porject-5

Chitra Karki

11/9/2021

Reading data

```
df.crime = read.csv("crime.csv")
```

1. Data Preparation: Start with the data set crime.csv from the class website. We will first dosome minor data preparation and exploration.

a. Remove the first five columns from the data since these are not predictive.

```
df.crime = df.crime[,-c(1:5)]
```

b. Take a look at the missing percentage of each remaining variable. Remove those heavy missing, say, over 60%.

```
nrows = dim(df.crime)[1]
col2drop = which(apply(df.crime,2,function(x) sum(is.na(x))/nrows * 100) > 60)
# columns containing more then 60 % NA values
col2drop
```

```
##         LemasSwornFT      LemasSwFTPerPop    LemasSwFTFieldOps
##                   97                   98                   99
## LemasSwFTFieldPerPop        LemasTotalReq    LemasTotReqPerPop
##                  100                  101                  102
##      PolicReqPerOffic          PolicPerPop    RacialMatchCommPol
##                  103                  104                  105
##        PctPolicWhite          PctPolicBlack         PctPolicHisp
##                  106                  107                  108
##        PctPolicAsian          PctPolicMinor  OfficAssgnDrugUnits
##                  109                  110                  111
##     NumKindsDrugsSeiz      PolicAveOTWorked            PolicCars
##                  112                  113                  117
##        PolicOperBudg  LemasPctPolicOnPatr  LemasGangUnitDeploy
##                  118                  119                  120
##      PolicBudgPerPop
##                  122
```

```
df.crime = df.crime[,-col2drop]
# checking dimension after droping cols with heavy NAs
dim(df.crime)
```

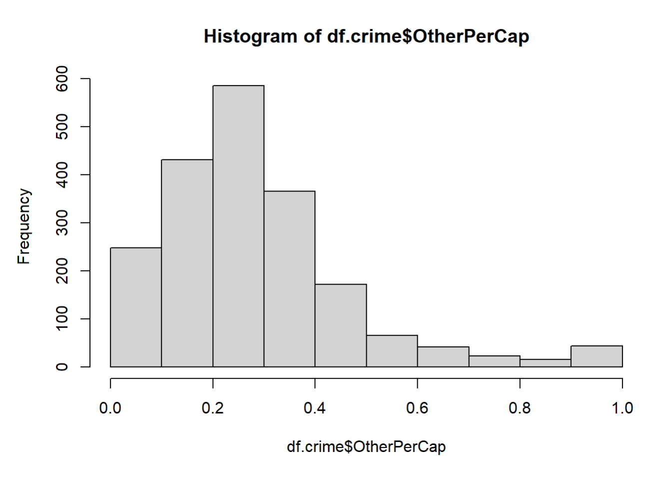
```
## [1] 1994  101
```

c. Impute or replace the remaining missing values appropriately.

```
which(apply(df.crime,2,function(x) sum(is.na(x))/nrows * 100) > 0)
```

```
## OtherPerCap
##          26
```

```
# this shows now we are left with single NA value at the 26 i.e "otherpercap" column.
# lets replace the missing with the median value.
hist(df.crime$OtherPerCap) # looking at the distribution of data points.
```

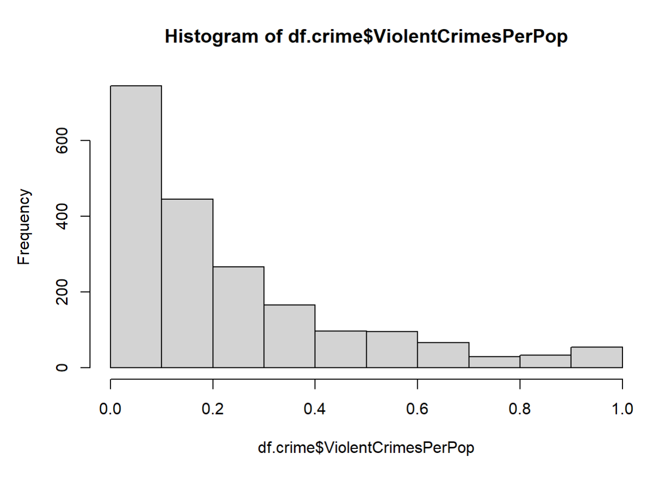



```
df.crime[which(is.na(df.crime$OtherPerCap)),"OtherPerCap"] = median(df.crime$OtherPerCap, na.rm = T)

# checking for nas
sum(is.na(df.crime)) # now the data is clean.
```

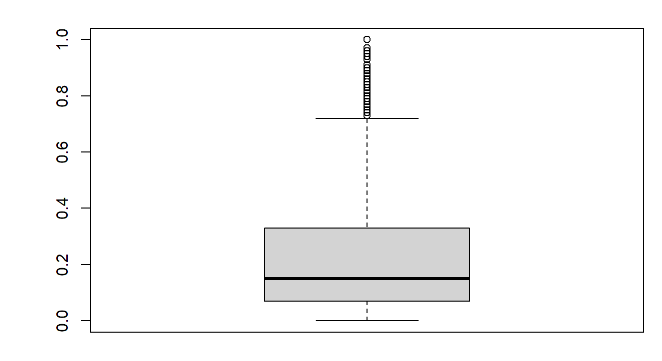
```
## [1] 0
```

d. Conduct some EDA (which could be involved). In particular, check the distribution of the target variable ViolentCrimesPerPop.

```
hist(df.crime$ViolentCrimesPerPop)
```




```
# the distribution is something like exp(-x)
boxplot(df.crime$ViolentCrimesPerPop)
```



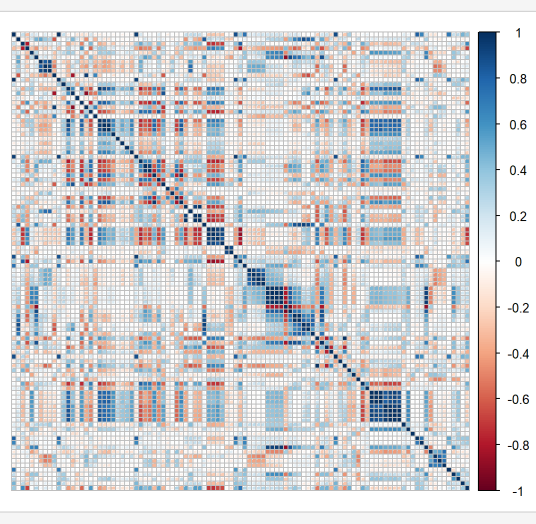
```
# there are ceratin outliers.More amount of data are greater then median
summary(df.crime$ViolentCrimesPerPop)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.070   0.150   0.238   0.330   1.000
```

```
library(corrplot)
```

```
## corrplot 0.90 loaded
```

```
corrplot(cor(df.crime),type = "full",tl.pos = "n",method = "square")
```



```
# there are several variables correlated with each other. Also some other variables are positively and negatively
correlated with the response variable.
```

2. Partitioning Data: Randomly partition your data into two sets: the training set D1 and the test set D2 with a ratio of 2:1. In order for your results to be reproducible, report the random seed that you use in the partitioning.

```
set.seed(1) # seed 123 for reproducibility
train.index = sample(1:nrows,2/3*nrows,replace = F)
test.index = -train.index
d1 = df.crime[train.index,]
d2 = df.crime[test.index,]
```

3. Predictive Modeling: Referring to the sample R code R09.R from the class website, fit at least five models of your own choice from the following models using the training set D1:

i. Linear regression with stepwise selection;
ii. LASSO
iii. Ridge Regression (RR)
iv. Principal Components Regression (PCR)
v. Partial least squares regression (PLSR)
vi. Weighted orthogonal components regression (WOCR)
vii. Total least squares regression (TLSR)
viii. Stagewise regression
ix. Least angle regression (LAR) You might want to describe how you select the tuning parameter if such decision needs to be made in each approach. Then apply the tted model to the test set D2. Report the mean square error of prediction (MSEP) and compare.

i> linear regression with stepwise selections

```
library(MASS)
fit.full = lm(d1$ViolentCrimesPerPop~.,data = d1)
fit.step <- stepAIC(fit.full,direction="both", k=log(nrow(d1)))
```

```
#fit.step$anova
best.model = summary(fit.step)
best.formula = as.formula(best.model$call)
best.fit = lm(best.formula,data = d1)
old.dat =as.data.frame(model.matrix(best.fit))
new.dat = d2[,names(old.dat)[-1]]
y.pre = predict(best.fit,newdata = new.dat)
y.obs = d2[,dim(d2)[2]]
plot(y.obs,y.pre,pch=19,col="red",main = "stepwise Regression")
abline(a=0,b=1,col="black")
```


```
MSE.step = mean((y.obs-y.pre)^2);MSE.step
```

```
## [1] 0.02101974
```

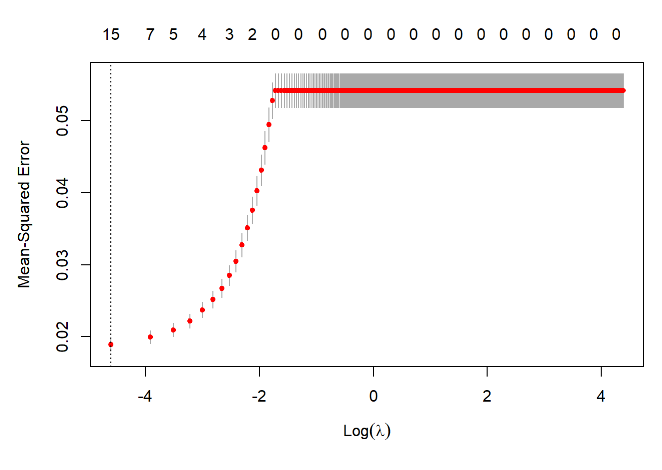
ii> LASSO

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

```
lambda= seq(0, 80.0, 0.01)   # AGAIN, APPROPRIATE ADJUSTMENT MIGHT BE NEEDED
# SETTING alpha=0 IN glmnet GIVES RIDGE REGRESSION
X.train = as.matrix(d1[,-dim(d1)[2]])
y.train = d1[,dim(d1)[2]]
cv.LR <- cv.glmnet(x=X.train, y=y.train, alpha = 1, lambda = lambda, nfolds=10)  # 10-FOLD CV BY DEFAULT
plot(cv.LR)
```



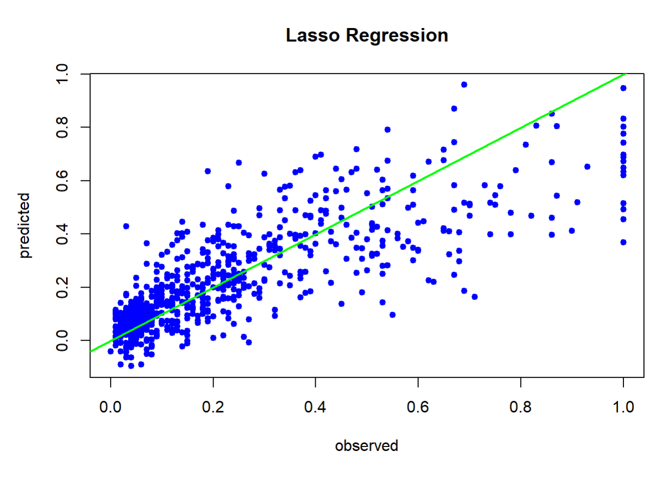

```
names(cv.LR)
```

```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"
##  [6] "nzero"      "call"       "name"       "glmnet.fit" "lambda.min"
## [11] "lambda.1se" "index"
```

```
lmbd0 <- cv.LR$lambda.min; lmbd0 # MINIMUM CV ERROR
```

```
## [1] 0
```

```
# lmbd0 <- cv.RR$lambda.1se  # 1SE RULE
fit.LR <- cv.LR$glmnet.fit
y.pred <- predict(fit.LR, s=lmbd0, newx = as.matrix(d2[,-dim(d2)[2]]))
yobs <- d2[,dim(d2)[2]]
plot(yobs, y.pred, xlab="observed", ylab="predicted",
    col="blue", pch=19, cex=0.8, main="Lasso Regression")
abline(a=0, b=1, col="green", lwd=2)
```

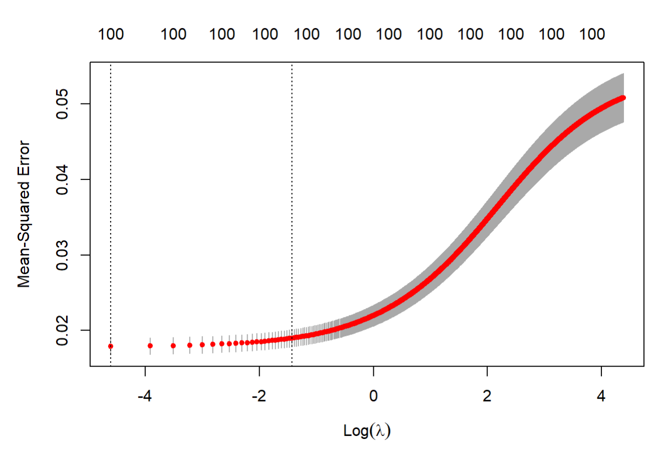



```
MSEP.LR <- mean((yobs -y.pred)^2)
MSEP.LR
```

```
## [1] 0.02041003
```

iii> Ridge Regression

```
library(glmnet)
lambda= seq(0, 80.0, 0.01)   # AGAIN, APPROPRIATE ADJUSTMENT MIGHT BE NEEDED
# SETTING alpha=0 IN glmnet GIVES RIDGE REGRESSION
X.train = as.matrix(d1[,-dim(d1)[2]])
y.train = d1[,dim(d1)[2]]
cv.RR <- cv.glmnet(x=X.train, y=y.train, alpha = 0, lambda = lambda, nfolds=10)  # 10-FOLD CV BY DEFAULT
plot(cv.RR)
```



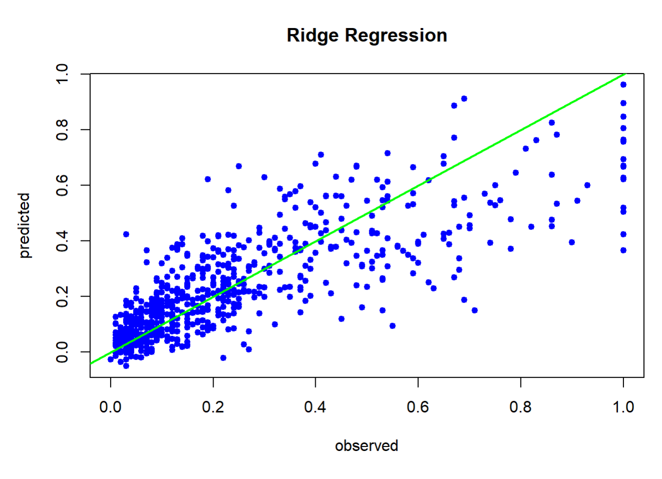

```
names(cv.RR)
```

```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"
##  [6] "nzero"      "call"       "name"       "glmnet.fit" "lambda.min"
## [11] "lambda.1se" "index"
```

```
lmbd0 <- cv.RR$lambda.min; lmbd0 # MINIMUM CV ERROR
```

```
## [1] 0.01
```

```
# lmbd0 <- cv.RR$lambda.1se  # 1SE RULE
fit.RR <- cv.RR$glmnet.fit
y.pred <- predict(fit.RR, s=lmbd0, newx = as.matrix(d2[,-dim(d2)[2]]))
yobs <- d2[,dim(d2)[2]]
plot(yobs, y.pred, xlab="observed", ylab="predicted",
    col="blue", pch=19, cex=0.8, main="Ridge Regression")
abline(a=0, b=1, col="green", lwd=2)
```




```
MSEP.RR <- mean((yobs -y.pred)^2); MSEP.RR
```

```
## [1] 0.01976178
```

iv> Principal Components Regression (PCR)

```
#install.packages("pls")
library(pls)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:corrplot':
##
##     corrplot
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
ncol(d1)
```

```
## [1] 101
```

```
fit.PCR <- pcr(d1$ViolentCrimesPerPop ~ ., ncomp=100, data=d1, method = pls.options()$pcralg,
    validation = "CV", segments = 10,  segment.type ="random", scale=TRUE)
#summary(fit.PCR)
#names(fit.PCR);
CV <- fit.PCR$validation; names(CV)
```

```
## [1] "method"       "pred"         "coefficients" "gammas"       "PRESS0"
## [6] "PRESS"        "adj"          "segments"     "ncomp"
```

```
par(mfrow=c(2,1), mar=rep(4,4))
plot(1:CV$ncomp, CV$PRESS,  xlab="number of PCs", ylab="PRESS", type="b", col="blue", lwd=2)
# plot(fit.PCR)
ncomp.best <- which.min(CV$PRESS); ncomp.best
```

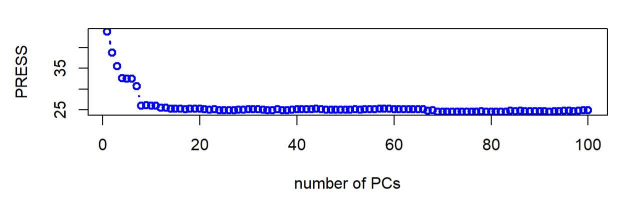
```
## [1] 73
```

```
# FIT THE BEST PCR MODEL WITHOUT V-FOLD CV (SETTING V=1 WOULD DO)
fit.PCR.best <- pcr(d1$ViolentCrimesPerPop ~ ., ncomp=ncomp.best, data=d1, method = pls.options()$pcralg,
     segments = 1, scale=TRUE)
#summary(fit.PCR.best)

# PREDICTION
?predict.mvr
```

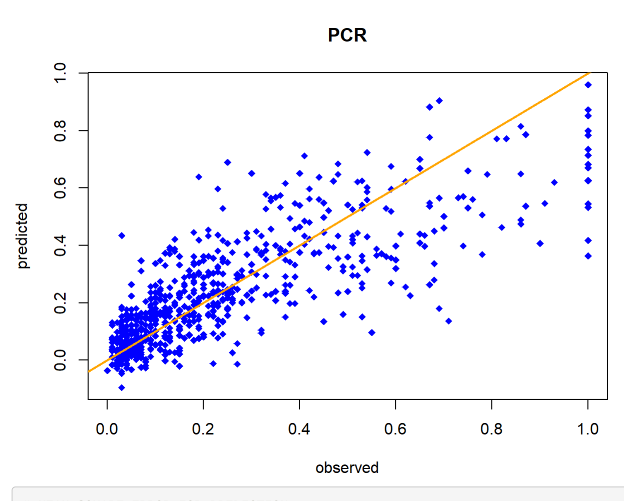
```
## starting httpd help server ...
```

```
##  done
```

```
yhat.PCR <- predict(fit.PCR.best, newdata=d2[,-dim(d2)[2]], comps=1:ncomp.best)   # THE ARGUMENT comps= IS IMPORT
ANT!
yobs <- d2[,dim(d2)[2]]

# PREDICTED VS. OBSERVED
par(mfrow=c(1,1), mar=rep(4,4))
```




```
plot(yobs, yhat.PCR, type="p", pch=18, col="blue",
    xlab="observed", ylab="predicted", main="PCR")
abline(a=0, b=1, col="orange", lwd=2)
```

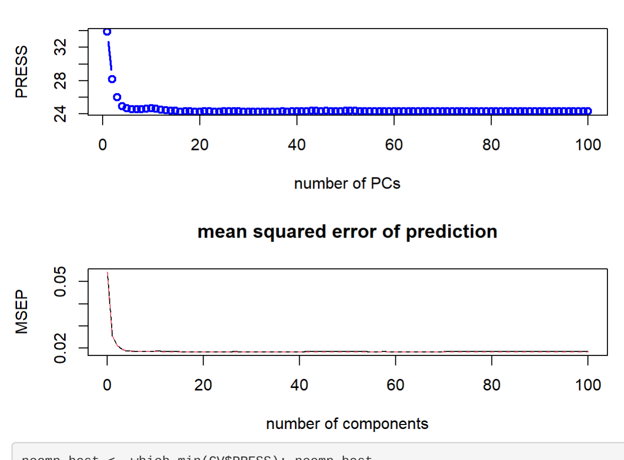



```
# MEAN SQUARE ERROR FOR PREDICTION
MSEP.PCR <- mean((yobs-yhat.PCR)^2); MSEP.PCR
```

```
## [1] 0.02004789
```

v> Partial least squares regression (PLSR)

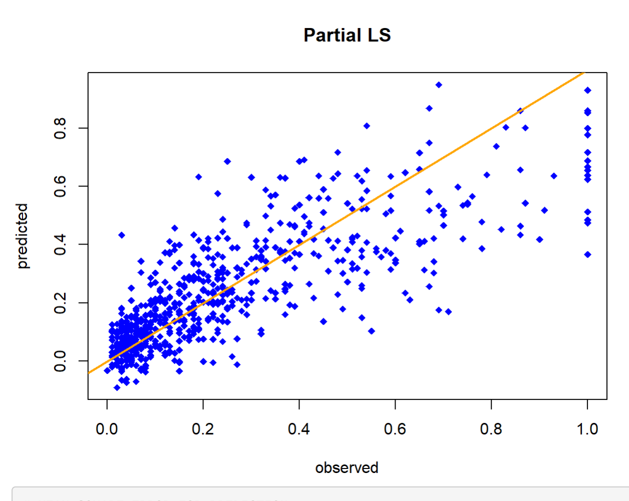
```
library(pls)
fit.PLS <- plsr(d1$ViolentCrimesPerPop ~ ., ncomp=100, data=d1, method = "simpls",
    validation = "CV", segments = 10,  segment.type ="random", scale=TRUE)
#summary(fit.PLS)

CV <- fit.PLS$validation
par(mfrow=c(2,1), mar=rep(4,4))
plot(1:CV$ncomp, CV$PRESS,  xlab="number of PCs", ylab="PRESS", type="b", col="blue", lwd=2)
# plot(fit.PLS)
# ?validationplot
validationplot(fit.PLS, val.type = "MSEP", main="mean squared error of prediction")
```




```
ncomp.best <- which.min(CV$PRESS); ncomp.best
```

```
## [1] 23
```

```
fit.PLSR.best <- plsr(d1$ViolentCrimesPerPop ~ ., ncomp=ncomp.best, data=d1, method = "simpls",
    validation = "none", scale=F)
#summary(fit.PLSR.best)

# MAKE PREDICTION
yhat.PLSR <- predict(fit.PLSR.best, newdata=d2[,-dim(d2)[2]], comps=1:ncomp.best)   # THE ARGUMENT comps= IS IMPO
RTANT!
yobs <- d2[,dim(d2)[2]]

# PREDICTED VS. OBSERVED
par(mfrow=c(1,1), mar=rep(4,4))
plot(yobs, yhat.PLSR, type="p", pch=18, col="blue",
    xlab="observed", ylab="predicted", main="Partial LS")
abline(a=0, b=1, col="orange", lwd=2)
```




```
# MEAN SQUARE ERROR FOR PREDICTION
MSEP.PLSR <- mean((yobs-yhat.PLSR)^2); MSEP.PLSR
```

```
## [1] 0.02031212
```

comparing mse of all the models implemented above

```
errors = c("MSE.step","MSEP.LR", "MSEP.RR","MSEP.PCR","MSEP.PLSR")
values = round(c(MSE.step,MSEP.LR, MSEP.RR,MSEP.PCR,MSEP.PLSR),5)
cbind(errors,values)
```

```
##      errors      values
## [1,] "MSE.step"  "0.02102"
## [2,] "MSEP.LR"   "0.02041"
## [3,] "MSEP.RR"   "0.01976"
## [4,] "MSEP.PCR"  "0.02005"
## [5,] "MSEP.PLSR" "0.02031"
```

The tuning parameters,if involved,were computed with cross validation method. The minimum error is for Ridge Regression.