STAT432_HW8

Taiga Hasegawa(taigah2)

2019/3/26

Question1

Following logstic regression was done by built-in function.

```
library(ElemStatLearn)
data(SAheart)

heart = SAheart
heart$famhist = as.numeric(heart$famhist)-1
n = nrow(heart)
p = ncol(heart)

heart.full = glm(chd~., data=heart, family=binomial)
round(summary(heart.full)$coef, dig=3)
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -6.151      1.308  -4.701    0.000
## sbp            0.007      0.006   1.135    0.256
## tobacco        0.079      0.027   2.984    0.003
## ldl            0.174      0.060   2.915    0.004
## adiposity      0.019      0.029   0.635    0.526
## famhist        0.925      0.228   4.061    0.000
## typea          0.040      0.012   3.214    0.001
## obesity       -0.063      0.044  -1.422    0.155
## alcohol        0.000      0.004   0.027    0.978
## age            0.045      0.012   3.728    0.000
```

```
# fitted value
yhat = (heart.full$fitted.values>0.5)
table(yhat, SAheart$chd)
```

```
##
## yhat      0   1
##   FALSE 256  77
##   TRUE   46  83
```

I'm going to replicate the above summary matrix using my own code.

```
# Gradient
my.gradient <- function(b, x, y)
{
    bm = as.matrix(b)
    expxb =  exp(x %*% bm)
    return(t(x) %*% (y - expxb/(1+expxb)))
}

# Hessian
my.hessian <- function(b, x, y)
{
```

```r
        bm = as.matrix(b)
        expxb =  exp(x %*% bm)
        x1 = sweep(x, 1, expxb/(1+expxb)^2, "*")
        return(-t(x) %*% x1)
    }

    #Newton-Raphson
    my.logistic <- function(b, x, y, tol = 1e-10, maxitr = 30, gr, hess, verbose = FALSE)
    {
        b_new = b

        for (j in 1:maxitr) # turns out you don't really need many iterations
        {
            b_old = b_new
            b_new = b_old - solve(hess(b_old, x, y)) %*% gr(b_old, x, y)

            if (verbose)
            {
                cat(paste("at iteration ", j,", current beta is \n", sep = ""))
                cat(round(b_new, 3))
                cat("\n")
            }
            if (sum(abs(b_old - b_new)) < tol) break;
        }
        return(b_new)
    }
```

```r
    x = as.matrix(cbind("intercept" = 1, heart[, 1:9]))
    y = as.matrix(heart[,10])

    # set up an initial value
    b = rep(0, ncol(x))

    mybeta = my.logistic(b, x, y, tol = 1e-10, maxitr = 20,
                         gr = my.gradient, hess = my.hessian, verbose = FALSE)
    mysd = sqrt(diag(solve(-my.hessian(mybeta, x, y))))

    # my summary matrix
    round(data.frame("beta" = mybeta, "sd" = mysd, "z" = mybeta/mysd,
          "pvalue" = 2*(1-pnorm(abs(mybeta/mysd)))), dig=3)
```

```
##              beta    sd      z pvalue
## intercept -6.151 1.308 -4.701  0.000
## sbp        0.007 0.006  1.135  0.256
## tobacco    0.079 0.027  2.984  0.003
## ldl        0.174 0.060  2.915  0.004
## adiposity  0.019 0.029  0.635  0.526
## famhist    0.925 0.228  4.061  0.000
## typea      0.040 0.012  3.214  0.001
## obesity   -0.063 0.044 -1.422  0.155
## alcohol    0.000 0.004  0.027  0.978
## age        0.045 0.012  3.728  0.000
```

```
# fitted value
yhat = (exp(x%*%mybeta)/(1+exp(x%*%mybeta))>0.5)
table(yhat, SAheart$chd)
```

```
##
## yhat      0   1
##   FALSE 256  77
##   TRUE   46  83
```

I got the exactly the same result with the one we found by built-in function.

```
# set up an initial value
b = rep(1, ncol(x))

mybeta = my.logistic(b, x, y, tol = 1e-10, maxitr = 20,
                     gr = my.gradient, hess = my.hessian, verbose = FALSE)

mysd = sqrt(diag(solve(-my.hessian(mybeta, x, y))))

# my summary matrix
round(data.frame("beta" = mybeta, "sd" = mysd, "z" = mybeta/mysd,
      "pvalue" = 2*(1-pnorm(abs(mybeta/mysd)))), dig=3)
```

I got the error because value in hessian matrix became too small to inverse.

Question2

a)

```
findRows <- function(zip, n) {
# Find n (random) rows with zip representing 0,1,2,...,9
res <- vector(length=10, mode="list")
names(res) <- 0:9
ind <- zip[,1]
for (j in 0:9) {
res[[j+1]] <- sample( which(ind==j), n ) }
return(res)
}
set.seed(1)
# find 100 samples for each digit for both the training and testing data
train.id <- findRows(zip.train, 100)
train.id = unlist(train.id)
test.id <- findRows(zip.test, 100)
test.id = unlist(test.id)
X = zip.train[train.id, -1]
Y = zip.train[train.id, 1]
dim(X)
```

```
## [1] 1000  256
```

```
Xtest = zip.test[test.id, -1]
Ytest = zip.test[test.id, 1]
dim(Xtest)
```

```
## [1] 1000  256
```

```r
#prior
pi_k=0.1

#centroid
mu_k=sapply(0:9, function(x) apply(X[Y==x,],2,FUN=mean))

#covariance
Sigma=1/(1000-10)*(99*cov(X[Y==0,])+99*cov(X[Y==1,])+99*cov(X[Y==2,])+99*cov(X[Y==3,])+99*cov(X[Y==4,])

#W_k
w_k=sapply(1:10, function(x) solve(Sigma)%*%mu_k[,x])

#b_k
b_k=sapply(1:10, function(x) -1/2*t(mu_k[,x])%*%solve(Sigma)%*%mu_k[,x]+log(pi_k))

#prediction
answer=sapply(1:1000,function(x) which.max(t(t(w_k)%*%t(Xtest)+b_k)[x,]))-1

#confusion matrix
table(Ytest, answer)
```

```
##      answer
## Ytest  0  1  2  3  4  5  6  7  8  9
##     0 93  1  0  0  0  0  6  0  0  0
##     1  0 93  0  0  2  0  0  0  1  4
##     2  1  2 77  3  3  1  3  2  8  0
##     3  2  0  3 77  2  8  0  1  5  2
##     4  4  2  4  0 78  0  3  1  0  8
##     5  4  0  2 13  4 73  2  1  0  1
##     6  2  1  1  1  2  3 89  0  1  0
##     7  0  1  0  1  6  1  0 84  0  7
##     8  3  0  0  8  6  3  0  1 75  4
##     9  0  0  0  0  4  0  0  5  2 89
```

```r
#prediction accuracy
mean(Ytest == answer)
```

```
## [1] 0.828
```

b)

QDA does not work on this dataset. So I used one of the regularized approaches provided in the lecture note.

```r
library(mlbench)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
library(glmnet)
```

```
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-16
```

```
library(klaR)
```

```
## Loading required package: MASS
```

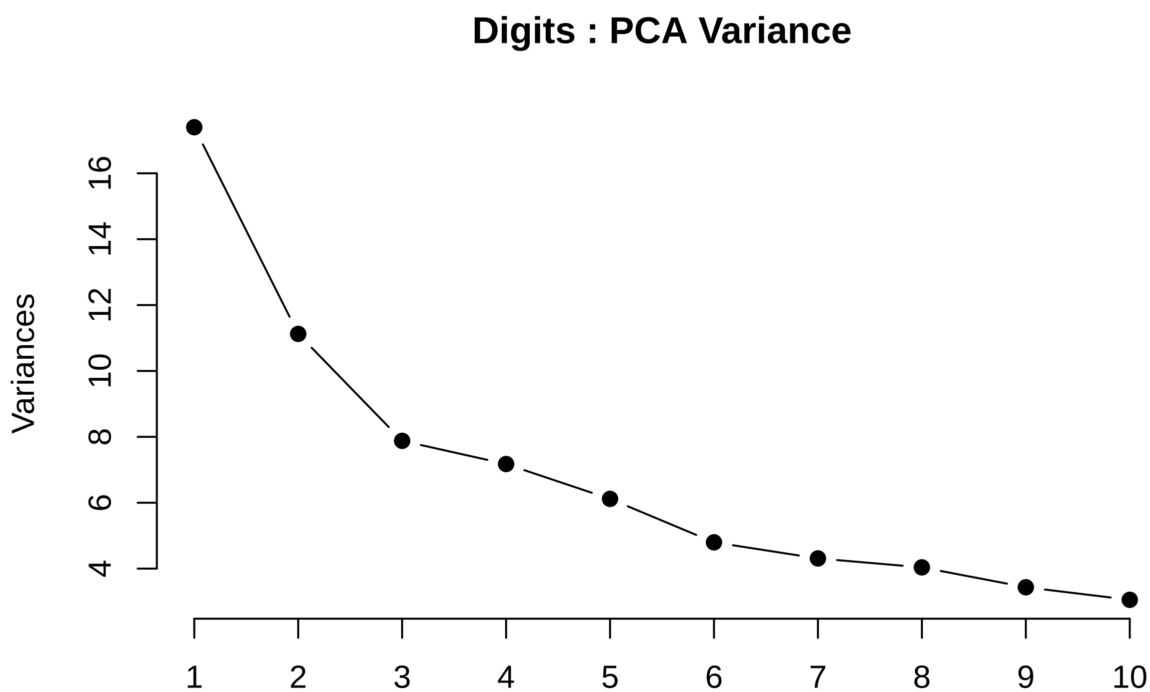
```
library(methods)
```

```
set.seed(1337)
data=data.frame(zip.train[train.id, ])
data$X1=as.factor(data$X1)
cv_5_grid = trainControl(method = "cv", number = 5)
set.seed(1337)
fit_rda_grid = train(X1 ~ ., data = data, method = "rda", trControl = cv_5_grid)
fit_rda_grid
```

```
## Regularized Discriminant Analysis
##
## 1000 samples
##  256 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 800, 800, 800, 800, 800
## Resampling results across tuning parameters:
##
##   gamma  lambda  Accuracy  Kappa
##   0.0    0.0       NaN           NaN
##   0.0    0.5     0.100     0.0000000
##   0.0    1.0     0.100     0.0000000
##   0.5    0.0     0.100     0.0000000
##   0.5    0.5     0.929     0.9211111
##   0.5    1.0     0.898     0.8866667
##   1.0    0.0     0.677     0.6411111
##   1.0    0.5     0.829     0.8100000
##   1.0    1.0     0.827     0.8077778
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were gamma = 0.5 and lambda = 0.5.
```

Next approach is that we reduced the dimension of the dataset first by PCA and then apply QDA.

```
zip=rbind(X,Xtest)
pca=prcomp(zip)
plot(pca, type = "l", pch = 19, main = "Digits : PCA Variance")
```




```r
pca=pca$x[,1:9]
X=pca[1:1000,]
Xtest=pca[1001:2000,]
dig.qda = qda(X, Y)
Ytest.pred=predict(dig.qda, Xtest)$class
table(Ytest, Ytest.pred)
```

```
##      Ytest.pred
## Ytest  0  1  2  3  4  5  6  7  8  9
##     0 89  0  0  0  0  3  7  0  1  0
##     1  0 88  1  0  5  0  1  0  4  1
##     2  2  0 91  3  1  1  0  0  2  0
##     3  1  0  3 81  0 12  0  0  3  0
##     4  2  0  4  0 81  1  1  4  5  2
##     5  3  0  0  4  2 84  2  0  4  1
##     6  5  0  1  0  2  1 89  0  2  0
##     7  0  0  2  0  6  0  0 81  2  9
##     8  0  0  1  5  2  5  0  0 82  5
##     9  0  0  1  0 10  0  0 13  4 72
```

```r
mean(Ytest == Ytest.pred)
```

```
## [1] 0.838
```

This approach is not as good as the previous one.

Question3

```r
#Prepare the dataset

y=c("No","No","Yes","Yes","Yes","No","Yes","No","Yes","Yes","Yes","Yes","Yes","No")
```

```r
Xtrain=data.frame(
  "outlook"=c("Rainy","Rainy","Overcast","Sunny","Sunny","Sunny","Overcast","Rainy","Rainy","Sunny","Ra
  "temperature"=c("Hot","Hot","Hot","Mild","Cool","Cool","Cool","Mild","Cool","Mild","Mild","Mild","Hot
  "humidity"=c("High","High","High","High","Normal","Normal","Normal","High","Normal","Normal","Normal"
  "windy"=c("False","True","False","False","False","True","True","False","False","False","Ture","True",
  "play_golf"=c("No","No","Yes","Yes","Yes","No","Yes","No","Yes","Yes","Yes","Yes","Yes","No")
  )

Xtest=c("Sunny","Hot","Normal","False")
```

```r
#Naive bayes
naivebayes=function(y,Xtrain,Xtest){
  prior_no=table(y)[1]/length(y)
  prior_yes=table(y)[2]/length(y)
  index=1
  condition_yes=rep(NA,length(Xtest))
  condition_no=rep(NA,length(Xtest))
  for(i in Xtest){
    X=Xtrain[Xtrain[,index]==i,]
    condition_yes[index]=sum(X[,ncol(X)]=="Yes")/table(y)[2]
    condition_no[index]=sum(X[,ncol(X)]=="No")/table(y)[1]
    index=index+1
  }
  posterior_yes=prod(condition_yes)*prior_yes
  posterior_no=prod(condition_no)*prior_no
  if(posterior_yes>posterior_no){
    cat("Yes")
  }else{cat("No")}
}
```

```r
naivebayes(y,Xtrain,Xtest)
```

```
## Yes
```