Relative location of CT slices - MScA 32014

Ye Zhou

7/26/2017

I. Peek at Data

In this project, we aim to use four different regreesion methods (linear model, PCA method, Lasso method, and regression tree method) to predict the location of the slice using the features.

```
# Loading libraries
suppressWarnings(library ('glmnet'))
```

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

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-10
```

```
suppressWarnings(library ('rpart'))
suppressWarnings(library ('rpart.plot'))
suppressMessages(library('relaimpo'))

# Import data
dataPath <- "~/Documents/Lecture/MachineLearning/ML_PA_SM17_Yuri/HW/project"
data <- read.csv(paste(dataPath,"slice_localization_data.csv",sep="/"),header=T)
coln = colnames(data)
nrow = nrow(data)
ncol = ncol(data)
sprintf('The data contains %i columns and %i observations.', ncol, nrow)
```

```
## [1] "The data contains 386 columns and 53500 observations."
```

The data contain 384 features and 53,500 observations. Each record characterizes a slice of an image. Each CT slice is described by two histograms in polar space. The first histogram describes the location of bone structures in the image, the second the location of air inclusions inside of the body. Both histograms are concatenated to form the final feature vector of length 384. The class output variable is numeric and denotes the relative location of the CT slice on the axial axis of the human body.

The column information:

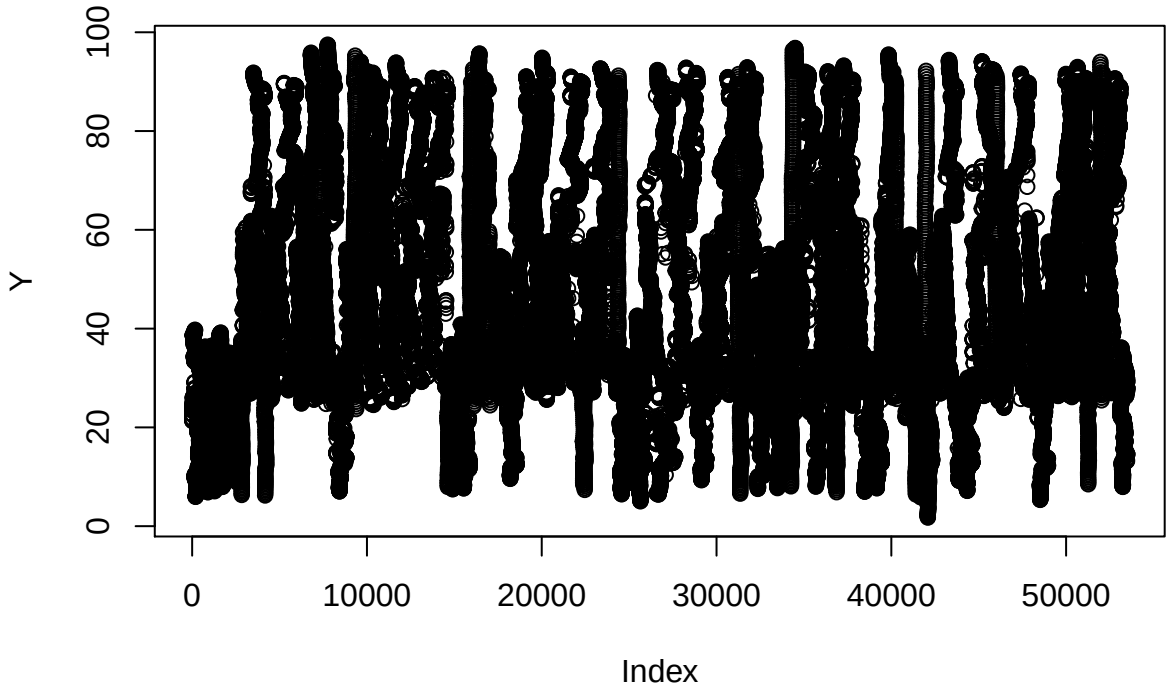
1: PatientId: Each ID identifies a different patient

2 - 241: Histogram describing bone structures

242 - 385: Histogram describing air inclusions

386: Reference: Relative location of the image on the axial axis (class value). Values are in the range [0; 180] where 0 denotes the top of the head and 180 the soles of the feet.

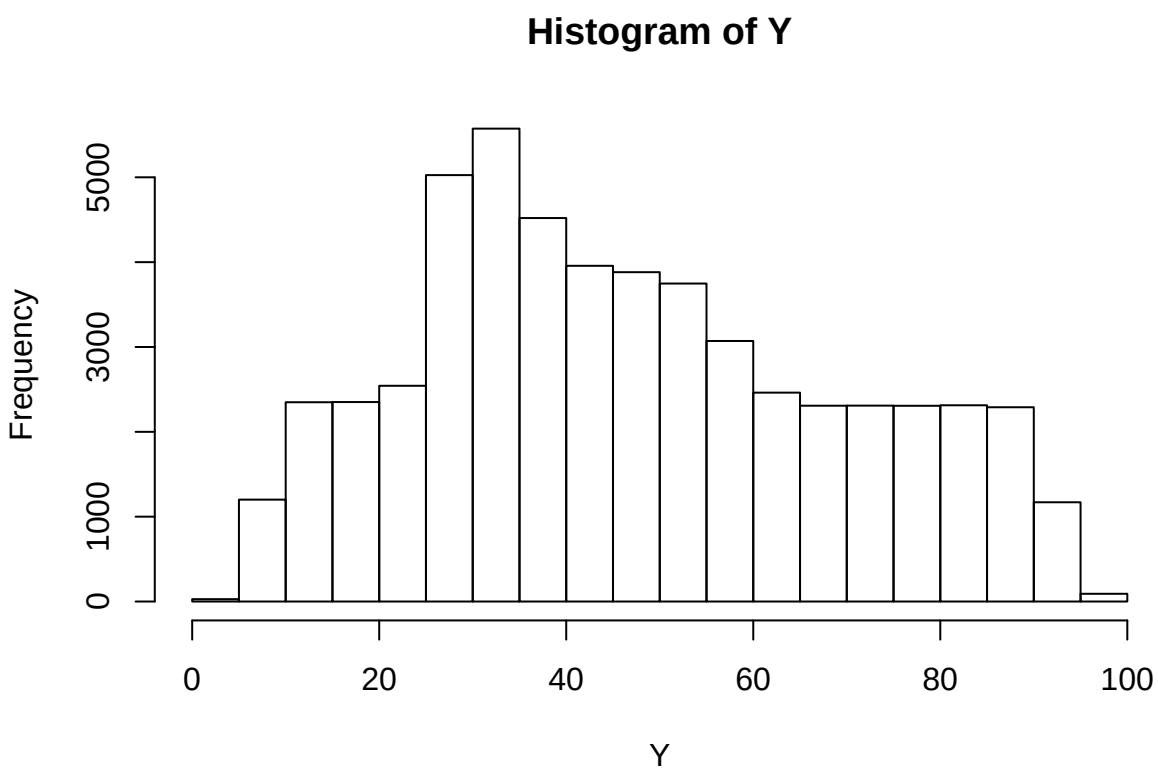
```
#define predictors and response
pred = data[,c(-1, -ncol)]
Y = data[, ncol]

# look at the distribution of Y
plot(Y)
```



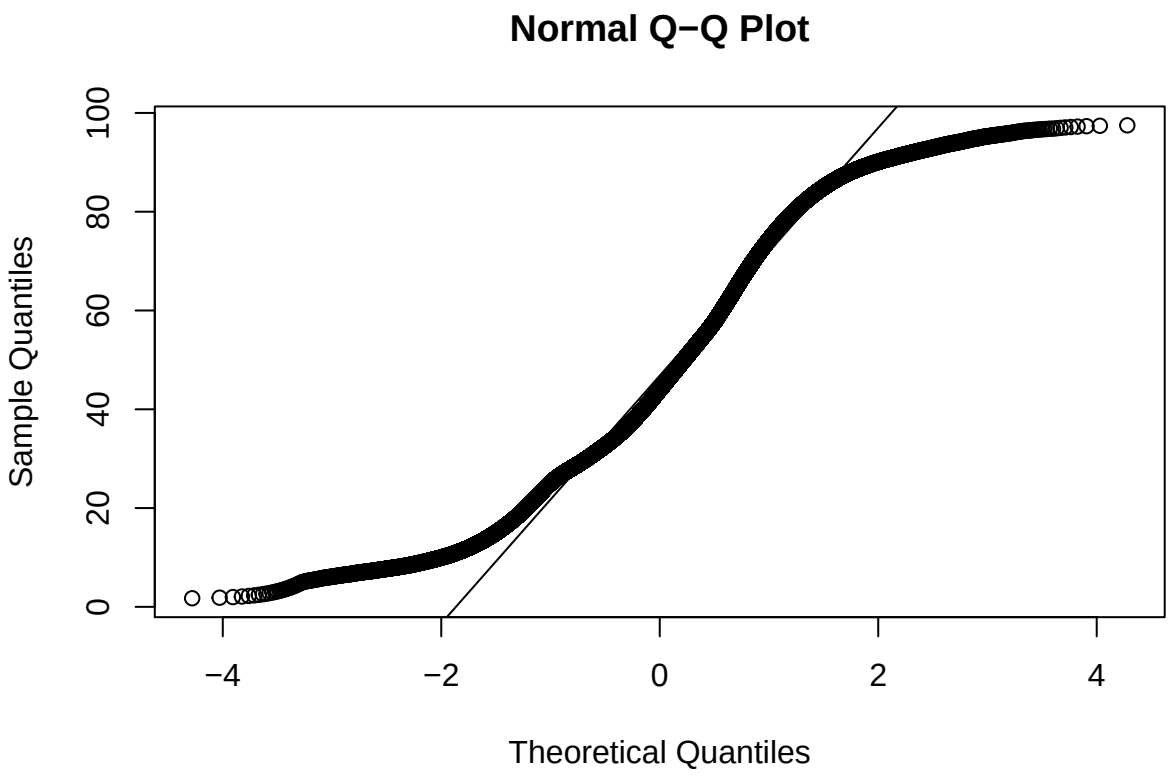

```
summary(Y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.739  29.890  43.990  47.030  63.740  97.490
```

```
hist(Y)
```




```
qqnorm(Y)
qqline(Y)
```




```
# Split data into 70% training and 30% testing set.
ntrain = round(nrow * 0.7)
sprintf('Training data size is %i.', ntrain)
```

```
## [1] "Training data size is 37450."
```

```
sprintf('Testing data size is %i.', nrow - ntrain)
```

```
## [1] "Testing data size is 16050."
```

```
set.seed(1)
train = sample(1:nrow, ntrain)

X_train = pred[train,]
X_test = pred[-train,]
Y_train = Y[train]
Y_test = Y[-train]
```

As shown above, the distribution of Y is rightly skewed, but the skewness is still tolerable and there is no need to perform data transformation here. In order to better examine the model performance, we randomly select the train sample and test sample in a ratio of 7:3 and both the mean squared error (MSE) for train sample and test sample will be reported for each method.

II. Linear Model

In this section, we apply a linear model to fit the data. We first fit the linear model with all predictors. Based on the regression result, we remove predictors insignificant with 5% level and those with NA as coefficients, then fit the reduced linear model.

```
# linear regression
newdata = data.frame(Y=Y_train, X_train)
```

```
lm.fit = lm(Y~., data=newdata)
# Select predictors significant with 5%
print('Summary for linear regression:')
```

```
## [1] "Summary for linear regression:"
```

```
res.lm0 = c(AIC=AIC(lm.fit),
            Rsq = summary(lm.fit)$r.squared,
            MSE_train = mean(lm.fit$residuals^2),
            #MSE_test = mean((Y_test - predict(lm.fit, X_test))^2),
            N_predictors = length(lm.fit$coefficients) - 1)
print(res.lm0)
```

```
##          AIC          Rsq    MSE_train N_predictors
## 2.648527e+05 8.645652e-01 6.765164e+01 3.840000e+02
```

```
coef_lm = coefficients(summary(lm.fit))
# linear regression with reduced data
lm.npred = coef_lm[,4] < 0.05
coef.na = which(is.na(lm.fit$coefficients))
reducedData = newdata[,-coef.na]
# alway keep y, no matter of the significance of intercept
lm.npred[1] = TRUE
reducedData = reducedData[,lm.npred]
lmReduced = lm(Y~., data=reducedData)
#Examine the residual of the fit
#hist(lmReduced$residuals)
#qqnorm(lmReduced$residuals)
#qqline(lmReduced$residuals)
y_pred = predict(lmReduced, X_test)

res.lm = c(AIC=AIC(lmReduced),
           Rsq = summary(lmReduced)$r.squared,
           MSE_train = mean(lmReduced$residuals^2),
           MSE_test = mean((Y_test - y_pred)^2),
           N_predictors = length(lmReduced$coefficients) - 1)
print(res.lm)
```

```
##          AIC          Rsq    MSE_train     MSE_test N_predictors
## 2.648382e+05 8.636964e-01 6.808564e+01 6.894445e+01 2.440000e+02
```

From the table above, 140 insignificant predicators with 5% level are removed in the reduced linear model. The squared R decreases, due to the decrease of predictor number. But AIC, as a metrics of adjusted error which penalize model with many predictors, decreases, indicating the model actually improves after the removal of the predictors.

III. PCA Method

In this section, we apply a PCA regression method to fit the data. After sorting the meta-features in the order of relative importances, we select the first 244 most important meta-features, for the purpose of direct comparison with the linear regression method in the previous section.

```
# Apply pca
PCA.comp = prcomp(X_train)
factorScores = PCA.comp$x
```

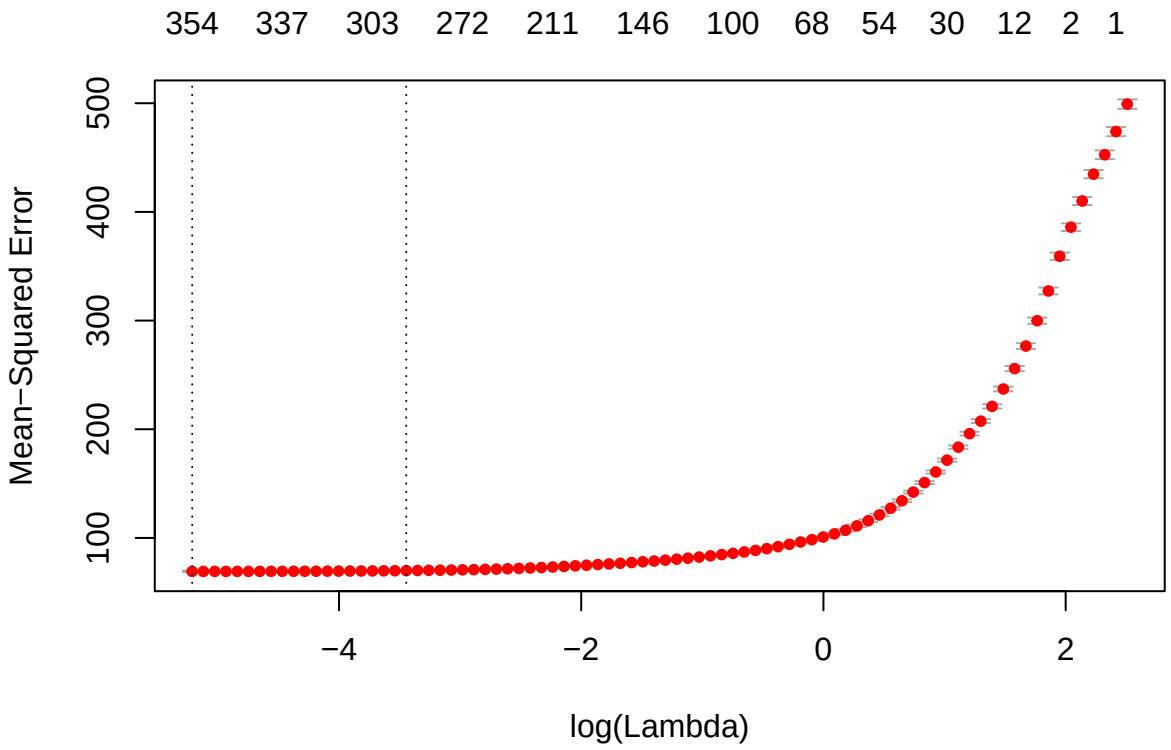
```
factorLoading = PCA.comp$rotation
pca.summary = summary(PCA.comp)

# Remove singular values
n_var = sum(pca.summary$importance[2,] > 0)
# Apply linear regresssion on metafeatures
PCA.fit<-lm(Y~.,data=data.frame(Y=Y_train, factorScores[,1:n_var]))
metrics.PCA <- calc.relimp(PCA.fit, type = "first")
first.PCA.rank<-metrics.PCA@first.rank

#Reorder the factors by the order of relative importance
orderedFactors<-factorScores[,order(first.PCA.rank)]
n_pred = 244
PCA.reduced = lm(Y~.,data=data.frame(Y=Y_train,orderedFactors[,1:n_pred]))
X_test_score = scale(as.matrix(X_test), PCA.comp$center, PCA.comp$scale) %*% factorLoading
X_test_score_red = X_test_score[,order(first.PCA.rank)][,1:n_pred]
y_pred = predict(PCA.reduced, newdata=as.data.frame(X_test_score_red))
res.pca = c(AIC=AIC(PCA.reduced),
            Rsq = summary(PCA.reduced)$r.squared,
            MSE_train = mean(PCA.reduced$residuals^2),
            MSE_test = mean((Y_test - y_pred)^2),
            N_predictors = length(PCA.reduced$coefficients)-1)
print(res.pca)
```

```
##          AIC          Rsq    MSE_train     MSE_test N_predictors
## 2.649258e+05 8.633772e-01 6.824510e+01 6.898930e+01 2.440000e+02
```

IV. Lasso Method

In this section we apply a Lasso regression method to fit the data. Using a cross-validation, we select the best lambda for the model, then fit the data using that value. The AIC of the Lasso mehtod is assigned to zero because it is not available.

```
# Use cross validation to find the optimal lambda for Lasso regression
cv.out=cv.glmnet(x=data.matrix(X_train),y=Y_train,alpha=1)
plot(cv.out)
```




```
bestlam =cv.out$lambda.min
# Fit the data with optimal lambda
out=glmnet(x=as.matrix(X_train),y=Y_train,alpha=1,lambda=bestlam)
y_pred_lasso = predict(out, newx=as.matrix(X_test))
res.lasso = c(AIC=0,
           Rsq = out$dev.ratio,
           MSE_train = deviance(out)/ntrain,
           MSE_test = mean((Y_test - y_pred_lasso)^2),
           n_predictors = out$df)
print(res.lasso)
```

```
##          AIC          Rsq    MSE_train     MSE_test n_predictors
##    0.0000000    0.8644236   67.7223858   68.4367852  359.0000000
```

V. Regression Tree Method

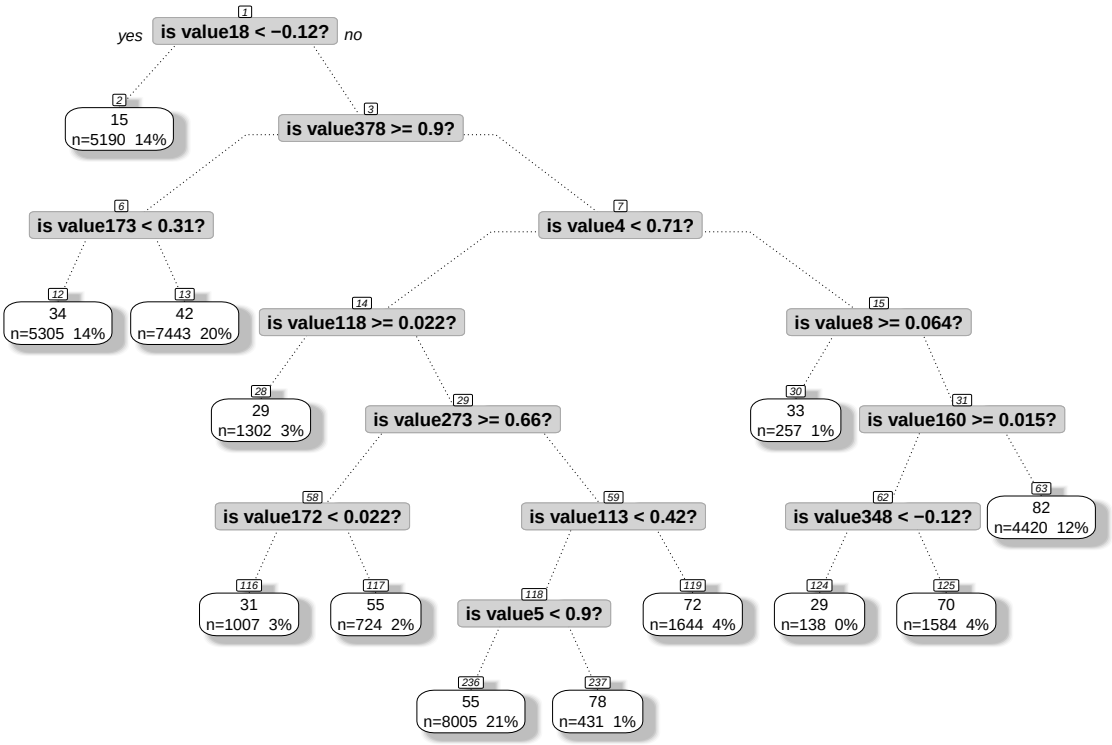
In this section, we apply a regression tree method to fit the data with 10-fold cross validation. The cross-validation error is minimized with the smallest cp in the table, which is 0.10. Therefore, we didn't further prune the tree.

```
set.seed(1)
tree = rpart(Y~., data=newdata, xval = 10)
print(tree$cptable)
```

```
##            CP nsplit rel error    xerror        xstd
## 1  0.32312355      0 1.0000000 1.0000804 0.005536056
## 2  0.19876935      1 0.6768764 0.6769543 0.004005943
## 3  0.11409133      2 0.4781071 0.4783041 0.003927638
## 4  0.04542094      3 0.3640158 0.3662619 0.003826068
## 5  0.02626099      4 0.3185948 0.3229277 0.003828989
```

```
## 6  0.02511566      5 0.2923338 0.2950700 0.003416353
## 7  0.01825308      6 0.2672182 0.2706889 0.003311641
## 8  0.01499512      7 0.2489651 0.2532064 0.003288056
## 9  0.01349578      8 0.2339700 0.2386422 0.003142731
## 10 0.01147265      9 0.2204742 0.2177213 0.003090329
## 11 0.01146998     10 0.2090016 0.2047768 0.002978660
## 12 0.01128015     11 0.1975316 0.2020493 0.002973941
## 13 0.01000000     12 0.1862514 0.1910394 0.002973941
```

```
prp(tree, extra=101, # display the number of observations that fall in the node
    branch=.5, # change angle of branch lines
    shadow.col="gray", # shadows under the leaves
    branch.lty=3, # draw branches using dotted lines
    split.cex=1.2, # make the split text larger than the node text
    split.prefix="is ", # put "is " before split text
    split.suffix="?", # put "?" after split text
    split.box.col="lightgray", # lightgray split boxes (default is white)
    split.border.col="darkgray", # darkgray border on split boxes
    split.round=.5,
    nn=TRUE)
```




```
tree.pred = predict(tree, X_test)
tree.Rsq = 1 - sum(resid(tree)^2) /sum((Y_train - mean(Y_train))^2)
res.tree = c( AIC = 0,
              Rsq = tree.Rsq,
              MSE_train = mean(resid(tree)^2),
              MSE_test = mean((tree.pred - Y_test)^2),
              n_predictors = 12)
print(res.tree)
```

```
##          AIC          Rsq    MSE_train     MSE_test n_predictors
```

```
##    0.0000000    0.8137486   93.0353045   94.8904465   12.0000000
```

VI. Conclusion

```
res = rbind(lm = res.lm, pca = res.pca, lasso = res.lasso, tree= res.tree)
print(res)
```

```
##            AIC       Rsq MSE_train MSE_test N_predictors
## lm    264838.2 0.8636964  68.08564 68.94445          244
## pca   264925.8 0.8633772  68.24510 68.98930          244
## lasso      0.0 0.8644236  67.72239 68.43679          359
## tree       0.0 0.8137486  93.03530 94.89045           12
```

In conclusion, the performance of four different regression methods is listed in the table above. Even though regression tree method has the lowest coefficient of determinaiton (R) and highest MSE in both training set and testing set, it only uses 12 predictors, which is much smaller than other three methods. Taking consideration of predictor number used, a R score of 0.814 for regression tree method is relatively good. The linear regression method with 5% level significant features slightly outperforms PCA regression method with the same predictor number. It reveals the drawback of PCA regression method, there is no guarantee that the direction best explain the predictors would also be the best direction for the use to predict the response. In the end, in this example, Lasso method has the best performance, but with the most predictor numbers.