lab1 block1 2.0

Zhixuan_Duan(zhidu838)

12/18/2019

Orange is the correction.

Green is the comments to the group report.

Load packages

```
library(readxl)
library(kknn)
library(dplyr)
library(ggplot2)
library(MASS)
library(glmnet)
```

Blue is my thoughts.

Thank you very much!

Assignment 1. Spam classification with nearest neighbors

1.

Read the data into R

2.

```
##                  train_acc
## train_observation   0   1
##                 0 804 127
##                 1  93 346
```

```
## [1] 0.1605839
```

```
##                 test_acc
## test_observation   0   1
##                0 808 143
##                1  92 327
```

```
## [1] 0.1715328
```

Add error rate for test dataset

The output shows the model performances when the classification rule set to be 50/50.

3.

```
##                  train_acc
## train_observation   0   1
##                 0 921  10
##                 1 333 106
```

```
## [1] 0.250365
```

```
##                 test_acc
## test_observation   0   1
##                0 931  20
##                1 314 105
```

```
## [1] 0.2437956
```

The output shows the model performances when the classification rule set to be 20/80.

Compared with the previous question, we can see after applying new rule which is the model will classify a email over 80% of probability, so from the result, the predict-spam number decrease even though the overall misclassification rate increase.

4.

```
##    train_pred1
##       0   1
##   0 779 152
##   1  77 362
```

```
## [1] 0.1671533
```

```
##    test_pred1
##       0   1
##   0 702 249
##   1 180 239
```

```
## [1] 0.3131387
```

Compared to the results in group report, points out the knn-method's impact.

Compared to the step 2, the misclassification rates for train dataset are almost the same, however, the mis_error differ dramatically when the model is deployed to the test dataset, the knn method clearly has more error rate.

5.

```
##    train_pred1
##       0   1
##   0 931   0
##   1   0 439
```

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

```
##    test_pred1
##       0   1
##   0 644 307
##   1 185 234
```

```
## [1] 0.3591241
```

Introduce the procedure of KNN method, explain the relationship between K and error rate.

Effect: Compared to question 4, the most significant change is the train accuracy increase to 100%, with no error, but the error rate increase on test datasets, it implys when K = 1, the model overfit train dataset.

Explaination: The k-nearest neighbors model will be very complex and locally optimised when k equal to one, since every point of dataset is considered as the seperate class. And when k increase, it would be smoother.

Assignment 3. Feature selection by cross-validation in a linear model.

1

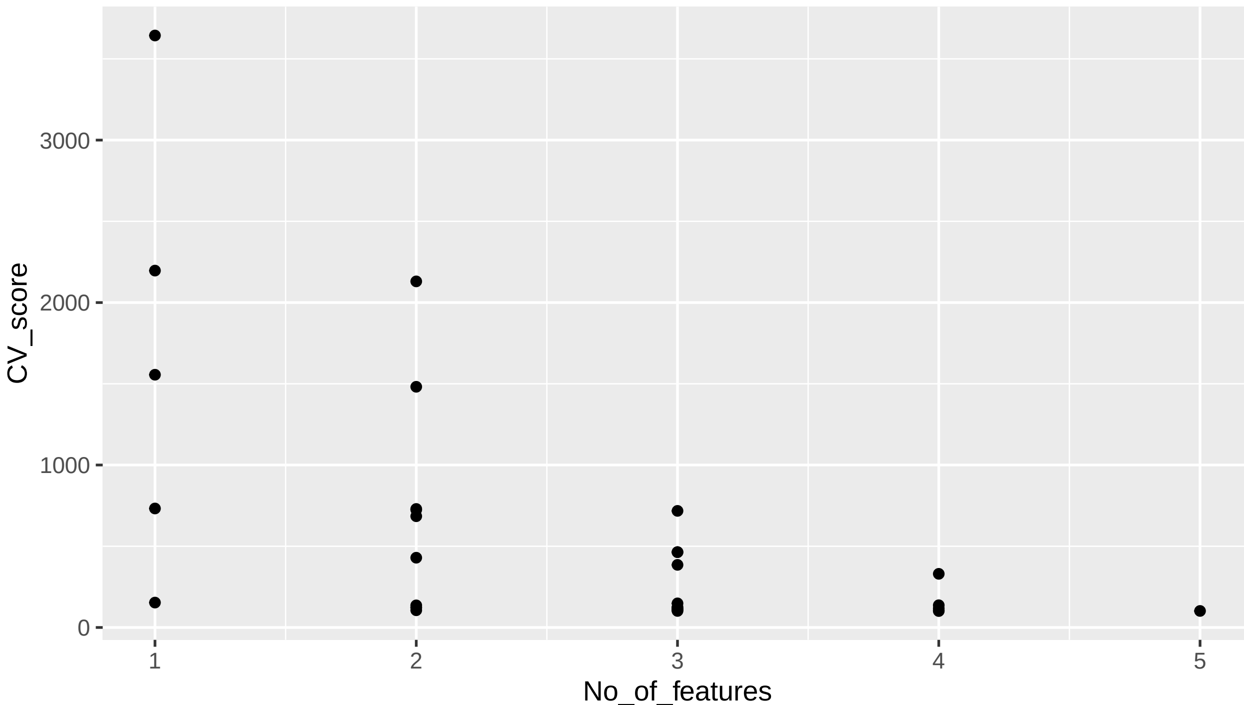
```r
fun1 <- function(x, y, nfolds){

# permute the dataset
df <- cbind(x, y)
set.seed(12345)
df1 <- df[sample(nrow(df)),]
folds <- sample(nfolds, size = nrow(df1), replace = TRUE, prob = rep(1/nfolds,nfolds))
df2 <- cbind(df1, folds)

# get the new x and y
x_new <- df2[,1:5]
y_new <- df2[,6]
f_new <- df2[,7]

mse_folds <- c()
cv_model <- c()
n_features <- c()
result <- NULL

comb <- expand.grid(c(T, F), c(T, F), c(T, F), c(T, F),
                    c(T, F))[-32,]

# Loop over each possible model
for (i in 1:nrow(comb)){
  #loop over each fold
  for (j in 1:nfolds){
    comb_i <- as.logical(c(as.logical(comb[i,]),'TRUE','TRUE'))
    n <- sum(comb_i)-2
    n_features[i] <- n
    temp <- df2[,c(comb_i)]
    x_temp <- temp[,1:n]

    #train and test datasets
    train <- temp[temp$folds != j,]
    train_x <- as.matrix(train[, 1:n])
    train_y <- train$y

    test <- temp[temp$folds == j,]
    test_x <- as.matrix(test[, 1:n])
    test_y <- test$y

    # linear regression
    X <- train_x
    beta_j <- solve(t(X) %*% X) %*% t(X) %*% train_y
    yfit_j <- test_x %*% beta_j
    res_j <- abs(yfit_j - test_y)
    mse_j <- mean(res_j^2)

    mse_folds[j] <- mse_j
  }
```

Compute cv score

```r
  cv_model[i] <- mean(mse_folds)
  n_features[i] <- n
}

temp1 <- cbind(cv_model,n_features)
colnames(temp1) <- c("CV_score", "No_of_features")
temp1 <- as.data.frame(temp1)

# return results consists of n_feature, model, cv_score and plot
# extracting the best model
min_cv <- min(temp1$CV_score)
best_model_index <- which.min(temp1$CV_score)
x <- as.logical(comb[best_model_index,])
final_model <- colnames(x_new)[x]
# the best n feature
final_n_feature <- length(final_model)
# plot
final_plot <- ggplot(temp1, aes(x = No_of_features, y = CV_score)) +
geom_point()

results <- list(final_n_feature,final_model,min_cv,final_plot)
return(results)
}
```

2

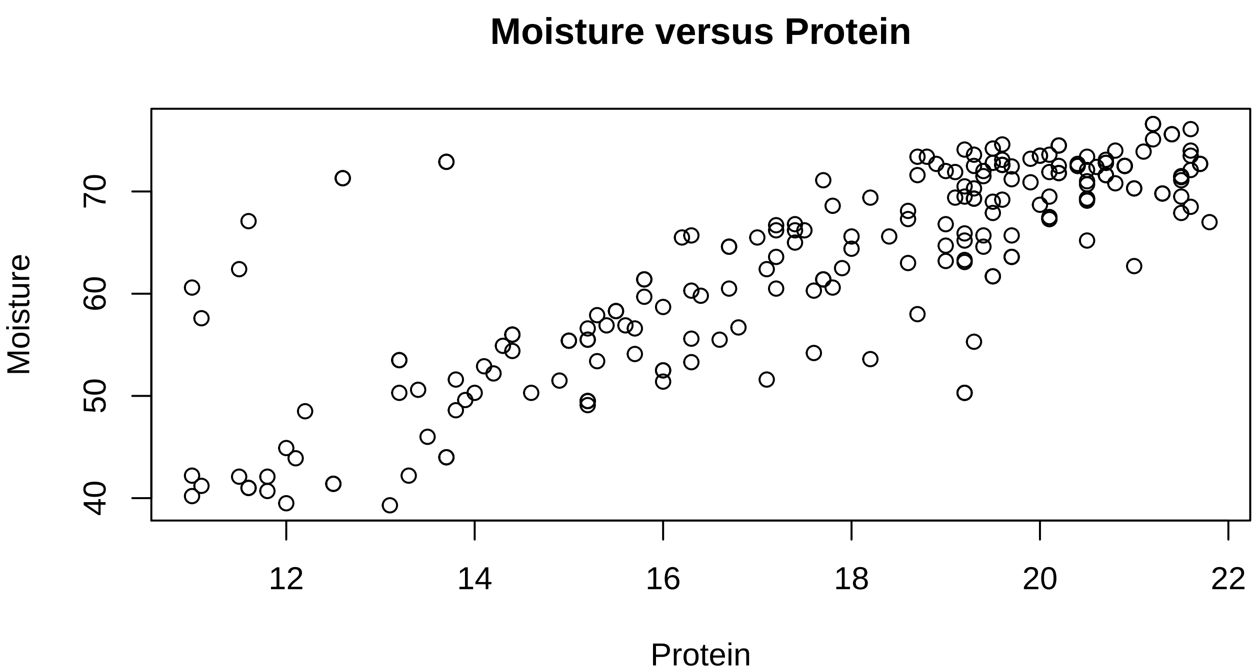
```
## [[1]]
## [1] 4
##
## [[2]]
## [1] "Examination"      "Education"        "Catholic"
## [4] "Infant.Mortality"
##
## [[3]]
## [1] 101.0744
##
## [[4]]
```


CV_score
3000
2000
1000
0
1
2
3
4
5
No_of_features

Assignment 4. Linear regression and regularization

1

Yes, from the scatter plot, we can see when the protein increase, the moisture rise as well in general, but there are some outliers need to be take care of.



2

The linear regression model is :

$$M_i = \beta_0 + \beta_1 P_1 + \beta_2 P_2^2 + \cdots + \beta_i P_i^i + \epsilon = \beta^T P + \epsilon$$

Where,

$$P = (1\ P_1\ P_2\ \ldots\ P_i)^T$$
$$\beta = (\beta_0\ \beta_1\ \beta_2\ \ldots\ \beta_i)^T$$
$$\epsilon \sim N(0, \theta)$$

Notice, we consider the moisture $M_i$is normally distributed. Add the probabilistic linear model

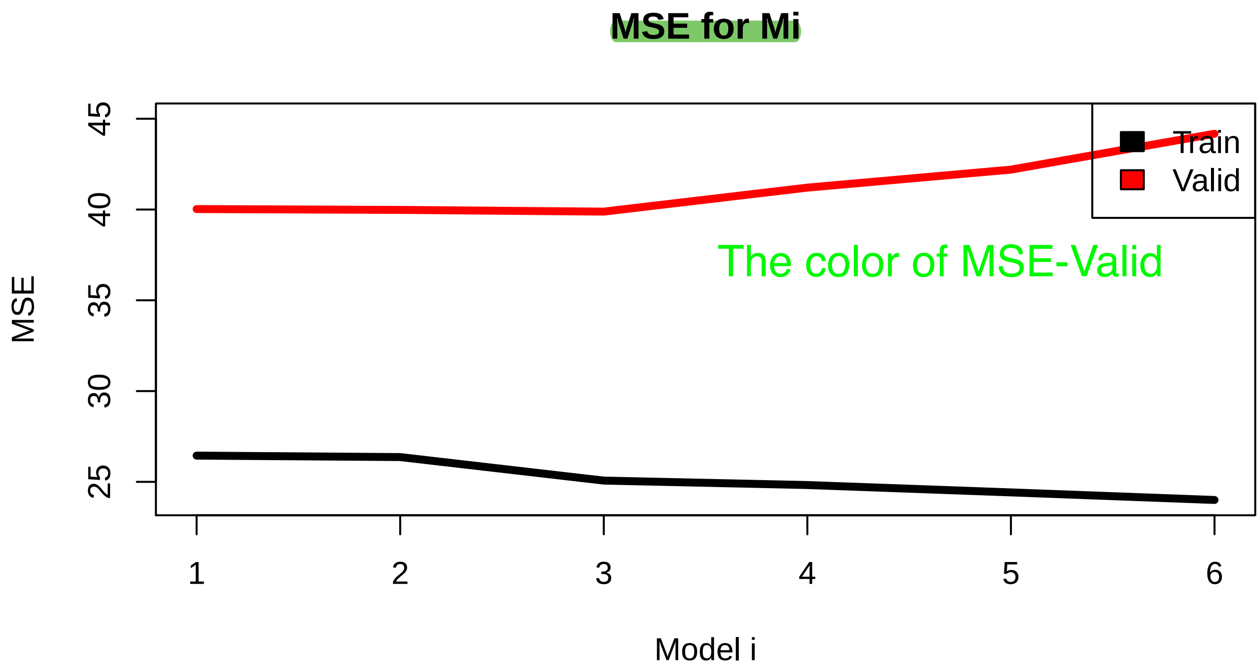
Then the probabilistic model is:

$$p(M_i|\beta) = N(M_i|P\beta, \sigma^2 I_n)$$

And we use MSE criterion because when the $\epsilon \sim N(0, \sigma^2)$, the parameters β which chosen to maximise the likelihood(MLE) are exactly the same which chosen to minimise the mean-squared error(MSE).

3 Explain why use MSE criterion here.

When the i equals 3, it should be the best model. From the plot we can see, the MSE for M3 is the lowest when predict test dataset and when it comes to 4, the model seems to over-fitted.



4

```
##
## Call:
## lm(formula = Fat ~ Channel1 + Channel2 + Channel4 + Channel5 +
##     Channel7 + Channel8 + Channel11 + Channel12 + Channel13 +
##     Channel14 + Channel15 + Channel17 + Channel19 + Channel20 +
##     Channel22 + Channel24 + Channel25 + Channel26 + Channel28 +
##     Channel29 + Channel30 + Channel32 + Channel34 + Channel36 +
##     Channel37 + Channel39 + Channel40 + Channel41 + Channel42 +
##     Channel45 + Channel46 + Channel47 + Channel48 + Channel50 +
##     Channel51 + Channel52 + Channel54 + Channel55 + Channel56 +
##     Channel59 + Channel60 + Channel61 + Channel63 + Channel64 +
##     Channel65 + Channel67 + Channel68 + Channel69 + Channel71 +
##     Channel73 + Channel74 + Channel78 + Channel79 + Channel80 +
##     Channel81 + Channel84 + Channel85 + Channel87 + Channel88 +
##     Channel92 + Channel94 + Channel98 + Channel99, data = temp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.82961 -0.57129 -0.00696  0.58152  2.86375
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)      7.093      1.453   4.882 2.64e-06 ***
## Channel1     10559.894   2333.430   4.525 1.21e-05 ***
## Channel2    -12636.967   3467.995  -3.644 0.000369 ***
## Channel4      8489.323   4637.993   1.830 0.069164 .
## Channel5    -10408.967   4771.350  -2.182 0.030689 *
## Channel7     -5376.018   3851.782  -1.396 0.164847
## Channel8      7215.595   4246.489   1.699 0.091342 .
```

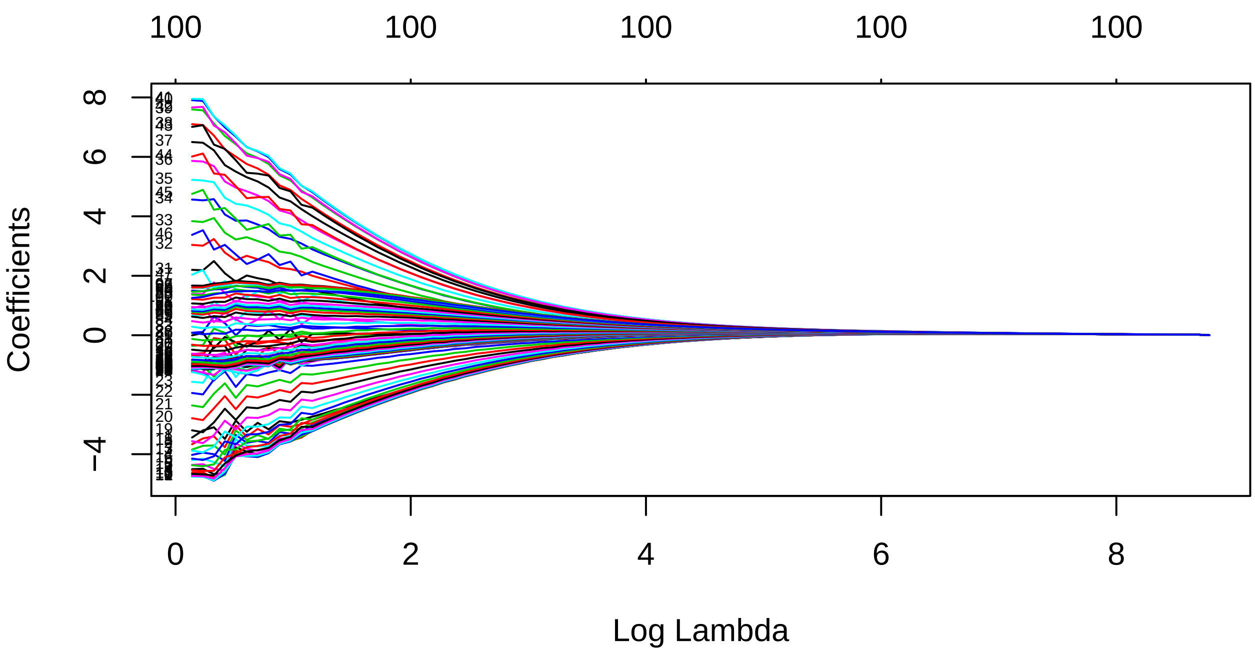
```
## Channel11    -9505.520   5721.115  -1.661 0.098692 .
## Channel12    37240.918  12290.648   3.030 0.002878 **
## Channel13   -41564.547  15892.375  -2.615 0.009817 **
## Channel14    34938.179  13290.454   2.629 0.009454 **
## Channel15   -23761.451   6584.006  -3.609 0.000417 ***
## Channel17     4296.572   3189.730   1.347 0.179998
## Channel19    14279.808   5017.407   2.846 0.005042 **
## Channel20   -23855.616   5153.161  -4.629 7.85e-06 ***
## Channel22    18444.906   3381.683   5.454 1.97e-07 ***
## Channel24   -20138.426   4946.417  -4.071 7.52e-05 ***
## Channel25    18137.432   5374.094   3.375 0.000938 ***
## Channel26    -7670.318   3859.006  -1.988 0.048660 *
## Channel28    20079.898   4991.631   4.023 9.06e-05 ***
## Channel29   -36351.014   7655.223  -4.749 4.72e-06 ***
## Channel30    18071.276   5863.802   3.082 0.002446 **
## Channel32     3838.013   2722.862   1.410 0.160729
## Channel34    -9242.884   2225.926  -4.152 5.48e-05 ***
## Channel36     8070.938   3317.588   2.433 0.016152 *
## Channel37    -9045.588   3536.621  -2.558 0.011522 *
## Channel39    18664.454   5986.730   3.118 0.002183 **
## Channel40   -20069.709  10701.902  -1.875 0.062677 .
## Channel41    22257.776  11122.533   2.001 0.047169 *
## Channel42   -21760.853   5833.811  -3.730 0.000270 ***
## Channel45    18145.804   2985.416   6.078 9.50e-09 ***
## Channel46    -8225.696   3715.367  -2.214 0.028330 *
## Channel47    -4986.549   2558.694  -1.949 0.053165 .
## Channel48     2876.075   2014.985   1.427 0.155546
## Channel50   -13009.410   4535.797  -2.868 0.004720 **
## Channel51    29251.161   6554.297   4.463 1.57e-05 ***
## Channel52   -26833.976   4389.473  -6.113 7.97e-09 ***
## Channel54    30954.862   4392.339   7.047 6.06e-11 ***
## Channel55   -35183.287   5646.314  -6.231 4.39e-09 ***
## Channel56    14912.986   2810.889   5.305 3.93e-07 ***
## Channel59    -8030.278   1887.431  -4.255 3.66e-05 ***
## Channel60    13071.416   2629.374   4.971 1.79e-06 ***
## Channel61    -7850.189   2246.864  -3.494 0.000625 ***
## Channel63    15059.275   3231.692   4.660 6.90e-06 ***
## Channel64   -19909.466   4727.696  -4.211 4.35e-05 ***
## Channel65     4190.184   3486.766   1.202 0.231346
## Channel67    13850.508   3909.121   3.543 0.000526 ***
## Channel68   -25873.365   5304.223  -4.878 2.69e-06 ***
## Channel69    18362.385   3331.483   5.512 1.50e-07 ***
## Channel71    -9223.910   1558.752  -5.917 2.11e-08 ***
## Channel73    12456.498   2386.255   5.220 5.82e-07 ***
## Channel74    -5624.411   1933.590  -2.909 0.004177 **
## Channel78    -7927.105   2176.860  -3.642 0.000372 ***
## Channel79    15473.188   3812.200   4.059 7.89e-05 ***
## Channel80   -22391.895   4490.714  -4.986 1.67e-06 ***
## Channel81    13852.453   3105.934   4.460 1.59e-05 ***
## Channel84   -11442.630   3457.064  -3.310 0.001167 **
## Channel85    20228.671   4081.863   4.956 1.91e-06 ***
## Channel87   -15938.315   4102.273  -3.885 0.000153 ***
## Channel88     5647.072   3236.286   1.745 0.083033 .
## Channel92     6595.995   1864.595   3.537 0.000537 ***
```

```
## Channel94    -5497.846   1847.113  -2.976 0.003397 **
## Channel98    -8728.596   2489.314  -3.506 0.000598 ***
## Channel99     8554.587   1898.010   4.507 1.31e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.107 on 151 degrees of freedom
## Multiple R-squared:  0.9947, Adjusted R-squared:  0.9925
## F-statistic: 447.9 on 63 and 151 DF,  p-value: < 2.2e-16
```

```
## [1] 63
```

From the result, 63 independent variables are selected in this linear model.

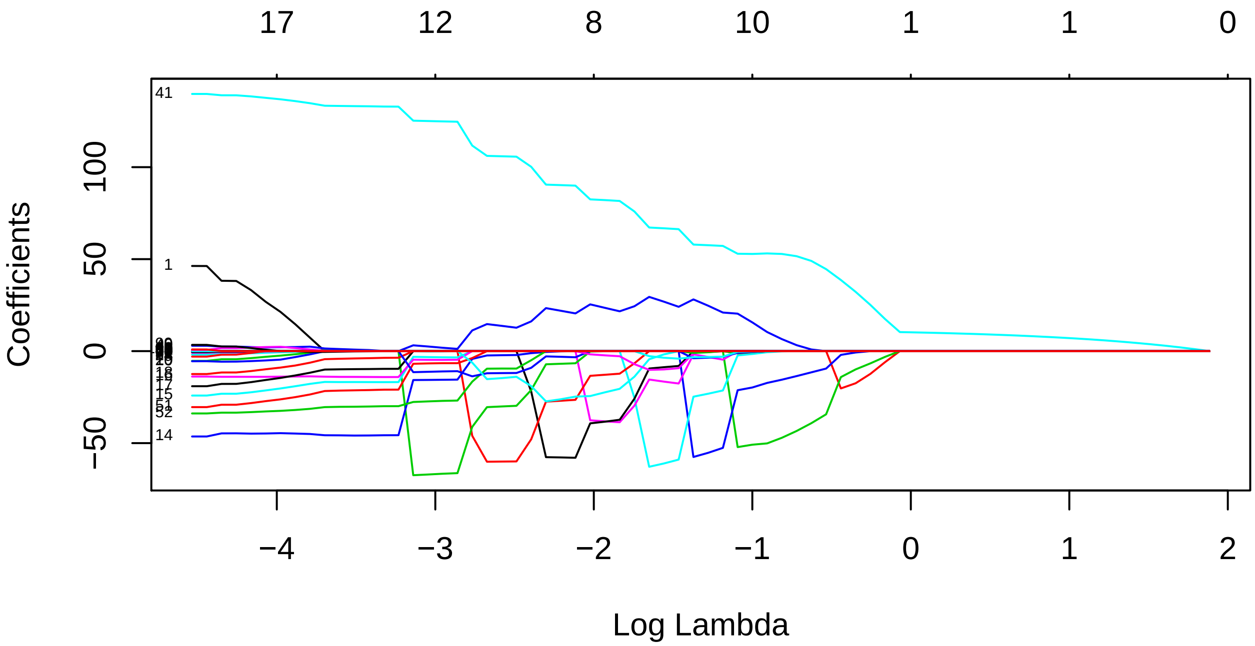
5



From the plot, we can see that when $log(\lambda)$ increases, all the coefficients press on zero.

And when penalty factor increase, the coefficients will decrease, and make the parameter smaller than before.

6



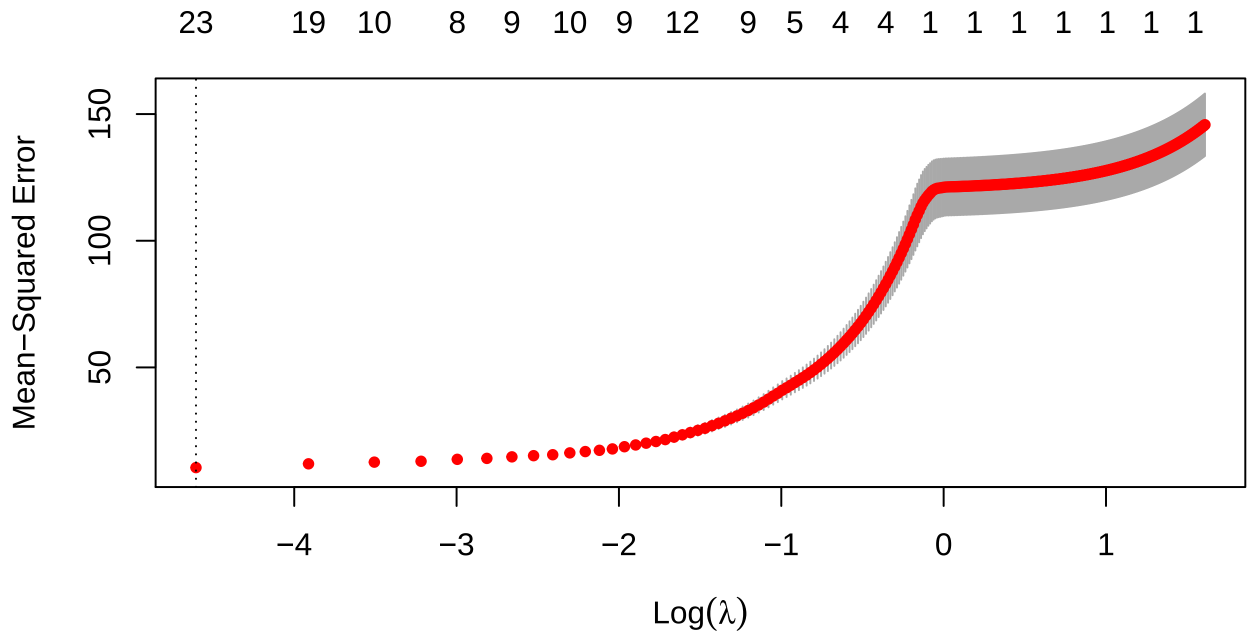
Both the **ridge** and the **lasso** models show that as $log(\lambda)$ increases, the coefficients converge to zero.

But the **lasso** model select less variables, and the coefficients converge much faster as log(lambda) increases than which in the **ridge** model.

Apparently, the **lasso** model can select fewer variables, and set more variables' weight to 0, compared to the **ridge** model. But it may underfit the data, considering the weights to variables in the **ridge** model will not be zero, which means all the variables will be included in the final model.

7

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



The optimal lambda is 0, and 22 independent variables were selected.

From the plot, we can see as $log(\lambda)$ increases, the **MSE** increase as well.

8.

The stepwise-AIC model selects a total of 63 independent variables, whereas the cross-validation generlised linear model selects 22 independent variables.

For the stepwise-AIC method, the final model shows a higher AIC score indicating a good fit.

However,the number of variables in stepwise-AIC model is more than the cv-glm model, also,the coefficents of cv-glm model is smaller, which means it has more robustness. So I think the cv-glm model is better.

Appendix

Sorry, but I don't think the AIC criterion is appropriate here to compare two models since the model in Q4 is chosen by the stepwise-AIC method.
So comparing the number of variables and their coefficients may be a suitable way.

```
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE,
                      fig.width = 7,
                      fig.height = 4,
                      fig.align = 'center')
library(readxl)
library(kknn)
library(dplyr)
library(ggplot2)
library(MASS)
library(glmnet)
data <-read_excel("/Users/darin/Desktop/ML/Lab/spambase.xlsx")

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

```r
fit <- glm(Spam ~ ., data = train, family = binomial)

pred_train <- predict(fit, newdata = train,type = "response")
train_acc <- ifelse(pred_train > 0.5, 1, 0)
train_observation <- train$Spam
table1 <- table(train_observation, train_acc)
table1
train_error <- 1 - (sum(diag(table1)) / nrow(train))
train_error


pred_test <- predict(fit, newdata = test, type = "response")
test_acc <- ifelse(pred_test > 0.5, 1, 0)
test_observation <- test$Spam
table2 <- table(test_observation, test_acc)
table2
train_error <- 1 - (sum(diag(table2)) / nrow(test))
train_error
train_acc <- ifelse(pred_train > 0.8, 1, 0)
table3 <- table(train_observation,train_acc)
table3
train_error <- 1 - (sum(diag(table3)) / nrow(train))
train_error

test_acc <- ifelse(pred_test > 0.8, 1, 0)
table4 <- table(test_observation,test_acc)
table4
test_error <- 1 - (sum(diag(table4)) / nrow(test))
test_error
fit.kknn <- train.kknn(Spam ~ ., data = train, kmax = 30)

train_pred <- predict(fit.kknn, train)
train_pred1<-round(train_pred)
table5 <- table(train$Spam, train_pred1)
table5
train_error <- 1 - (sum(diag(table5)) / nrow(train))
train_error

test_pred <- predict(fit.kknn, test)
test_pred1 <-round(test_pred)
table6 <- table(test$Spam, test_pred1)
table6
test_error <- 1 - (sum(diag(table6)) / nrow(test))
test_error
fit.kknn <- train.kknn(Spam ~ ., data = train, kmax = 1)

train_pred <- predict(fit.kknn, train)
train_pred1<-round(train_pred)
table7 <- table(train$Spam, train_pred1)
table7
train_error <- 1 - (sum(diag(table7)) / nrow(train))
train_error
```

```r
test_pred <- predict(fit.kknn, test)
test_pred1 <-round(test_pred)
table8 <- table(test$Spam, test_pred1)
table8
test_error <- 1 - (sum(diag(table8)) / nrow(test))
test_error
fun1 <- function(x, y, nfolds){

# permute the dataset
df <- cbind(x, y)
set.seed(12345)
df1 <- df[sample(nrow(df)),]
folds <- sample(nfolds, size = nrow(df1), replace = TRUE, prob = rep(1/nfolds,nfolds))
df2 <- cbind(df1, folds)

# get the new x and y
x_new <- df2[,1:5]
y_new <- df2[,6]
f_new <- df2[,7]

mse_folds <- c()
cv_model <- c()
n_features <- c()
result <- NULL

comb <- expand.grid(c(T, F), c(T, F), c(T, F), c(T, F),
                    c(T, F))[-32,]

# Loop over each possible model
for (i in 1:nrow(comb)){
  #loop over each fold
  for (j in 1:nfolds){
    comb_i <- as.logical(c(as.logical(comb[i,]),'TRUE','TRUE'))
    n <- sum(comb_i)-2
    n_features[i] <- n
    temp <- df2[,c(comb_i)]
    x_temp <- temp[,1:n]

    #train and test datasets
    train <- temp[temp$folds != j,]
    train_x <- as.matrix(train[, 1:n])
    train_y <- train$y

    test <- temp[temp$folds == j,]
    test_x <- as.matrix(test[, 1:n])
    test_y <- test$y

    # linear regression
    X <- train_x
    beta_j <- solve(t(X) %*% X) %*% t(X) %*% train_y
    yfit_j <- test_x %*% beta_j
    res_j <- abs(yfit_j - test_y)
    mse_j <- mean(res_j^2)
```

```
    mse_folds[j] <- mse_j
  }

  cv_model[i] <- mean(mse_folds)
  n_features[i] <- n
}

temp1 <- cbind(cv_model,n_features)
colnames(temp1) <- c("CV_score", "No_of_features")
temp1 <- as.data.frame(temp1)

# return results consists of n_feature, model, cv_score and plot
# extracting the best model
min_cv <- min(temp1$CV_score)
best_model_index <- which.min(temp1$CV_score)
x <- as.logical(comb[best_model_index,])
final_model <- colnames(x_new)[x]
# the best n feature
final_n_feature <- length(final_model)
# plot
final_plot <- ggplot(temp1, aes(x = No_of_features, y = CV_score)) +
geom_point()

results <- list(final_n_feature,final_model,min_cv,final_plot)
return(results)
}
fun1(x = swiss[,2:6], y = swiss[,1], nfolds = 5)
library(readxl)
library(MASS)
data<-read_excel("/Users/darin/Desktop/ML/Lab/tecator.xlsx")
plot(data$Protein,data$Moisture,xlab="Protein",ylab="Moisture", main = "Moisture versus Protein")
n<-nrow(data)
set.seed(12345)
id<-sample(n,n%/%2)
train<-data[id,]
valid<-data[-id,]
xtrain<-rep(1,length(id))
xvalid<-rep(1,length(id))
mse<-list(train=NULL,valid=NULL)

for(i in 1:6){
  xtrain<-cbind(xtrain,as.matrix(train["Protein"]^i))
  xvalid<-cbind(xvalid,as.matrix(valid["Protein"]^i))
  esti<-solve(t(xtrain)%*%xtrain,tol=1e-25) %*%
    t(xtrain) %*% as.matrix(train["Moisture"])
  mse[["train"]]<-append(mse[["train"]],
                         sum((xtrain%*%esti-train["Moisture"])^2)/nrow(xtrain))
  mse[["valid"]]<-append(mse[["valid"]],
                         sum((xvalid%*%esti-valid["Moisture"])^2)/nrow(xvalid))
}

plot(1:6,mse$train,xlab="Model i",
     ylab="MSE",main="MSE for Mi",type="l", col = "black", lwd = 4, ylim=range(c(24,45)))
```

```
lines(1:6,mse$valid,add=TRUE,type="l", col = "red", lwd = 4)
legend('topright',legend=c("Train","Valid"),fill=c("black","red"))
temp <- data[, 2:102]
l_m <- lm(Fat ~ ., data = temp)
stepwise <- stepAIC(l_m, direction = "both", trace = FALSE)
summary(stepwise)

# exclude the intercept term
no_var <- length(stepwise$coefficients) - 1
no_var
cov <- as.matrix(temp[, 1:100])
response <- as.matrix(temp[, 101])

# fit a ridge model
model_ridge <- glmnet(cov, response, alpha = 0, family = "gaussian")
plot(model_ridge, xvar = "lambda", label = TRUE)
# fit a lasso model
model_lasso <- glmnet(cov, response, alpha = 1, family = "gaussian")
plot(model_lasso, xvar = "lambda", label = TRUE)
# choose different lambda using cv.glmnet function
model_cv <- cv.glmnet(cov, response, alpha = 1, family = "gaussian",
                      lambda = seq(0,5,0.01))
model_cv$lambda.min
# the number of variables selected in the model
a <- as.matrix(coef(model_cv, lambda = model_cv$lambda.min))
a1 <- data.frame(colname = row.names(a), a)
row.names(a1) <- NULL
b <- a1[a1$X1 != 0,]
no_variable<-nrow(b) - 1
# plot the cv-error versus log(lambda)
plot(model_cv)
```