Nicholas Orriols

Statistical Learning

Homework 4

1. (i) The following code finds the lasso solution for . It then outputs which indices of the solution have nonzero values, and then outputs those nonzero values.

> data=read.csv("Problem1.csv", header=T)

> X=as.matrix(data[,-ncol(data)])

> Y=as.matrix(data[, ncol(data)])

>

> soft = function(x, l){ return (max(abs(x)-l, 0)\*sign(x))}

> lassoDescent = function(X, Y, lambda, tol) {

+ p = ncol(X)

+ B = matrix(rep(1,times=p),nrow=p)

+ r = Y - X %\*% B

+ err = 1

+ Bprev=B

+

+ while(err >= tol){

+ Bprev = B

+ for(j in 1:p) {

+ Xj = as.matrix(X[,j])

+ rj = r + Xj \* B[j]

+ Bp = soft((t(rj)%\*%Xj)/(t(Xj)%\*%Xj), lambda)

+ B[j] = Bp

+ r = rj - Xj %\*% B[j]

+ }

+ err = norm(Bprev - B, type="f")

+ }

+ return(B)

+ }

>

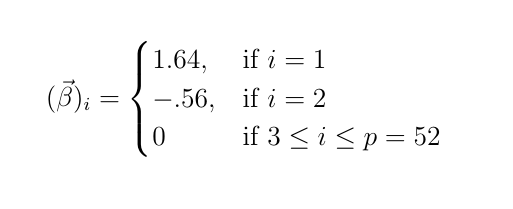
> B = lassoDescent(X, Y, .5, 10\*\*(-7))

> which(B != 0)

[1] 1 2

> B[B != 0]

[1] 1.6427195 -0.5610157

 The output says the lasso estimate for is where

(ii) The following code computes the glmnet lasso solution for and then outputs which indices have nonzero values, and then it outputs those nonzero values.

> library(glmnet)

> Bv = glmnet(X, Y, family="gaussian", alpha=1, intercept=F, lambda=c(.5))

> which(Bv$beta@x != 0)

[1] 1 2

> Bv$beta@x

[1] 1.6406751 -0.5589713

The solution from glmnet is nearly identical to the manual solution.

(iii) The following code generates a sequence of 100 equally spaced lambda values between 0 and 2, calculates the manual lasso solution, and outputs the graph on the left. The code also computes the glmnet solutions for each lambda and outputs the graph on the right. The graphs are nearly identical.

> lambdaSeq=as.matrix(seq(from=0, to=2, length.out = 100))

> B.mat = matrix(NA, nrow=length(lambdaSeq), ncol=ncol(X))

>

> for(i in 1:length(lambdaSeq))

+ B.mat[i,] = lassoDescent(X, Y, lambdaSeq[i], 10\*\*(-7))

>

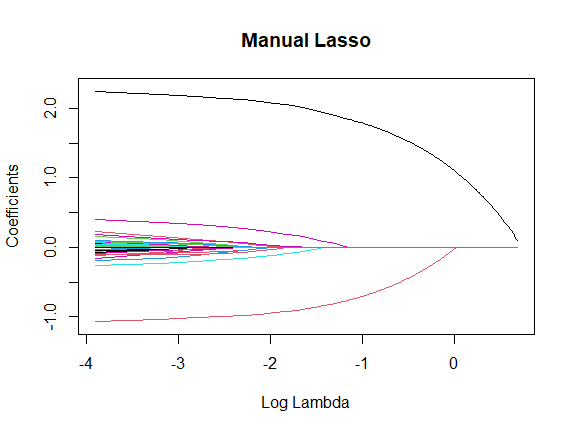
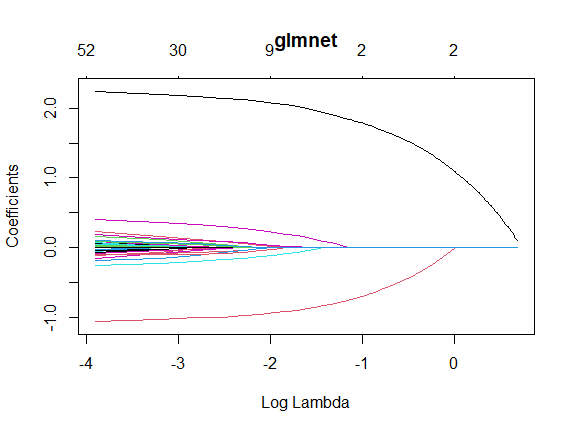
> matplot(log(lambdaSeq), B.mat, type="l", lty="solid",

+ ylab="Coefficients", xlab="Log Lambda", main="Manual Lasso")

>

> lassoFit = glmnet(X, Y, alpha=1, lambda=lambdaSeq)

> plot(lassoFit, xvar="lambda", main="glmnet")



1. (i) The following code loads and stores the data, and then produces the following heatmap.

> library(glmnet)

> library(ggplot2)

> library(reshape2)

> cor.mat = cor(X.train)

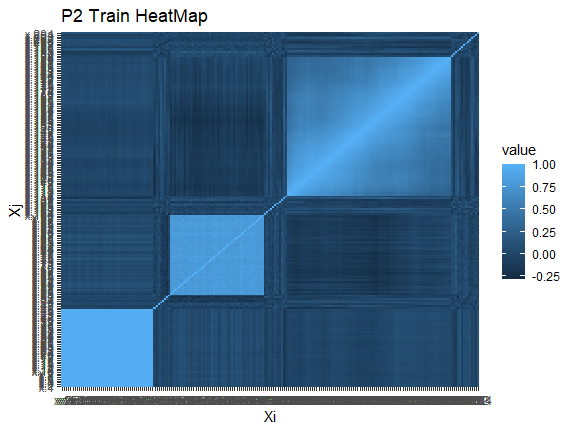
> melt.mat = melt(cor.mat)

> ggplot(data = melt.mat, aes(x=Var1, y=Var2, fill=value)) +

+ geom\_tile() +

+ ggtitle("P2 Train HeatMap") +

+ xlab("Xi") + ylab("Xj")



Since there are large areas of very high correlation, PCA will likely need only a few principal components to model a large portion of covariates’ variability. Whether this is helpful depends on whether the highly correlated variables are predictive of the response.

(ii) The following code computes the percentage of the covariates’ variance explained by the first 10 principal components. It also outputs a graph which shows the magnitude of variance for each of the first 10 PCs (these should sum to the 1st output quantity).

> modPCA = prcomp(X.train)

>

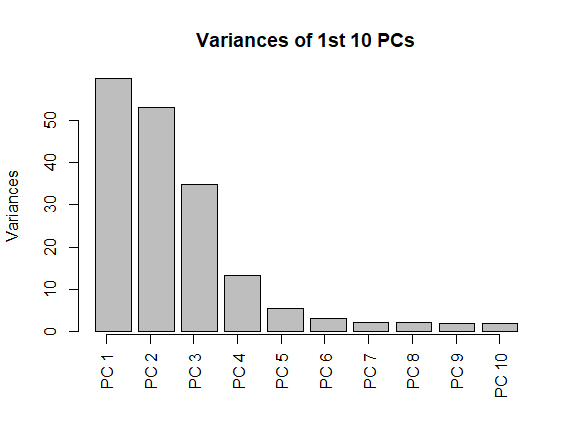
> summary(modPCA)$importance[3,10]

[1] 0.81907

>

> plot(modPCA, main="Variances of 1st 10 PCs")

> axis(1, at=seq(.5, 11.5, length.out = 10), labels=paste("PC", 1:10, sep=" "), las=2)



The first quantity indicates about 80% of the variation in the orginial covariates is explained by the 1st 10 PCs. From the plot, the 1st 3 PCs have comparatively large variances, while the rest of the (140) PCs have comparatively low variances. However, from just the graph, we could not infer PCA would be helpful for 2 reasons.

Firstly, it is possible the 1st 10 PCs are not predictive of the response, in which case PCA with 10 PCs would likely not perform well on the testing set. Secondly, although the 1st 3 PCs seemingly have large variances, the graph does not indicate what percentage of the covariates’ variation they explain; for instance, if the data has 10,000 PCs all with variances about equal to the variance of the 10th PC, the 1st 10 PCs would likely only model a small portion of the data’s variability. It is only by knowing the percentage of the variation explained by the 1st 10 PCs (.819) and the number of PCs (140) that the graph indicates PCA captures a large portion of the covariates’ variation with a small number of PCs; and even then, there is no guaruntee the PCs will capture the variability which is predictive of the response.

(iii) The following code fits each of the 6 models requested and outputs their testing MSEs.

> set.seed(4)

>

> # Lasso Regression

> lassoMod = cv.glmnet(X.train, Y.train, alpha=1)

> lassoPred = predict(lassoMod, newx=X.test)

> mean((lassoPred - Y.test)^2)

[1] 1.836522

>

> # Ridge Regression

> ridgeMod = cv.glmnet(X.train, Y.train, alpha=0)

> ridgePred = predict(ridgeMod, newx=X.test)

> mean((ridgePred - Y.test)^2)

[1] 3.932382

>

> # PCR w ncomp chosen by CV

> PCR = pcr(y ~ ., data=dataTrain, validation="CV")

> minNcomp.cv = as.numeric(which.min(RMSEP(PCR)$val[1,,])-1)

> PCR.cv.pred = predict(PCR, newdata=dataTest, ncomp=minNcomp.cv)

> mean((PCR.cv.pred[,1,1] - Y.test)^2)

[1] 1.595888

>

> # PCR w min ncomp explaining 95% variance

> pcr2=prcomp(X.train)

> minNcomp.var = min(which(summary(pcr2)$importance[3,] >= .95))

> PCR.var.pred = predict(PCR, newdata = dataTest, ncomp=minNcomp.var)

> mean((PCR.var.pred[,1,1] - Y.test)^2)

[1] 1.818899

>

> # PLSR w # of covariates chosen by CV

> PLSR = plsr(y~., data=dataTrain, validation="CV")

> PLSR.pred = predict(PLSR, newdata = dataTest)

> minCV = min(which.min(RMSEP(PLSR)$val[1,,])-1)

> mean((PLSR.pred[,1,minCV] - Y.test)^2)

[1] 1.607353

>

> # (Relaxed) Lasso w lambda and gamma chosen w CV

> lasso.R = cv.glmnet(X.train, Y.train, alpha=1, relax = T)

> lasso.R.pred = predict(lasso.R, newx=X.test)

> mean((lasso.R.pred - Y.test)^2)

[1] 1.764613

The two best approaches were PCR with the number of components chosen by CV (15 PCs) and PLSR with number of components chosen by CV (5 components); both testing MSEs are about 1.6.

PCR performed well probably because the variation explained by the 1st 15 PCs was predictive of the response. Since PCR chooses PCs to maximize the amount of explained variation in the covariates, usually highly correlated covariates’ variability is explained by the first several PCs. Since only 15 PCs (out of 140) were most predictive of the response, this indicates that the highly correlated covariates were predictive of the response. In contrast, the other PCR used more (38) PCs and had a testing MSE of about 1.8, which would indicate overfitting.

> PCR.var.pred.tr = predict(PCR, newdata=dataTrain, ncomp=minNcomp.var)

> mean((PCR.var.pred.tr[,1,1] - Y.train)^2)

[1] 0.9140008

The training MSE is much lower than the testing MSE, so PCR with 38 PCs was overfit to the training data. Usage of CV is the reason for the discrepancy between the testing performances of the two PCR approaches.

PCR’s performance indicates that the highly correlated covariates were predictive of the response. Consequently, PLSR, which transforms each covariate to maximize its individual correlation with the response, should have also captured this relationship with a relatively small number of components. Accordingly, PLSR performed well with only 5 components. Additionally, since PLSR includes the response in the calculation of the model (whereas PCR does not), PLSR typically reduces bias from PCR; since PCR performed well in this situation anyway, then it was likely PLSR would also perform well.