ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΡΩΤΗ ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ
ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

Αναστασία Χριστίνα Λίβα
03119029
anachriliva@gmail.com

1

Λόγω της υψηλής διάστασης του χώρου δε μπορώ να έχω γραφική απεικόνιση. Θεωρώ πως βρίσκομαι σε ευκλείδιο χώρο διαστάσεων n και πως Π είναι το target παραλληλόγραμμο, δηλαδή ο ιδανικός ταξινομητής. Τα δεδομένα παρέχονται στον learner σε μορφή τυχαίων σημείων p στο χώρο μαζί με μία ετικέτα n η οποία δηλώνει εάν το εκάστοτε σημείο p_i βρίσκεται μέσα στο Π(ετικέτα+1) ή έξω από αυτό (ετικέτα -1)

Θεωρώ τον ταξινομητή h ο οποίος είναι ένα υπερπαραλληλόγραμμομε την ιδιότητα να είναι το πιο στενό υπερπαραληλόγραμμο που περικλείει όλα τα δεδομένα που έχουν ετικέτα +1, οπότε λόγω της ιδιότητας αυτής ο h είναι πάντα υποσύνολο του Π.

Βάσει της παραπάνω κατασκευής η περιοχή σφάλματος θα είναι η συμμετρική διαφορά των Π και h. Στον 2διαστατο χώρο η συμμετρική διαφορά χψρίστηκε σε 4 υποχώροτς. Στη γενική περίπτωση όπου η διάσταση είναι n η συμμετρική διαφορά θα χωρίζεται πλέον σε 2 υποχώρους.

Δεδομένου ότι τα δείγματα που επιλέγω από την κατανομή είναι ανεξάρτητα μεταξύ τους φράζω την πιθανότητα ενός κακού γεγονότος, δηλ τα δείγματά μου να βρίσκονται στη συμμετρική διαφορά των Π και h από το δ. Οπότε

$$2n(1-\frac{\varepsilon}{2n})^m \leq \delta \Rightarrow$$
$$(1-\frac{\varepsilon}{2n})^m \leq \ 2n\delta \Rightarrow$$
$$e^{-\frac{\varepsilon}{2n}m} \leq \frac{\delta}{2n} \Rightarrow$$
$$m \geq \frac{2n}{\varepsilon}ln(\frac{2n}{\delta})$$

Εάν ισχύει η ανισότητα αυτή τότε με πιθανότητα 1-δ η h θα έχει πραγματικό σφάλμα το πολύ ε σε σχέση με την Π. Επιπλέον εδόσον η επεξεργασία κάθε δείγματος απαιτεί το πολύ $2n$ συγκρίσεις το running time είναι γραμμικώς φραγμένο από το m. Οπότε ο αλγόριθμος είναι φραγμένος με πολυπλοκότητα n από ένα πολυώνυμο του m, $\frac{1}{\varepsilon}$, $\frac{1}{\delta}$. Άρα PAC εκπαιδεύσιμη κλάση.

1 Άσκηση 2

α)

$$F(u,\theta,\lambda) = J(u,\theta) - \lambda(\sum_{j=1}^{m} n_{i,j} - 1)$$

$$\frac{\vartheta F}{\vartheta U_{kp}} = 0 \Rightarrow u_{i,j} = (\frac{l}{qd(x_i, \theta_j}^{\frac{1}{q-1}}$$

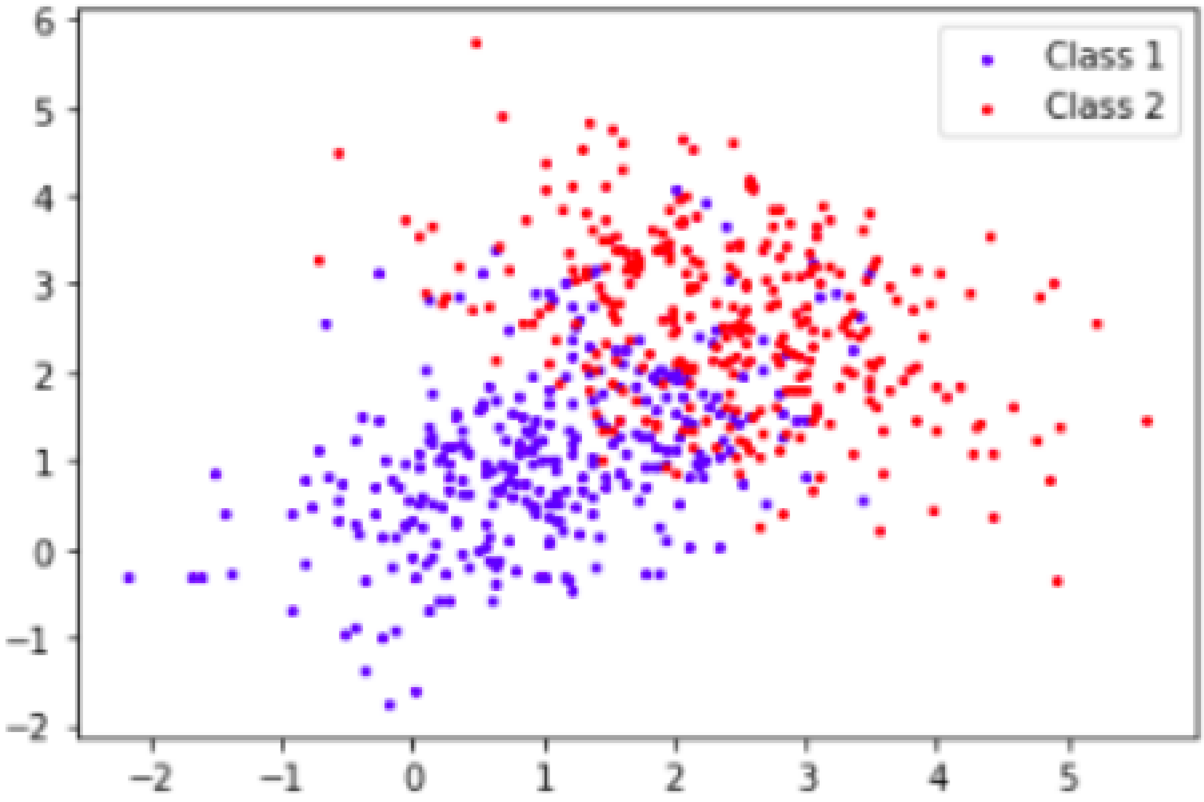
$$\sum_{j=1}^{m} u_{ij} = 1 \Rightarrow \lambda^{\frac{1}{q-1}} \sum_{j=1}^{m} \frac{1}{(qd(x_i, \theta_i))^{\frac{1}{q-1}}} = 1$$

οπότε:

$$u_{ij} = \frac{1}{\sum_{k=1}^{m} (\frac{d(x_i,\theta_i)}{d(x_i,\theta_k})^{\frac{1}{q-1}}}$$

$$\frac{\vartheta f}{\vartheta \theta = 0} \Rightarrow \theta_i = \frac{\sum_{i=1} N u_{ij}^q x_i}{\sum_{i=1}^{N} u_{ij}^q}$$

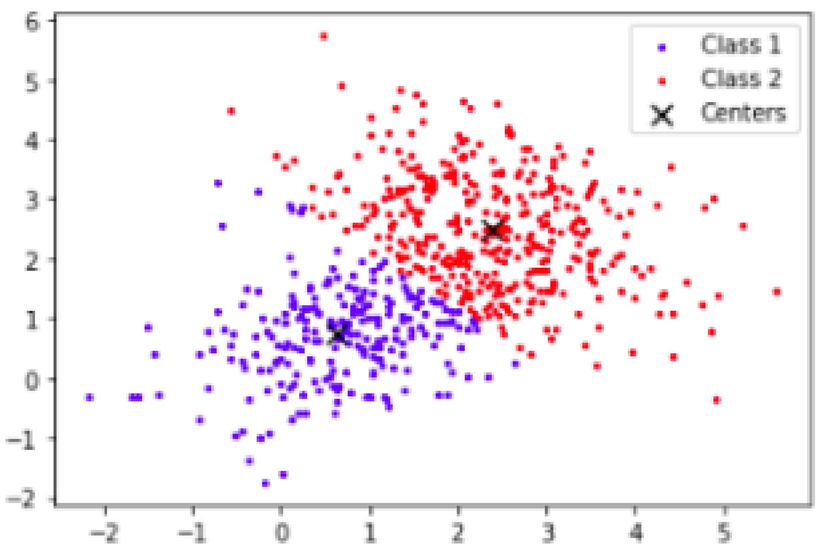
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

mi=[1,1]
cl=[[1,0.5],[0.5,1]]
m2=[2.5,2.5]
c2=[[1,-0.4],[-0.4,1]]

datacl=np.random.multivariate_normal(ml, cl, 300)
datac2=np.random.multivariate_normal(m2, c2, 300)

data=list(datacl)+list(datac2)
Xcl=list(datac1[:,0])
Ycl=list(datacl[:,1])
Xc2=list(datac2[:,0])
Yc2=list(datac2[:,1])

fig=plt. figure()
axl=fig.add_subplot(111)

axl.scatter(Xcl,Ycl,s=4, c='b', label='Class 1')
axl.scatter(Xc2,Yc2,s5=4, c='r',label='Class 2')
ax1.legend()

plt.show()
```




```
def randomcenter(i,j):
ans=[]
t=np.random.uniform(i,j);
ans .append(t)
t=np.random.uniform(i,j);
ans .append(t)
return ans

X=[]
for (i,j) in zip(Xc1,Yc1):
X.append([i,j])

for (i,j) in zip(Xc2,Yc2):
X.append([i,j])

centers=[randomcenter(-2,5),randomcenter(-2,5)]
def dist(p1.p2):
ans=o.q
for (i,3) in zip(pl,p2):
```

```
anste()-i)*2
return ans

def kneans(x,cen, rec):
tempx=[]
for point in x:
clymind=-1,100
classno-o
for j in cen:
tedist (point, j)
if (temind):
leclassno
ind=t
classnot=1

tempX.append(cl)



tenpcen=[]
for i in range(len(cen)):
    tx=np.mean([X{j][0]] for j in range(Len(x)) if tempx{jJ==il)
    ty=np.mean(X{j][1] for j in range(en(x)) if tempxtjl==il)
    tempcen.append( [tx, ty])

diff=0.0
for (i,j) in zip(cen, tempcen):
diffeedist (i,j)
if diffeiors(-4):
return tempx, tempcen, rec
else:
return kmeans(X, teapcen, rece)

8k, precentersk, recskekneans(X,centers, 1)

<classk1, classk2=0,1
if precentersk[0][0]>precentersk[1][0]
    classk1,classk2=classk2, classk1
    precentersk[0] ,precentersk[1]=precentersk[ 1] precentersk[0]
predxct=[X{il{9] for i in range(Len(x)) if BkLi}==classk1]
predvet=[X{ill1] for i in range(Len(X)) if BkLil==classk1]
PredXc2=[X{il{0] for i in range(Len(X)) if BkLil==classk2]
predve2=[X{ill1] for i in rango(Len(X)) if BkLil==classk2]
cenX=[precentersk[0] 0] ,precentersk[1][0]]
cenY=[precentersk[0] [1] ,precentersk[1] [1]



sucratek=0.0
```

```
for i in range(600):
    if(4<308 and Bk[i]==classk1) :
        sucratekr=1
    elif(>299 and BkLi}==classk2):
        sucratekt+=1
sucratek=sucratek/600

indistkenp.mean(Inp.sqrt(dist(i,j)) for (i,j) in zip(precentersk, (
   m,m2)1)

figeplt.figure()
axlefig-add_subplot(111)

axl. scatter (predxcl, predycl,se4,c=' b  ,Label='Class 1")
axl. scatter (predxc2, predyc2,s=t,c='r*  label='Class 2")

axl. scatter (cenX, cenY, 5=80,c='biack' ynarker='x' , abel='
    Centers')
axl. Legend()

plt.show()
```



2.2 γ)

```
def randomcenter(i,j):
  ans=[]
  t=np.random.uniform(i,j);
  ans.append(t)
  t=np.random.uniform(i,j);
  ans.append(t)
  return ans
```

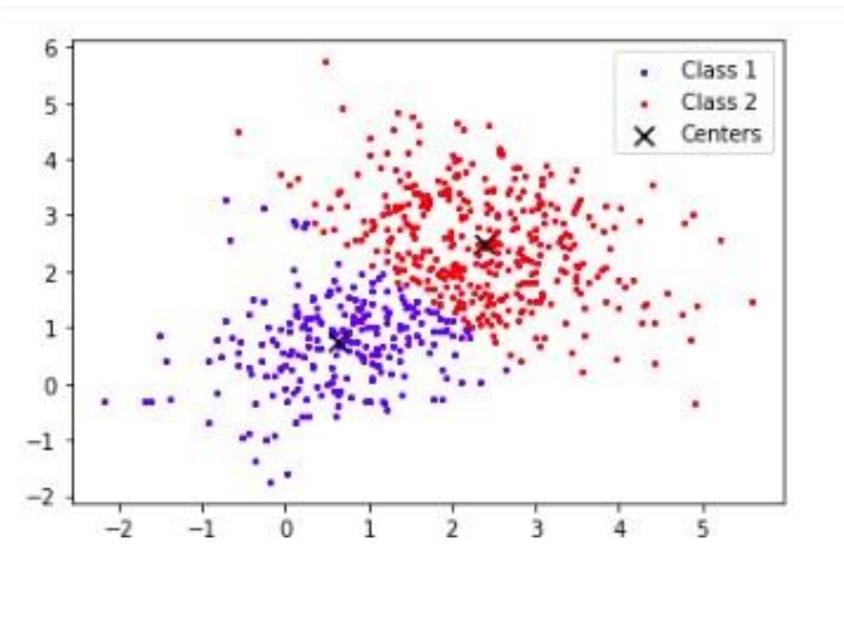
```
X=[]
for (i,j) in zip(Xc1,Yc1):
  X.append([i,j])

for (i,j) in zip(Xc2,Yc2):
  X.append([i,j])

centers=[randomcenter(-2,5),randomcenter(-2,5)]
```

```
def dist(p1,p2):
  ans=0.0
  for (i,j) in zip(p1,p2):
    ans+=(j-i)**2
  return ans

def kmeans(X,cen,rec):
  tempX=[]
  for point in X:
    cl,mind=-1,100
    classno=0
    for j in cen:
      t=dist(point,j)
      if(t<mind):
        cl=classno
        mind=t
      classno+=1
    tempX.append(cl)

  tempcen=[]
  for i in range(len(cen)):
    tx=np.mean([X[j][0] for j in range(len(X)) if tempX[j]==i])
    ty=np.mean([X[j][1] for j in range(len(X)) if tempX[j]==i])
    tempcen.append([tx,ty])

  diff=0.0
  for (i,j) in zip(cen,tempcen):
    diff+=dist(i,j)
  if diff<10**(-4):
    return tempX,tempcen,rec
  else:
    return kmeans(X,tempcen,rec+1)
```

```
Bk,precentersk,recsk=kmeans(X,centers,1)
```

```
classk1,classk2=0,1
if precentersk[0][0]>precentersk[1][0]:
  classk1,classk2=classk2,classk1
  precentersk[0],precentersk[1]=precentersk[1],precentersk[0]
predXc1=[X[i][0] for i in range(len(X)) if Bk[i]==classk1]
predYc1=[X[i][1] for i in range(len(X)) if Bk[i]==classk1]
predXc2=[X[i][0] for i in range(len(X)) if Bk[i]==classk2]
predYc2=[X[i][1] for i in range(len(X)) if Bk[i]==classk2]
cenX=[precentersk[0][0],precentersk[1][0]]
cenY=[precentersk[0][1],precentersk[1][1]]

sucratek=0.0
for i in range(600):
  if(i<300 and Bk[i]==classk1):
    sucratek+=1
  elif(i>299 and Bk[i]==classk2):
    sucratek+=1
sucratek=sucratek/600

mdistk=np.mean([np.sqrt(dist(i,j)) for (i,j) in zip(precentersk,[m1,m2])])

fig=plt.figure()
ax1=fig.add_subplot(111)

ax1.scatter(predXc1,predYc1,s=4,c='b',label='Class 1')
ax1.scatter(predXc2,predYc2,s=4,c='r',label='Class 2')
ax1.scatter(cenX,cenY,s=80,c='black',marker='x',label='Centers')
ax1.legend()
plt.show()
```

Λαμβάνωτην εξής γραφική παράσταση της ομαδοποίησης στην οποία καταλήγει ο k-means καιτων κέντρων των κλάσεων που εκτιμά ο αλγόριθμος :



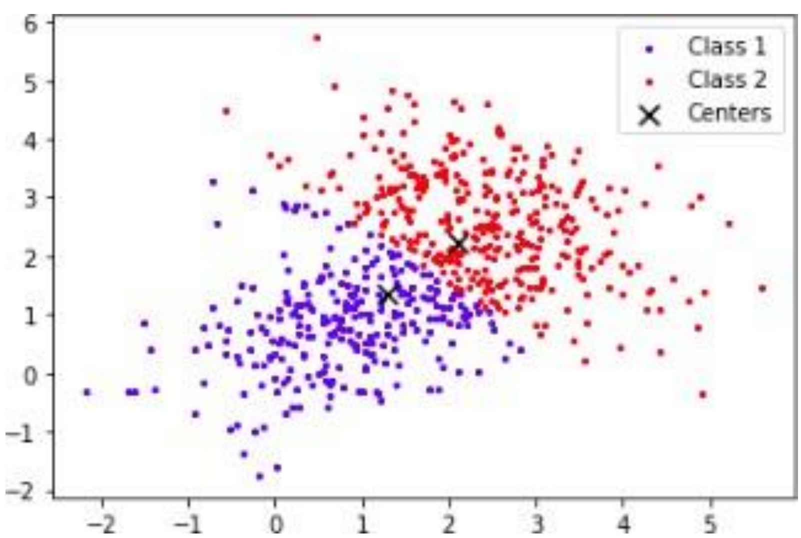
2.2 δ)

```
def fuzzcmeans(X,cen,rec):
  tempU=[]
  for point in X:
    s1d=0.0
    tempX=[]
    for i in cen:
      s1d+=(1/dist(point,i))

    for j in cen:
      t=dist(point,j)
      uij=1/(t*s1d)
      tempX.append(uij)
    tempU.append(tempX)

  tempcen=[]
  for i in range(len(cen)):
    tx=0.0
    sumu=0.0
    ty=0.0
    for j in range(len(X)):
      t=tempU[j][i]
      sumu+=t
      tx+=t*X[j][0]
      ty+=t*X[j][1]
    tx=tx/sumu
    ty=ty/sumu
    tempcen.append([tx,ty])

  diff=0.0
  for (i,j) in zip(cen,tempcen):
    diff+=dist(i,j)
  if diff<10**(-4):
    B=[]
    for i in tempU:
      cl,mind=-1,-1
      for j in range(len(cen)):
        if(i[j]>mind):
          cl=j
          mind=i[j]
      B.append(cl)
    return B,tempcen,rec
  else:
    return fuzzcmeans(X,tempcen,rec+1)
```

```
Bf,precentersf,recsf=fuzzcmeans(X,centers,1)
```

```
classf1,classf2=0,1
if precentersf[0][0]>precentersf[1][0]:
  classf1,classf2=classf2,classf1
  precentersf[0],precentersf[1]=precentersf[1],precentersf[0]
predXc1=[X[i][0] for i in range(len(X)) if Bf[i]==classf1]
predYc1=[X[i][1] for i in range(len(X)) if Bf[i]==classf1]
predXc2=[X[i][0] for i in range(len(X)) if Bf[i]==classf2]
predYc2=[X[i][1] for i in range(len(X)) if Bf[i]==classf2]
cenX=[precentersf[0][0],precentersf[1][0]]
cenY=[precentersf[0][1],precentersf[1][1]]


sucratef=0.0
for i in range(600):
  if(i<300 and Bf[i]==classf1):
    sucratef+=1
  elif(i>299 and Bf[i]==classf2):
    sucratef+=1
sucratef=sucratef/600

mdistf=np.mean([np.sqrt(dist(i,j)) for (i,j) in zip(precentersf,[m1,m2])]

fig=plt.figure()
ax1=fig.add_subplot(111)

ax1.scatter(predXc1,predYc1,s=4,c='b',label='Class 1')
ax1.scatter(predXc2,predYc2,s=4,c='r',label='Class 2')
ax1.scatter(cenX,cenY,s=80,c='black',marker='x',label='Centers')
ax1.legend()
plt.show()
```

Λαμβάνω την εξής γραφική παράσταση της ομαδοποίησης στην οποία καταλήγει ο fuzzyc-means και των κέντρων των κλάσεων που εκτιμά ο αλγόριθμος :



2.2 ε)

Παραθέτω παρακάτω τα αποτελέσματα των συγκρίσεων των δύο αλγορίθμων ομαδοποίησης :

```
print("K-means Algorithm:")
print("Number of recursions:",recsk)
print("Centers of Classes:",precentersk[0],precentersk[1])
print("Mean distance from real centers:",mdistk)
print("Success rate:",sucratek,"\n")
print("Fuzzy-c-means Algorithm:")
print("Number of recursions:",recsf)
print("Centers of Classes:",precentersf[0],precentersf[1])
print("Mean distance from real centers:",mdistf)
print("Success rate:",sucratef,"\n")
```

```
K-means Algorithm:
Number of recursions: 14
Centers of Classes: [0.6367214189266467, 0.7319110505820643] [2.3881426397416807, 2.5018821901675388]
Mean distance from real centers: 0.28168148224885137
Success rate: 0.8383333333333334

Fuzzy-c-means Algorithm:
Number of recursions: 26
Centers of Classes: [1.2859770951999703, 1.3693670319110673] [2.1133592477849494, 2.2416270003277052]
Mean distance from real centers: 0.46607958257639104
Success rate: 0.8283333333333334
```

2.5 α)

Παραθέτω παρακάτω κώδικα :

```
from sklearn.datasets import load_iris
data = load_iris()
label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']
```

2.5 β)

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
df=pd.DataFrame(features,columns=feature_names)
c=0
for i in feature_names:
    scl=ColumnTransformer([('standard',StandardScaler(),[c])],remainder='drop')
    df[i]=scl.fit_transform(df)
    c+=1

features=df.to_numpy(copy=True)
```

2.5 γ)

```
C=np.zeros((4,4))
for x in features:
  t1=np.array([x,np.zeros(4)])
  t1t=t1.transpose()
  temp=np.matmul(t1t,t1)
  C+=temp/150
print(C)
```

```
[[ 1.         -0.11756978  0.87175378  0.81794113]
 [-0.11756978  1.         -0.4284401  -0.36612593]
 [ 0.87175378 -0.4284401   1.          0.96286543]
 [ 0.81794113 -0.36612593  0.96286543  1.        ]]
```

2.5 δ)

```python
from numpy import linalg
u,s,v=linalg.svd(C)
print(u,"\n",s,"\n",v)
```

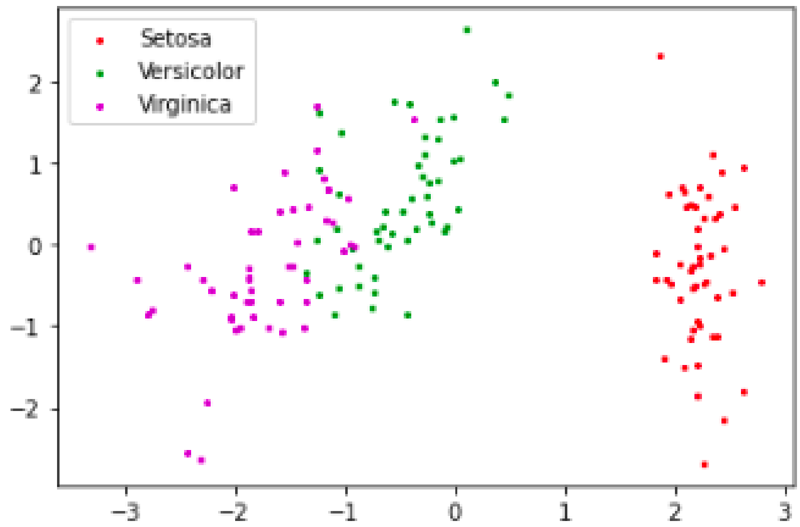
```
[[-0.52106591 -0.37741762  0.71956635  0.26128628]
 [ 0.26934744 -0.92329566 -0.24438178 -0.12350962]
 [-0.5804131  -0.02449161 -0.14212637 -0.80144925]
 [-0.56485654 -0.06694199 -0.63427274  0.52359713]]
 [2.91849782 0.91403047 0.14675688 0.02071484]
 [[-0.52106591  0.26934744 -0.5804131  -0.56485654]
 [-0.37741762 -0.92329566 -0.02449161 -0.06694199]
 [ 0.71956635 -0.24438178 -0.14212637 -0.63427274]
 [ 0.26128628 -0.12350962 -0.80144925  0.52359713]]
```

2.5 ε)

```python
f2c=np.array([v[0],v[1]])
f2c=f2c.transpose()
features2=np.matmul(features,f2c)
```

```python
setos=[]
versi=[]
virgi=[]
setos.append([])
setos.append([])
versi.append([])
versi.append([])
virgi.append([])
virgi.append([])
c=0
for i in features2:
  if c<50:
    setos[0].append(i[0])
    setos[1].append(i[1])
  elif c<100:
    versi[0].append(i[0])
    versi[1].append(i[1])
  else:
    virgi[0].append(i[0])
    virgi[1].append(i[1])
  c+=1

fig=plt.figure()
ax1=fig.add_subplot(111)
ax1.scatter(setos[0],setos[1],s=4,c='r',label='Setosa')
ax1.scatter(versi[0],versi[1],s=4,c='g',label='Versicolor')
ax1.scatter(virgi[0],virgi[1],s=4,c='m',label='Virginica')
ax1.legend()
plt.show()
```

Προβάλω τα δεδομένα πάνω στις δύο πρώτες κύριες συνιστώσες και σχεδιάζω τα αποτελέσματα που προκύπτουν :



2.5 στ)

```
ssum=0.
c=0
for i in s:
  c+=1
  ssum+=i
  print("Variance of dataset with",c,"principal component(s):",ssum/s.sum())
```

```
Variance of dataset with 1 principal component(s): 0.729624454132999
Variance of dataset with 2 principal component(s): 0.9581320720000164
Variance of dataset with 3 principal component(s): 0.9948212908928452
Variance of dataset with 4 principal component(s): 1.0
```