M3S7-Statistical Pattern Recognition
Project 2

Alexandre ELKRIEF
CID: 00732974

March 9, 2015

1 EM-Algorithm

The density of a finite mixture distribution has the form:

$$p(\mathbf{x}) = \sum_{i=1}^{K} \pi_i f_i(\mathbf{x}; \boldsymbol{\theta}_i)$$

where $f_i(.)$ are the K component densities, and π_j are mixing proportions. For fixed K, the EM algorithm can be used to estimate the parameters, $\boldsymbol{\theta}_i$, π_j, for $i = 1, ...K$, from an iid sample. In this question we will restrict to all component densities being p-dimensional normal, with density

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{p}{2}}|\boldsymbol{\Sigma}|^{1/2}} \exp\Bigg(-\frac{1}{2}((\mathbf{x}-\boldsymbol{\mu})^t\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\Bigg)$$

(a) Write an R function that uses the EM algorithm to find parameters which maximise the likelihood (or minimise the negative log-likelihood) for a sample of size n from $p(\mathbf{x})$, for a given choice of K. The function prototype should be

em.norm(x,means,covariances,mix.prop)

where **x** is an $n \times p$ matrix of data, **means**, **covariances**, and **mix.prop** are the initial values for the K mean vectors, covariance matrices and mixing proportions. Consider including arguments, with sensible defaults, for the convergence criterion and the maximum number of iterations.

For the code, see appendix. I have used a convergence criterion of 0.1 between to iterations of the log-likelihood and a upper limit for the number of iterations at 100. These choices kept the computation time within reasonable bounds.

(b) This question will use the first two columns of the object **synth.te** in the **MASS** library:
x <− synth.te[,-3]
For $K = 2, 3, 4, 5, 6$, use your function to compute the maximum likelihood estimates for the finite mixture of normal distributions, for these data. Select initial parameters either randomly, or by selecting from a plot of the data.

i. Construct a table that reports, for each choice of K, the maximised likelihood, and the AIC.

ii. On the basis of this table, which choice of K provides the best density estimate? For this choice, construct a contour plot of the estimated density, along with the data.

iii. Briefly discuss any problems you anticipate using the EM algorithm for computing a mixture model with more components, or in higher dimensions.

i. For this question, I set the initial parameters randomly. Moreover, I chose the same AIC as in the lecture notes, namely $M = 6K - 1$ where K corresponds to the number of mixture components. The code outputs a matrix corresponding to the following table:

K	ML	AIC
2	-555.4818	-566.4818
3	-529.8571	-546.8571
4	-486.0654	-509.0654
5	-486.3345	-515.3345
6	-485.1242	-520.1242

ii. On the basis of the above table, $K = 4$ seems to be the best choice. The following plot corresponds to such a choice. It displays a contour plot of the multivariate normal density and the data **x** from the the **MASS** library.

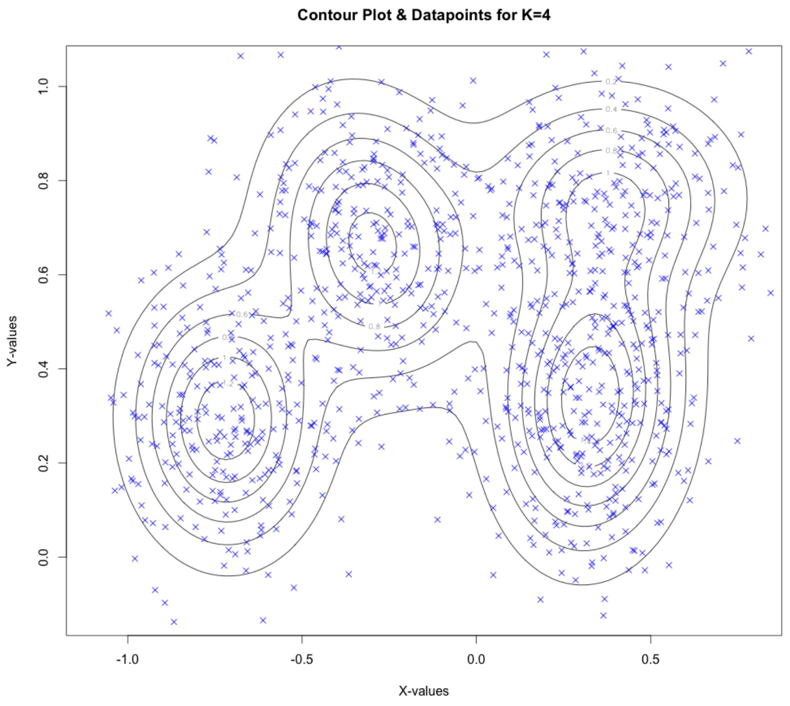

Figure 1: Contour plot for optimal choice of K

iii. The use of double for loops in my EM-Algorithm would cause the computational time to increase exponentially when working in higher dimensions or with more mixture components.

2 Rayleigh Distribution

Consider a two-class bivariate classification problem, with equal prior probabilities and class conditional densities given by:

$$f(x,y|C_i) = 4\theta_i^2 xy \exp(-\theta_i(x^2+y^2)) \qquad x,y>0$$

and $\theta_i > 0$ for $i = 1,2$. Note that this joint density is the product of Rayleigh distributions.

(a) Write an R function that generates a random sample of size n from class C_1 and a random sample of size n for class C_2. The function should return both the feature vectors and the class indicator. A function for generating Rayleigh distributed random variables is available.

(b) Obtain an expression for the decision boundary for minimum error. Suppose we are interested in the situation where the decision boundary for minimum error intersects with the midpoint of the line connecting the two class mean vectors. Derive an expression for θ_1 and θ_2 to satisfy this situation.

$$\frac{f(x,y|C_2)}{f(x,y|C_1)} = \frac{p(C_1)}{p(C_2)}$$
$$\Leftrightarrow \frac{4\theta_2^2 xy \exp(-\theta_2(x^2+y^2)}{4\theta_1^2 xy \exp(-\theta_1(x^2+y^2))} = 1$$
$$\Leftrightarrow \exp\{(x^2+y^2)(\theta_1-\theta_2)\} = \frac{\theta_1^2}{\theta_2^2}$$
$$\Leftrightarrow (x^2+y^2)(\theta_1-\theta_2) = 2\log(\frac{\theta_1}{\theta_2})$$
$$\Leftrightarrow T_{min}^2 = x^2+y^2 = 2\log(\frac{\theta_1}{\theta_2})(\frac{1}{\theta_1-\theta_2})$$

As we are in 2 dimensional-space the decision boundary for minimum error is a circle on the plane.

(c) Derive an expression in terms of θ_1 and θ_2 for the Bayes error rate. Now, suppose $\theta_2 = 1$ and $\theta_1 > \theta_2$. Use the golden ratio search algorithm developed in question 4 of project 1, to determine the value of θ_1 that gives a Bayes error rate of 15%. The solution occurs in the interval [3,10]. (Hint: The target function does not have to be differentiable at the minimum for the golden ratio search to work).

Integrating $f(x,y)$ we get:

$$F(x,y|C_i) = \int_x \int_y 4\theta_i^2 xy \exp(-\theta_i(x^2+y^2))\ dxdy$$
$$\Leftrightarrow F(x,y|C_i) = \int_0^{\pi/2} 2\sin(\phi)\cos(\phi)\ d\phi \int_0^u 2\theta_i^2 r^3 \exp(-\theta_i(r^2))\ dr$$
$$\Leftrightarrow F(x,y|C_i) = 1 - \theta_i u^2 \exp(-\theta_i u^2) - \exp(-\theta_i u^2)$$

Now the expression for the Bayes error rate is the following:

$$e_B = p(C_1)p(x|C_1 \geq T) + p(C_2)p(x|C_2 \leq T)$$
$$\Rightarrow e_B = 0.5[(1-F_1(T_{min})) + F_2(T_{min})]$$
$$\Rightarrow e_B = 0.5[(\theta_1 T_{min}^2 \exp(-\theta_1 T_{min}^2) + \exp(-\theta_i T_{min}^2) + 1 - \theta_2 T_{min}^2 \exp(-\theta_2 T_{min}^2) - \exp(-\theta_2 T_{min}^2))]$$

where $T_{min} = \sqrt{2\log(\frac{\theta_1}{\theta_2})(\frac{1}{\theta_1-\theta_2})}$

In order to find the value of θ_1 that gives a Bayes error rate of 15% we use the golden ratio search algorithm to minimize the function

$$f(\theta_1) = |e_B - 0.15|$$

This gives a value of $\theta_1 = 4.652476$

(d) Write down a discriminant function for each class, treating the parameter θ_i as unknown.

The code in the appendix uses the following discriminant function:

$$g_i(\mathbf{x}) = \log(p(C_i) + \log(p(\mathbf{x}|C_i))$$

(e) Let $\theta_1 = 4$ and $\theta_2 = 2$. Construct a plot of the unconditional density

$$f(x, y) = p(C_1)f(x, y|C_1) + p(C_2)f(x, y|C_2)$$

for the specified parameter values. Obtain a sample of 50 observations from each class. Add these data and the Bayes optimal decision boundary to the plot.

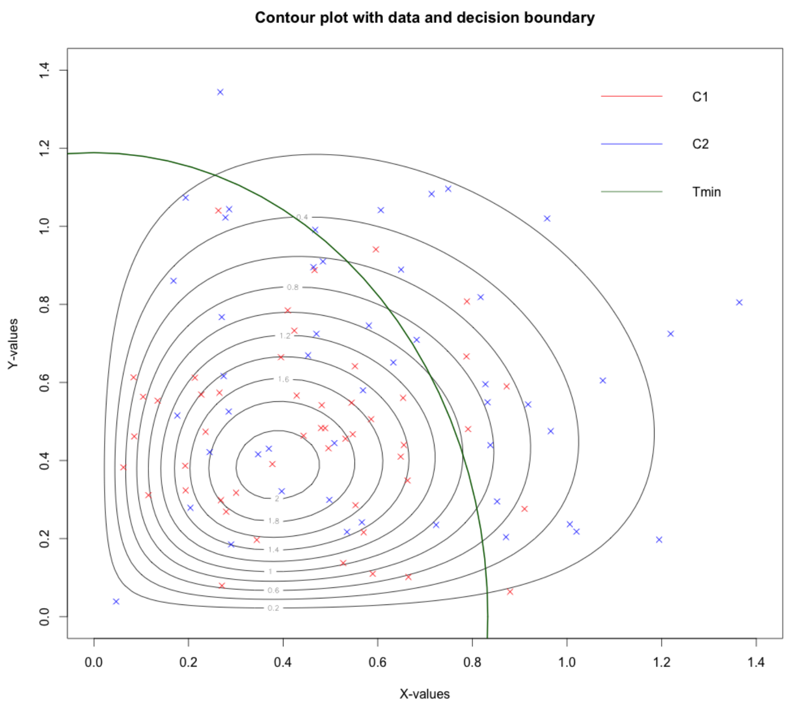

Figure 2: Unconditional Density and Decision Boundary

(f) Derive the maximum likelihood estimators for the parameters of each class, given a sample of size n from each class.

$$l(\theta_j) = \sum_{i=1}^{n} \log(f(x_i, y_i|\theta_j)$$

$$\Leftrightarrow l(\theta_j) = \sum_{i=1}^{n} \log(4\theta_j^2 x_i y_i \exp(-\theta_j(x_i^2 + y_i^2)))$$

$$\Leftrightarrow l(\theta_j) = \sum_{i=1}^{n} -\theta_j(x_i^2 + y_i^2) + 2\log(2\theta_j) + \log(x_i y_i)$$

$$\Leftrightarrow l(\theta_j) = -\theta_j \sum_{i=1}^{n}(x_i^2 + y_i^2) + \sum_{i=1}^{n} \log(x_i y_i) + 2n\log(2\theta_j)$$

$$\Rightarrow \frac{\partial l}{\partial \theta_j} = 0$$

$$\Rightarrow -\sum_{i=1}^{n}(x_i^2 + y_i^2) + \frac{2n}{2\theta_j} = 0$$

$$\Rightarrow \widehat{\theta_j} = \frac{2n}{\sum_{i=1}^{n}(x_i^2 + y_i^2)}$$

(g) Write two R functions, the first for computing the maximum likelihood estimates in (f) from a set of data generated by the function in (a), and the second for evaluating the discriminant function for each class, using the maximum likelihood estimates (the estimative discriminant function). Compute the discriminant scores for the data generated in (e) and estimate the error rate of this classifier on this training data.

The discriminant scores are computed within the discriminant function which allocates each points to one of the 2 classes.
The error rate of this classifier ranges between 25% and 37% depending on the set of points generated.
(see code in appendix)

(h) Obtain a training sample of size $n = 200$ and a test sample of size $n = 10000$, using the parameter values in part (e). Retain these training and test samples for use in Questions 3 and 4. Using these data sets, compute the training and test set error rates for

i. the estimative version of the true model, using the functions in part (g),

ii. Linear discriminant analysis,

iii. Quadratic discriminant analysis.

Provide a table of these error rates for the different models. Comment on the results.

The table of error rates is the following:

Dataset	ML model	LDA	QDA
Training	29%	32%	28%
Testing	31.8%	32.4%	31.7%

All 3 models seem to be in the same range of error. The values in the table are only indicative as the error depends on each random set generated by the R-code. The error from the training data is slightly lower than the one from the testing data which could be due to the ratio training points vs. testing points although this remains quite negligeable.

3 Kernel Density Estimation

This question is concerned with product kernel classifiers, based on a density estimate of the form:

$$\hat{p}(\boldsymbol{x}^*) = \frac{1}{n}\frac{1}{h_1 h_2 ... h_d}\sum_{i=1}^{n}\prod_{j=1}^{n} K\left(\frac{[\boldsymbol{x}^* - \boldsymbol{x}_i]_j}{h_j}\right)$$

where d is the dimension of the feature vector, $K()$ is a kernel function, n is the number of observations in the sample, and h_j, $j = 1, 2, ..., d$, are bandwidth parameters. Restrict attention to the normal kernel:

$$K(z) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right)$$

(a) Write a function with prototype **kde(x.star,data,bw)** that will perform a kernel density estimate for a single d-dimensional feature vector **x.star**, using an $n \times d$ data set data and d bandwidth parameters contained in a vector **bw**.

(see code in appendix)

(b) A classifier proceeds by evaluating a probability density estimate for each class, given bandwidth parameters, and combines these with information about prior probabilities, to obtain estimated posterior probabilities of class membership. In this question, we will use the same bandwidth parameters for both classes.
Using the data you generated in Question 2 part (h) (so $d = 2$), perform a 2-fold cross-validation for bandwidth selection in the following manner (you may take the class priors as $P(C_1) = P(C_2) = 0.5$):

i. Randomly split the training data generated in Question 2 (h) into two blocks of 100 samples each, D_1 and D_2, say.

ii. Construct a 20×20 grid of values for $H = (h1, h2)$ defined on $[0.05, 1] \times [0.05, 1]$, where h_j is the bandwidth parameter for variable j. For each selection of H:

- Classify each observation in D_2, using D_1 as the training set. Compute and retain the error rate.
- Classify each observation in D_1, using D_2 as the training set. Compute and retain the error rate.
- Combine these two error rates to give an average error rate for this selection of H.

(c) What is the minimum error rate found in part (b), and what are the associated bandwidths h_1 and h_2?

The minimum error rate is 30%, corresponding to $(h_1, h_2) = (0.3, 0.3)$

(d) For this h_1 and h_2 use all your training data and calculate the error rate on the 10000 test sample generated in Question 2 (h).

The error rate we obtain using the the best choice of (h_1, h_2) is 33.3% .

4 Distance-weighted K-NN

In an early attempt to combine the features of kernel methods and k-nearest neighbour methods, Dudani $(1976)^2$ proposed a *distance weighted* version of the k-nearest neighbour classifier. In this version, weights are assigned to the k-nearest neighbours, with closer neighbours being weighted more heavily. A feature vector $\mathbf{x}$ is then assigned to the class for which the weights of the representatives among the k neighbours sum to the greatest value.
One implementation of this strategy is as follows. For a specific feature vector, $\mathbf{x}$, let the number of neighbours of class i among the k-nearest neighbours be $k_i, i = 1, 2, ..., K$, where K is the number of classes. Note that $\sum_{i=1}^{K} k_i = k$. Let the Euclidean distance of $\mathbf{x}$ from each of these class i neighbours be d_i^j, for $j = 1, 2, ..., k_i$. The *weight* associated with class i is

$$w_i = \sum_{j=1}^{k_i} f(d_i^j) \quad i = 1, 2, ..., K$$

A popular choice for the weighting function f (in the nonparametric smoothing literature) is the *tricube* function

$$f(x) = (1 - |x|^3)^3$$

for $|x| \leq 1$, and 0 otherwise. This requires that we re-scale the distances, d_i^j , such that the largest, for all i and j, is 1. The allocation rule is then: allocate to class i if $w_i > w_j$ for all $j \neq i$.

(a) Write an R function that implements the procedure described above. The function prototype should be of the form

knn.dist(train,test,class,k)

where **train** is a matrix of training data (the rows are feature vectors), **class** is the associated vector of class indicators, **test** is a matrix of test data, and **k** is the number of neighbours.

(b) Using the training data obtained in Question 2 (h), perform a 2- fold cross-validation to select k from $k = 3, 7, 11, 15, 19, 23, 27, 31, 35$ or 39 for your distance weighted k-nearest neighbour classifier.

According to the cross-validation the best choice of k is 31.

(c) For your selection of k use all your training data and calculate the error rate on the 10000 test sample from Question 2 (h).

Using the value of k found above we get an error rate on the 10000 test sample of 32.9%

(d) You have now tried various methods on these data. Discuss the method you prefer for this problem and issues that motivate your preference.

Considering methods from Q2, Q3 and Q4, the Kernel density estimation method from question 2 is the most computationally expensive. Working in higher dimensions and with a bigger range for each bandwidth parameter increases the computational time significantly as the number of combinations of $h_1, h_2, ..., h_d$ increased exponentially. Moreover, this method doesn't produce significantly better error rates than the other models, at least in the case of our data.
The other methods give similar error rates and are similarly quick to run (although finding the optimal choice of K in Q4 isn't instantaneous). I would therefore use a combination of these 4 methods (ML, LDA, QDA and Distance-weighted K-NN) depending on the set of data that I need to test.

5 Appendix

```
#M3S7-Statistical Pattern Recognition
#Project 1

install.packages("mvtnorm")
install.packages("plotrix")
library("MASS")
library("mvtnorm")


em.norm<-function(x,means,covariances,mix.prop){
  "loglike"<-function(datapoints,pi,mu,sigma){
    takelog<-rep(0,k)
    log.like.inter<-rep(0,n)
    for (i in 1:n){
      for (j in 1:k){
        takelog[j]<-pi[j]*dmvnorm(datapoints[i,],mu[[j]],sigma[[j]])
      }
      log.like.inter[i]<-log(sum(takelog))
    }
    log.like<-sum(log.like.inter)
    return(log.like)
  }

  old.pii<-mix.prop
  old.mui<-means
  old.covi<-covariances
  n<-length(x[,1])
  k=length(old.pii)
  old.log.like<-loglike(x,old.pii,old.mui,old.covi)
  gam.nom<-matrix(0,nrow=n,ncol=k)
  new.mui<-list(0)[rep(1,k)]
  new.pii<-rep(0,k)
  new.covi<-list(0)[rep(1,k)]
  inter.mat<-list(0)[rep(1,n)]
  new.log.like<-0
  z<-0

  while (z<=100 & abs(old.log.like-new.log.like)>0.1){
    old.log.like<-new.log.like
    for (j in 1:n){
      for (kk in 1:k){
        gam.nom[j,kk]<-old.pii[kk]*dmvnorm(x[j,],old.mui[[kk]],old.covi[[kk]])
      }
    }
    #gam.nom
    gam.denom<-1/apply(gam.nom,1,sum)
    #gam.denom
    gamma<-gam.nom*gam.denom

    nvec<-apply(gamma,2,sum)

    for (i in 1:k){
      new.mui[[i]]<-(1/nvec[i])*apply(gamma[,i]*x,2,sum)
      new.pii[i]<-nvec[i]/n
    }
    new.mui
    new.pii

    for (i in 1:k){
      for (j in 1:n){
        inter.mat[[j]]<-gamma[j,i]*(x[j,]-new.mui[[i]])%*%t(x[j,]-new.mui[[i]])
      }
      new.covi[[i]]<-Reduce("+",inter.mat)*(1/nvec[i])
    }

    new.log.like<-loglike(x,new.pii,new.mui,new.covi)
    #print(new.log.like)

    old.pii<-new.pii
    old.mui<-new.mui
    old.covi<-new.covi
    z<-z+1
  }
  result<-list(means=new.mui,covariances=new.covi,mix.prop=new.pii,ML.est=new.log.like,iterations=z
      )
  return(result)
}

x<-as.matrix(synth.te[,-3])
#Initial covariances chose randomly
cov1<-matrix(c(1,0.6,0.6,1),nrow=2,ncol=2)
cov2<-matrix(c(1.5,0.7,0.7,1.5),nrow=2,ncol=2)
cov3<-matrix(c(2.8,1.2,1.2,2.8),nrow=2,ncol=2)
cov4<-matrix(c(0.8,0.2,0.2,0.8),nrow=2,ncol=2)
cov5<-matrix(c(1.3,0.4,0.4,1.3),nrow=2,ncol=2)
cov6<-matrix(c(3.1,0,0,3.1),nrow=2,ncol=2)

#K=2
old.pii2<-c(7/10,3/10)
old.mui2<-list(c(1,1.3),c(3,2))
old.covi2<-list(cov1,cov2)
k2<-list(old.mui2,old.covi2,old.pii2)
#K=3
old.pii3<-c(4/10,3/10,3/10)
old.mui3<-list(c(1,1.3),c(1,4),c(3,0))
old.covi3<-list(cov1,cov2,cov3)
k3<-list(old.mui3,old.covi3,old.pii3)
#K=4
old.pii4<-c(3/10,2/10,1/10,4/10)
old.mui4<-list(c(1,1.3),c(1,4),c(3,0),c(2.5,1.3))
old.covi4<-list(cov1,cov2,cov3,cov4)
k4<-list(old.mui4,old.covi4,old.pii4)
#K=5
old.pii5<-c(2/10,3/10,1/10,2/10,2/10)
old.mui5<-list(c(1,1.3),c(1,4),c(3,0),c(2.5,1.3),c(0.2,0.2))
old.covi5<-list(cov1,cov2,cov3,cov4,cov5)
k5<-list(old.mui5,old.covi5,old.pii5)
#K=6
old.pii6<-c(9/100,12/100,27/100,18/100,12/100,22/100)
old.mui6<-list(c(1,1.3),c(1,4),c(3,0),c(2.5,1.3),c(0.2,0.2),c(0.4,1.6))
old.covi6<-list(cov1,cov2,cov3,cov4,cov5,cov6)
k6<-list(old.mui6,old.covi6,old.pii6)

biglist<-list(k2,k3,k4,k5,k6)
#biglist

table.report<-matrix(c(2,3,4,5,6),nrow=5,ncol=3)
Sys.time()->start
for (i in 1:5){
  em<-em.norm(x,biglist[[i]][[1]],biglist[[i]][[2]],biglist[[i]][[3]])
  table.report[i,2]<-em$ML.est
  table.report[i,3]<-em$ML.est-(6*(i+1)-1)
  if (i>1){
    if (table.report[i,3]>table.report[i-1,3]){
      em.best<-em
    }
  }
}

table.report
print(Sys.time()-start)

#em.best<-em.norm(x,biglist[[3]][[1]],biglist[[3]][[2]],biglist[[3]][[3]])

em.best

#Q1b-iii ploting with best choice of K
xseq<-seq(-1.1,0.8,0.02)
yseq<-seq(-0.12,1.05,0.02)
"zvalues"<-function(x.star){
  zvalues1<-em.best$mix.prop[1]*dmvnorm(x.star,em.best$means[[1]],em.best$covariances[[1]])+em.
      best$mix.prop[2]*dmvnorm(x.star,em.best$means[[2]],em.best$covariances[[2]])+em.best$mix.prop
      [3]*dmvnorm(x.star,em.best$means[[3]],em.best$covariances[[3]])+em.best$mix.prop[4]*dmvnorm(x
      .star,em.best$means[[4]],em.best$covariances[[4]])
  return(zvalues1)
}
em.best
zvalues.tot<-matrix(0,nrow=length(xseq),ncol=length(yseq))

for (i in 1:length(xseq)){
  for (j in 1:length(yseq)){
    zvalues.tot[i,j]<-zvalues(c(xseq[i],yseq[j]))
  }
}

contour(xseq,yseq,zvalues.tot,main='Contour Plot & Datapoints for K=4',xlab='X-values',ylab='Y-
    values')
points(x[,1],x[,2],pch= 4 ,col='blue')

#Q2
#(a)
"rrayleigh" <- function (n,theta) {
  u <- runif(n,0,1)
  sqrt(-2*log(u))/sqrt(2*theta)
}
```

```
"rsample"<-function(n,theta1,theta2){
  #class1 sample
  rray.x1<-rrayleigh(n,theta1)
  rray.y1<-rrayleigh(n,theta1)

  rray.x2<-rrayleigh(n,theta2)
  rray.y2<-rrayleigh(n,theta2)

  rrayc1.class<-rep(1,n)
  rrayc2.class<-rep(2,n)

  rand.sample<-data.frame(rray.x1,rray.y1,rrayc1.class,rray.x2,rray.y2,rrayc2.class)
}
#the function returns a data frame for a better clarity and readability of my code.
#I later transform the samples generated by this function into matrices or lists to answer the
#following questions in an easier way.


#(b)-(c)
#the following function returns a value for the function |e_B-0.15| for a given theta1.
#we then use the golden ratio bracketing algorithm to find the value for theta1
#that minimizes it

"bayes.error"<-function(theta1){
  theta2=1
  Tmin<-sqrt(log(theta1/theta2)*(2/(theta1-theta2)))

  cdf.c1<-theta1*(Tmin^2)*exp(-theta1*Tmin^2)+exp(-theta1*Tmin^2)
  cdf.c2<-1-theta2*(Tmin^2)*exp(-theta2*Tmin^2)-exp(-theta2*Tmin^2)
  b.error<-abs(0.5*(cdf.c1+cdf.c2)-0.15)
  return(b.error)
}


#golden search algorithm
grbrack<-function(f,x1,x3,eps){
  ratio<-(1+sqrt(5))/2
  x2<-x1+(x3-x1)/ratio
  x2
  i=0

  while ((x3-x1)>eps){
    i=i+1

    f.x1=f(x1)
    f.x2=f(x2)
    f.x3=f(x3)
    if (f.x2<f.x1 && f.x2<f.x3){
      x3<-x2
      x2<-x1+(x3-x1)/ratio
    }
    else{
      x1<-x2
      x2<-x1+(x3-x1)/ratio
    }
  }
  return(x2)
}
grbrack(bayes.error,3,10,0.0000001)

#(d)
#the following function computes the discriminant score conditional on the 2-classes
#it then compares the 2 discriminant scores and allocates the tested point to
#the class with the highest score.
#discriminant function used: g_i(x)=log(p(c_i))+log(p(x|c_i))
"discriminant"<-function(x,y,theta1,theta2){

  p.c1<-4*theta1^2*x*y*exp(-theta1*(x^2+y^2))
  p.c2<-4*theta2^2*x*y*exp(-theta2*(x^2+y^2))

  g1<-log(0.5)+log(p.c1)
  g2<-log(0.5)+log(p.c2)
  discr<-g1-g2
  if (g1>=g2){
    class<-1
  }
  else{
    class<-2
  }
  return(class)
}

#points<-rsample(100,4,2)
# test<-discriminant(points$rray.x2[5],points$rray.y2[5],4,2)
# test

#(e)
#f(x,y) = p(C1)f(x,y|C1)+p(C2)f(x,y|C2)

theta1=4
theta2=2
xvals=seq(0,1.4,0.02)
yvals=xvals
uncond.cdf<-matrix(0,nrow=length(xvals),ncol=length(yvals))
length(xvals)
for (i in 1:length(xvals)){
  for (j in 1:length(yvals)){
    f.c1<-4*theta1^2*xvals[i]*yvals[j]*exp(-theta1*(xvals[i]^2+yvals[j]^2))
    f.c2<-4*theta2^2*xvals[i]*yvals[j]*exp(-theta2*(xvals[i]^2+yvals[j]^2))
    print(f.c1)
    uncond.cdf[i,j]<-0.5*(f.c1+f.c2)
  }
}
uncond.cdf
obs50<-rsample(50,4,2)

contour(xvals,yvals,uncond.cdf,main='Contour plot with data and decision boundary',xlab='X-values',
    ylab='Y-values')
points(obs50$rray.x1,obs50$rray.y1,col="red",pch=4)
points(obs50$rray.x2,obs50$rray.y2,col="blue",pch=4)
Tmin<-sqrt(log(4/2)*(2/(4-2)))
```

```
require("plotrix")
draw.circle(0,0,Tmin,nv=100,border="dark green",lty=1,lwd=2)
legend("topright",legend=c("C1","C2","Tmin"),col=c("red","blue","dark green"),lty=1,bty='n')


#(g)
"ml"<-function(data){
  n<-length(data$rray.x1)
  thetahat.c1<-2*n/sum((data$rray.x1)^2+(data$rray.y1)^2)
  thetahat.c2<-2*n/sum((data$rray.x2)^2+(data$rray.y2)^2)
  thetahat.est<-cbind(thetahat.c1,thetahat.c2)
  return(thetahat.est)
}


ml.est<-ml(obs50)
ml.est

"eval.discr"<-function(data,estimate){
  discr.alloc1<-rep(0,length(data$rray.x1))
  discr.alloc2<-rep(0,length(data$rray.x1))
  for (i in 1:length(data$rray.x1)){
    discr.alloc1[i]<-discriminant(data$rray.x1[i],data$rray.y1[i],estimate[1],estimate[2])
    discr.alloc2[i]<-discriminant(data$rray.x2[i],data$rray.y2[i],4,2)
  }
  data.new<-data.frame(data$rray.x1,data$rray.y1,data$rrayc1.class,discr.alloc1,data$rray.x2,
       data$rray.y2,data$rrayc2.class,discr.alloc2)
  return(data.new)
}
points.new<-eval.discr(obs50,ml.est)
#comparing the class allocated by the ml classifier and the true class:

tablec1<-table(points.new$data.rrayc1.class,points.new$discr.alloc1)
tablec2<-table(points.new$data.rrayc2.class,points.new$discr.alloc2)
errorc1<-tablec1[2]/(tablec1[1]+tablec1[2])
errorc2<-tablec2[1]/(tablec2[1]+tablec2[2])
errorc1
errorc2
classifier.error<-0.5*(errorc2+errorc1)
#the error of this classifier is the following:
classifier.error


#Q2h
training.sample<-rsample(100,4,2)
test.sample<-rsample(5000,4,2)

#Estimative using ML
training.ml<-ml(training.sample)
test.ml<-ml(test.sample)

training.eval<-eval.discr(training.sample,training.ml)
test.eval<-eval.discr(test.sample,test.ml)

training.class.tot<-c(training.eval$data.rrayc1.class,training.eval$data.rrayc2.class)
training.class.est<-c(training.eval$discr.alloc1,training.eval$discr.alloc2)
```

```
test.class.tot<-c(test.eval$data.rrayc1.class,test.eval$data.rrayc2.class)
test.class.est<-c(test.eval$discr.alloc1,test.eval$discr.alloc2)

training.ml.table<-table(training.class.tot,training.class.est)
test.ml.table<-table(test.class.tot,test.class.est)

training.ml.error<-(training.ml.table[2,1]+training.ml.table[1,2])/200
test.ml.error<-(test.ml.table[2,1]+test.ml.table[1,2])/10000

training.ml.error
test.ml.error

#LDA
#training
training2.data<-cbind(c(training.sample$rray.x1,training.sample$rray.x2),c(training.sample$rray.y1,
    training.sample$rray.y2))
training2.data

training2.class<-c(training.sample$rrayc1.class,training.sample$rrayc2.class)
#training2.class
training2.lda<-lda(training2.data,training2.class)
#training2.lda

prediction.training2<-predict(training2.lda)
training2.lda.table<-table(prediction.training2$class,training2.class)
training2.lda.error<-(training2.lda.table[1,2]+training2.lda.table[2,1])/200
training2.lda.error

#LDA-testing
test2.data<-cbind(c(test.sample$rray.x1,test.sample$rray.x2),c(test.sample$rray.y1,test.sample$rray
    .y2))
test2.class<-c(test.sample$rrayc1.class,test.sample$rrayc2.class)

test2.lda<-lda(test2.data,test2.class)
prediction.test2<-predict(test2.lda)
test2.lda.table<-table(prediction.test2$class,test2.class)
test2.lda.error<-(test2.lda.table[1,2]+test2.lda.table[2,1])/10000
test2.lda.error

#QDA
#training
training2.qda<-qda(training2.data,training2.class)
training2.predict.qda<-predict(training2.qda)
training2.table.qda<-table(training2.predict.qda$class,training2.class)
training2.qda.error<-(training2.table.qda[1,2]+training2.table.qda[2,1])/200
training2.qda.error

#testing
test2.qda<-qda(test2.data,test2.class)
test2.predict.qda<-predict(test2.qda)
test2.table.qda<-table(test2.predict.qda$class,test2.class)
test2.qda.error<-(test2.table.qda[1,2]+test2.table.qda[2,1])/10000
test2.qda.error


#Q3
```

```
"kernel"<-function(z){
  return((1/sqrt(2*pi))*exp(-z^2/2))
}

"kde"<-function(x.star,data,bw){
  d<-length(x.star)
  n<-length(data[,1])
  h<-prod(bw)
  m<-matrix(rep(0,n*d),nrow=n,ncol=d)
  for (i in 1:n){
    for (j in 1:d){
      m[i,j]<-(x.star[j]-data[i,j])/bw[j]
    }
  }
  m2<-apply(m,1:2,kernel)
  vec<-m2[,1]*m2[,2]
  s<-sum(vec)
  p<-(1/(n*h))*s
  return(p)
}
#h<-c(0.2,0.5)
#x.star<-c(points$rray.x2[7],points$rray.y2[7])

dataset<-cbind(training.sample$rray.x1,training.sample$rray.y1)
dataset2<-cbind(training.sample$rray.x2,training.sample$rray.y2)

#Q3b
#Dividing the training set in D1 and D2 randomly
dataset.c1<-cbind(training.sample$rray.x1,training.sample$rray.y1)
dataset.c2<-cbind(training.sample$rray.x2,training.sample$rray.y2)
d1.c1<-sample(1:100,50,replace=F)
d1.c2<-sample(1:100,50,replace=F)

d2.c1<-rep(0,100)
d2.c2<-rep(0,100)

for (i in 1:100){
  if (i %in% d1.c1==FALSE){
    d2.c1[i]<-i
  }
  if (i %in% d1.c2==FALSE){
    d2.c2[i]<-i
  }
}
d2.c1<-d2.c1[d2.c1!=0]
d2.c2<-d2.c2[d2.c2!=0]

cross1.c1<-matrix(0,nrow=50,ncol=2)
cross1.c2<-cross1.c1
cross2.c1<-cross1.c1
cross2.c2<-cross1.c1

for (i in 1:50){
  cross1.c1[i,]<-dataset.c1[d1.c1[i],]
  cross2.c1[i,]<-dataset.c1[d2.c1[i],]
  cross1.c2[i,]<-dataset.c2[d1.c2[i],]
  cross2.c2[i,]<-dataset.c2[d2.c2[i],]
}
D1<-rbind(cross1.c1,cross1.c2)
D1.true.c<-c(rep(1,50),rep(2,50))
D2<-rbind(cross2.c1,cross2.c2)
D2.true.c<-D1.true.c
H<-expand.grid(seq(0.05,1,0.05),seq(0.05,1,0.05))



"error.class"<-function(h,testdata,traindata.c1,traindata.c2,true.class.vec){
  T<-matrix(0,nrow=length(testdata[,1]),ncol=3)
  for (i in 1:length(testdata[,1])){
    T[i,1]<-kde(testdata[i,],traindata.c1,h)
    T[i,2]<-kde(testdata[i,],traindata.c2,h)
    if (T[i,1]>T[i,2]){
      T[i,3]<-1
    }
    else{
      T[i,3]<-2
    }
  }
  tt<-table(T[,3],true.class.vec)
  error<-(tt[1,2]+tt[2,1])/length(T[,3])
  return(error)
}

error.vec12<-rep(0,400)
error.vec21<-rep(0,400)
for (i in 1:400){
  h<-c(H[i,1],H[i,2])
  error.vec12[i]<-error.class(h,D2,cross1.c1,cross1.c2,D2.true.c)
  error.vec21[i]<-error.class(h,D1,cross2.c1,cross2.c2,D1.true.c)
}

# h<-c(H[1,1],H[1,2])
# min(error.vec12)
# min(error.vec21)
error.tot<-0.5*(error.vec12+error.vec21)
error.tot
min(error.tot)
b<-which.min(error.tot)
h.best<-c(H[b,1],H[b,2])
h.best

#Q3d
#test.sample
datatrain.c1<-cbind(training.sample$rray.x1,training.sample$rray.y1)
datatrain.c2<-cbind(training.sample$rray.x2,training.sample$rray.y2)
datatest.c1<-cbind(test.sample$rray.x1,test.sample$rray.y1)
datatest.c2<-cbind(test.sample$rray.x2,test.sample$rray.y2)
datatest.tot<-rbind(datatest.c1,datatest.c2)
#datatest.tot
datatest.true.c<-c(test.sample$rrayc1.class,test.sample$rrayc2.class)
```

```
error.class(h.best,datatest.tot,datatrain.c1,datatrain.c2,datatest.true.c)


#Q4

#training.sample
#Redefining my training and test sets of data
q4train.c1<-cbind(training.sample$rray.x1,training.sample$rray.y1)
q4train.c2<-cbind(training.sample$rray.x2,training.sample$rray.y2)
q4train<-rbind(q4train.c1,q4train.c2)
q4train.class<-c(training.sample$rrayc1.class,training.sample$rrayc2.class)


#knn.alloc(c(0.3,0,3),q4train,q4train.class,11)

"knn.dist"<-function(train,test,class,k){
  "dist"<-function(x,y){
    return(sqrt((x[1]-y[1])^2+(x[2]-y[2])^2))
  }
  "dist.vector"<-function(x.star,datapoints){
    dist.vec<-rep(0,length(datapoints[,1]))
    for (i in 1:length(datapoints[,1])){
      dist.vec[i]<-dist(x.star,datapoints[i,])
    }
    return(dist.vec)
  }
  "tricube"<-function(x){
    y<-(1-abs(x)^3)^3
    return(y)
  }

  "knn.alloc"<-function(x.star,traindata,classvec,K){

    dist.vec<-dist.vector(x.star,traindata)
    scale.dist.vec<-dist.vec/max(dist.vec)

    knn.index<-order(scale.dist.vec)[1:K]
    knn.points<-matrix(0,nrow=K,ncol=6)

    for (i in 1:K){
      #first points of the knn.points is x.star
      knn.points[i,1]<-traindata[knn.index[i],1]
      knn.points[i,2]<-traindata[knn.index[i],2]
      knn.points[i,3]<-classvec[knn.index[i]]
      knn.points[i,4]<-scale.dist.vec[knn.index[i]]

      if (knn.points[i,3]==1){
        knn.points[i,5]<-tricube(knn.points[i,4])
      }
      else{
        knn.points[i,6]<-tricube(knn.points[i,4])
      }
    }
    #print(knn.points)
    weights.vec1<-knn.points[,5]
    weights.vec2<-knn.points[,6]
    #need to remove 1

    w1<-sum(weights.vec1)
    w2<-sum(weights.vec2)
    if (w1>w2){
      x.star.alloc<-1
    }
    else if (w1<w2){x.star.alloc<-2}
    else{x.start.alloc<-sample(1:2,1)}

    #   print(knn.points)
    #   print(weights.vec1)
    #   print(weights.vec2)
    return(x.star.alloc)
  }
  alloc<-apply(test,1,knn.alloc,traindata=train,classvec=class,K=k)
  return(alloc)
}

#Q4b
kvec<-c(3,7,11,15,19,23,27,31,35,39)
#D1
#D2
#D1.true.c
#D2.true.c
"knn.cross.err"<-function(D1,D2,D1.class,D2.class,K){
  knn12<-knn.dist(D1,D2,D1.class,K)
  knn12.table<-table(D1.class,knn12)
  knn12.err<-(knn12.table[1,2]+knn12.table[2,1])/100

  knn21<-knn.dist(D2,D1,D2.class,K)
  knn21.table<-table(D2.class,knn21)
  knn21.err<-(knn21.table[1,2]+knn21.table[2,1])/100

  knn.average<-0.5*(knn12.err+knn21.err)
  return(knn.average)
}



"knn.k.opt"<-function(D1,D2,D1.class,D2.class,kvec){
  knn.err.vec<-rep(0,length(kvec))
  for (i in 1:length(kvec)){
    knn.err.vec[i]<-knn.cross.err(D1,D2,D1.true.c,D2.true.c,kvec[i])
  }

  z<-which.min(knn.err.vec)
  #print(knn.err.vec)
  #print(z)
  best<-kvec[z]
  return(best)
}

k.best<-knn.k.opt(D1,D2,D1.true.c,D2.true.c,kvec)
k.best
```

```
#Q4c
#Using the k.best found above
q4test.c1<-cbind(test.sample$rray.x1,test.sample$rray.y1)
q4test.c2<-cbind(test.sample$rray.x2,test.sample$rray.y2)
q4test<-rbind(q4test.c1,q4test.c2)
q4test.class<-c(test.sample$rrayc1.class,test.sample$rrayc2.class)

q4c<-knn.dist(q4train,q4test,q4train.class,k.best)
q4c.table<-table(q4test.class,q4c)
q4c.error<-(q4c.table[1,2]+q4c.table[2,1])/sum(q4c.table)
q4c.error
```