HW3_xz2735

Xiaofan Zhang(xz2735)

2/28/2019

Problem 1

To show that the Bayes-optimal classifier is the classifier which minimizes the probability of error:

Firstly, I decompose the risk $R(f)$ into conditional risks$R(f|x)$:

$$R(f|x) = \sum_{y\in[k]} L^{0-1}(y, f(x))p(y|x)$$

and

$$R(f) = \int_{\mathbf{R}^d} R(f|x)p(x)dx$$

. Here,

$$R(f|x) = \sum_{y\in[k]} I(\hat{y}_i \neq y_i)p(y_i|x) = \sum_{x\in\chi}\sum_{\hat{y}_i\neq y_i} p(y_i|x)$$

and to minimize $\sum_{x\in\chi}\sum_{\hat{y}_i\neq y_i} p(y_i|x)$, I have to minimize

$$\sum_{f_*(x)=\hat{y}_i\neq y_i} p(y_i|x)$$

, so I get a bayes optimal:

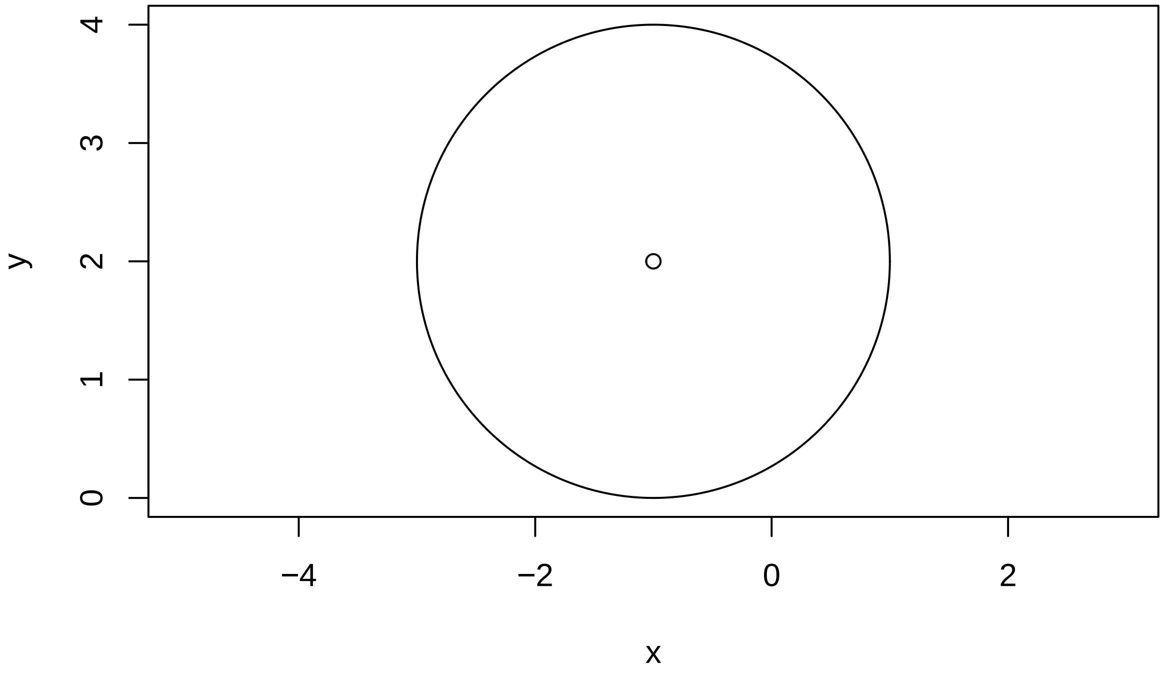
$$f_* = arg_f min \sum_{\hat{y}_i\neq y_i} p(y_i|x) = arg_f max \sum_{\hat{y}_i=y_i} p(y_i|x) = arg_f max p(y|x)$$

. As the intergral for R(f) is monotonically, so f_* is also the minimizor of R(f). Thus, $f_* = arg_f max p(y_i|x)$ is the optimal one that minimize the probability of error.

Problem 2

(a)

```r
library("plotrix")
x = -1
y = 2
plot(x, y, asp = 1, xlim = c(-4, 2),ylim=c(0,4))
draw.circle(-1, 2, 2, nv = 1000, border = NULL, col = NA, lty = 1, lwd = 1)
```

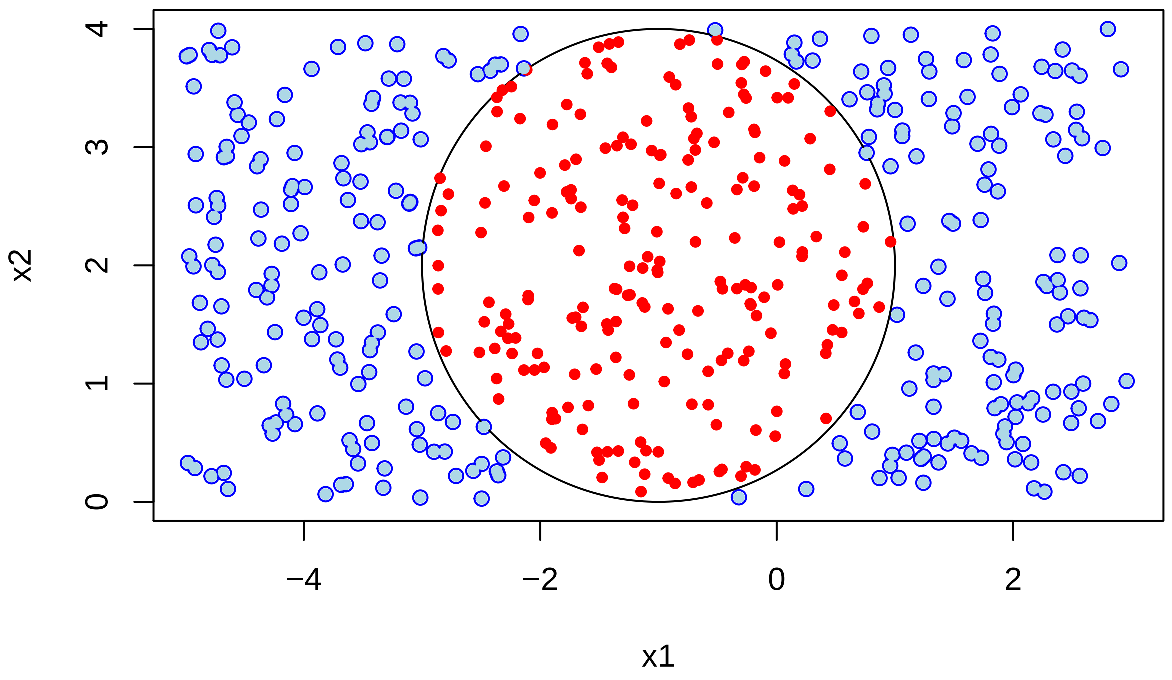


(b)

Basically, $(1+X_1)^2+(2-X_2)^2 > 4$ represents points that outside the circle, whereas $(1+X_1)^2+(2-X_2)^2 \leq 4$ represents points inside and in the circle.

(c)

```
x = runif(500, -5, 3)
y = runif(500, 0, 4)
fun <- function(x,y){return((1+x)^2+(2-y)^2)}
data = data.frame('x'=x, 'y'=y)
data$class = ifelse(fun(x,y)>4,'blue','red')
plot(data[data$class=='blue',]$x, data[data$class=='blue',]$y, asp = 1, xlim = c(-4, 2),ylim=c(0,4),col=
draw.circle(-1, 2, 2, nv = 1000, border = NULL, col = NA, lty = 1, lwd = 1)
points(data[!data$class=='blue',]$x, data[!data$class=='blue',]$y,col='red',pch=20)
points(data[data$class=='blue',]$x, data[data$class=='blue',]$y,col='lightblue',pch=20)
```



(0,0)

(3,8) and (22) are in blue class and (-1,1) is in red class.

(d)

If I decompose the equation of the graph, it will be

$$(1 + X_1)^2 + (2 - X_2)^2 = X_1^2 + X_2^2 + 2X_1 - 4X_2 + 5$$

. As the coefficients are all constant, so this function and the deexision boundary is linear towards X_1, X_2, X_1^2, X_2^2.

Problem 3

```
train_3 <- read.csv("~/Desktop/Statistical-machine-learning/HW3/train_3.txt", header=FALSE)
train_5 <- read.csv("~/Desktop/Statistical-machine-learning/HW3/train_5.txt", header=FALSE)
train_8 <- read.csv("~/Desktop/Statistical-machine-learning/HW3/train_8.txt", header=FALSE)
testdf = read.table("~/Desktop/Statistical-machine-learning/HW3/zip_test.txt", quote="\"", comment.char
xtest = as.matrix(testdf[testdf$V1==3|testdf$V1==5|testdf$V1==8,-1])
ytest = as.matrix(testdf[testdf$V1==3|testdf$V1==5|testdf$V1==8,1])
xtrain = rbind(train_3,train_5,train_8)
ytrain  = as.matrix(c(rep(3,dim(train_3)[1]),rep(5,dim(train_5)[1]),rep(8,dim(train_8)[1])))
colnames(xtrain)=1:256
colnames(xtest) = 1:256
```

1.

```
library("MASS")
train.error = c()
test.error = c()
r1 = lda(xtrain,ytrain)
train.error[1] = sum(predict(r1)$class != ytrain)/dim(xtrain)[1]
test.error[1] = sum(predict(r1,xtest)$class != ytest)/dim(xtest)[1]
```

2.

```
xtrain.scale = scale(xtrain,center=T,scale=F)
xtest.scale = scale(xtest,center=T,scale=F)
pca = svd(xtrain.scale)$v[,1:49]
pcaxtrain= xtrain.scale%*%pca
pcaxtest = xtest.scale%*%pca
r2 = lda(pcaxtrain,ytrain)
train.error[2] = sum(predict(r2)$class!=ytrain)/dim(pcaxtrain)[1]
test.error[2] = sum(predict(r2,pcaxtest)$class != ytest)/dim(pcaxtest)[1]
```

3.

```
filter = function(x){
  x=matrix(x,16,16)
  vec = rep(1:2,8)
  x = x[vec==1,]+x[vec==2,]
  x = x[,vec==1]+x[,vec==2]
  as.vector(x/4)
}
xtrain.filter = t(apply(xtrain,1,filter))
xtest.filter = t(apply(xtest,1,filter))
r3 = lda(xtrain.filter,ytrain)
train.error[3] = sum(predict(r3)$class != ytrain)/dim(xtrain.filter)[1]
test.error[3] = sum(predict(r3,xtest.filter)$class != ytest)/dim(xtest.filter)[1]
```

4.

```
library(glmnet)
r4 = glmnet(xtrain.filter,factor(ytrain),family="multinomial",alpha=0)
train.error[4] = sum(predict(r4, xtrain.filter, type = "class",s=0) != ytrain)/dim(xtrain.filter)[1]
test.error[4] = sum(predict(r4, xtest.filter, type = "class",s=0) != ytest)/dim(xtest.filter)[1]
```

Here, I use Ridge(alpha=0) to fit the multinomial logistic model, as the test error is smaller than lasso.

```
r5 = glmnet(xtrain.filter,factor(ytrain),family="multinomial",alpha=1)
```

```
## Warning: from glmnet Fortran code (error code -93); Convergence for 93th
## lambda value not reached after maxit=100000 iterations; solutions for
## larger lambdas returned
```

```
train.error[5] = sum(predict(r5, xtrain.filter, type = "class",s=0) != ytrain)/dim(xtrain.filter)[1]
test.error[5] = sum(predict(r5, xtest.filter, type = "class",s=0) != ytest)/dim(xtest.filter)[1]
```

Here, I use Lasso to fit the multinomial logistic model.

conclusion

```
error = cbind(train.error,test.error)
rownames(error)=c("LDA","PCA.LDA","Filter.LDA","Filter.Logistic.Ridge","Filter.Logistic.Lasso")
error
```

```
##                          train.error test.error
## LDA                      0.015945330 0.08739837
## PCA.LDA                  0.043849658 0.09146341
## Filter.LDA               0.033599089 0.07520325
## Filter.Logistic.Ridge 0.031890661 0.07926829
## Filter.Logistic.Lasso 0.002277904 0.09552846
```