[2],rowvar=False)
```

Χρησιμοποιώντας αυτά τώρα ξανά-υπολογίζουμε με τον ίδιο τρόπο τα σφάλματα ταξινόμησης για κάθε ταξινομητή (για να μην ξαναβάλω όλο το κώδικα στο report υπάρχει στο “Ex3_4.py”.

Τώρα έχουμε τα εξής αποτελέσματα:

```
Euclidian Minimun Distance Classifier Error: 0.086  Using calculated Maximun Likelihood Mean and Covariance
Mahalanobis Minimun Distance Classifier Error: 0.083  Using calculated Maximun Likelihood Mean and Covariance
Bayesian Classifier Error: 0.082  Using calculated Maximun Likelihood Mean and Covariance
```

Εδώ βλέπουμε ότι είναι σχεδόν ολόιδια με το να χρησιμοποιήσουμε τα θεωρητικά μ,Σ που γνωρίζαμε εξαρχής. Το αποτέλεσμα αυτό μου φαίνεται λογικό καθώς τα train_data ακολουθούν κανονική κατανομή όποτε αυτή η εκτίμηση των παραμέτρων μ,Σ είναι πολύ ακριβής.

Δ. Τώρα ξανατρέχω τον κώδικα που έκανα στο αρχείο ''Ex3_4.py'' αλλάζοντας τις παραμέτρους για την δημιουργία των δεδομένων. Το αρχείο αυτό είναι το "Ex3_4d.py" με την μόνη αλλαγής την δημιουργία των δεδομένων:

```
trainsize=10000
testsize=1000

p1=1/6
p2=1/6
p3=2/3
listofP=[p1,p2,p3]

m1=np.array([0,0,0])
m2=np.array([1,2,2])
m3=np.array([3,3,4])
listofm=[m1,m2,m3]

s1=np.array([[0.8,0.2,0.1],[0.2,0.8,0.2],[0.1,0.2,0.8]])
s2=np.array([[0.6,0.2,0.01],[0.2,0.8,0.01],[0.01,0.01,0.6]])
s3=np.array([[0.6,0.1,0.1],[0.1,0.6,0.1],[0.1,0.1,0.6]])
listofS=[s1,s2,s3]

train_data=createMultivariateClassData(3,trainsize,listofP,listofm,listofS)
test_data=createMultivariateClassData(3,testsize,listofP,listofm,listofS)
```

και τρέχοντας τον κώδικα έχουμε τα εξής αποτελέσματα:

```
Euclidian Minimun Distance Classifier Error: 0.059  Using theoretical Mean and Covariance
Mahalanobis Minimun Distance Classifier Error: 0.06  Using theoretical Mean and Covariance
Bayesian Classifier Error: 0.049  Using theoretical Mean and Covariance

EROTIMA G

Euclidian Minimun Distance Classifier Error: 0.059  Using calculated Maximun Likelihood Mean and Covariance
Mahalanobis Minimun Distance Classifier Error: 0.06  Using calculated Maximun Likelihood Mean and Covariance
Bayesian Classifier Error: 0.048  Using calculated Maximun Likelihood Mean and Covariance
```

Εδώ παρατηρούμε πάλι ότι δεν παίζει ρόλο αν είμαστε με τα θεωρητικά Σ,μ ή με αυτά που εκτιμήσαμε εμείς. Όλοι οι ταξινομητές έχουν καλά αποτελέσματα, αν και εδώ βλέπουμε λίγο εμφανέστερα τον Bayesian classifier να είναι καλύτερος. Θεωρώ ότι αυτό προκύπτει από το γεγονός ότι πλέον οι κλάσεις μας δεν είναι ισοπίθανες (έχουν διαφορετικές a priori πιθανότητες) κάτι που λαμβάνει υπόψιν ο Bayesian classifier ενώ οι άλλοι 2 όχι.