STA9750_Project_Code

Soonmo Seong

12/15/2019

Project Code

```r
library(dplyr)
library(MASS)
library(glmnet)
library(ggplot2)
library(reshape)
library(gridExtra)
library(xtable)

a = read.csv("pa.csv")
dim(a)
t.data = a[,-(1:2)]

# standardize the dataset
t.data[,-(1:2)] = scale(t.data[,-(1:2)])
t.data = t.data[,-2]
colMeans(t.data)
apply(t.data, 2, sd)
mean(t.data$Status)
attach(t.data)

# data structure and rates
n1 = dim(t.data %>% filter(Status == 1))[1]
n0 = dim(t.data %>% filter(Status == 0))[1]
n = n1 + n0
p = dim(t.data)[2]-1

#######################
#######################
# Modelling factors
iterations = 100
Dlearn_rate = 0.8
sampling.rate = 1

# train, test, and cv error rate matrix
train_error = matrix(0, nrow = iterations, ncol = 3)
colnames(train_error) = c("Log", "Log-LASSO","Log-Ridge")

cv_error = matrix(0, nrow = iterations, ncol = 2)
colnames(cv_error) = c("Log-LASSO","Log-Ridge")

test_error = matrix(0, nrow = iterations, ncol = 3)
colnames(test_error) = c("Log", "Log-LASSO","Log-Ridge")

cv_error_nfold = matrix(0, nrow = iterations, ncol = 30)
```

```r
colnames(cv_error_nfold) = c("3_Log-LASSO",
                             "4_Log-LASSO","5_Log-LASSO",
                             "6_Log-LASSO","7_Log-LASSO",
                             "8_Log-LASSO","9_Log-LASSO",
                             "10_Log-LASSO",
                             "15_Log-LASSO","20_Log-LASSO",
                             "25_Log-LASSO","30_Log-LASSO",
                             "40_Log-LASSO","50_Log-LASSO",
                             "LOO_Log-LASSO",
                             "3_Log-Ridge",
                             "4_Log-Ridge","5_Log-Ridge",
                             "6_Log-Ridge","7_Log-Ridge",
                             "8_Log-Ridge","9_Log-Ridge",
                             "10_Log-Ridge",
                             "15_Log-Ridge","20_Log-Ridge",
                             "25_Log-Ridge","30_Log-Ridge",
                             "40_Log-Ridge","50_Log-Ridge",
                             "LOO_Log-Ridge")

lasso.coef = matrix(0, ncol = iterations, nrow = p+1)
ridge.coef = matrix(0, ncol = iterations, nrow = p+1)

# convert to data frame
train_error = data.frame(train_error)
test_error = data.frame(test_error)
cv_error = data.frame(cv_error)
cv_error_nfold = data.frame(cv_error_nfold)

# time of cv and fit
time.cv = matrix(0, nrow = iterations, ncol = 2)
colnames(time.cv) = c("LASSO", "Ridge")

time.cv.nfold = matrix(0, nrow = iterations, ncol = 30)
colnames(time.cv.nfold) = c("3_Log-LASSO",
                            "4_Log-LASSO","5_Log-LASSO",
                            "6_Log-LASSO","7_Log-LASSO",
                            "8_Log-LASSO","9_Log-LASSO",
                            "10_Log-LASSO",
                            "15_Log-LASSO","20_Log-LASSO",
                            "25_Log-LASSO","30_Log-LASSO",
                            "40_Log-LASSO","50_Log-LASSO",
                            "LOO_Log-LASSO",
                            "3_Log-Ridge",
                            "4_Log-Ridge","5_Log-Ridge",
                            "6_Log-Ridge","7_Log-Ridge",
                            "8_Log-Ridge","9_Log-Ridge",
                            "10_Log-Ridge",
                            "15_Log-Ridge","20_Log-Ridge",
                            "25_Log-Ridge","30_Log-Ridge",
                            "40_Log-Ridge","50_Log-Ridge",
                            "LOO_Log-Ridge")

time.fit = matrix(0, nrow = iterations, ncol = 3)
```

```r
colnames(time.fit) = c("Log", "LASSO", "Ridge")

# sampling from t.data(ntrain, ntest)
sampling = sample(n,n*sampling.rate)
sampling.data = data.frame(t.data[sampling,])
sampling.n = dim(sampling.data)[1]

# preparation for lasso and ridge
X = model.matrix(Status ~., sampling.data)[,-1]
y = sampling.data$Status


# 100 iteration for training error rate and testing error rate
for(m in 41:iterations){


  # create a training data vector for dividing the data set.
  train = sample(sampling.n, sampling.n*Dlearn_rate)

  dat   = data.frame(sampling.data[train,])
  datt  = data.frame(sampling.data[-train,])

  # logistic regression
  log.mod = glm(Status ~., data = dat, family = "binomial")
  log.pred = predict(log.mod, newdata = sampling.data[train,], type = "response")
  log.pred = ifelse(log.pred > 0.5, 1, 0)
  train_error[m,1] = mean( sampling.data[train,1]!= log.pred)
  log.pred = predict(log.mod, newdata = sampling.data[-train,], type = "response")
  log.pred = ifelse(log.pred > 0.5, 1, 0)
  test_error[m,1] = mean( sampling.data[-train,1]!= log.pred)

  # lasso cross validation and tune lambda
  # record lasso cv time and error rate
  ptm = proc.time()
  cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                       intercept = T, type.measure="class")
  ptm = proc.time() - ptm
  time.cv[m,1]  = ptm["elapsed"]

  cv_error[m,1] = min(cv.lasso$cvm)
  bestlam = cv.lasso$lambda.min
  # record lasso fit time
  ptm = proc.time()
  lasso.mod = glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, lambda = bestlam,
                     standardize = F)
  ptm = proc.time() - ptm
  time.fit[m,2]  = ptm["elapsed"]

  lasso.coef[,m] = coef(lasso.mod)[,1]
  lasso.pred = predict(lasso.mod, s = bestlam, newx = X[train,], type ="class")
  train_error[m,2] = mean(y[train] != lasso.pred)
  lasso.pred = predict(lasso.mod, s = bestlam, newx = X[-train,], type="class",)
```

```
test_error[m,2] = mean(y[-train] != lasso.pred)

# record lasso cv time and error rate by nfolds
# 3fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 3)
ptm = proc.time() - ptm
time.cv.nfold[m,1]  = ptm["elapsed"]

cv_error_nfold[m,1] = min(cv.lasso$cvm)
# 4fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 4)
ptm = proc.time() - ptm
time.cv.nfold[m,2]  = ptm["elapsed"]

cv_error_nfold[m,2] = min(cv.lasso$cvm)
# 5fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 5)
ptm = proc.time() - ptm
time.cv.nfold[m,3]  = ptm["elapsed"]

cv_error_nfold[m,3] = min(cv.lasso$cvm)
# 6fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 6)
ptm = proc.time() - ptm
time.cv.nfold[m,4]  = ptm["elapsed"]

cv_error_nfold[m,4] = min(cv.lasso$cvm)
# 7fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 7)
ptm = proc.time() - ptm
time.cv.nfold[m,5]  = ptm["elapsed"]

cv_error_nfold[m,5] = min(cv.lasso$cvm)
# 8fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 8)
ptm = proc.time() - ptm
time.cv.nfold[m,6]  = ptm["elapsed"]

cv_error_nfold[m,6] = min(cv.lasso$cvm)
# 9fold cv lasso
ptm = proc.time()
```

```r
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 9)
ptm = proc.time() - ptm
time.cv.nfold[m,7]  = ptm["elapsed"]

cv_error_nfold[m,7] = min(cv.lasso$cvm)
# 10fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 10)
ptm = proc.time() - ptm
time.cv.nfold[m,8]  = ptm["elapsed"]

cv_error_nfold[m,8] = min(cv.lasso$cvm)
# 15fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 15)
ptm = proc.time() - ptm
time.cv.nfold[m,9]  = ptm["elapsed"]

cv_error_nfold[m,9] = min(cv.lasso$cvm)
# 20fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 20)
ptm = proc.time() - ptm
time.cv.nfold[m,10]  = ptm["elapsed"]

cv_error_nfold[m,10] = min(cv.lasso$cvm)
# 25fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 25)
ptm = proc.time() - ptm
time.cv.nfold[m,11]  = ptm["elapsed"]

cv_error_nfold[m,11] = min(cv.lasso$cvm)
# 30fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 30)
ptm = proc.time() - ptm
time.cv.nfold[m,12]  = ptm["elapsed"]

cv_error_nfold[m,12] = min(cv.lasso$cvm)
# 40fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 40)
ptm = proc.time() - ptm
time.cv.nfold[m,13]  = ptm["elapsed"]
```

```r
cv_error_nfold[m,13] = min(cv.lasso$cvm)
# 50fold cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 50)
ptm = proc.time() - ptm
time.cv.nfold[m,14]  = ptm["elapsed"]
cv_error_nfold[m,14] = min(cv.lasso$cvm)
# LOO cv lasso
ptm = proc.time()
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = length(train))
ptm = proc.time() - ptm
time.cv.nfold[m,15]  = ptm["elapsed"]

cv_error_nfold[m,15] = min(cv.lasso$cvm)

# ridge cross validation and tune lambda
# record ridge cv time
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class")
ptm = proc.time() - ptm
time.cv[m,2]  = ptm["elapsed"]

cv_error[m,2] = min(cv.ridge$cvm)
bestlam = cv.ridge$lambda.min

# record ridge fit time
ptm = proc.time()
ridge.mod = glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                   intercept = T, lambda = bestlam,
                   standardize = F)
ptm = proc.time() - ptm
time.fit[m,3]  = ptm["elapsed"]

ridge.coef[,m] = as.matrix(coef(ridge.mod))
ridge.pred = predict(ridge.mod, s = bestlam, newx = X[train,], type = "class")
train_error[m,3] = mean(y[train] != ridge.pred)
ridge.pred = predict(ridge.mod, s = bestlam, newx = X[-train,], type = "class")
test_error[m,3] = mean(y[-train] != ridge.pred)

# record ridge cv time and error rate by nfolds
# 3fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 3)
ptm = proc.time() - ptm
time.cv.nfold[m,16]  = ptm["elapsed"]

cv_error_nfold[m,16] = min(cv.ridge$cvm)
# 4fold cv ridge
ptm = proc.time()
```

```r
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 4)
ptm = proc.time() - ptm
time.cv.nfold[m,17]  = ptm["elapsed"]

cv_error_nfold[m,17] = min(cv.ridge$cvm)
# 5fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 5)
ptm = proc.time() - ptm
time.cv.nfold[m,18]  = ptm["elapsed"]

cv_error_nfold[m,18] = min(cv.ridge$cvm)
# 6fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 6)
ptm = proc.time() - ptm
time.cv.nfold[m,19]  = ptm["elapsed"]

cv_error_nfold[m,19] = min(cv.ridge$cvm)
# 7fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 7)
ptm = proc.time() - ptm
time.cv.nfold[m,20]  = ptm["elapsed"]

cv_error_nfold[m,20] = min(cv.ridge$cvm)
# 8fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 8)
ptm = proc.time() - ptm
time.cv.nfold[m,21]  = ptm["elapsed"]

cv_error_nfold[m,21] = min(cv.ridge$cvm)
# 9fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 9)
ptm = proc.time() - ptm
time.cv.nfold[m,22]  = ptm["elapsed"]

cv_error_nfold[m,22] = min(cv.ridge$cvm)
# 10fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 10)
ptm = proc.time() - ptm
time.cv.nfold[m,23]  = ptm["elapsed"]
```

```
cv_error_nfold[m,23] = min(cv.ridge$cvm)
# 15fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 15)
ptm = proc.time() - ptm
time.cv.nfold[m,24]  = ptm["elapsed"]

cv_error_nfold[m,24] = min(cv.ridge$cvm)
# 20fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 20)
ptm = proc.time() - ptm
time.cv.nfold[m,25]  = ptm["elapsed"]

cv_error_nfold[m,25] = min(cv.ridge$cvm)
# 25fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 25)
ptm = proc.time() - ptm
time.cv.nfold[m,26]  = ptm["elapsed"]

cv_error_nfold[m,26] = min(cv.ridge$cvm)
# 30fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 30)
ptm = proc.time() - ptm
time.cv.nfold[m,27]  = ptm["elapsed"]

cv_error_nfold[m,27] = min(cv.ridge$cvm)
# 40fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 40)
ptm = proc.time() - ptm
time.cv.nfold[m,28]  = ptm["elapsed"]

cv_error_nfold[m,28] = min(cv.ridge$cvm)
# 50fold cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 50)
ptm = proc.time() - ptm
time.cv.nfold[m,29]  = ptm["elapsed"]

cv_error_nfold[m,29] = min(cv.ridge$cvm)
# LOO cv ridge
ptm = proc.time()
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = length(train))
```

```
  ptm = proc.time() - ptm
  time.cv.nfold[m,30]  = ptm["elapsed"]

  cv_error_nfold[m,30] = min(cv.ridge$cvm)
  }

# tune lambda for lasso and ridge with optimal k-fold cross validation
# create a training data vector for dividing the data set.
train = sample(sampling.n, sampling.n*Dlearn_rate)

dat   = data.frame(sampling.data[train,])
datt  = data.frame(sampling.data[-train,])

# lasso 25 fold cross validation and tune lambda
cv.lasso = cv.glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 25)
# ridge 10 fold cross validation and tune lambda
cv.ridge = cv.glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, type.measure="class", nfolds = 10)
lasso.cv.error = data.frame(lambda = cv.lasso$lambda, error = cv.lasso$cvm)
ridge.cv.error = data.frame(lambda = cv.ridge$lambda, error = cv.ridge$cvm)

# train and test error with best lambda
tune_train_error = matrix(0, nrow = iterations, ncol = 3)
colnames(tune_train_error) = c("Log", "Log-LASSO","Log-Ridge")

tune_test_error = matrix(0, nrow = iterations, ncol = 3)
colnames(tune_test_error) = c("Log", "Log-LASSO","Log-Ridge")

tune_train_error = data.frame(train_error)
tune_test_error = data.frame(test_error)

tune_lasso.coef = matrix(0, nrow = iterations, ncol = p+1)
tune_ridge.coef = matrix(0, nrow = iterations, ncol = p+1)

# 100 iteration with best lambdas
for(m in 1:iterations){


  # create a training data vector for dividing the data set.
  train = sample(sampling.n, sampling.n*Dlearn_rate)

  dat   = data.frame(sampling.data[train,])
  datt  = data.frame(sampling.data[-train,])

  # logistic regression
  log.mod = glm(Status ~., data = dat, family = "binomial")
  log.pred = predict(log.mod, newdata = sampling.data[train,], type = "response")
  log.pred = ifelse(log.pred > 0.5, 1, 0)
  tune_train_error[m,1] = mean( sampling.data[train,1]!= log.pred)
  log.pred = predict(log.mod, newdata = sampling.data[-train,], type = "response")
  log.pred = ifelse(log.pred > 0.5, 1, 0)
  tune_test_error[m,1] = mean( sampling.data[-train,1]!= log.pred)
```

```r
  # lasso regression
  bestlam = cv.lasso$lambda.min
  lasso.mod = glmnet(X[train,], y[train], alpha = 1, family = "binomial",
                     intercept = T, lambda = bestlam,
                     standardize = F)
  tune_lasso.coef[m,] = coef(lasso.mod)[,1]
  lasso.pred = predict(lasso.mod, s = bestlam, newx = X[train,], type ="class")
  tune_train_error[m,2] = mean(y[train] != lasso.pred)
  lasso.pred = predict(lasso.mod, s = bestlam, newx = X[-train,], type="class",)
  tune_test_error[m,2] = mean(y[-train] != lasso.pred)

  # ridge regression
  bestlam = cv.ridge$lambda.min
  ridge.mod = glmnet(X[train,], y[train], alpha = 0, family = "binomial",
                     intercept = T, lambda = bestlam,
                     standardize = F)
  tune_ridge.coef[m,] = coef(ridge.mod)[,1]
  ridge.pred = predict(ridge.mod, s = bestlam, newx = X[train,], type = "class")
  tune_train_error[m,3] = mean(y[train] != ridge.pred)
  ridge.pred = predict(ridge.mod, s = bestlam, newx = X[-train,], type = "class")
  tune_test_error[m,3] = mean(y[-train] != ridge.pred)
}

# best model fit
bestlam = cv.ridge$lambda.min
ridge.mod = glmnet(X, y, alpha = 0, family = "binomial",
                   intercept = T, lambda = bestlam,
                   standardize = F)
ridge.pred = predict(ridge.mod, s = bestlam, newx = X, type = "class")
mean(y != ridge.pred)
ridge.confusion = data.frame(table(ridge.pred, y))
xtable(ridge.confusion)


#######################################
#Graph Drawing
####################

#general comparison between logistic, lasso, and ridge regression
  f1_1 = ggplot(melt(train_error), aes(x = variable, y = value, color = variable)) +
    geom_boxplot() + ylim(0,.5) + labs(x = element_blank(), y = "Misclassification Error Rate",
                                       title = "Train Error Rate") +
    scale_color_brewer(palette="Dark2") + theme(legend.position="none")
  f1_2 = ggplot(melt(test_error), aes(x = variable, y = value, color = variable)) +
    geom_boxplot() + ylim(0,.5)  + labs(x = element_blank(), y = "Misclassification Error Rate",
                                        title = "Test Error Rate") +
    scale_color_brewer(palette="Dark2") + theme(legend.position="none")
  f1_3 = ggplot(melt(cv_error), aes(x = variable, y = value, color = variable)) +
    geom_boxplot() + ylim(0,.5)  + labs(x = element_blank(), y = "Misclassification Error Rate",
                                        title = "CV Error Rate") +
    scale_color_brewer(palette="Dark2") + theme(legend.position="none")
  f1 = grid.arrange(f1_1, f1_2, f1_3, nrow =1, widths = c(3,3,2))
```

```r
# cross validation folds for lasso
  lasso.average.cv.time.and.error = data.frame(error = round(colMeans(cv_error_nfold[,1:15]),4),
                                                time = round(colMeans(time.cv.nfold[,1:15]),2),
                                                fold = factor(c("3","4","5","6","7",
                                                                "8","9","10","15","20",
                                                                "25","30","40","50","LOO"),
                                                              levels = c("3","4","5","6","7",
                                                                         "8","9","10","15","20",
                                                                         "25","30","40","50","LOO")))
  f2_1 = ggplot() +
    geom_line(data = lasso.average.cv.time.and.error, aes(x = fold, y = error, group = 1,
                                                          color = "line")) + ylim(0.15,.2) +
    geom_point(data = lasso.average.cv.time.and.error, aes(x = fold[which.min(error)],
                                                       y = min(error), color = "min")) +
    labs(x = "Cross Validation Folds", y = "Misclassification Error Rate",
         title = "LASSO CV Error Rate") + theme(legend.position="none") +
    scale_color_manual(name = element_blank(), values = c("line" = "blue", "min" = "red"))
  f2_2 = ggplot() +
    geom_line(data = lasso.average.cv.time.and.error, aes(x = fold, y = time, group = 1,
                                                          color = "line")) + ylim(0,65) +
    geom_point(data = lasso.average.cv.time.and.error, aes(x = fold[which.min(time)],
                                                         y = min(time), color = "min")) +
    labs(x = "Cross Validation Folds", y = "Time[sec]", title = "LASSO CV Running Time") +
    theme(legend.position="none") +
    scale_color_manual(name = element_blank(), values = c("line" = "blue", "min" = "red"))
  f2 = grid.arrange(f2_1, f2_2, nrow =1, widths = c(1,1))

# cross validation folds for ridge
  ridge.average.cv.time.and.error = data.frame(error = round(colMeans(cv_error_nfold[,16:30]),4),
                                                      time = round(colMeans(time.cv.nfold[,16:30]),2),
                                                      fold = factor(c("3","4","5","6","7",
                                                                      "8","9","10","15","20",
                                                                      "25","30","40","50","LOO"),
                                                                    levels = c("3","4","5","6","7",
                                                                               "8","9","10","15","20",
                                                                               "25","30","40","50","LOO")))
  f3_1 = ggplot() +
    geom_line(data = ridge.average.cv.time.and.error, aes(x = fold, y = error, group = 1,
                                                          color = "line")) + ylim(0.15,.2) +
    geom_point(data = ridge.average.cv.time.and.error, aes(x = fold[which.min(error)],
                                                           y = min(error), color = "min")) +
    labs(x = "Cross Validation Folds", y = "Misclassification Error Rate",
         title = "Ridge CV Error Rate") + theme(legend.position="none") +
    scale_color_manual(name = element_blank(), values = c("line" = "blue", "min" = "red"))
  f3_2 = ggplot() +
    geom_line(data = ridge.average.cv.time.and.error, aes(x = fold, y = time, group = 1,
                                                          color = "line")) + ylim(0,65) +
    geom_point(data = ridge.average.cv.time.and.error, aes(x = fold[which.min(time)],
                                                           y = min(time), color = "min")) +
    labs(x = "Cross Validation Folds", y = "Time[sec]", title = "Ridge CV Running Time") +
    theme(legend.position="none") +
    scale_color_manual(name = element_blank(), values = c("line" = "blue", "min" = "red"))
  f3 = grid.arrange(f3_1, f3_2, nrow =1, widths = c(1,1))
```

```r
# cv error rate with lambda
  f4_1 = ggplot() +
    geom_line(data = lasso.cv.error, aes(x = lambda, y = error, group = 1,
                                                         color = "line")) + ylim(0,.75) +
    geom_point(data = lasso.cv.error, aes(x = lambda[which.min(error)],
                                                          y = min(error), color = "min")) +
    labs(x = "Lambda", y = "CV Misclassification Error Rate",
         title = "LASSO CV Error Rate") + theme(legend.position="none") +
    scale_color_manual(name = element_blank(), values = c("line" = "blue", "min" = "red"))
  f4_2 = ggplot() +
    geom_line(data = ridge.cv.error, aes(x = lambda, y = error, group = 1,
                                         color = "line")) + ylim(0,.75) +
    geom_point(data = ridge.cv.error, aes(x = lambda[which.min(error)],
                                          y = min(error), color = "min")) +
    labs(x = "Lambda", y = "CV Misclassification Error Rate",
         title = "Ridge CV Error Rate") + theme(legend.position="none") +
    scale_color_manual(name = element_blank(), values = c("line" = "blue", "min" = "red"))
  f4 = grid.arrange(f4_1, f4_2, nrow =1, widths = c(1,1))

#comparison between logistic, tuned lasso, and tuned ridge regression
  f5_1 = ggplot(melt(tune_train_error), aes(x = variable, y = value, color = variable)) +
    geom_boxplot() + ylim(0,.5) + labs(x = element_blank(), y = "Misclassification Error Rate",
                                     title = "Train Error Rate with Optimal Lambda") +
    scale_color_brewer(palette="Dark2") + theme(legend.position="none")
  f5_2 = ggplot(melt(tune_test_error), aes(x = variable, y = value, color = variable)) +
    geom_boxplot() + ylim(0,.5)  + labs(x = element_blank(), y = "Misclassification Error Rate",
                                      title = "Test Error Rate with Optimal Lambda") +
    scale_color_brewer(palette="Dark2") + theme(legend.position="none")
  f5 = grid.arrange(f5_1, f5_2, nrow =1, widths = c(1,1))

# variable importance
  tune_lasso.coef.mean = data.frame(name = dimnames(coef(lasso.mod)),
                                    coef = colMeans(tune_lasso.coef) )
  tune_lasso.coef.mean = tune_lasso.coef.mean[,-2]
  tune_ridge.coef.mean = data.frame(name = dimnames(coef(ridge.mod)),
                                    coef = colMeans(tune_ridge.coef) )
  tune_ridge.coef.mean = tune_ridge.coef.mean[,-2]

  top5.positive.lasso = tune_lasso.coef.mean %>% arrange(-coef) %>% top_n(5)
  top5.negative.lasso = tune_lasso.coef.mean %>% arrange(coef) %>% top_n(-5)
  top5.lasso = cbind(top5.positive.lasso,top5.negative.lasso)
  xtable(top5.lasso)
  top5.positive.ridge = tune_ridge.coef.mean %>% arrange(-coef) %>% top_n(5)
  top5.negative.ridge = tune_ridge.coef.mean %>% arrange(coef) %>% top_n(-5)
  top5.ridge = cbind(top5.positive.ridge,top5.negative.ridge)
  xtable(top5.ridge)

  f6_1 = ggplot(tune_lasso.coef.mean, aes(x = tune_lasso.coef.mean[,1], y = coef)) +
    geom_bar(stat="identity") + ylim(-1,1) + theme(legend.position="none")+
    labs(x = element_blank(), y = "Average Coefficients",
         title = "LASSO Variable Importance") +
    theme(axis.text.x=element_blank(), axis.ticks.x=element_blank())
  f6_2 = ggplot(tune_ridge.coef.mean, aes(x = tune_ridge.coef.mean[,1], y = coef)) +
    geom_bar(stat="identity") + ylim(-1,1) + theme(legend.position="none")+
    labs(x = element_blank(), y = "Average Coefficients",
         title = "Ridge Variable Importance") +
    theme(axis.text.x=element_blank(), axis.ticks.x=element_blank())
  f6 = grid.arrange(f6_1, f6_2, nrow = 1)
```