STOR 767 Spring 2019 Hw5: Computational Part

Due on 03/18/2019 in Class

Zhenghan Fang

Instruction.

- Please use **RMarkdown** to create a formatted report for the **Computational Part** of this homework.

Exercise 1

Define soft thresholding function and backfitting function.

```
soft.thresholding = function(z, lambda){
  if(abs(z) - lambda > 0){
    z.th = z / abs(z) * (abs(z) - lambda)
  }
  else{
    z.th = 0
  }
  return(z.th)
}

backfitting.LASSO = function(x, y, lambda, thresh=1e-20){
  N.pred = length(x[1,])
  n = length(x[,1])
  alpha = mean(y)
  beta = rep(0, N.pred)
  while(T){
    change = rep(0,N.pred)
    for(j in 1:N.pred){
      y.t = y - alpha - x[,-j] %*% beta[-j]
      beta.ls = (x[,j] %*% y.t) / n
      beta.lasso = soft.thresholding(beta.ls, lambda)
      change[j] = beta[j] - beta.lasso
      beta[j] = beta.lasso
    }
    if(sum(change^2) < thresh){
      break
    }
  }
  return(list("alpha" = alpha, "beta" = beta))
}
```

Perform backfitting. Left column: result of glmnet. Right column: result of my lasso.

```
x = matrix(rnorm(10000), 100, 100)
x.norm <- as.matrix(apply(x, 2, function(x) (x - mean(x))/sqrt(mean(x^2))))
y = rnorm(100)
lambda = 1/10
lasso.my = backfitting.LASSO(x, y, lambda)
lasso.glmnet = glmnet(x, y, alpha=1, lambda = 1/10, thresh=1e-20)
```

```r
cbind(glmnet = cbind(glmnet = coef(lasso.glmnet), my = rbind(matrix(lasso.my$alpha), matrix(lasso.my$be
```

```
## 101 x 2 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -0.110992175 -0.073078150
## V1          -0.027362148 -0.013308731
## V2           .            .
## V3           .            .
## V4           .            .
## V5           .            .
## V6           .            .
## V7           .            .
## V8           0.064776768  0.080489424
## V9           .            .
## V10          .            0.003502036
## V11          .            .
## V12          .            .
## V13          .            .
## V14          .            .
## V15          .            .
## V16          .            .
## V17         -0.011923815 -0.018826084
## V18          .            .
## V19          .            .
## V20          .            .
## V21          .            .
## V22          .            .
## V23          .            .
## V24          .            .
## V25          .            .
## V26          .            .
## V27          0.126737391  0.120936510
## V28          .            .
## V29          .            .
## V30          .            .
## V31          .            .
## V32          .            .
## V33          .            .
## V34          .            .
## V35          .            .
## V36          .            .
## V37          .            .
## V38          0.028759797  0.049783112
## V39          .            .
## V40         -0.017245754 -0.024655788
## V41          .            .
## V42          .            .
## V43          .            .
## V44          .            .
## V45          .            .
## V46          .            .
## V47          .            .
## V48          .            .
```

```
## V49           0.009007847  .
## V50           .            .
## V51           .            .
## V52           .            .
## V53          -0.180926676 -0.175595109
## V54           .            .
## V55           .            .
## V56           0.030402036  0.038209447
## V57           .            .
## V58          -0.017400607 -0.014057820
## V59           .            .
## V60           .            .
## V61           .            .
## V62           .            .
## V63           .            .
## V64           .            .
## V65           .            .
## V66           .            .
## V67           .            .
## V68           0.053150935  0.059852968
## V69          -0.045121408 -0.042835456
## V70           .            .
## V71           .            .
## V72           .            .
## V73           .            .
## V74           .            .
## V75           0.087606274  0.106623489
## V76           .            .
## V77           .            .
## V78           .            .
## V79           .            .
## V80           .            .
## V81           .            .
## V82           .            .
## V83           .            .
## V84           .            .
## V85           .            .
## V86           .            .
## V87           .            .
## V88           0.027219825  0.044945446
## V89           .            .
## V90           .            .
## V91           .            .
## V92           .            .
## V93           0.044784635  0.041944630
## V94           .            .
## V95           .            .
## V96           .            .
## V97           0.108262897  0.143128693
## V98           .            .
## V99           .            .
## V100          .            .
```

Exercise 2

Read data.

```
data.prost <- read.table('prostate.data.txt')
pred.name <- c("lcavol","lweight","age","lbph","svi","lcp","gleason","pgg45")
resp.name <- "lpsa"
train.idx <- which(data.prost$train)
test.idx <- which(!data.prost$train)
pred.N <- length(pred.name)
formula.full <- as.formula(paste(resp.name, paste(pred.name, collapse=" + "), sep=" ~ "))
```

Standardize the predictors to unit variance and zero mean.

```
for (i in pred.name){
  t <- data.prost[, i]
  data.prost[ ,i] <- (t-mean(t)) / sqrt(var(t))
}
```

Split the data into training and test set.

```
data.train <- data.prost[train.idx, ]
data.test <- data.prost[test.idx, ]
```

Cross-validation

Define function "bs.cv" for cross validation. Inputs: data.train: training data; K.cv: number of folds.

```
bs.cv = function(data.train, K.cv){
  N.train = length(data.train[,1])
  cv.idx = sample(1:K.cv,N.train,replace=T)
  err.cv=c()
  for (k in 1:K.cv){
    bs = regsubsets(formula.full, data = data.train[cv.idx!=k,], nvmax=8)
    test.mat = model.matrix(formula.full, data = data.train[cv.idx==k,])
    err.cv.temp = c()
    for (size.bs in 1:8){
      coefi= coef(bs, id = size.bs)
      pred=test.mat[,names(coefi)]%*%coefi
      err=(data.train[cv.idx==k,]$lpsa-pred)^2
      err.cv.temp[size.bs] = mean(err)
    }
    err.cv = cbind(err.cv, err.cv.temp)
  }
  err.cv = rowMeans(err.cv)
  return(err.cv)
}
```

Perform 5-fold CV.

```
K.cv = 5
err.cv = bs.cv(data.train, K.cv)
```

The best subset size is

```
best.size.cv = which.min(err.cv)
best.size.cv
```

```
## [1] 7
```

The test error (MSE) is

```
bs = regsubsets(formula.full, data = data.train, nvmax=8)
test.mat = model.matrix(formula.full, data = data.test)
coefi= coef(bs, id = best.size.cv)
pred=test.mat[,names(coefi)]%*%coefi
err=(data.test$lpsa-pred)^2
mean(err)
```

```
## [1] 0.5165135
```

Perform 10-fold CV.

```
K.cv = 10
err.cv = bs.cv(data.train, K.cv)
```

The best subset size is

```
best.size.cv = which.min(err.cv)
best.size.cv
```

```
## [1] 7
```

The test error is

```
coefi= coef(bs, id = best.size.cv)
pred=test.mat[,names(coefi)]%*%coefi
err=(data.test$lpsa-pred)^2
mean(err)
```

```
## [1] 0.5165135
```

AIC and BIC

```
bs = regsubsets(formula.full, data = data.train, nvmax=8)
AIC.bs = c()
BIC.bs = c()
for(size.bs in 1:8){
  pred.name <- names(coef(bs,id=size.bs))
  formula.t <- as.formula(paste(resp.name, paste(pred.name[-1], collapse=" + "), sep=" ~ "))
  model.t <- lm(formula.t, data.train)
  AIC.bs[size.bs] = AIC(model.t)
  BIC.bs[size.bs] = BIC(model.t)
}
```

The best subset size from AIC is

```
best.size.AIC = which.min(AIC.bs)
best.size.AIC
```

```
## [1] 7
```

Test error is

```
coefi= coef(bs, id = best.size.AIC)
pred=test.mat[,names(coefi)]%*%coefi
err=(data.test$lpsa-pred)^2
mean(err)
```

```
## [1] 0.5165135
```

The best subset size from BIC is

```
best.size.BIC = which.min(BIC.bs)
best.size.BIC
```

```
## [1] 2
```

Test error is

```
coefi= coef(bs, id = best.size.BIC)
pred=test.mat[,names(coefi)]%*%coefi
err=(data.test$lpsa-pred)^2
mean(err)
```

```
## [1] 0.4924823
```

Bootstrap .632 estimator

```
bs = regsubsets(formula.full, data = data.train, nvmax=8)

err.632.bs = c()
theta.fit <- function(x,y){lsfit(x,y)}
theta.predict <- function(fit,x){
  cbind(1,x)%*%fit$coef
}
sq.err <- function(y,yhat) { (y-yhat)^2}
for(size.bs in 1:8){
  pred.name <- names(coef(bs,id=size.bs))
  x = data.train[,pred.name[-1]]
  y = data.train[,resp.name]
  results <- bootpred(x,y,200,theta.fit,theta.predict, err.meas=sq.err)
  err.632.bs[size.bs] = results[[3]]
}
```

The best subset size from bootstrap .632 estimator is

```
best.size.632 = which.min(err.632.bs)
best.size.632
```

```
## [1] 7
```

Test error is

```
coefi= coef(bs, id = best.size.632)
pred=test.mat[,names(coefi)]%*%coefi
err=(data.test$lpsa-pred)^2
mean(err)
```

```
## [1] 0.5165135
```

Exercise 3.

Load dataset.

```r
SAheart = read.table('SAheart.data.txt', sep=",", header=T, row.names=1)
SAheart$chd = factor(SAheart$chd)
```

Use half of the dataset as training data.

```r
train = sample(1:462, 231)
```

SVM with various kernels

SVM with linear kernel.

```r
model.svm = svm(chd~., data=SAheart[train,], kernel = "linear")
summary(model.svm)
```

```
##
## Call:
## svm(formula = chd ~ ., data = SAheart[train, ], kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.1
##
## Number of Support Vectors:  133
##
##  ( 67 66 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Define error as number of misclassified samples/total number of samples.

Test error is

```r
pred.svm = predict(model.svm, newdata = SAheart[-train, -10])
err = sum(pred.svm != SAheart$chd[-train] ) / length(pred.svm)
err
```

```
## [1] 0.3073593
```

SVM with radial basis function kernel. Tune gamma by 10-fold cross validation.

```r
model.svm = best.tune(svm, chd~., data=SAheart[train,], kernel = "radial", gamma = 0.05*2^(-1:3))
summary(model.svm)
```

```
##
## Call:
## best.tune(svm, chd ~ ., data = SAheart[train, ], kernel = "radial",
##     gamma = 0.05 * 2^(-1:3))
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.025 0.05 0.1 0.2 0.4
##
## Number of Support Vectors:  149
##
##  ( 76 73 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Test error is

```
pred.svm = predict(model.svm, newdata = SAheart[-train, -10])
err = sum(pred.svm != SAheart$chd[-train] ) / length(pred.svm)
err
```

```
## [1] 0.2770563
```

SVM with sigmoid kernel. Tune gamma by 10-fold cross validation.

```
model.svm = best.tune(svm, chd~., data=SAheart[train,], kernel = "sigmoid", gamma = 0.05*2^(-1:3))
summary(model.svm)
```

```
##
## Call:
## best.tune(svm, chd ~ ., data = SAheart[train, ], kernel = "sigmoid",
##     gamma = 0.05 * 2^(-1:3))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  sigmoid
##        cost:  1
##       gamma:  0.025 0.05 0.1 0.2 0.4
##      coef.0:  0
##
## Number of Support Vectors:  151
##
##  ( 76 75 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Test error is

```
pred.svm = predict(model.svm, newdata = SAheart[-train, -10])
err = sum(pred.svm != SAheart$chd[-train] ) / length(pred.svm)
err
```

```
## [1] 0.2900433
```

LDA, QDA, and logistic regression

Perform LDA.

```r
model = lda(formula=chd~., data=SAheart, subset=train)
model
```

```
## Call:
## lda(chd ~ ., data = SAheart, subset = train)
##
## Prior probabilities of groups:
##         0         1
## 0.6753247 0.3246753
##
## Group means:
##        sbp  tobacco      ldl adiposity famhistPresent    typea  obesity
## 0 137.2051 2.555192 4.286154  23.06404      0.2756410 52.91667 25.52622
## 1 145.8533 4.627333 5.682267  28.03667      0.5866667 54.52000 27.02280
##    alcohol      age
## 0 15.46923 38.67308
## 1 19.78573 50.78667
##
## Coefficients of linear discriminants:
##                         LD1
## sbp             0.001626368
## tobacco         0.040463910
## ldl             0.225681772
## adiposity      -0.015040937
## famhistPresent  1.004167679
## typea           0.022398522
## obesity         0.009389706
## alcohol         0.002239492
## age             0.044142741
```

The test error of LDA is

```r
pred = predict(object = model, newdata = SAheart[-train, ])
err = sum(pred$class != SAheart$chd[-train] ) / length(pred$class)
err
```

```
## [1] 0.2597403
```

Perform QDA.

```r
model = qda(formula=chd~., data=SAheart, subset=train)
model
```

```
## Call:
## qda(chd ~ ., data = SAheart, subset = train)
##
## Prior probabilities of groups:
##         0         1
## 0.6753247 0.3246753
##
## Group means:
##        sbp  tobacco      ldl adiposity famhistPresent    typea  obesity
## 0 137.2051 2.555192 4.286154  23.06404      0.2756410 52.91667 25.52622
## 1 145.8533 4.627333 5.682267  28.03667      0.5866667 54.52000 27.02280
```

```
##    alcohol      age
## 0 15.46923 38.67308
## 1 19.78573 50.78667
```

The test error of QDA is

```
pred = predict(object = model, newdata = SAheart[-train, ])
err = sum(pred$class != SAheart$chd[-train] ) / length(pred$class)
err
```

```
## [1] 0.3073593
```

Perform Logistic regression.

```
model = lrm(formula=chd~., data=SAheart, subset=train)
model
```

```
## Logistic Regression Model
##
##  lrm(formula = chd ~ ., data = SAheart, subset = train)
##
##                        Model Likelihood     Discrimination    Rank Discrim.
##                           Ratio Test           Indexes           Indexes
##  Obs           231    LR chi2      68.94    R2       0.360    C       0.813
##   0            156    d.f.             9    g        1.734    Dxy     0.626
##   1             75    Pr(> chi2) <0.0001    gr       5.660    gamma   0.626
##  max |deriv| 1e-06                          gp       0.279    tau-a   0.276
##                                             Brier    0.161
##
##                  Coef    S.E.   Wald Z Pr(>|Z|)
##  Intercept       -7.7548 1.9187 -4.04  <0.0001
##  sbp              0.0013 0.0078  0.17  0.8637
##  tobacco          0.0404 0.0433  0.93  0.3516
##  ldl              0.2439 0.0875  2.79  0.0053
##  adiposity       -0.0120 0.0430 -0.28  0.7795
##  famhist=Present  1.1164 0.3342  3.34  0.0008
##  typea            0.0357 0.0179  1.99  0.0463
##  obesity          0.0100 0.0602  0.17  0.8686
##  alcohol          0.0051 0.0060  0.85  0.3956
##  age              0.0662 0.0186  3.57  0.0004
##
```

The test error of Logistic regression is

```
pred = predict(object = model, newdata = SAheart[-train, ])
err = sum((pred>0)*1 != SAheart$chd[-train] ) / length(pred)
err
```

```
## [1] 0.2943723
```