

HW 7:

Problem 45.

a) by default, we use 10-fold cross-validation here. The range of knn classifier goes from 1 to 21 here.

For N = 400 case:

Here is R-code output:

Parameter tuning of 'knn.wrapper':

- sampling method: 10-fold cross validation

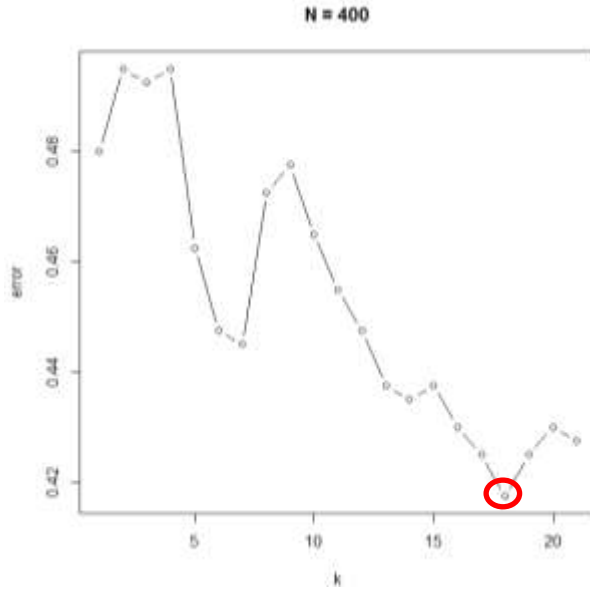
- best parameters:

k
18

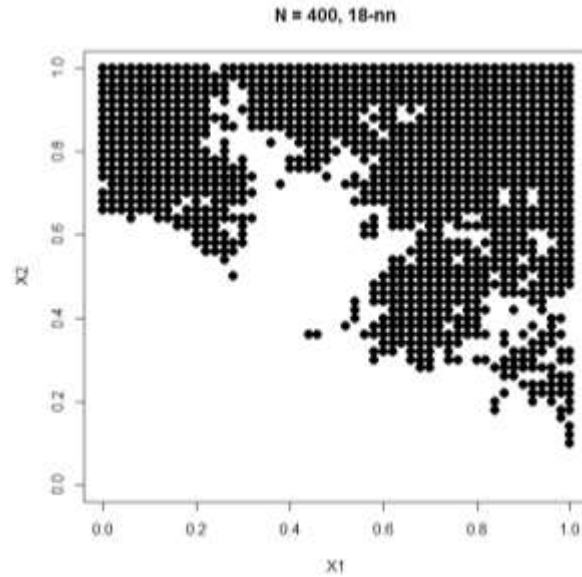
- best performance: 0.4175 : this is the error rate on training for 18-nn

- Detailed performance results:

	k	error	dispersion
1	1	0.4800	0.06540472
2	2	0.4950	0.07710022
3	3	0.4925	0.08502451
4	4	0.4950	0.09339284
5	5	0.4625	0.07288690
6	6	0.4475	0.07495369
7	7	0.4450	0.08563488
8	8	0.4725	0.07857233
9	9	0.4775	0.07212066
10	10	0.4650	0.09442810
11	11	0.4550	0.06749486
12	12	0.4475	0.05945353
13	13	0.4375	0.06795628
14	14	0.4350	0.07564537
15	15	0.4375	0.07927624
16	16	0.4300	0.07434903
17	17	0.4250	0.09354143
18	18	0.4175	0.09431773
19	19	0.4250	0.08164966
20	20	0.4300	0.08482007
21	21	0.4275	0.06609127



Problem 45 a) (left) N = 400, error rate vs k



(right) performance for 18-nn classifier

N = 4000 case

Parameter tuning of 'knn.wrapper':

- sampling method: 10-fold cross validation

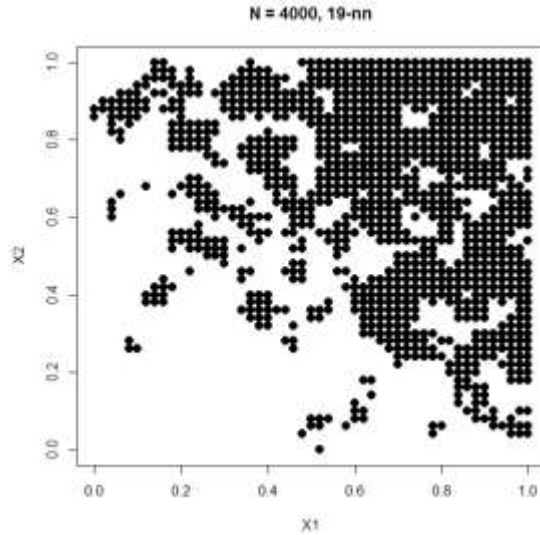
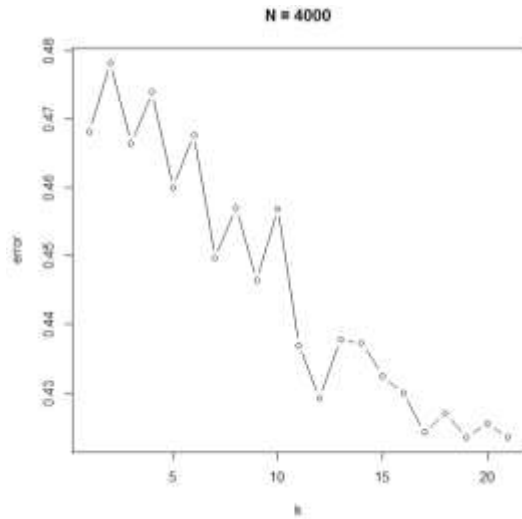
- best parameters:

k
19

- best performance: 0.4235151

- Detailed performance results:

	k	error	dispersion
1	1	0.4682212	0.02253985
2	2	0.4782365	0.03093815
3	3	0.4664650	0.03301308
4	4	0.4739852	0.03412457
5	5	0.4599736	0.03545803
6	6	0.4677338	0.02969492
7	7	0.4497405	0.03139836
8	8	0.4569993	0.02543870
9	9	0.4464797	0.02895795
10	10	0.4569327	0.03305593
11	11	0.4369600	0.03982293
12	12	0.4291999	0.03763466
13	13	0.4377415	0.03264795
14	14	0.4372490	0.03182687
15	15	0.4324977	0.03055444
16	16	0.4299763	0.03085898
17	17	0.4242318	0.03250511
18	18	0.4269932	0.03063469
19	19	0.4235151	0.03351471
20	20	0.4255077	0.02641613
21	21	0.4235265	0.02740486



Problem 45 a) (left) N = 4000, error rate vs k

(right) performance for 19-nn classifier

N = 40000 case:

Parameter tuning of 'knn.wrapper':

- sampling method: 10-fold cross validation

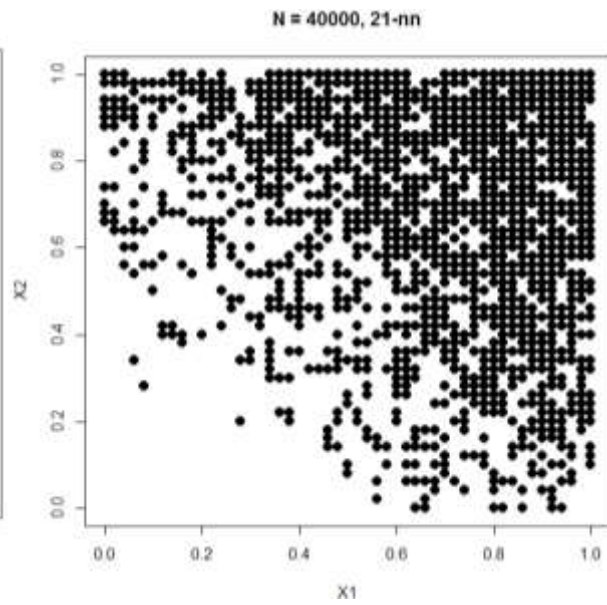
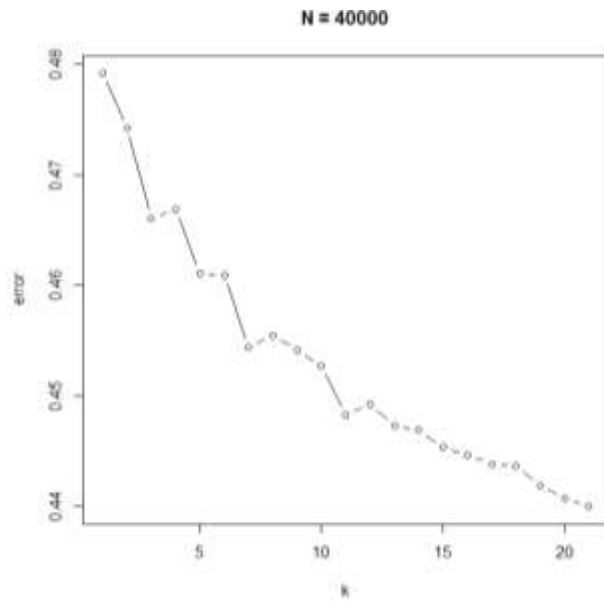
- best parameters:

k
21

- best performance: 0.4399952

- Detailed performance results:

k	error	dispersion
1	0.4792544	0.004300586
2	0.4742554	0.007722119
3	0.4660932	0.004998922
4	0.4668402	0.006495078
5	0.4610545	0.008436912
6	0.4609399	0.005941747
7	0.4544198	0.006458519
8	0.4555058	0.004496446
9	0.4541618	0.004192825
10	0.4527109	0.005711383
11	0.4482829	0.004081356
12	0.4492207	0.006519132
13	0.4473096	0.005975858
14	0.4469034	0.007396434
15	0.4453103	0.008082419
16	0.4446289	0.005514593
17	0.4438597	0.006902594
18	0.4436067	0.006316764
19	0.4418846	0.007076715
20	0.4407180	0.009567875
21	0.4399952	0.010289518



Problem 45 a) (left) N = 40000, error rate vs k

(right) performance for 21-nn classifier

b) First generate a test set of size =100,000. We can use the same procedure as creating a training set of size N. Below is the code:

```
N_test = 100000
```

```
test <- matrix(0,nrow=N_test,ncol=3)
```

```
for(i in 1:N_test){
```

```
  test[i,]<-observ(2)
```

```
}
```

The outcome for **N = 400, 18nn classifier** is :

```
> k=18  
> knn.pred=knn(train[,2:3],test[,2:3],train[,1],k=k)  
> table(knn.pred,test[,1])
```

```
knn.pred    0    1  
      0 27817 21414  
      1 22050 28719
```

It means there are 22050 points, which is supposed to be 0, but 18nn classifier classify it as 1; there are 21414 points, which is supposed to be 1, but 18nn classifier classify it as 0. So the error rate on this test set is

```
> (22050+21414)/100000  
[1] 0.43464
```

The outcome for **N = 4000, 19nn** classifier is:

```
> k=19
> knn.pred<-knn(train[,2:3],test[,2:3], train[,1],k=k)
> table(knn.pred,test[,1])
```

```
knn.pred  0  1
0 27786 21885
1 22096 28233
```

So the error rate on this test set is

```
> (22096+21885)/100000
[1] 0.43981
```

The outcome for **N = 40000, 21nn** classifier is:

```
> k=21
> knn.pred<-knn(train[,2:3],test[,2:3], train[,1],k=k)
> table(knn.pred,test[,1])
```

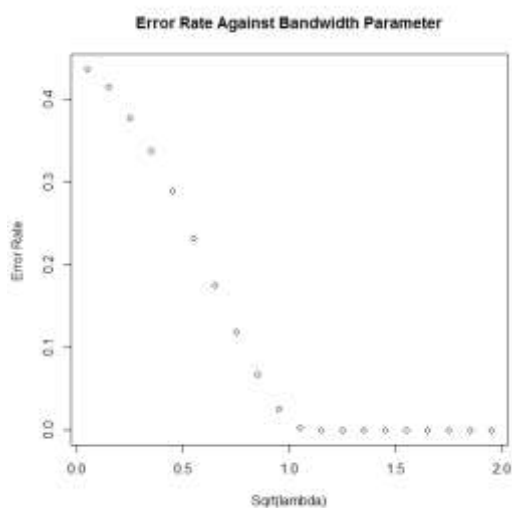
```
knn.pred  0  1
0 26148 20578
1 23839 29435
```

The error rate on this test set is :

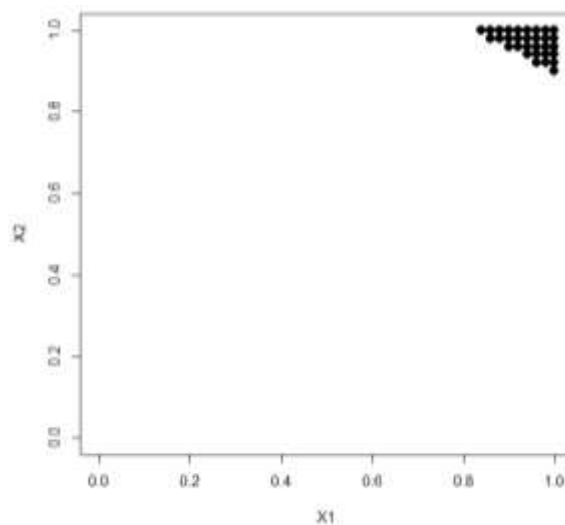
```
> (23839+20578)/100000
[1] 0.44417
```

All of those error rates are higher than the value in problem 43 a).

Problem 46.



Error rate against Bandwidth



classifier plot when lambda = 1

Here is the code:

```
rm(list = ls())
set.seed(1)
#Enter the prior and define the two pdf's
#We'll use here f0 the uniform density and f1
the density that is the sum of the two
#input coordinates
g0<-.5
g1<-1-g0
density0 <- function(x1,x2) {
  y<-1
  return(y)
}
density1 <- function(x1,x2) {
  y<-(x1+x2)
  return(y)
}
## generating a training set with size = 400
rejsamp = function(A){
  while(1){
    # three independent uniforms
    u = runif(3)
    # Accept or reject candidate value; if rejected
    try again
    if(A*u[3]< density1(u[1],u[2]))
      return(u[1:2])
  }
}
#Here is code for making a training sample
sample of N from g0*density0 + g1*density1
N=400
train<-matrix(0,nrow=N,ncol=3)
```

```
observ = function (A) {
  u = runif(3)
  if(u[1]<g0)
    return(c(0,u[2:3]))
  else
    return(c(1,rejsamp(2)))
}
for(i in 1:N){
  train[i,]<-observ(2)
}
## update g0 and f0, f1
## get the number of N0 and N1 respectively
N0 = sum(train[,1]==0)
N1 = sum(train[,1]==1)
cat("there are ", N0, " 0s in this training set.\n")
cat("there are ", N1, " 1s in this training set.\n")
g0_hat = N0/N
g1_hat = N1/N
cat("g0_hat is updated to ", g0_hat, "\n")
cat("g1_hat is updated to ", g1_hat, "\n")
## get the training set with y = 0 , y = 1
respectively
idx_0 = which(train[,1]==0)
train_0 = train[idx_0,]
idx_1 = which(train[,1]==1)
train_1 = train[idx_1,]
## update density
density0_hat = function(x1,x2,lambda) {
  aux = rep(0,nrow(train_0))
  for(i in 1:nrow(train_0)){
```

```

    tmp = exp(-0.5*(1/lambda)*((x1-
train_0[i,2])^2+(x2-train_0[i,3])^2))
    aux[i] = tmp/(2*pi*lambda)
  }
y = sum(aux)/N0
return(y)
}

density1_hat = function(x1,x2,lambda) {
  aux = rep(0,nrow(train_1))
  for(i in 1:nrow(train_1)){
    tmp = exp(-0.5*(1/lambda)*((x1-
train_1[i,2])^2+(x2-train_1[i,3])^2))
    aux[i] = tmp/(2*pi*lambda)
  }
y = sum(aux)/N1
return(y)
}

#Here is the new classifier
myclassifier <- function(x1,x2,lambda) {
  if (g1_hat*density1_hat(x1,x2,lambda) >
g0_hat*density0_hat(x1,x2,lambda)) {result<-1}
  else {result<-0}
  return(result)
}

#Specify the number of values of both x1 and x2
to use in a grid for plotting
h<-51

#Make vectors to hold both coordinates for the
grid
X1<-rep(seq(1:h)-1,h)/(h-1)
X2<-rep(0:(h-1),each=h)/(h-1)
## lambda change from 0.05^2 to 2^2

```

```

## err_vec is a vector, recording the error for
each lambda case
a = seq(0.05,2,by = 0.1)
err_vec = rep(0,length(a))
for( s in 1:length(a)){
  lambda = a[s]*a[s]
  #Make a vector of values of the new classifier
for points on the grid
  Class<-rep(0,h*h)
  for (i in 1:(h*h)) {
    Class[i] = myclassifier(X1[i],X2[i],lambda)
  }
  err = sum(Class)/(h*h)
  err_vec[s]= err
}
## Plot the error rate against sqrt(lambda)
x11()
plot(a,err_vec,main = "Error Rate Against
Bandwidth Parameter",xlab =
"sqrt(lambda)",ylab = "Error Rate ")
## choose lambda = 1
lambda = 1
Class<-rep(0,h*h)
for (i in 1:(h*h)) {
  Class[i] = myclassifier(X1[i],X2[i],lambda)
}

#Make a plot of the values of the Bayes
classifier for points on the grid
x11()
plot(X1,X2,pch=20,col= Class,cex = 2,lwd = 1)

```

Problem 47.

a). (need to install a R package called “tree” first)

- fit classification trees using tree() function with default parameter settings:

```
> treeclass1<-tree(y ~ x1+x2,training)
> summary(treeclass1)
```

Classification tree:

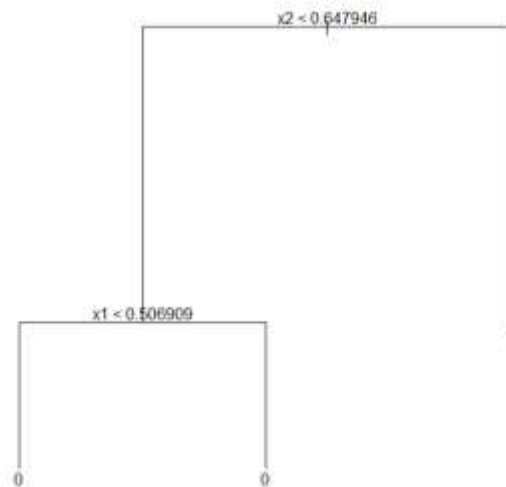
```
tree(formula = y ~ x1 + x2, data = training)
```

```
Number of terminal nodes: 3
```

```
Residual mean deviance: 1.31 = 520 / 397
```

```
Misclassification error rate: 0.38 = 152 / 400 this is the error rate on training set
```

Below is the corresponding plot :



Classification tree with default settings

(Note: in tree() function in “tree” package, the default setting is: mincut = 5, minsize = 10, mindev = 0.01.

mincut means the minimum number of observations to include in either child node , default value is 5

minsize means the smallest allowed node size, default value is 10

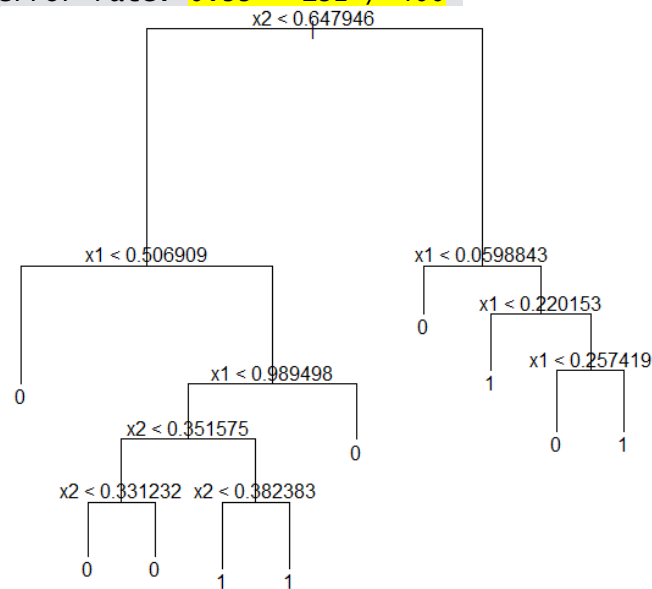
mindev means the within-node deviance must be at least this times that of the root node for the node to be split default value is 0.01)

- If use the setting in the R-code:

```
> treeclass<-tree(y ~ x1+x2,training,control=tree.control(nobs=N,mincut=1,minsize=2,mindev=.008))
> summary(treeclass)
```

Classification tree:

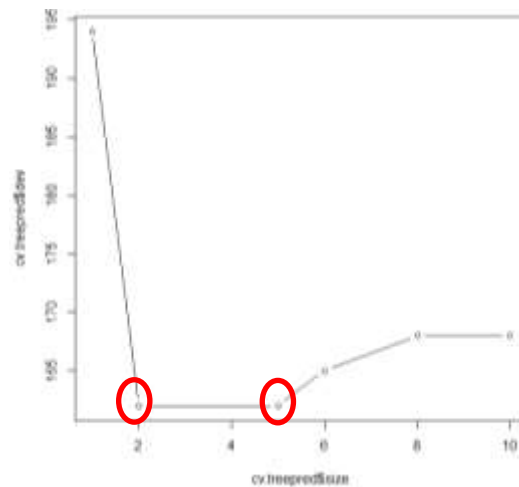

```
tree(formula = y ~ x1 + x2, data = training, control = tree.control(nobs = N,
  mincut = 1, minsize = 2, mindev = 0.008))
Number of terminal nodes: 10
Residual mean deviance: 1.234 = 481.3 / 390
Misclassification error rate: 0.33 = 132 / 400
```



Classification tree with settings in R code

This second tree with less error rate on training set, also with more number of nodes.

Use 10-fold cross validation to find a good sub-tree of the full tree with settings , the following plot suggests that the optimal tree size would be 2 or 5.



The error rate on training set for node = 2

`> summary(prune.best)`

```
Classification tree:
snip.tree(tree = treeclass, nodes = c(3L, 2L))
Variables actually used in tree construction:
```

```
[1] "x2"
Number of terminal nodes: 2
Residual mean deviance: 1.335 = 531.2 / 398
Misclassification error rate: 0.38 = 152 / 400
```

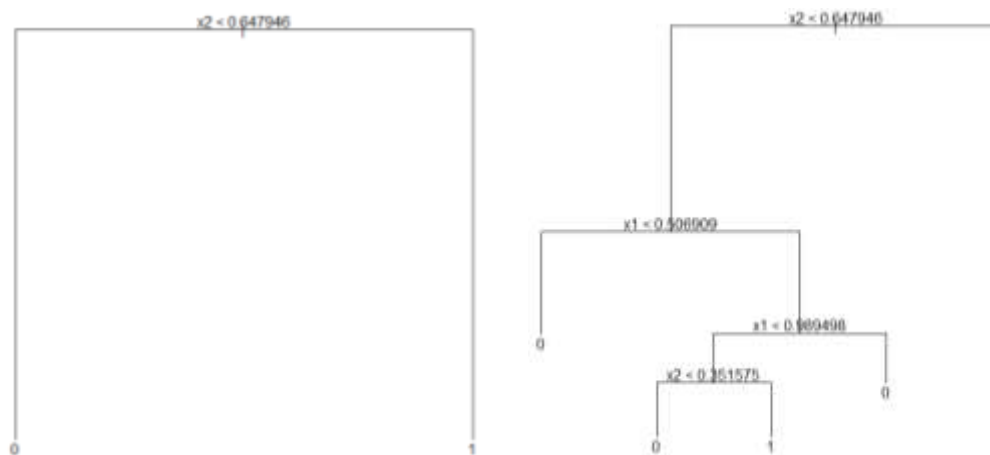
The error rate on training set for node = 5:

```
> summary(prune.best2)
```

```
Classification tree:
snip.tree(tree = treeclass, nodes = c(20L, 21L, 3L))
Number of terminal nodes: 5
Residual mean deviance: 1.288 = 508.7 / 395
Misclassification error rate: 0.3475 = 139 / 400
```

Higher complexity with lower error rate on training set here.

Here is the tree plot for node = 2 , node = 5, respectively.



b). training size N = 400

```
> rf.example
```

```
Call:
randomForest(formula = y ~ x1 + x2, data = training, type = "classification",
             ntree = 500, mtry = 2)
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 2
```

OOB estimate of error rate: 42%

```
Confusion matrix:
  0  1 class.error
0 118 90  0.4326923
1  78 114  0.4062500
```

(**Note** : class.error in the confusion matrix is obtained as followings:

For the first row, the predicted class is 0s (there are $118+90 = 208$ 0s in the predicted class. Among those 208 0s, there actually 118 0s, , 90 1s in the original class. So the class.error = $90/(118+90) = 0.4326923$)
 So is the class.error in the second row: $78/(78+114) = 0.40625$)

When the sample size N = 4000,

Call:

```
randomForest(formula = y ~ x1 + x2, data = training, type = "classification",
  ntree = 500, mtry = 2)
  Type of random forest: classification
  Number of trees: 500
  No. of variables tried at each split: 2
```

OOB estimate of error rate: 44.47%

Confusion matrix:

	0	1	class.error
0	1097	914	0.4545002
1	865	1124	0.4348919

Error rate on training set increase as increasing the size of training set.

c) use sample size N = 400

alpha = 1:

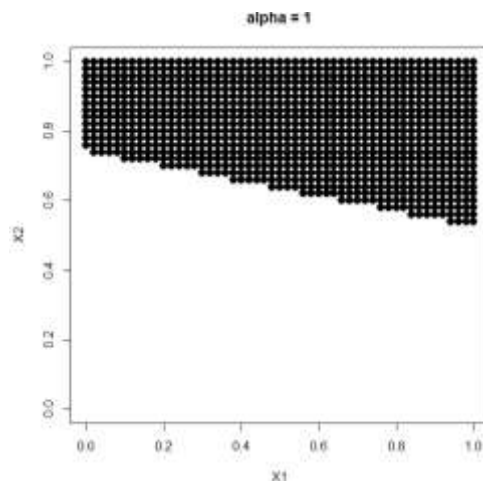
calculate error rate on training set:

```
> lrfit2.train<-predict(lrfit2,as.matrix(training[,2:3]), s = "lambda.1se",type="class")
> table(lrfit2.train,training[,1])
```

lrfit2.train	0	1
0	144	90
1	64	102

Again, error rate on training set would be

```
> (64+90)/(400)
[1] 0.385
```



Classification plot for alpha 1

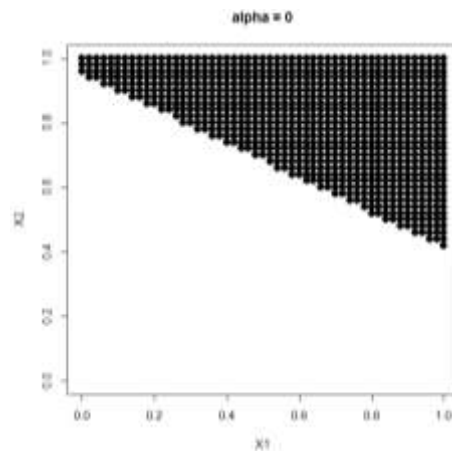
alpha = 0

```
> lrfit3.train<-predict(lrfit3,as.matrix(training[,2:3]), s = "lambda.1se",type="class")  
> table(lrfit3.train,training[,1])
```

```
lrfit3.train  0  1  
0 156 110  
1  52  82
```

Error rate is

```
> (52+110)/400  
[1] 0.405
```



Classification plot for alpha 0

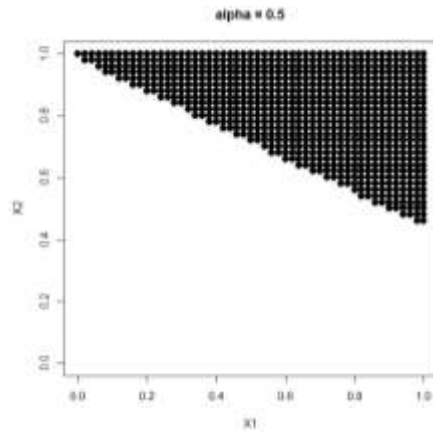
Alpha = 0.5

```
> lrfit4.train<-predict(lrfit4,as.matrix(training[,2:3]), s = "lambda.1se",type="class")  
> table(lrfit4.train,training[,1])
```

```
lrfit4.train  0  1  
0 141  91  
1  67 101
```

Error rate is :

```
> (67+91)/400  
[1] 0.395
```



Classification plot for alpha 0.5

d) for the original logistic regression classification, the error rate on training set is :

```
lrfit<-glm(y~x1+x2,training,family="binomial")
```

```
lrfit
```

```
summary(lrfit)
```

```
## get the probability of logistic regression
```

```
theProbs_train = predict(lrfit,training[,2:3], type = "response")
```

```
#calculate error rate
```

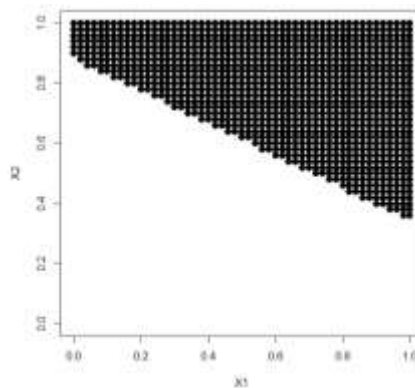
```
table(theProbs_train>.5, training[,1])
```

```
> table(theProbs_train>.5, training[,1])
```

```
      0   1
FALSE 137 84
TRUE   71 108
```

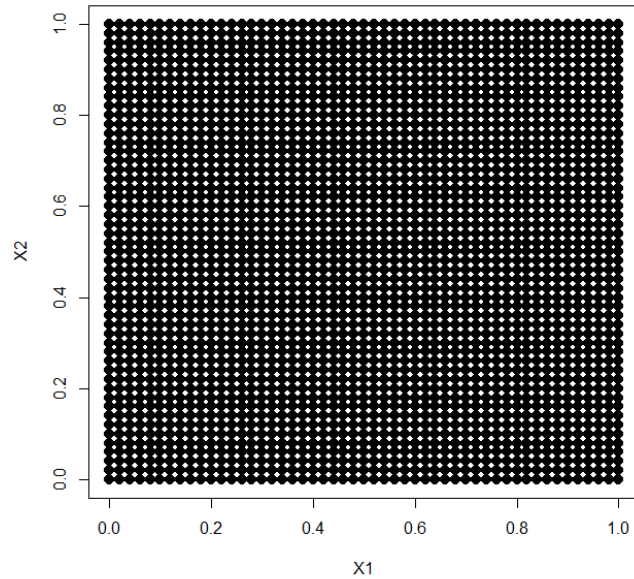
```
> (71+84)/400
```

```
[1] 0.3875
```

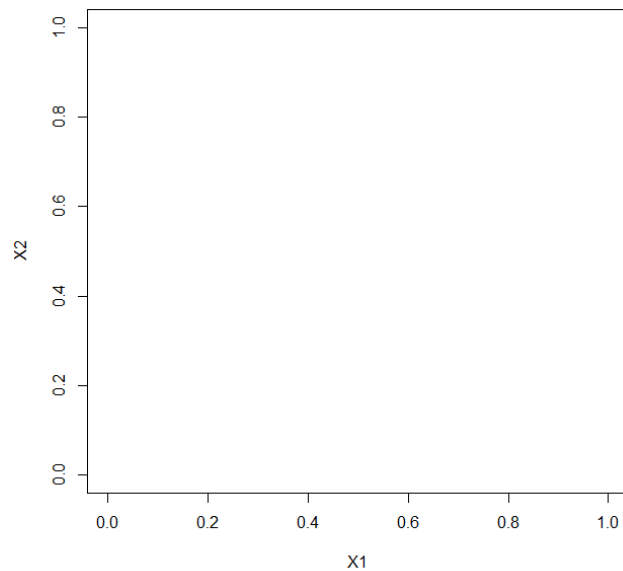


Logistic regression classification

From the discussion of case-control studies on P34-P35 on course outline, probably we would starting a sandom sample of N_0 cases with $y = 0$, and a random sample of N_1 controls with $y = 1$. Usually, N_1 is on the order of 5/6 times N_0 . Let $N_0 = 4$, $N_1 = 24$. If $g(0) = 0.01$, sample size of $N = 400$, logistic regression will yield the classification plot as below:



But if apply case-control ideal, the modify logistic regression will yield the classification plot as below



which is more reasonable since the $g(0)$ is very small.

Ps: the corresponding R code for this part is

```
## get a sample with N0 0s and N1 1s
N0 = 4
N1 = 24
train_0 =
training[sample(which(training[,1]==0),N0),]
train_1 =
training[sample(which(training[,1]==1),N1),]
train_new = rbind(train_0,train_1)
## logist regression on this set
lrfit_new =
glm(y~x1+x2,train_new,family="binomial")
summary(lrfit_new)
beta_0_cc = coef(lrfit)[1] + log(N0/N1)-
log(0.01/(1-0.01))
beta_1_cc_1 = coef(lrfit)[2]
beta_2_cc_2 = coef(lrfit)[3]
## case_control classifier
case_control_classifier = function(x1,x2){
  aux =
exp(beta_0_cc+x1*beta_1_cc_1+x2+beta_2_cc
_2)
```

```
p0 = aux/(1+aux)
p1 = 1/(1+aux)
if (p0<0.5 ) {result=1}
else {result=0}
return(result)
}
pred_train = rep(0,nrow(train))
for(i in 1:nrow(train)){
  tr_x1 = train[i,2]
  tr_x2 = train[i,3]
  pred_train[i] =
case_control_classifier(tr_x1,tr_x2)
}
Class<-rep(0,h*h)
for (i in 1:(h*h)) {
  Class[i]<-case_control_classifier(X1[i],X2[i])
}
x11()
plot(X1,X2,pch=20,col=Class,cex=2,lwd=1)
```

e).

```
> summary(svmfit)
```

```
> summary(tune.out)
```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:

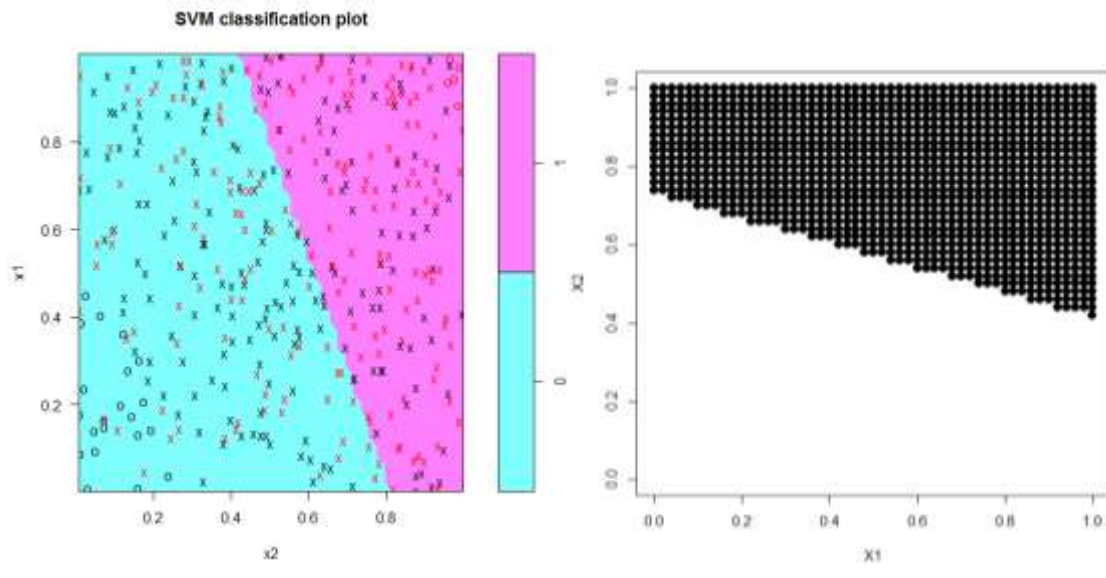
```
cost  
0.1
```

- best performance: 0.395

- Detailed performance results:

	cost	error	dispersion
1	1e-03	0.480	0.07888106
2	1e-02	0.420	0.05749396
3	1e-01	0.395	0.06540472
4	1e+00	0.395	0.07051399
5	5e+00	0.395	0.07051399
6	1e+01	0.395	0.07051399
7	1e+02	0.395	0.07051399
8	1e+03	0.395	0.07051399

The good svm is achieved when cost = 0.1, with error rate 0.395 on training set of size N = 400. This is associated classification plot



f) ensemble average method:

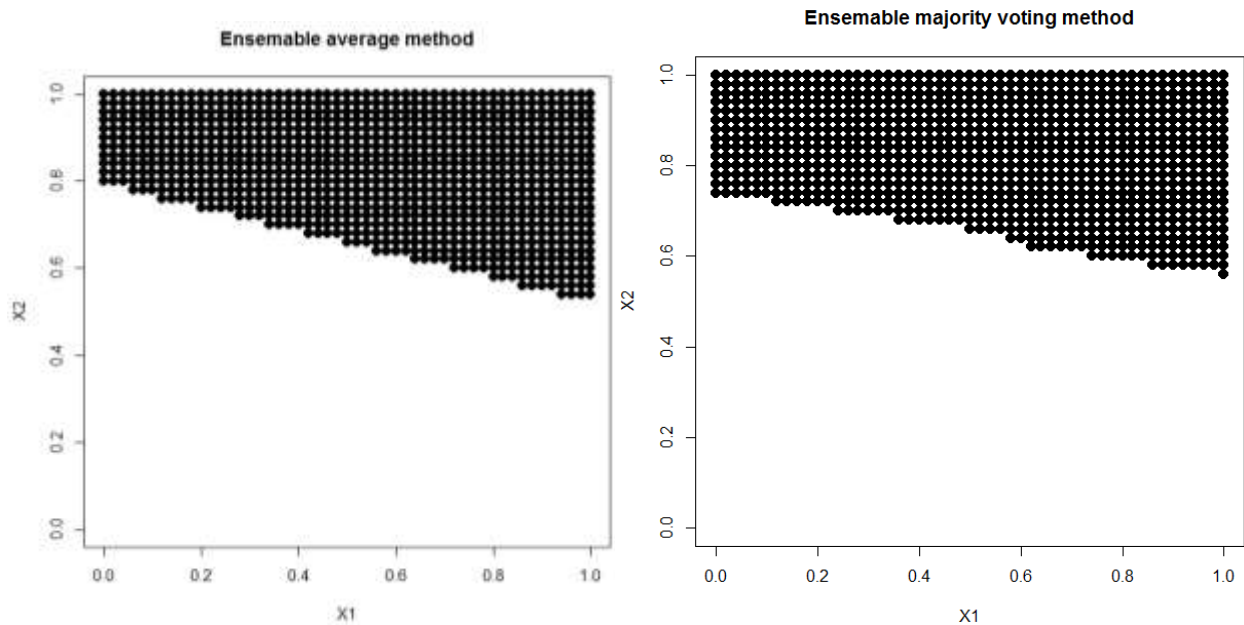
$$p_{new}(y = 0|x) = \frac{1}{3} * [p_{\{\alpha=0\}}(y = 0|x) + p_{\{\alpha=0.5\}}(y = 0|x) + p_{\{\alpha=1\}}(y = 0|x)]$$

ensemble majority voting method:

$$p_{new}(x) = \frac{1}{3} * [p_{\{\alpha=0\}}(x) + p_{\{\alpha=0.5\}}(x) + p_{\{\alpha=1\}}(x)]$$

$$y_{new}(x) = I(p_{new}(x) > 0.5)$$

first plot the classification ensemble average method and ensemble majority voting method on 51*51 grid points,



Next, compare these two method on a test set of size 100000,

For ensemble average method, the error rate is

```
> table(ensemble_average_test,test_set[,1])
ensemble_average_test  0    1
0 32708 26371
1 17163 23758
> (17163+26371)/100000
[1] 0.43534
```

For ensemble majority voting method, the error rate is

```
> table(ensemble_voting_test,test_set[,1])
ensemble_voting_test  0    1
0 32316 26254
1 17555 23875
> (17555+26254)/100000
[1] 0.43809
```

Here is the R code for this part:

```
lrfit2<-
cv.glmnet(as.matrix(training[,2:3]),training[,1],f
amily="binomial",alpha=1,standardize=TRUE)
#lrfit2
summary(lrfit2)
## get the probability: note this is the  $p(y=1|x)$ 
lrfit2.preds<-predict(lrfit2,as.matrix(newdata), s
="lambda.1se",type="response")

#Setting alpha to 0 gives the penalty involving
squares of the coefficients)
lrfit3<-
cv.glmnet(as.matrix(training[,2:3]),training[,1],f
amily="binomial",alpha=0,standardize=TRUE)
lrfit3.preds<-predict(lrfit3,as.matrix(newdata), s
="lambda.1se",type="response")

#Setting alpha to 0.5 gives the penalty involving
squares of the coefficients)
lrfit4<-
cv.glmnet(as.matrix(training[,2:3]),training[,1],f
amily="binomial",alpha=0.5,standardize=TRUE)

lrfit4.preds<-predict(lrfit4,as.matrix(newdata), s
="lambda.1se",type="response")

Ensemble_average_classify = rep(0,h*h)

for (i in 1:(h*h)) {
  p = c(lrfit2.preds[i],lrfit3.preds[i],lrfit4.preds[i])
```

```
## get the  $p_{\hat{y}=0|x}$ 
p0 = 1-p
p0_new = (1/3)*(sum(p0))

if(p0_new<0.5){
  Ensemble_average_classify[i] = 1
}else{
  Ensemble_average_classify[i] = 0
}
}

x11()
plot(X1,X2,pch=20,main = "Ensemble average
method",col=Ensemble_average_classify,cex=2,
lwd=1)

Ensemble_voting_classify = rep(0,h*h)

for (i in 1:(h*h)) {
  p = c(lrfit2.preds[i],lrfit3.preds[i],lrfit4.preds[i])
  ## get class
  y_pre = as.numeric(p>0.5)
  p0_new = (1/3)*sum(y_pre)
  if(p0_new>0.5){
    Ensemble_voting_classify[i] = 1
  }else{
    Ensemble_voting_classify[i] = 0
  }
}
```

```

}

x11()
plot(X1,X2,pch=20,main = "Ensemble majority
voting
method",col=Ensemble_voting_classify,cex=2,l
wd=1)

## condier error rate on a test set of size n =
100000
## generate a test set first
N_test = 100000

test_set = matrix(0,nrow=N_test,ncol=3)

for(i in 1:N_test){
  test_set[i,]= observ(2)
}

## compute predictor probability on this test
set using lrfit2 model
lrfit2.preds_test =
predict(lrfit2,as.matrix(test_set[,2:3]), s =
"lambda.1se",type="response")
lrfit3.preds_test =
predict(lrfit3,as.matrix(test_set[,2:3]), s =
"lambda.1se",type="response")
lrfit4.preds_test =
predict(lrfit4,as.matrix(test_set[,2:3]), s =
"lambda.1se",type="response")

```

```

ensemble_average_test = rep(0,N_test)

for (i in 1:N_test) {
  p =
c(lrfit2.preds_test[i],lrfit3.preds_test[i],lrfit4.pre
ds_test[i])
  ## get the p_hat(y=0|x)
  p0 = 1-p
  p0_new = (1/3)*(sum(p0))

  if(p0_new<0.5){
    ensemble_average_test[i] = 1
  }else{
    ensemble_average_test[i] = 0
  }
}

## get the performance of ensemble average
method on this test set
table(ensemble_average_test,test_set[,1])

### ensemble majority voting method on this
test set

ensemble_voting_test = rep(0,N_test)

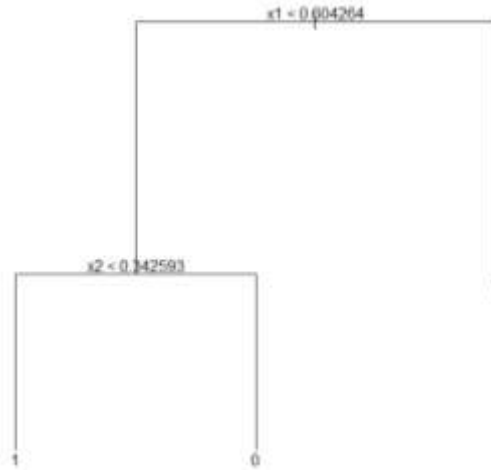
for (i in 1:N_test) {
  p =
c(lrfit2.preds_test[i],lrfit3.preds_test[i],lrfit4.pre
ds_test[i])

```

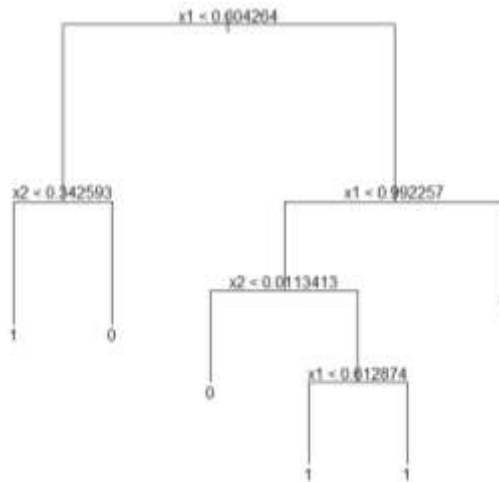
```
## get the  $p_{\hat{y}=0|x}$ 
y_pre = as.numeric(p > 0.5)
p0_new = (1/3)*sum(y_pre)
if(p0_new>0.5){
  ensemble_voting_test[i] = 1
}else{
  ensemble_voting_test[i] = 0
}
}
## get the performance of ensemble voting
method on this test set
table(ensemble_voting_test,test_set[,1])
```

Problem 48

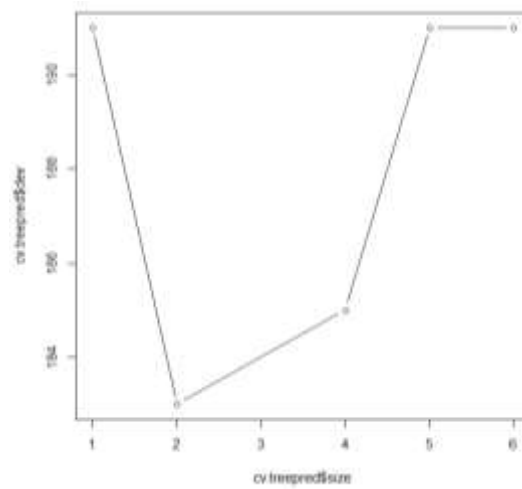
Here is the tree classification with default settings :



Here is the tree classification with complexity settings;



Cross validation to choose the best number of node:



So the best tree size would be 2.

