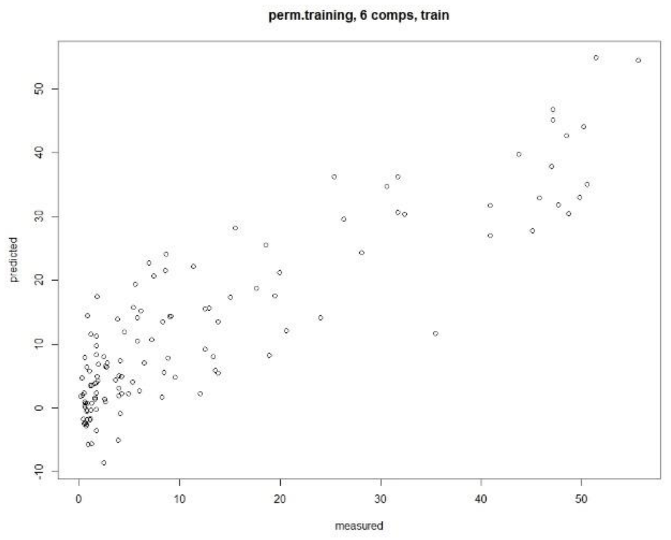
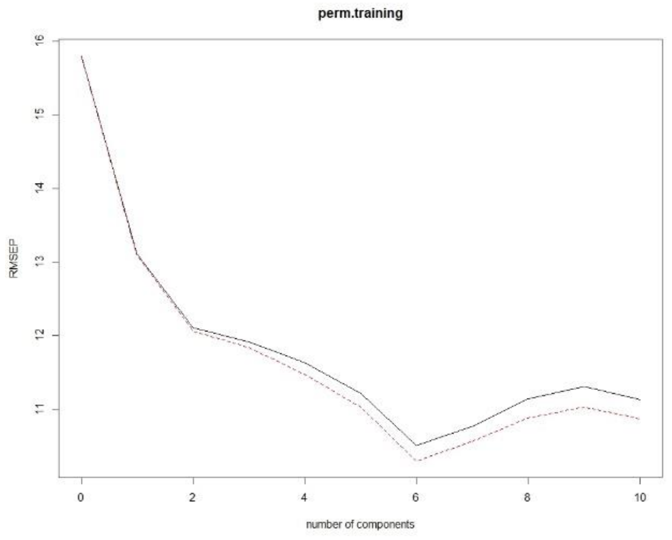
```
#==================================================================================
#                          Using R: Predicting permeability
#==================================================================================
#Developing a model to predict permeability could save significant resources for a
pharmaceutical company, while at the same time more rapidly identifying molecules that
have a sufficient permeability to become a drug:

#This pharmaceutical data set was used to develop a model for predicting compounds'
permeability. In short, permeability is the measure of a molecule's ability to cross a
membrane. The body, for example, has notable membranes between the body and brain, known
as the blood-brain barrier, and between the gut and body in the intestines. These
membranes help the bodyguard critical regions from receiving undesirable or detrimental
substances. For an orally taken drug to be effective in the brain, it first must pass
through the intestinal wall and then must pass through the blood-brain barrier in order
to be present for the desired neurological target. Therefore, a compound's ability to
permeate relevant biological membranes is critically important to understand early in the
drug discovery process. Compounds that appear to be effective for a particular disease in
research screening experiments but appear to be poorly permeable may need to be altered
in order to improve permeability and thus the compound's ability to reach the desired
target. Identifying permeability problems can help guide chemists towards better
molecules.
Permeability assays such as PAMPA and Caco-2 have been developed to help measure
compounds' permeability (Kansy et al. 1998). These screens are effective at quantifying a
compound's permeability, but the assay is expensive labor intensive. Given a sufficient
number of compounds that have been screened, we could develop a predictive model for
permeability in an attempt to potentially reduce the need for the assay. In this project
there were 165 unique compounds; 1,107 molecular fingerprints were determined for each. A
molecular fingerprint is a binary sequence of numbers that represents the presence or
absence of a specific molecular substructure. The response is highly skewed, the
predictors are sparse (15.5% are present), and many predictors are strongly associated.

#----------------------------------------------------------------------------------
#(a) load the data:
library(AppliedPredictiveModeling)
data(permeability)
#The matrix fingerprints contain 1,107 binary molecular predictors for the 165 Compounds,
while permeability contains permeability response. In the following part we perform pre-
process on the data. The number of observations is much smaller than the number of
predictors and they may going to cause us problem.

#----------------------------------------------------------------------------------
#(b) The fingerprint predictors indicate the presence or absence of substructures of a
molecule and are often sparse, meaning that relatively few of the molecules contain each
substructure. Using nearZeroVar function we remove the predictors who have almost zero or
near zero variance. They may go to cause us some problem if we do not remove them. After
implementing the function 388 out of the original 1107 predictors are left.
library(caret)
nzv <- nearZeroVar(fingerprints, saveMetrics= TRUE)
nzv[nzv$nzv,]
dim(fingerprints)
nzv <- nearZeroVar(fingerprints)
fil.finger <- fingerprints[, -nzv]
dim(fil.finger)
```

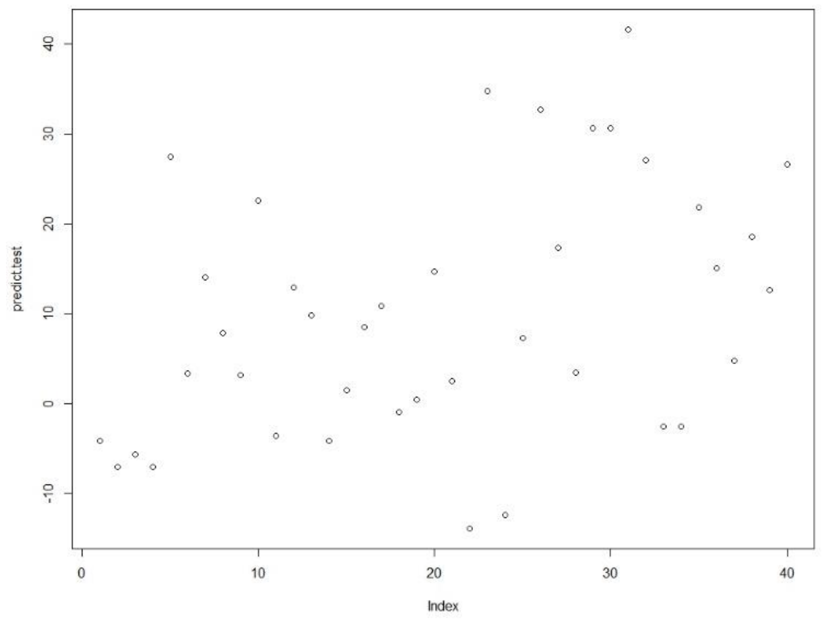
```
#-------------------------------------------------------------------------------------
#(c) Splitting the data into a training and a test set, pre-processing the data, and
tunning a PLS model.

# Now, we use findLinearCombos function to find if there is any linear combination
between predictors.
comboInfo <- findLinearCombos(fil.finger)
fil.finger <- fil.finger[, -comboInfo$remove]
dim(fil.finger)
#After removing linear correlation 106 predictors will be remain.

#Here, we use Simple Splitting. As long as we do not need validating set, we split data
in two parts of training and test. We assign the weight for training in the way that we
have enough observation and the numbers are larger than predictors. So, about 75% for
training and 25% for test would be used here.

mean(is.na(fil.finger))
#There is no missing data.

#Spliting Data
set.seed(100)
tra <- createDataPartition(permeability, p=.75, list = FALSE, times = 1)
head(tra)
finger.training <- data.frame(fil.finger[tra,])
finger.test <- data.frame(fil.finger[-tra,])
perm.training <- permeability[tra,]
perm.test <- permeability[-tra,]

#Assessing Correlation
correlation <- round(cor(cbind(perm.training,finger.training)),3)
highCorr <- sum(abs(correlation[upper.tri(correlation)]) > 0.999)
highCorr
#As can be seen after removing low variance and linear dependent predictors, there are
only 2 descriptors that are almost perfectly correlated (|correlation| > 0.999). Here, we
are going to use the PLS, this method can thrive the correlation.

#Partial Least Squares
library(pls)
fit.pls <- plsr(perm.training ~ ., 10, data=finger.training, method="oscorespls",
validation="CV")
plot(fit.pls)
summary(fit.pls)
dev.off()
plot(RMSEP(fit.pls))
#It seems using only 7 components is enough
ncom <- 7
fit.pls <- plsr(perm.training ~ ., ncom, data=finger.training, method="oscorespls")
summary(fit.pls)
#R2 (R Square)
SStot <- sum((perm.training-mean(perm.training))^2)
SSres <- sum((perm.training-drop(predict(fit.pls, finger.training, ncom)))^2)
Rsqu.pls <- 1-(SSres/SStot)
resid.pls <- drop(fit.pls$resid)[,ncom]
MSE.pls <- mean(resid.pls^2)
```

```
#As can be seen from the figure below-right, 6 components are recommended to choose by
using cross validation method. In the figure below-left the predicted versus the measured
response is plotted. This fitting has MSE=54.413 and R2=0.779.
```

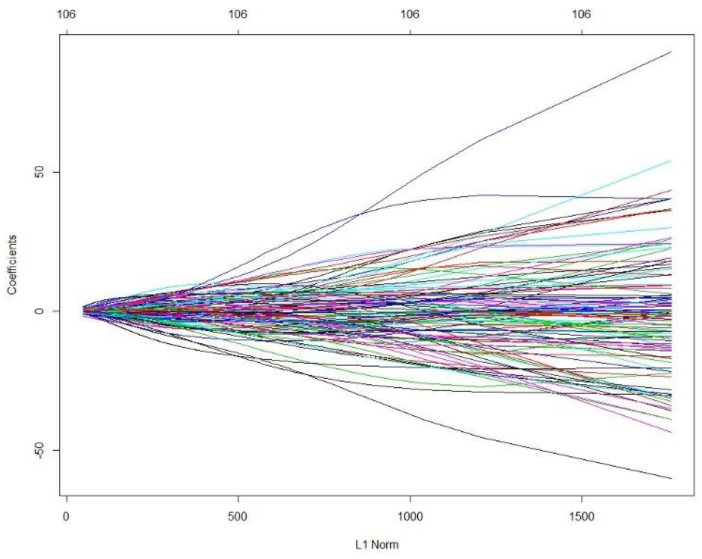


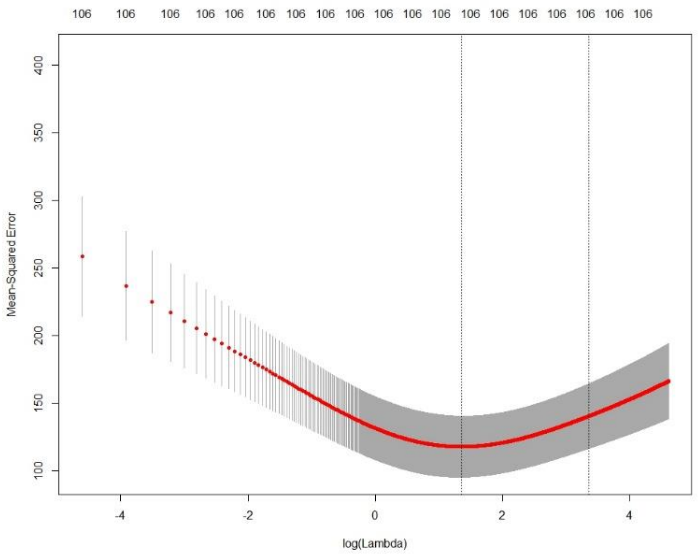


```
#------------------------------------------------------------------------------------
#(d) Predict the response for the test set. What is the test set estimate of R2?
predict.test <- drop(predict(fit.pls, finger.test, ncom))
#R2 (R Square)
SStot <- sum((perm.test-mean(perm.test))^2)
SSres <- sum((perm.test-drop(predict(fit.pls, finger.test, ncom)))^2)
Rsqu.pls.test <- 1-(SSres/SStot)
resid.pls <- perm.test-drop(predict(fit.pls, finger.test, ncom))
test.MSE.pls <- mean(resid.pls^2)

#As can be seen from the following graph, it does not have a very good and consistent
prediction in the test data. And its MSE and R2 have suffered compared to training data.
MSE=133.507
R2=0.411
```

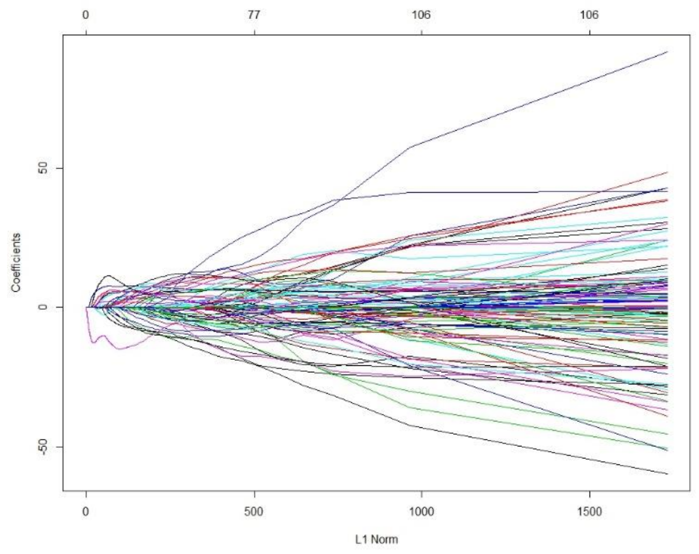


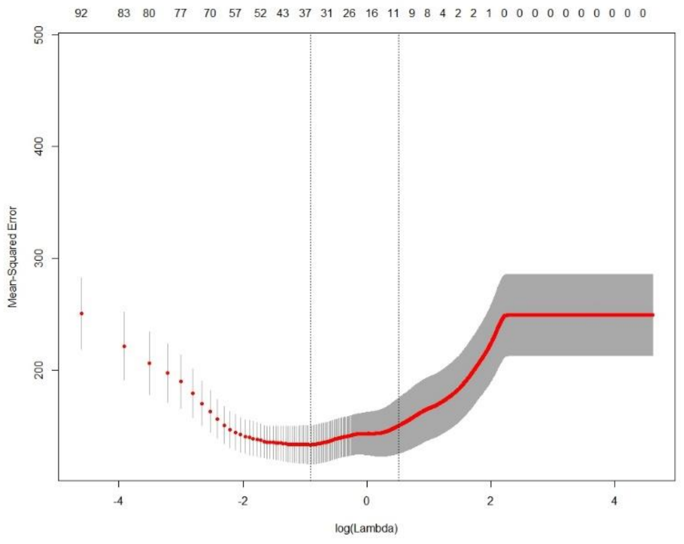
```r
#--------------------------------------------------------------------------------------
#(e) Here we are going to build ridge and lasso regressions and compare it with pls
calculated before.

#Ridge Regression
#we use glmnet with alpha equal to 0 to perform ridge regression
library(glmnet)
fit.ridge <- glmnet(as.matrix(finger.training), perm.training, alpha=0,
lambda=seq(0,100,0.01))
plot(fit.ridge)
cv.out <- cv.glmnet(as.matrix(finger.training), perm.training, alpha=0,
lambda=seq(0,100,0.01))
plot(cv.out)
lambda.ridge <- cv.out$lambda.min
fit.ridge <- glmnet(as.matrix(finger.training), perm.training, alpha=0,
lambda=lambda.ridge)
SStot <- sum((perm.training-mean(perm.training))^2)
SSres <- sum((perm.training-predict(fit.ridge, s=lambda.ridge
,newx=as.matrix(finger.training)))^2)
Rsqu.ridge <- 1-(SSres/SStot)
resid.ridge <- perm.training-predict(fit.ridge, s=lambda.ridge ,
newx=as.matrix(finger.training))
MSE.ridge <- mean(resid.ridge^2)

#test data
SStot <- sum((perm.test-mean(perm.test))^2)
SSres <- sum((perm.test-predict(fit.ridge, s=lambda.ridge
,newx=as.matrix(finger.test)))^2)
Rsqu.ridge.test <- 1-(SSres/SStot)
resid.ridge <- perm.test-predict(fit.ridge, s=lambda.ridge , newx=
as.matrix(finger.test))
test.MSE.ridge <- mean(resid.ridge^2)

#Ridge regression shrinkage coefficients and tuning parameter graphs are shown below. For
the training data it has MSE=47.224 and R2=0.808. For the test data they have MSE=122.407
and R2=0.460. It has almost less error in both the training and test data compared to
PLS.
```





```
#LASSO Regression by help of glmnet with alpha equal to 1
fit.lasso <- glmnet(as.matrix(finger.training), perm.training, alpha=1,
lambda=seq(0,100,0.01))
plot(fit.lasso)
cv.out <- cv.glmnet(as.matrix(finger.training), perm.training, alpha=1,
lambda=seq(0,100,0.01))
plot(cv.out)
lambda.lasso <- cv.out$lambda.min
fit.lasso <- glmnet(as.matrix(finger.training), perm.training, alpha=1,
lambda=lambda.lasso)
resid.lasso <- perm.training - predict(fit.lasso, s=lambda.lasso ,
newx=as.matrix(finger.training))
MSE.lasso <- mean(resid.lasso^2)
SStot <- sum((perm.training-mean(perm.training))^2)
SSres <- sum((perm.training-predict(fit.lasso, s=lambda.lasso ,
newx=as.matrix(finger.training)))^2)
Rsqu.lasso <- 1-(SSres/SStot)
resid.lasso <- perm.training-predict(fit.lasso, s=lambda.lasso ,
newx=as.matrix(finger.training))
MSE.lasso <- mean(resid.lasso^2)

#test data
SStot <- sum((perm.test-mean(perm.test))^2)
SSres <- sum((perm.test-predict(fit.lasso, s=lambda.lasso
,newx=as.matrix(finger.test)))^2)
Rsqu.lasso.test <- 1-(SSres/SStot)
resid.lasso <- perm.test-predict(fit.lasso, s=lambda.lasso , newx=as.matrix(finger.test))
test.MSE.lasso <- mean(resid.lasso^2)

#LASSO regression shrinkage coefficients and tuning parameter graphs are shown below. For
the training data it has MSE=50.072 and R2=0.760. For the test data it has MSE=129.144
and R2=0.431. The errors are so similar to the ridge regression and both of these methods
are slightly better than the PLS. However, these three methods have overall the same
error rate.
```





```
#-------------------------------------------------------------------------------------
#(f) Among the assessed method, LASSO has the simplest model. At the same time, it has
almost the same accuracy. So, this method is preferred. I believe this model can be
improved much more. Because the error on the test data is not satisfactory.
For the purpose of substituting any of this model by the permeability laboratory
experiment, I think it is possible to do so. However, a more rigorous cost analyses
should be performed to calculate the associated cost of type I and type II errors
associated with replacing this model with the experiment.
```