STAT-721 Take Home Part of Mid-Term

Segbehoe, Lawrence Sethor

October 17, 2018

Midterm 721 - FA2018

Take home part

1) Use the training data below to build a **Gaussian Process classifier** Section 6.4.5 and 6.4.6 in the text book You do not need to optimise the parameters of the kernel function using equations 6.89 and following The kernel function that you need to use is

$$k(\boldsymbol{a}, \boldsymbol{b}) = exp\Big(-0.5(\boldsymbol{a} - \boldsymbol{b})^T(\boldsymbol{a} - \boldsymbol{b})\Big),$$

where $\boldsymbol{a}$,$\boldsymbol{b}$ are multidimensional vectors or $\boldsymbol{a}, \boldsymbol{b} \in R^D$.

Goal: The main deliverable is a function that accepts two data frames as arguments. The first data frame contains training data in 3 columns (x,y,class) and the second data frame contains any number of new samples in 2 columns (x,y) The function has to return the probability OF CLASS 1 for all the points in the second data frame

You can use the `optim()` function in R. The rest of the algorithm needs to be coded from scratch apart from trivial R functions

2) Provide the COMPLETE derivation for exercise 6.22 in the text book

DEADLINE: Friday October 19 @ 0900

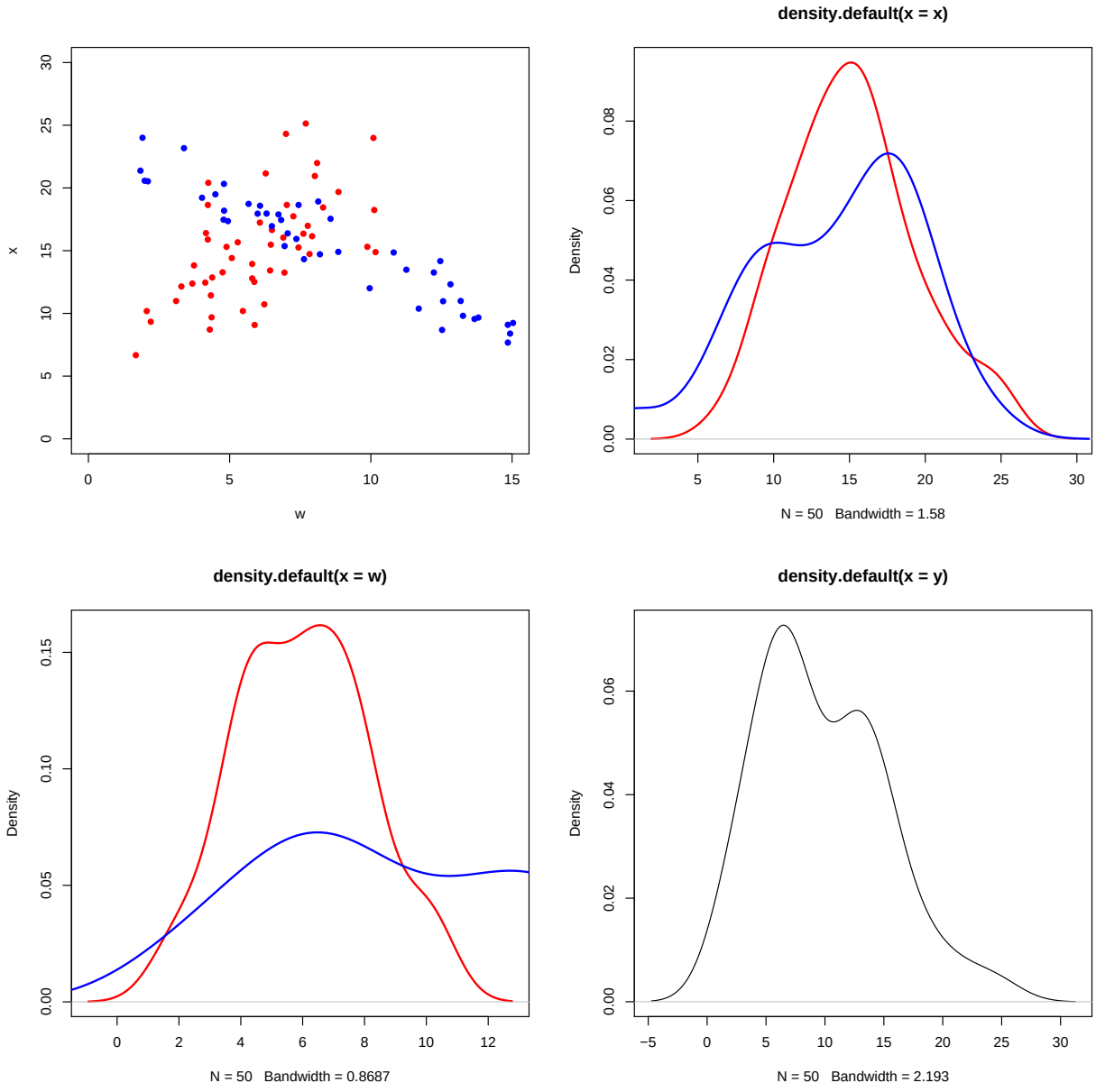
There will be no extension unless medically related

WORK BY YOURSELF!!!!

Code will be audited, students with duplicated code or analytical solution for the derivation will receive an F

```
### Seed set correctly to:
set.seed(7677)

### generate data
n <- 50
w <- rgamma(n,shape=6,rate=1)
x <- w + rgamma(n,10,1)
y <- rgamma(n,shape=3,rate=0.3)
z <- 20-y + rgamma (n,8,2)

### look at the data
# quartz()
par(mfrow=c(2,2))
plot(w,x,col="red",pch=16,xlim=c(0,15),ylim=c(0,30))
points(y,z,col="blue",pch=16)
plot(density(x),col="red",lwd=2)
lines(density(z),col="blue", lwd=2)
plot(density(w),col="red",lwd=2)
```

```
lines(density(y),col="blue", lwd=2)
plot(density(y))
```




```
par(mfrow=c(1,1))

### put the data together
data.df <- data.frame(x=c(w,y), y=c(x,z), class = rep(c(1,2),each=n))

### use data.df
```

Gaussian Process Classifier

```
# Function required in the Gaussian Process Classification function

# Gaussian Kernel (GK)
GK <- function(Xn, Xm){
  kern = exp(-0.5*((as.matrix(Xn-Xm))%*%t(as.matrix(Xn-Xm))))
  return(kern)
}

# Equation 4.59: Sigmoid function
sig.fun <- function(x){
  1/(1 + exp(-x))
}


# Equation 4.154: The Kappa function
kapa <- function(X){
  1/ sqrt(1 + (pi*X)/8)
}


# Equation 6.80: Laplace Approximation
Laplace.aN <- function(a.N, t.N, C.N){

  a.N # vector of mode
  t.N # vector of target values
  C.N # Gram Matrix
  N = dim(C.N)[1]

  Laplace =  (
                -0.5*t(a.N)%*%solve(C.N)%*%a.N
                - (N/2)*log(2*pi)
                - 0.5*log(det(C.N))
                + t(t.N)%*%a.N
                - sum(log(1 + exp(a.N)))
              )

  return(Laplace)
}


# Equation 6.81: Gradient of the Laplace Approximaton
Gradient.a.N <- function(a.N, t.N, C.N){

  a.N # vector of mode
  t.N # vector of target values
  C.N # Gram Matrix

  Gradient =  (
                  t.N
                - sig.fun(a.N)
                - solve(C.N)%*%a.N
```

```
              )

  return(Gradient)
}
```

```
#############################################################################
##############    Gaussian Process Classification (GPC)      #################
#############################################################################

GPC <- function(training.data, test.data){

  # extracting feactures from the training data

  N = dim(training.data)[1] # number of observations
  target = training.data[,3] - 1 # vector of classes
  X.train  = training.data[,-3] # training data
  colnames(X.train)  <- c("x","y") # column labels


  # Now extract the feature of the test data

  N2 <- dim(test.data)[1]# number of observation of test data
  X.test <- test.data # test data
  colnames(X.test)  <- NULL # removing column labels
  colnames(X.test) <- c("x","y")

  X <- rbind(X.train,X.test, deparse.level = 1) # both training and test data
  X <- data.frame(X)
  N_N2 <- dim(X)[1]

  # get the Gram matrix for both training and test data
  Gram.matrix.all <- matrix(NA,nrow = (N_N2), ncol= (N_N2))

  for (i in 1:(N_N2)) {
    for (j in 1:(N_N2)) {
      Gram.matrix.all[i,j] <- GK(X[i,], X[j,])
    }
  }


  # get the Gram matrix for training
  Gram.matrix <- Gram.matrix.all[1:N,1:N]


  # Initial values for par of the optim function
  a.N.initial = rep(-0.5,N)

  # Optim function --- with gradient
```

```r
optim.values <- optim(par    = a.N.initial,
                      fn     = Laplace.aN,
                      gr     = Gradient.a.N,
                      # further arguments to be passed to fn and gr
                      t.N    = target,
                      C.N    = Gram.matrix,
                      # quasi-Newton method
                      method = "BFGS",
                      # set control parameters
                      control= list(maxit   = 300000, # set iteration limit
                                    fnscale = -1  # change optim to maximize fn
                                    )
                      )

# extract mode from the optimation result
mode.N <- optim.values$par # mode.N

# get diagonal element of W.N
Derivative.1.sig.fun <- sig.fun(mode.N)*(1 - sig.fun(mode.N))

# get W.N
W.N <- diag(Derivative.1.sig.fun)

# get Equation 6.85: The Hessian at the mode.N
H <- W.N + solve(Gram.matrix)

# get k matrix associated with the test data
matrix.k <- Gram.matrix.all[1:N, (N+1):(N_N2)]

# get c matrix associated with the test data
matrix.c <- Gram.matrix.all[(N+1):(N_N2), (N+1):(N_N2)]

# get Equation 6.87: The Expection of a.N+1 given t.N
E.a.Nplus1 <- t(matrix.k)%*%(target - sig.fun(mode.N))

# get Equation 6.88: The Variance of a.N+1 given t.N
Var.a.Nplus1 <- matrix.c - t(matrix.k)%*%solve(solve(W.N) + Gram.matrix)%*%matrix.k



# The testbook talks about ignoring the effect of variance and
# consider the only the mean if we are interested in the decision boundry

# Equation 4.155: Probability of Class 1
Prob.class1.ignored <- sig.fun(kapa(Var.a.Nplus1)%*%E.a.Nplus1)



# we now put only the mean in sigmoid function -- sig.fun
# to get the required classification

# Equation 4.155: Probability of CLASS 1
Prob.class1.due.to.mean <- sig.fun(E.a.Nplus1) # Required output
```

```
  # Classification
  class.observation <- apply(Prob.class1.due.to.mean, 1, function(x)ifelse(x<0.5,1,2))

  # Create a data frame for the prob. of class of the classification due to the prob.
  Prob.out <- data.frame(Prob.CLASS1 = Prob.class1.due.to.mean,
                         CLASSES     = class.observation)
  return(list(Prob.Class = Prob.out))
}


# checking the GPC classifier using -- data.df -- as the test data

GPC.res = GPC(training.data = data.df, test.data = data.df[,-3] )
GPC.res
```

```
## $Prob.Class
##     Prob.CLASS1 CLASSES
## 1     0.2253234       1
## 2     0.3329558       1
## 3     0.2469485       1
## 4     0.3971538       1
## 5     0.2297622       1
## 6     0.4640180       1
## 7     0.5359851       2
## 8     0.4082261       1
## 9     0.3302382       1
## 10    0.2849074       1
## 11    0.2040470       1
## 12    0.2219675       1
## 13    0.2178063       1
## 14    0.2674412       1
## 15    0.2797830       1
## 16    0.3235357       1
## 17    0.2775483       1
## 18    0.4021333       1
## 19    0.3625033       1
## 20    0.4132033       1
## 21    0.2381891       1
## 22    0.4047844       1
## 23    0.3288644       1
## 24    0.2350308       1
## 25    0.2520184       1
## 26    0.3990484       1
## 27    0.6208697       2
## 28    0.3602469       1
## 29    0.4231878       1
## 30    0.3357123       1
## 31    0.3593385       1
## 32    0.6331991       2
## 33    0.4726695       1
## 34    0.1996784       1
## 35    0.3975939       1
```

```
## 36    0.2829479       1
## 37    0.3213455       1
## 38    0.4664163       1
## 39    0.4313692       1
## 40    0.2576539       1
## 41    0.4864247       1
## 42    0.1993767       1
## 43    0.3945725       1
## 44    0.5432669       2
## 45    0.3205806       1
## 46    0.3466490       1
## 47    0.2920323       1
## 48    0.5497024       2
## 49    0.6063719       2
## 50    0.3667534       1
## 51    0.6345271       2
## 52    0.6398836       2
## 53    0.5291682       2
## 54    0.6961120       2
## 55    0.7334292       2
## 56    0.6000629       2
## 57    0.5680623       2
## 58    0.7374217       2
## 59    0.4650338       1
## 60    0.6404936       2
## 61    0.7481847       2
## 62    0.7069973       2
## 63    0.7713962       2
## 64    0.6378744       2
## 65    0.6686186       2
## 66    0.5312401       2
## 67    0.7725591       2
## 68    0.6775781       2
## 69    0.6838103       2
## 70    0.6827692       2
## 71    0.4859371       1
## 72    0.6206564       2
## 73    0.6455240       2
## 74    0.4250501       1
## 75    0.6778125       2
## 76    0.6424184       2
## 77    0.4187523       1
## 78    0.5154819       2
## 79    0.7353458       2
## 80    0.6938847       2
## 81    0.7582030       2
## 82    0.7412388       2
## 83    0.5356976       2
## 84    0.7384350       2
## 85    0.6159711       2
## 86    0.5535501       2
## 87    0.6000629       2
## 88    0.6927704       2
## 89    0.7165415       2
```

```
## 90    0.6074942       2
## 91    0.5625340       2
## 92    0.7540007       2
## 93    0.7742799       2
## 94    0.6821075       2
## 95    0.4187353       1
## 96    0.5765067       2
## 97    0.7135553       2
## 98    0.6187640       2
## 99    0.7089077       2
## 100   0.6803174       2
```

```
## Table of misclassification
Pred = GPC.res$Prob.Class[,2]

table(Pred, classes = data.df[,3])
```

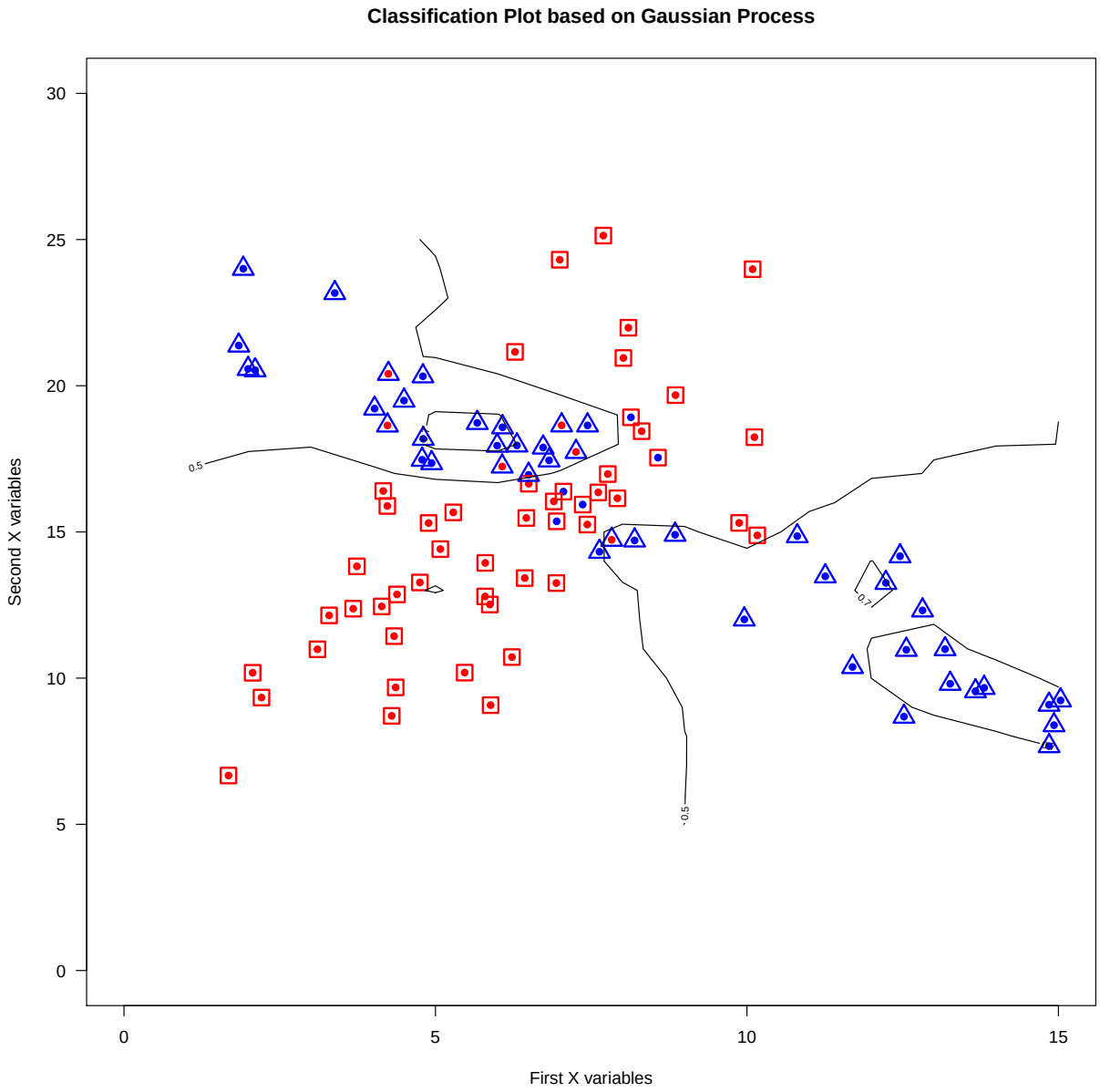
```
##     classes
## Pred  1  2
##    1 44  5
##    2  6 45
```

Testing the Gaussian Process Classifier

Using given data as test data

```
################### Testing the GPC function #########################

# Inverse Logistic Sigmoid function
sig.inv <- function(X){
  log(X/(1 - X))
}

# Generate a grid for contour lines
u = seq(1,15, by = 1)
v = seq(5,25, by = 1)

Newdata = expand.grid(u,v)

# dim(Newdata) 315 * 2
# class(Newdata) "data.frame"

# Generating Prob for "Newdata" from the grid data
GPC.res.newdata = GPC(training.data = data.df, test.data = Newdata)
ProbofCLASS1 = GPC.res.newdata$Prob.Class[,1]


## make matrix for the coutour plot
matrix.uv = matrix(ProbofCLASS1, nrow = length(u), ncol = length(v))


colors.pred = ifelse(Pred == 1, 2, 4)
pch.pred = colors.pred - 2
```

```
plot(w,x,col="red",pch=16, xlim=c(0,15),ylim=c(0,30),
     xlab = "First X variables",
     ylab = "Second X variables", las = 1,
     main = "Classification Plot based on Gaussian Process")
points(y,z,col="blue",pch=16)
points(data.df[,-3], col= colors.pred, pch = pch.pred,lwd = 2, cex = 2 )
graphics::contour(x = u, y = v, z = matrix.uv,
                  levels = c(0.1,0.2 ,0.5, 0.7), add = T)
```



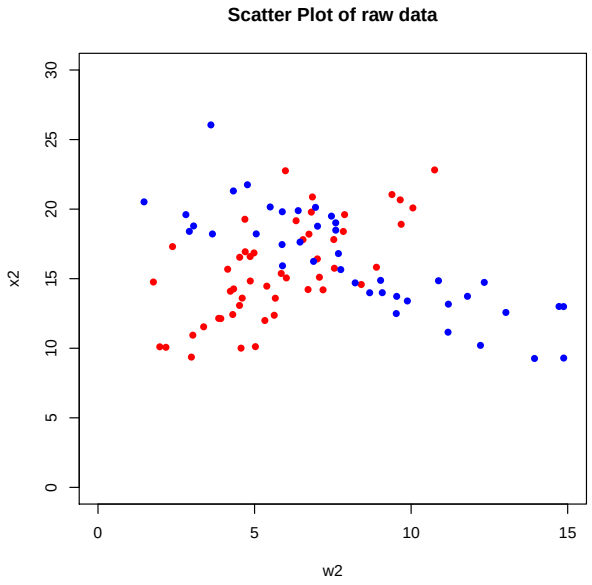
Using a new data generated with a different seed number

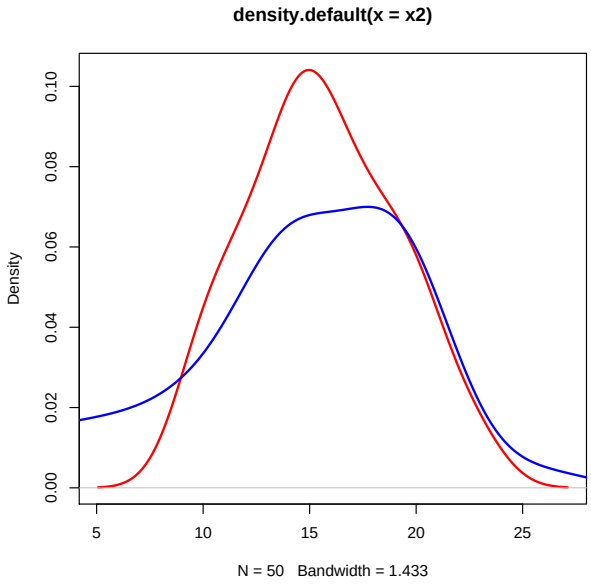
```
################### Testing the GPC function #########################

### Seed set correctly to:
set.seed(777)

### generate data
n <- 50
w2 <- rgamma(n,shape=6,rate=1)
x2 <- w2 + rgamma(n,10,1)
y2 <- rgamma(n,shape=3,rate=0.3)
# plot(density(y2))
z2 <- 20-y2 + rgamma (n,8,2)


### put the data together
data.df2 <- data.frame(x=c(w2,y2), y=c(x2,z2), class = rep(c(1,2),each=n))

### use data.df
GPC.res2 = GPC(training.data = data.df2, test.data = data.df2[,-3])

## Table of misclassification
Pred2 = GPC.res2$Prob.Class[,2]
table(Pred2, classes = data.df2[,3])
```

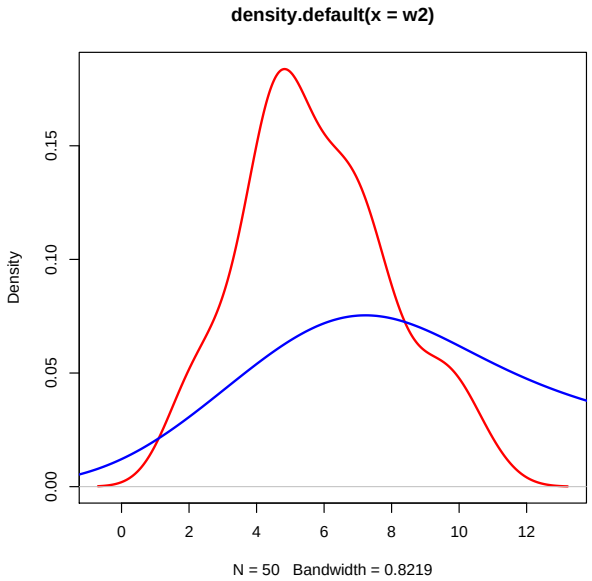
```
##       classes
## Pred2  1  2
##     1 43  3
##     2  7 47
```

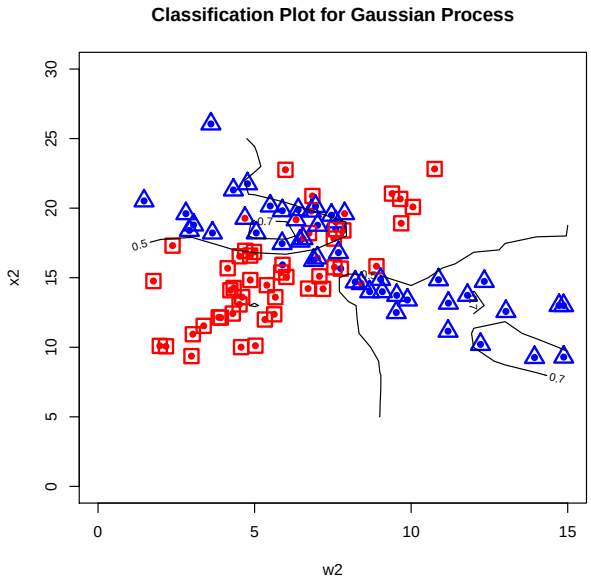
```
colors.pred2 = ifelse(Pred2 == 1, 2, 4)
pch.pred2 = colors.pred2 - 2

# Generating Prob for "Newdata" from the grid data
GPC.res2.newdata = GPC(training.data = data.df2, test.data = Newdata)
ProbofCLASS2 = GPC.res.newdata$Prob.Class[,1]

## make matrix for the coutour plot
matrix.uv = matrix(ProbofCLASS2, nrow = length(u), ncol = length(v))


### look at the data
# quartz()
par(mfrow=c(2,2))

plot(w2,x2,col="red",pch=16,xlim=c(0,15),ylim=c(0,30),
     main = "Scatter Plot of raw data")
points(y2,z2,col="blue",pch=16)
plot(density(x2),col="red",lwd=2)
lines(density(z2),col="blue", lwd=2)
plot(density(w2),col="red",lwd=2)
lines(density(y2),col="blue", lwd=2)
```

```
plot(w2,x2,col="red",pch=16,xlim=c(0,15),ylim=c(0,30),
     main = "Classification Plot for Gaussian Process")
points(y2,z2,col="blue",pch=16)
points(data.df2[,-3], col= colors.pred2, pch = pch.pred2,lwd = 2, cex = 2 )
graphics::contour(x = u, y = v, z = matrix.uv,
                  levels = c(0.1,0.2 ,0.5, 0.7),
                  add = T)
```










```
par(mfrow=c(1,1))
```