SMM636 Machine Learning (PRD2 A 2019/20)

R exercises 2: k nearest neighbours (Part 1)

DR. Rui Zhu, DR. Feng Zhou

2020-01-16

In R exercises 2, you will know:

- How to get training and test sets
- How to perform kNN in R
- How to scale data

Don't forget to change your working directory!

Contents

1 R Packages and datasets required

1.1 Use Package class or anything else

- Type the following code to install and activate the package:
  - install.packages("class")
  - library(class)

1.2 Use Package caret or anything else

- Type the following code to install and activate the package:
  - install.packages("caret")
  - library(caret)

1.3 Dataset is given as 'Edgar Anderson's Iris Data'

```
#This data frame is already contained in the R
#Therefore, no need to read from other source
#Use "?iris" to check the detail about this dataset
```

s # kNN We can perform kNN using the `knn()` function in the `class` package. Install the package and use it in a new workspace. After installing the package this time, you just need to run `library(class)` next time to use the functions in the package. There is no need to install it again. Now we can use the `knn()` function. Use `?knn` to see the help information of the `knn()` function. Read the help information carefully to understand what are the input options and outputs.

```
?knn
```

Here is the example from the help of `knn()`:

```
# get training and test set
train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
# get class factor
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
# classification using knn, with k=3
set.seed(47)
knn_pred=knn(train, test, cl, k = 3, prob=TRUE)
# see the attributes of the knn model
attributes(knn_pred)
```

```
## $levels
## [1] "c" "s" "v"
##
## $class
## [1] "factor"
##
## $prob
##  [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [8] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [22] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667
## [29] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667 1.0000000
## [36] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [43] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [50] 1.0000000 1.0000000 0.6666667 0.7500000 1.0000000 1.0000000 1.0000000
## [57] 1.0000000 1.0000000 0.5000000 1.0000000 1.0000000 1.0000000 1.0000000
## [64] 0.6666667 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [71] 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667
```

Exercise: Use `?iris3` to see the structure of `iris3` and think about why we can get the training and test sets using the corresponding commands.

We have to use `set.seed` before `knn()` to make the results reproducible, because if several observations are tied as nearest neighbours, then `knn()` will randomly break the tie. Change seed to see if there's change in the prediction.

Make sure you understand the outputs from `knn()`. What happens if we change `prob=FALSE`? Play with the `knn()` function by changing the inputs, e.g. `k=1` etc.

Note that if you run the following code after the above one, the original `knn_pred` (with `k=3` and `prob=TRUE`) will be covered by the new `knn_pred` (with `k=5` and `prob=FALSE`).

```
knn_pred=knn(train, test, cl, k = 5, prob=FALSE)
knn_pred
```

```
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s s s c c v c c c c c v c
## [36] c c c c c c c c c c c c c c c v c c v v v v v c v v v v c v v v v v v
## [71] v v v v v
## Levels: c s v
```

```
attributes(knn_pred)
```

```
## $levels
## [1] "c" "s" "v"
##
## $class
## [1] "factor"
```

To get two `knn` predictions from two models, you can simply change the name of the prediction:

```
knn_pred2=knn(train, test, cl, k = 5, prob=FALSE)
```

Play with the above codes, to see how change of training/test data, change of variables or change of k will affect the prediction.

Exercise: Randomly sample 25 observations from each class of the iris data to form the training set and the rest as the test set. Obtain the predictions of the test set of `knn` with `k=5` on this training/test split. Make sure your result is reproducible.

`knn1()` in the `class` package provides an easy way to do 1NN:

```
set.seed(47)
knn1_pred=knn1(train, test, cl)
```

If we set `prob=TRUE`, `knn()` returns the probabilities associated with the predicted class, i.e. only the largest probabilities are returned. If we want to see all probabilities, we can use the `knn3Train()` function in the `caret` package.

```
#If 'caret' package is already installed in the beginning, otherwise
#install.packages("caret")
#library(caret)
set.seed(47)
knn3_pred=knn3Train(train, test, cl, k = 3, prob=TRUE)
attributes(knn3_pred)
```

```
## $prob
##                c s         v
##  [1,] 0.0000000 1 0.0000000
##  [2,] 0.0000000 1 0.0000000
##  [3,] 0.0000000 1 0.0000000
##  [4,] 0.0000000 1 0.0000000
##  [5,] 0.0000000 1 0.0000000
##  [6,] 0.0000000 1 0.0000000
##  [7,] 0.0000000 1 0.0000000
##  [8,] 0.0000000 1 0.0000000
##  [9,] 0.0000000 1 0.0000000
## [10,] 0.0000000 1 0.0000000
## [11,] 0.0000000 1 0.0000000
## [12,] 0.0000000 1 0.0000000
## [13,] 0.0000000 1 0.0000000
```

```
## [14,] 0.0000000 1 0.0000000
## [15,] 0.0000000 1 0.0000000
## [16,] 0.0000000 1 0.0000000
## [17,] 0.0000000 1 0.0000000
## [18,] 0.0000000 1 0.0000000
## [19,] 0.0000000 1 0.0000000
## [20,] 0.0000000 1 0.0000000
## [21,] 0.0000000 1 0.0000000
## [22,] 0.0000000 1 0.0000000
## [23,] 0.0000000 1 0.0000000
## [24,] 0.0000000 1 0.0000000
## [25,] 0.0000000 1 0.0000000
## [26,] 1.0000000 0 0.0000000
## [27,] 1.0000000 0 0.0000000
## [28,] 0.3333333 0 0.6666667
## [29,] 1.0000000 0 0.0000000
## [30,] 1.0000000 0 0.0000000
## [31,] 1.0000000 0 0.0000000
## [32,] 1.0000000 0 0.0000000
## [33,] 1.0000000 0 0.0000000
## [34,] 0.3333333 0 0.6666667
## [35,] 1.0000000 0 0.0000000
## [36,] 1.0000000 0 0.0000000
## [37,] 1.0000000 0 0.0000000
## [38,] 1.0000000 0 0.0000000
## [39,] 1.0000000 0 0.0000000
## [40,] 1.0000000 0 0.0000000
## [41,] 1.0000000 0 0.0000000
## [42,] 1.0000000 0 0.0000000
## [43,] 1.0000000 0 0.0000000
## [44,] 1.0000000 0 0.0000000
## [45,] 1.0000000 0 0.0000000
## [46,] 1.0000000 0 0.0000000
## [47,] 1.0000000 0 0.0000000
## [48,] 1.0000000 0 0.0000000
## [49,] 1.0000000 0 0.0000000
## [50,] 1.0000000 0 0.0000000
## [51,] 0.0000000 0 1.0000000
## [52,] 0.6666667 0 0.3333333
## [53,] 0.7500000 0 0.2500000
## [54,] 0.0000000 0 1.0000000
## [55,] 0.0000000 0 1.0000000
## [56,] 0.0000000 0 1.0000000
## [57,] 0.0000000 0 1.0000000
## [58,] 0.0000000 0 1.0000000
## [59,] 0.5000000 0 0.5000000
## [60,] 0.0000000 0 1.0000000
## [61,] 0.0000000 0 1.0000000
## [62,] 0.0000000 0 1.0000000
## [63,] 0.0000000 0 1.0000000
## [64,] 0.6666667 0 0.3333333
## [65,] 0.0000000 0 1.0000000
## [66,] 0.0000000 0 1.0000000
## [67,] 0.0000000 0 1.0000000
```

```
## [68,] 0.0000000 0 1.0000000
## [69,] 0.0000000 0 1.0000000
## [70,] 0.0000000 0 1.0000000
## [71,] 0.0000000 0 1.0000000
## [72,] 0.3333333 0 0.6666667
## [73,] 0.0000000 0 1.0000000
## [74,] 0.0000000 0 1.0000000
## [75,] 0.3333333 0 0.6666667
```

2 Standardise data

When the scales of the variables are different, we have to scale the data before applying kNN. This is to make sure that the Euclidean distance is not dominated by the variables with large scales. To scale the dataset, we can use the `scale` function. Type `?scale` to see the help of this function.

```
?scale
```

Now suppose we change the measurement of `Sepal.Width` to milimeters and those of other variables to meters.

```
# Suppose we change Sepal.width to mm while others to m
iris_c=iris[,1:4]
iris_c[,2]=iris[,2]*10
iris_c[,-2]=iris[,c(1,3,4)]/10
```

Use `summary` to see the different scales of variables

```
summary(iris_c)
```

```
##   Sepal.Length     Sepal.Width     Petal.Length     Petal.Width
##  Min.   :0.4300   Min.   :20.00   Min.   :0.1000   Min.   :0.0100
##  1st Qu.:0.5100   1st Qu.:28.00   1st Qu.:0.1600   1st Qu.:0.0300
##  Median :0.5800   Median :30.00   Median :0.4350   Median :0.1300
##  Mean   :0.5843   Mean   :30.57   Mean   :0.3758   Mean   :0.1199
##  3rd Qu.:0.6400   3rd Qu.:33.00   3rd Qu.:0.5100   3rd Qu.:0.1800
##  Max.   :0.7900   Max.   :44.00   Max.   :0.6900   Max.   :0.2500
```

Now we apply 5NN:

```
# get indexes for training data
n=25 # the number of training data in each class
NN=dim(iris)[1]/3# the total number of observations in each class
set.seed(983)
index_s=sample(which(iris$Species=="setosa"),n)
index_c=sample(which(iris$Species=="versicolor"),n)
index_v=sample(which(iris$Species=="virginica"),n)
# get training and test set
train_rand_c = rbind(iris_c[index_s,], iris_c[index_c,], iris_c[index_v,])
test_rand_c = rbind(iris_c[-c(index_s,index_c,index_v),])
# get class factor for training data
train_label= factor(c(rep("s",n), rep("c",n), rep("v",n)))
# get class factor for test data
test_label_true=factor(c(rep("s",NN-n), rep("c",NN-n), rep("v",NN-n)))
# classification using knn, with k=5
kk=5 # number of nearest neighbours
set.seed(275)
```

```
knn_pred=knn(train=train_rand_c,test=test_rand_c,cl=train_label, k=kk, prob=FALSE)
knn_pred
```

```
##  [1] c s s s s s s s s s s s s s s c s s s s s c s s c v c c c c c c v c c
## [36] c v c c c v c c c c c c c c c c v v v v c v v c v s v c v v s v s c v c
## [71] v v v s c
## Levels: c s v
```

Exercise: Have a look at the predictions and compare them with the ground truth, what do you find?

Now we standardise the data first using `scale( )`:

```
iris_s=scale(iris_c)
```

Check the standard deviation and mean of the variables are all 1 and 0, respectively.

Then if we apply 5NN using the same training and test indexes:

```
# get training and test set
train_rand_s = rbind(iris_s[index_s,], iris_s[index_c,], iris_s[index_v,])
test_rand_s = rbind(iris_s[-c(index_s,index_c,index_v),])
# get class factor for training data
train_label= factor(c(rep("s",n), rep("c",n), rep("v",n)))
# get class factor for test data
test_label_true=factor(c(rep("s",NN-n), rep("c",NN-n), rep("v",NN-n)))
# classification using knn, with k=5
kk=5 # number of nearest neighbours
set.seed(275)
knn_pred=knn(train=train_rand_s,test=test_rand_s,cl=train_label, k=kk, prob=FALSE)
knn_pred
```

```
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s c c c c c c c c c c v
## [36] c c v c c c c c c c c c c c c v v v v c v v v v v v v v v v c v c v v
## [71] v v v v c
## Levels: c s v
```

Exercise: Compare the predictions with those without scaling. What do you find?

3 How to draw classification boundary for kNN

Let's see how we can draw the classification boundary when applying kNN to classify the `Default` data in the `ISLR` library.

First of all, have a look at the `Default` data.

```
library(ISLR)
str(Default)
```

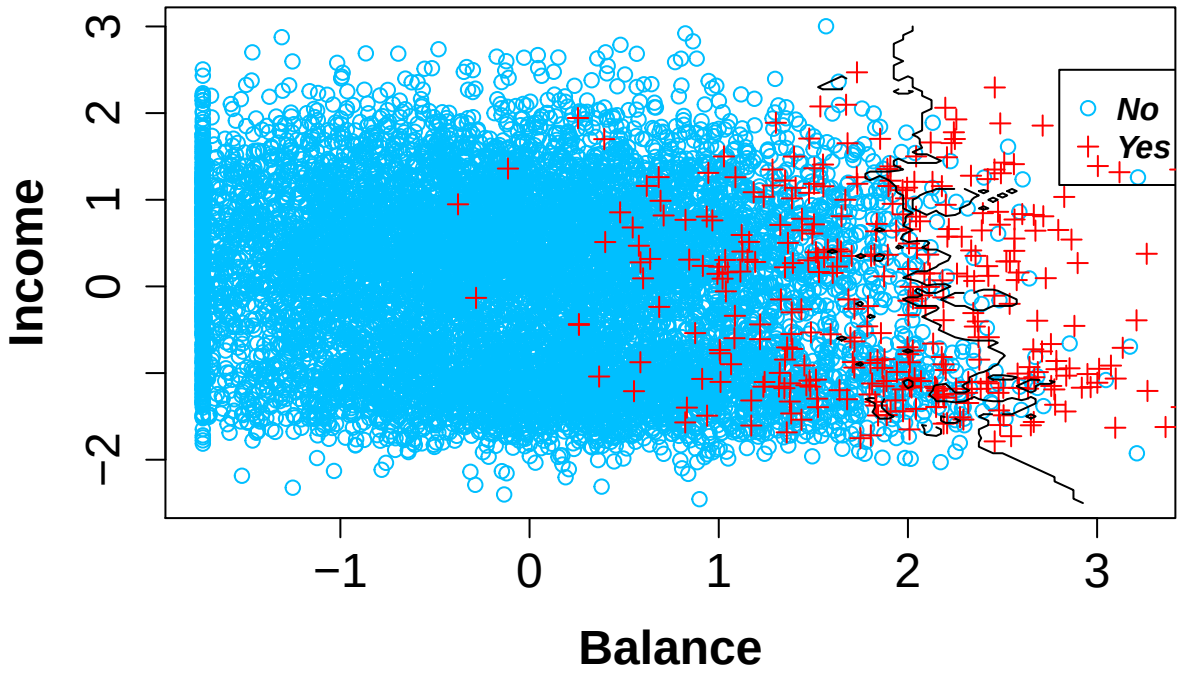
```
## 'data.frame':    10000 obs. of  4 variables:
##  $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
##  $ balance: num  730 817 1074 529 786 ...
##  $ income : num  44362 12106 31767 35704 38463 ...
```

```
summary(Default)
```

```
##  default    student       balance           income
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772
```

```
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
##                        Median : 823.6   Median :34553
##                        Mean   : 835.4   Mean   :33517
##                        3rd Qu.:1166.3   3rd Qu.:43808
##                        Max.   :2654.3   Max.   :73554
```

```
library(ISLR)
library(class)
Default[,c(3,4)]=scale(Default[,c(3,4)])
no=Default[Default$default=="No",]
yes=Default[Default$default=="Yes",]
##### scatter plot of the training data
plot(no$balance,no$income,pch=1,col="deepskyblue1",
     xlab="Balance",ylab="Income",cex.lab=1.5,cex.axis=1.5,font.lab=2)
points(yes$balance,yes$income,pch=3,col="red")
legend(2.8,2.5, legend=c("No", "Yes"),
       col=c("deepskyblue1", "red"),pch=c(1,3),cex=1,text.font=4)
##### train.feature, train.label
train.feature=Default[,c(3,4)]
train.label=Default$default
##### generate test data
new.x1=seq(-2,3.5,by=0.05)
new.x2=seq(-2.5,3,by=0.05)
new=expand.grid(new.x1,new.x2)
##### get predictions from 5NN
set.seed(12)
pred = knn(train.feature, new, train.label, k=5, prob=TRUE)
##### get P(Y=1|new)
prob = attr(pred, "prob")
prob = ifelse(pred=="Yes", prob, 1-prob)
##### get contour P(Y=1|X)=0.5
contour(new.x1, new.x2, matrix(prob, nrow = length(new.x1),ncol = length(new.x2)),
        levels=0.5,label="", axes=FALSE, add=TRUE)
```



4 Real data exercises

4.1 German Credit Data

These data have two classes for the credit worthiness: good or bad. There are predictors related to attributes, such as: checking account status, duration, credit history, purpose of the loan, amount of the loan, savings accounts or bonds, employment duration, Installment rate in percentage of disposable income, personal information, other debtors/guarantors, residence duration, property, age, other installment plans, housing, number of existing credits, job information, Number of people being liable to provide maintenance for, telephone, and foreign worker status.

This dataset is also included in the `caret` package.

```
library(caret)
#load german credit data from caret package
data(GermanCredit)
# classify two status: good or bad
## Show the first 10 columns
str(GermanCredit[, 1:10])
```

```
## 'data.frame':    1000 obs. of  10 variables:
##  $ Duration                : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                  : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration       : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                     : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits   : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone               : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker           : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                   : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
```

```
## Delete two variables where all values are the same for both classes
GermanCredit[,c("Purpose.Vacation","Personal.Female.Single")] <- list(NULL)
```

Now we divide the dataset to training and test sets.

```
# create training and test sets
set.seed(12)
trainIndex = createDataPartition(GermanCredit$Class, p = 0.7, list = FALSE, times = 1)
train.feature=GermanCredit[trainIndex,-10] # training features
train.label=GermanCredit$Class[trainIndex] # training labels
test.feature=GermanCredit[-trainIndex,-10] # test features
test.label=GermanCredit$Class[-trainIndex] # test labels
```

Exercise What's next?

4.2 Fatty Acid Composition Data

A set of data where seven fatty acid compositions were used to classify commercial oils as either pumpkin (labeled A), sunflower (B), peanut (C), olive (D), soybean (E), rapeseed (F) and corn (G). Our aim is to classify an unknown commercial oil to one of the seven categories based on its seven fatty acid compositions.

This dataset is in the `caret` package. We can load the data by using the `data( )` command.

```r
library(caret)
#load oil data from caret package
data(oil)
```

4.3 Pima Indians Diabetes Data

In this dataset, we aim to predict the onset of diabetes in female Pima Indians from medical record data. This dataset can be loaded from the `mlbench` package.

```r
library(mlbench)
#load Pima Indians Diabetes data from mlbench package
data(PimaIndiansDiabetes)
```

5 Shiny from R studio

Shiny is an **R** package that makes it easy to build interactive web apps straight from **R**. You can host standalone apps on a webpage or embed them in **R** Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions. There are good tutorials on the official website https://shiny.rstudio.com/tutorial/.

The first lesson is a good tutorial to read, to help you understand the structure of a Shiny app https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/.

The following app can visualise the k nearest neighbours of the green point.

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(
  # App title ----
  titlePanel("Find k nearest neighbours"),
  # Sidebar layout with input and output definitions ----
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
      # Input: Slider for the number of bins ----
      sliderInput(inputId = "k",
                  label = "Number of nearest neighbours:",
                  min = 1,
                  max = 11,
                  value = 30,
                  step=2)
    ),
    # Main panel for displaying outputs ----
    mainPanel(
      # Output: scatter plot ----
      plotOutput(outputId = "sccaterPlot")
    )
  )
)
# Define server logic required to draw a histogram ----
server <- function(input, output) {
  output$sccaterPlot <- renderPlot({
```

```
    ##### load training data
    set.seed(10)
    library(MASS)
    class1=mvrnorm(n = 30, mu=c(1,1), Sigma=matrix(c(1,0,0,1),2,2))
    class2=mvrnorm(n = 30, mu=c(2,2), Sigma=matrix(c(1,0,0,1),2,2))
    train.feature=rbind(class1,class2)
    train.label=c(rep(1,30),rep(2,30))
    ##### scatter plot of the training data
    plot(train.feature,col=ifelse(train.label==1, "blue", "red"),
         pch=ifelse(train.label==1, 16, 17),xlab="x1",ylab="x2",cex=1.5)
    points(2.1,1.5,pch=15,col="green",cex=1.5) ## test point
    ##### find k nearest neighbours
    library("FNN")
    test.feature=t(c(2.1,1.5))
    k=input$k
    knearest=get.knnx(train.feature,test.feature, k)
    neighbour=train.feature[knearest$nn.index,]
    if(k==1){
      points(t(neighbour),pch=0,col="black",cex=1.8)}else{
        points((neighbour),pch=0,col="black",cex=1.8)
        }
  })
}
shinyApp(ui = ui, server = server)
```

We can also visualise the classificaiton boundary with different k.

```
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(
  # App title ----
  titlePanel("Classification boundary of k nearest neighbours"),
  # Sidebar layout with input and output definitions ----
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
      # Input: Slider for the number of bins ----
      sliderInput(inputId = "k",
                  label = "Number of nearest neighbours:",
                  min = 1,
                  max = 51,
                  value = 30,
                  step=5)
    ),
    # Main panel for displaying outputs ----
    mainPanel(
      # Output: scatter plot ----
      plotOutput(outputId = "sccaterPlot")
    )
  )
)
# Define server logic required to draw a histogram ----
server <- function(input, output) {
```

```
  output$sccaterPlot <- renderPlot({
    library(class)
    ##### load training data
    simulate=read.table("simulate.txt",header=TRUE)
    train.feature=simulate[,-3]
    train.label=simulate[,3]
    ##### scatter plot of the training data
    plot(train.feature,col=ifelse(train.label==1, "blue", "red"),
         pch=ifelse(train.label==1, 16, 17))
    ##### generate test data
    new.x1=seq(-3,4.5,by=0.1)
    new.x2=seq(-2,4,by=0.1)
    new=expand.grid(new.x1,new.x2)
    ##### get predictions from 15-NN
    k=input$k
    pred = knn(train.feature, new, train.label,k, prob=TRUE)
    ##### get P(Y=1|new)
    prob = attr(pred, "prob")
    prob = ifelse(pred=="1", prob, 1-prob)
    ##### get contour P(Y=1|X)=0.5
    contour(new.x1, new.x2, matrix(prob, nrow = length(new.x1),ncol = length(new.x2)),
            levels=0.5,label="", axes=FALSE, add=TRUE)
  })

}
shinyApp(ui = ui, server = server)
```