Chapter 6

Wenduo Wang

July 20, 2016

Problem 9

```
library(dplyr)
College <- read.csv("College.csv", header=TRUE)
```

(a) Split the data set into a training set and a test set.

Split the College dataset into a training set containing 80% of total rows, and a test set containing the rest 20%.

```
training_index <- sample(nrow(College), round(nrow(College)*0.8))
training_set <- College[training_index,]
test_set <- College[-training_index,]
```

(b) Fit a linear model using least squares on the training set, and report the test error obtained.

Fit a linear model using all variables except school names (`X`). This simplifies the problem, but because each school appears only once in the dataset, not including school names in the training set will not affect prediction on other schools.

```
lm_model <- lm(Apps ~ ., data=training_set[, -1])
prediction_lm <- predict(lm_model, test_set)
RMSE <- mean((prediction_lm - test_set$Apps)^2)^.5
cat("The prediction RMSE using simple linear model is", RMSE)
output>>> The prediction RMSE using simple linear model is 1311.398
```

(c)Fit a ridge regression model on the training set, with chosen by cross-validation. Report the test error obtained.

We will use the `cv.glmnet()` function supplied in `glmnet` package to do the ridge regression, with λ chosen by cross-validation.

```
library(glmnet)
output>>> Loading required package: Matrix
output>>> Loading required package: foreach
output>>> Loaded glmnet 2.0-5
training_set_4ridge <- model.matrix(~., data=training_set[,-c(1, 3)])
ridge_model <- cv.glmnet(x=training_set_4ridge, y=training_set$Apps, nfolds=10, alpha=0)
ridge_model_lambda <- ridge_model$lambda.1se
prediction_ridge <- predict(ridge_model, model.matrix(Apps~., test_set[, -1]))
RMSE <- mean((prediction_ridge - test_set$Apps)^2)^.5
cat("The prediction RMSE using Ridge regression is", RMSE)
output>>> The prediction RMSE using Ridge regression is 2199.06
```

(d) Fit a lasso model on the training set, with chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

For Lasso regression, the function is the same as Ridge regression, with the difference of setting `alpha=1`. The following code implements Lasso regression on the training set, and calculate the prediction RMSE on the test set.

The λ is determined by cross-validation, all of which are printed with corresponding non-zero coefficient counts.

```
lasso_model <- cv.glmnet(x=training_set_4ridge, y=training_set$Apps, nfolds=10, alpha=1)
lambda_ind <- match(lasso_model$lambda.1se, lasso_model$lambda)
prediction_lasso <- predict(lasso_model, model.matrix(Apps~., test_set[, -1]))
RMSE <- mean((prediction_lasso - test_set$Apps)^2)^.5
cat("The prediction RMSE using Lasso regression is", RMSE)
## The prediction RMSE using Lasso regression is 1386.883
cat("Non-zero coefs corresponding to the chosen lambda is", lasso_model$nzero[[lambda_ind]])
## Non-zero coefs corresponding to the chosen lambda is 4
```

(e) Fit a PCR model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

Here `pcr()` function from `pls` library is used to fit PCA model on the training data. Before fitting the model, missing values are dropped from the training set, as required by the function. Within `pcr()` function, feature scaling and cross-validation are enforced, and therefore the number of component M is chosen automatically by the function. Before making prediction on test set, the model is trained on the whole training set with 2 principal components.

```
library(pls)
output>>>
output>>> Attaching package: 'pls'
output>>> The following object is masked from 'package:stats':
output>>>
output>>>     loadings
training_set_4pcr <- na.omit(training_set)[, -1]
set.seed(1)
pc_model <- pcr(Apps~., data=training_set_4pcr, scale=TRUE, validation="CV")
cat("The M chosen by cross-validation is", pc_model$ncomp)
output>>> The M chosen by cross-validation is 17
pc_model <- pcr(Apps~., data=training_set_4pcr, scale=TRUE, ncomp=pc_model$ncomp)
prediction_pca <- predict(pc_model, test_set)
RMSE <- mean((prediction_pca - test_set$Apps)^2)^.5
cat("The prediction RMSE using PCA is", RMSE)
output>>> The prediction RMSE using PCA is 2438.857
```

(f) Fit a PLS model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

The process of using partial least squares is very similar to PCA, which applies on the same dataset, except for `plsr` function is used in place of `pcr`. With `plsr`, M is chosen by specifying cross-validation.

```
pls_model <- plsr(Apps~., data=training_set_4pcr, scale=TRUE, validation="CV")
cat("The M chosen by cross-validation is", pls_model$ncomp)
output>>> The M chosen by cross-validation is 17
pls_model <- plsr(Apps~., data=training_set_4pcr, scale=TRUE, ncomp=pls_model$ncomp)
```

```
prediction_pls <- predict(pls_model, test_set)
RMSE <- mean((prediction_pls - test_set$Apps)^2)^.5
cat("The prediction RMSE using PLS is", RMSE)
output>>> The prediction RMSE using PLS is 1601.086
```

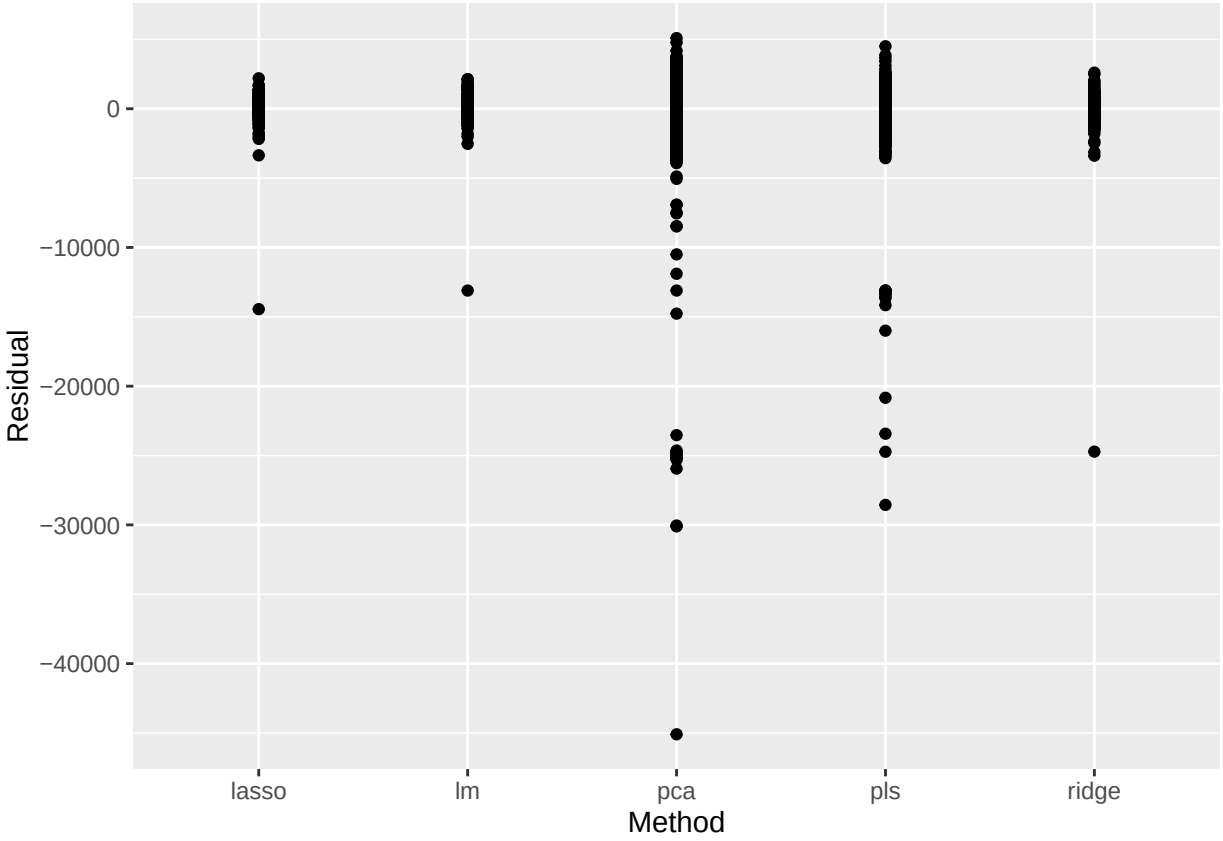
(g) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

To make a cross-sectional comparison between the five methods, a points plot has been made. The vertical axis represents residuals.

It is observed that in terms of residual pattern/distribution, simple linear regression, Lasso regression and Ridge regression are similar, and relatively tight fit and small spread. PLS has a relatively larger spread, but PCA has the largest deviation and spread, probably due to the chosen number of principle components is small so that a large portion of variability is not captured in the model.

Generally, with the five models, the prediction accuracy in terms of RMSE is around 1200 for linear regression, Lasso and Ridge regression, while PLS is slightly higher at ~1500, and PCA result in an RMSE in the range of 3000~4000.

```
library(ggplot2)
Residual <- c(prediction_lm, prediction_ridge, prediction_lasso, prediction_pca, prediction_pls)

Method <- c(rep("lm", length(prediction_lm)),
            rep("ridge", length(prediction_ridge)),
            rep("lasso", length(prediction_lasso)),
            rep("pca", length(prediction_pca)),
            rep("pls", length(prediction_pls)))

Method <- as.factor(Method)

residual_df <- data.frame(Residual, Method)

residual_df[, 1] <- residual_df[, 1] - test_set$Apps

qplot(x=Method, y=Residual, data=residual_df, geom="auto")
## Note: no visible global function definition for '.'
```



Problem 11

(a) Try out some of the regression methods explored in this chapter, such as best subset selection, the lasso, ridge regression, and PCR. Present and discuss results for the approaches that you consider.

First load the libraries used in the following section.

```
rm(list=ls())
gc()
```

```
##           used (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 1473246 78.7    2637877 140.9  2164898 115.7
## Vcells 2122988 16.2    3851194  29.4  3851194  29.4
```

```
library(MASS)
library(glmnet)
library(pls)
```

Before delving into the modeling process, the data is normalized by function `stdize` as expressed as $\frac{x-\mu}{\sigma}$, where μ and σ are the mean and standard deviation of the whole dataset. This is because by observation it is found that the column variables within the dataset have different scales, and by normalizing the variables, all the variables are brought to the same level, which is very important for regularized regression methods, e.g. Lasso and Ridge. Then the full dataset is randomly split into a training set and a test set, where 80% of the data is allocated for training.

```
mean_b <- sapply(Boston, mean)
sd_b <- sapply(Boston, sd)

stdize <- function(dfx, meanx=mean_b, sdx=sd_b){
    mean_mat <- matrix(rep(meanx, nrow(dfx)), nrow=nrow(dfx), byrow=TRUE)
    sd_mat <- matrix(rep(sdx, nrow(dfx)), nrow=nrow(dfx), byrow=TRUE)
    std_dfx <- (dfx - mean_mat)/sd_mat
    mtx <- as.matrix(std_dfx)
    # mtx <- mtx[, colSums(is.na(mtx))==0]
    return(mtx)
}

Boston_std <- stdize(Boston)
training_index <- sample(length(Boston), round(length(Boston)*0.8))
training_set <- Boston_std[training_index,]
test_set <- Boston_std[-training_index,]
```

Here plain linear regression is used as a baseline, and dimension reduction techniques including Lasso regression and PCA are employed to evaluate their effectiveness. Ridge and PLS are not selected because by principle they are very much similar to Lass and PCA, respectively.

The most straight forward way is to apply plain linear regression on the training set. The fitted linear model is then used to predict on the test set. The RMSE on the test set is printed at the end of the code.

```
lm_fit <- lm(crim~., data=as.data.frame(training_set))
prediction_lm <- predict(lm_fit, as.data.frame(test_set))
RMSE_lm <- mean(((prediction_lm-test_set[, 1])*sd_b[1])^2)^.5
cat("The plain linear regression method returns an RMSE on the test set of", RMSE_lm)
output>>> The plain linear regression method returns an RMSE on the test set of 8.575652
```

A more advanced way to build a robust prediction model is Lasso regression, and this is done with `cv.glmnet` which incorporates k-fold cross-validation. The prediction RMSE on the test set is also printed.

```
lasso_fit <- cv.glmnet(training_set[, -match("crim", colnames(training_set))],
                       y=training_set[, "crim"],
                       nfold=length(training_index),
                       alpha=1)

prediction_lasso <- predict(lasso_fit, test_set[, -1])
RMSE_lasso <- mean(((prediction_lasso - test_set[, 1])*sd_b[1])^2)^0.5
cat("Lasso regression returns an RMSE on the test set of", RMSE_lasso)
output>>> Lasso regression returns an RMSE on the test set of 9.364586
```

Principal Component regression is also tried here with the hope to improve prediction accuracy, with `ncomp` chose by cross-validation within `pcr` function.

```
training_set_4pcr <- as.data.frame(na.omit(training_set[, -4]))
pc_fit <- pcr(crim~., data=training_set_4pcr, scale=TRUE, validation="CV" )
prediction_pcr <- predict(pc_fit, na.omit(test_set[, -c(1, 4)]))
RMSE_pcr <- mean(((prediction_pcr - test_set[, 1])*sd_b[1])^2)^0.5
cat("Principal component regression returns an RMSE on the test set of", RMSE_pcr)
output>>> Principal component regression returns an RMSE on the test set of 8.955165
```

The result can be viewed from a relative perspective by dividing the RMSE by the standard deviation of `crim`. The RMSE are typically about 1 standard deviation of `crim`.

```
rel_RMSE <- sapply(c(RMSE_lm, RMSE_lasso, RMSE_pcr), function(x) x/sd_b[1])
method <- factor(c("lm", "Lasso", "PCA"))
rel_RMSE_df <- data.frame("Method"=method, "Rel_RMSE"=rel_RMSE)
print(rel_RMSE_df)
output>>>   Method  Rel_RMSE
output>>> 1     lm 0.9969897
output>>> 2  Lasso 1.0887097
output>>> 3    PCA 1.0411113
```

(b), (c)

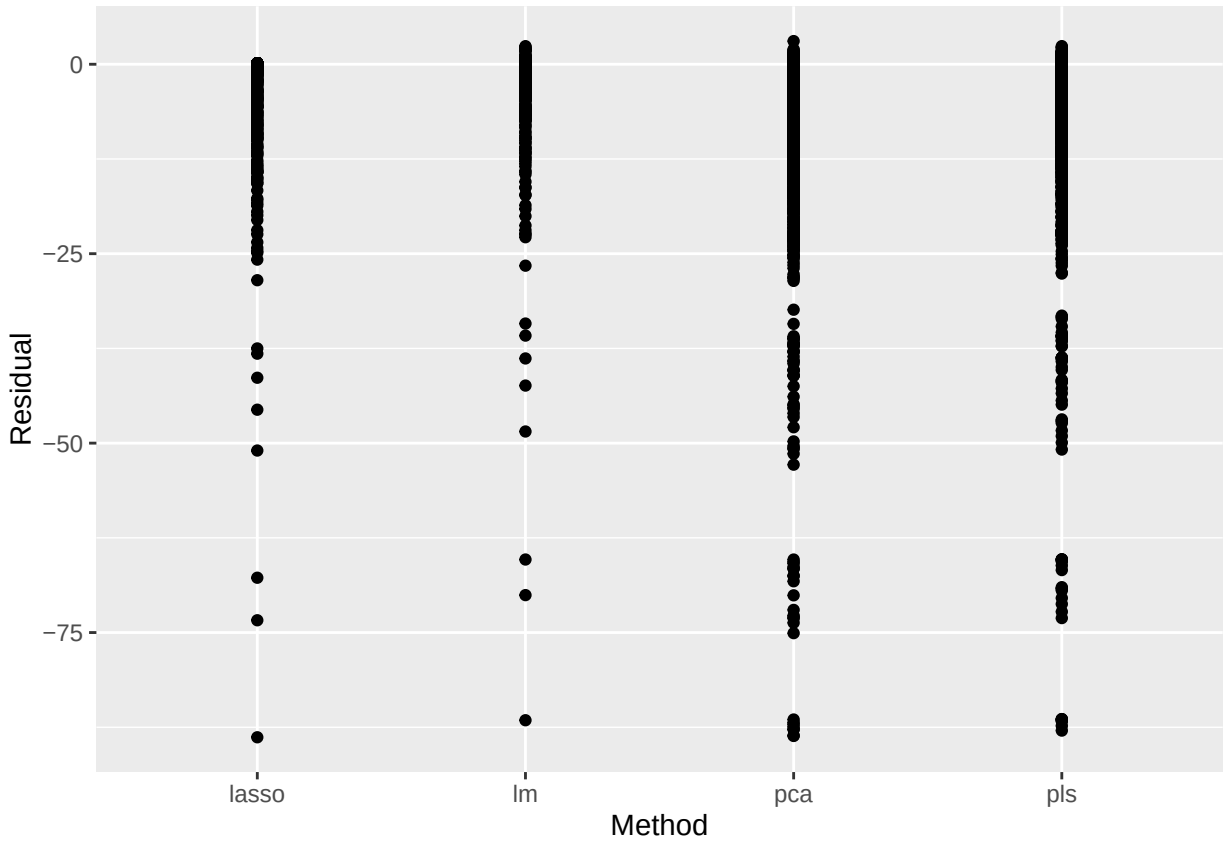
Comparing the test set RMSE of the three models, it is interesting that plain linear model reports smallest error. However, the difference between the 3 models in terms of RMSE is not large enough to warrant a complex model instead of a reduced one. Here we tentatively choose Partial Least Square regression, and compare its prediction capability with the previous 3 models.

First a PLS model is fitted on the same dataset as PCA. The `chas` column is dropped, because PCA and PLS require the dataset to be clean from missing values, and also, a constant column causes error while the function tries to assign a unique coefficient to it. Since `chas` variable is sparsely populated, it should be dropped as a result. Other than `chas`, other variables are all kept within the model, since PLS will automatically work out the contribution of each variable to `crim`, and by assigning small coefficients to less important variables, thus reducing the dimension in question.

```
pls_fit <- plsr(crim~., data=training_set_4pcr, scale=TRUE, validation="CV" )
prediction_pls <- predict(pls_fit, na.omit(test_set[, -c(1, 4)]))
RMSE_pls <- mean(((prediction_pls - test_set[, 1])*sd_b[1])^2)^0.5
cat("Partial least square regression returns an RMSE on the test set of", RMSE_pls)
output>>> Partial least square regression returns an RMSE on the test set of 8.589293
```

To intuitively compare the results from the four models, another points plot is made. A similar pattern to what was shown in Problem 9 appeared, with simple linear regression and Lasso regression exhibiting relatively concentrated and small residuals, which the PCA and PLS methods displaying larger error spread. Despite of this disadvantage, PLS is still useful when the data volumn is small compared with the number of features. The mechanism of PLS assists solving overfitting as usually caused by small $\frac{data}{feature}$ ratio.

```
library(ggplot2)
Residual <- c(prediction_lm, prediction_lasso, prediction_pcr, prediction_pls)

Method <- c(rep("lm", length(prediction_lm)),
            rep("lasso", length(prediction_lasso)),
            rep("pca", length(prediction_pcr)),
            rep("pls", length(prediction_pls)))

Method <- as.factor(Method)

residual_df <- data.frame(Residual, Method)

residual_df[, 1] <- (residual_df[, 1] - test_set[, 1])*sd_b[1]

qplot(x=Method, y=Residual, data=residual_df, geom="auto")
```


0
−25
−50
−75
Residual
lasso
lm
pca
pls
Method