Machine Learning LAB 01

Yusur AL-Mter, LIU ID yusal621

November 19, 2018

Assignment 1. Spam classification with nearest neighbors

Part 1.1

Importing the data and dividing into training and test sets (50%/50%):

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

Part 1.2

Logistic regression classifying the test data by the classification principle **Y_hat = 1 if p(Y = 1 | X) > 0.5, otherwise Y_hat = 0**, and obtaining the confusion matrices and the misclassification rates for training and test data.

Confusion matrix and misclassification rate for test data:

```
Con_mat_test
```

```
##    test_data
##       0   1
##   0 791 146
##   1  97 336
```

```
mis_rate_test
```

```
## [1] 0.1773723
```

Confusion matrix and misclassification rate for test data:

```
Con_mat_train
```

```
##    train_data
##       0   1
##   0 803 142
##   1  81 344
```

```
mis_rate_train
```

```
## [1] 0.1627737
```

By analyzing the two confusion matrices for both training and test data our misclassification rate increased from 0.1627737 to 0.1773723 for train and test data respectively which concludes that our model (glm function) fits the data better for training data than the test data.

As we are using train data for modeling, the misclassification rate for the train data will be less as compared to the test data because of over fitting for train data.

Part 1.3

Logistic regression classifying the test data by the classification principle **Y__hat = 1 if p(Y = 1 | X) > 0.9, otherwise Y__hat = 0**, and obtaining the confusion matrices and the misclassification rates for training and test data.

Confusion matrix and miss classification rate for test data:

```
Con_mat_test09
```

```
##    test_data09
##       0   1
##   0 936   1
##   1 427   6
```

```
mis_rate_test09
```

```
## [1] 0.3124088
```

Confusion matrix and misclassification rate for train data:

```
Con_mat_train09
```

```
##    train_data09
##       0   1
##   0 944   1
##   1 419   6
```

```
mis_rate_train09
```

```
## [1] 0.3065693
```

changing the classification principle and increasing the probability from 0.5 to 0.9 for an unseen data of being spam or not, the classification accuracy defined by the ratio of correct predictions to total predictions for both test and train data is 0.6875912409 and 0.6934306569, respectively; Hence the misclassification rate can be calculated by (1-classification accuracy), and the calculations obviously indicates that the misclassification rate for training data decreased while increased for the test data.

Part 1.4

Using standard classifier kknn() with K=30

Confusion matrix and misclassification rate for train data using kknn for k=30:

```
Con_mat_train_kknn30
```

```
##
##       0   1
##   0 594 351
##   1 265 160
```

```
mis_rate_train_kknn30
```

```
## [1] 0.449635
```

Confusion matrix and misclassification rate for test data using kknn for k=30:

```
Con_mat_test_kknn30
```

```
##
##       0   1
##   0 672 265
##   1 187 246
```

```
mis_rate_test_kknn30
```

```
## [1] 0.329927
```

Using kknn classifier with k= 30, the misclassification rates increased drastically for both training and test data compared to the results we have in *Logistic regression*, with classification principle *p(Y = 1 | X) >0.5*. The increasing rates for training and test data are approximately 176% and 86% respectively.

Part 1.5

Using standard classifier kknn() with K=1

Confusion matrix and misclassification rate for train data using kknn for k=1:

```
Con_mat_train_kknn_k1
```

```
##
##       0   1
##   0 559 386
##   1 258 167
```

```
mis_rate_train_kknn_k1
```

```
## [1] 0.470073
```

Confusion matrix and misclassification rate for test data using kknn for k=1:

```
Con_mat_test_kknn_k1
```

```
##
##       0   1
##   0 640 297
##   1 177 256
```

```
mis_rate_test_kknn_k1
```

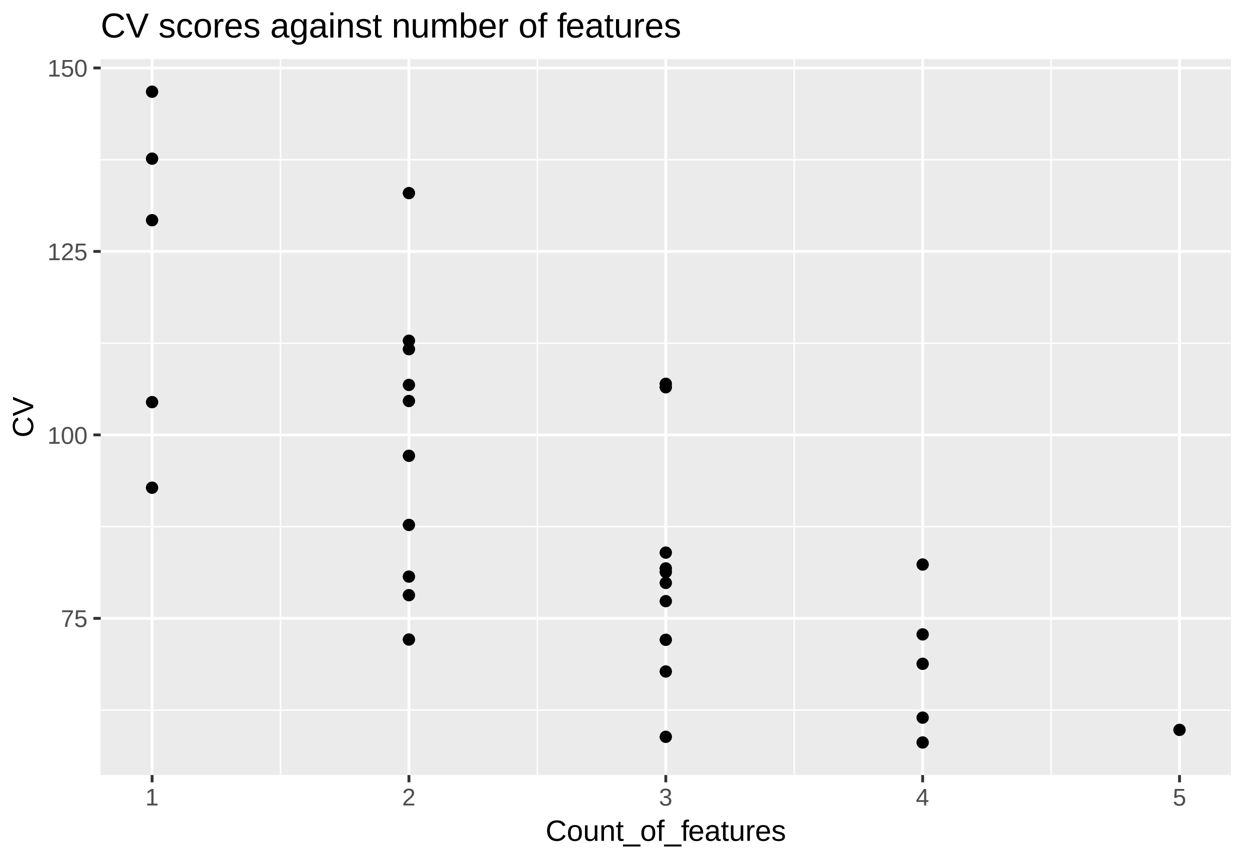
```
## [1] 0.3459854
```

Using kknn classifier with k= 1, the misclassification rates increased for both training and test data as compared to the model with k=30. Since the kknn algorithm clearly illustrates that, when K increases to infinity, the model is simplest. All test data point will belong to the same class: the majority class, and this represents the under-fit, that is, high bias and low variance. While when K decreases to K=1, the granularity or resolution is too fine, which is overfit and the overfit causes high variance and increasing the misclassification rate.

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

Implementing an R function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation

```
## Best Subset:  1, 3, 4, 5    CV:  58.0877264293099
```



The plot demonstrates the implemented function for best subset features selection in linear regression, using k-fold cross-validation. The data set swiss where fertility was the Y *response* and all other variables *predictors* were the columns of the matrix X.The method randomly partitioned the data by the number of folds with dimensions *9, 10, 10, 9 and 9* number of rows for each fold, respectively.

The plot obtained shows that the model with lowest MSE is the one with four features. And our model considered those variables mostly since each has a relationship with the response, although the relationship involves multiple dimensions that are difficult to disentangle empirically or even identify theoretically. we briefly illustrate some of them below:

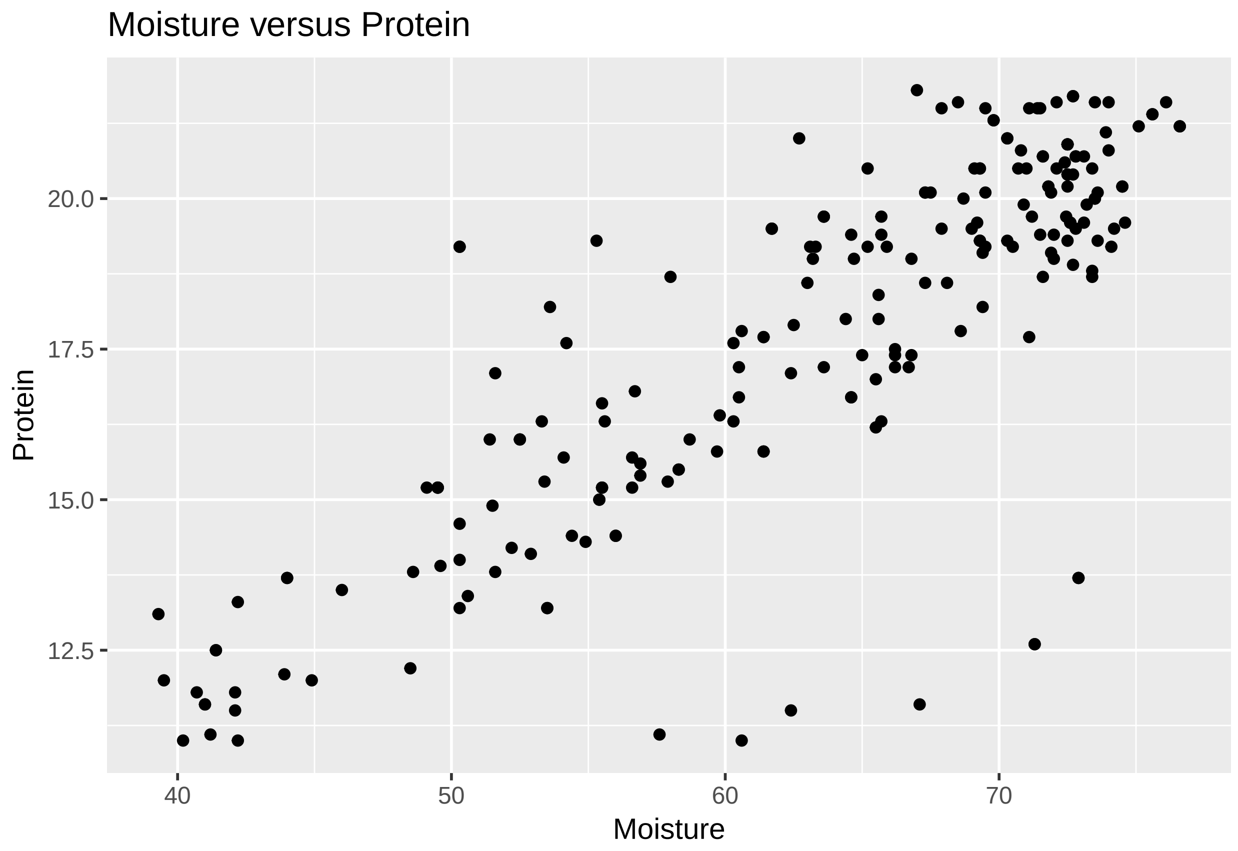
1. *Agriculture*, the type and quality of nutrition will affect proportionally the rate of fertility.
2. *Education*, has many factors affecting positively and negatively (inversely) on the rate of fertility. Such as, increased education in countries with higher literacy levels is more likely to relate inversely with fertility than in less literate countries.
3. *Catholic*, less committed and more committed Catholics, age at first marriage and contraceptive use, are all factors may effect differently in total fertility rate.
4. *Infant Mortality*, there are the effects of fertility on infant and child mortality, a relation depending mostly, on influences of length of birth intervals and related mediating mechanisms such as breast-feeding. On the other hand, there is an effect of infant and child mortality on fertility which depends mostly on mechanisms directly associated with birth intervals.

Assignment 4. Linear regression and regularization

Part 4.1

A plot of Moisture versus Protein:

```r
ggplot(tecator, aes(x = Moisture, y = Protein)) + geom_point()+ ggtitle("Moisture versus Protein")
```



Plotting Moisture and Protein as our target and observation, and by looking at the data points we can say that linear regression is a good model for the data although there are some outliers in the data.

Part 4.2

MSE criterion for a polynomial function:

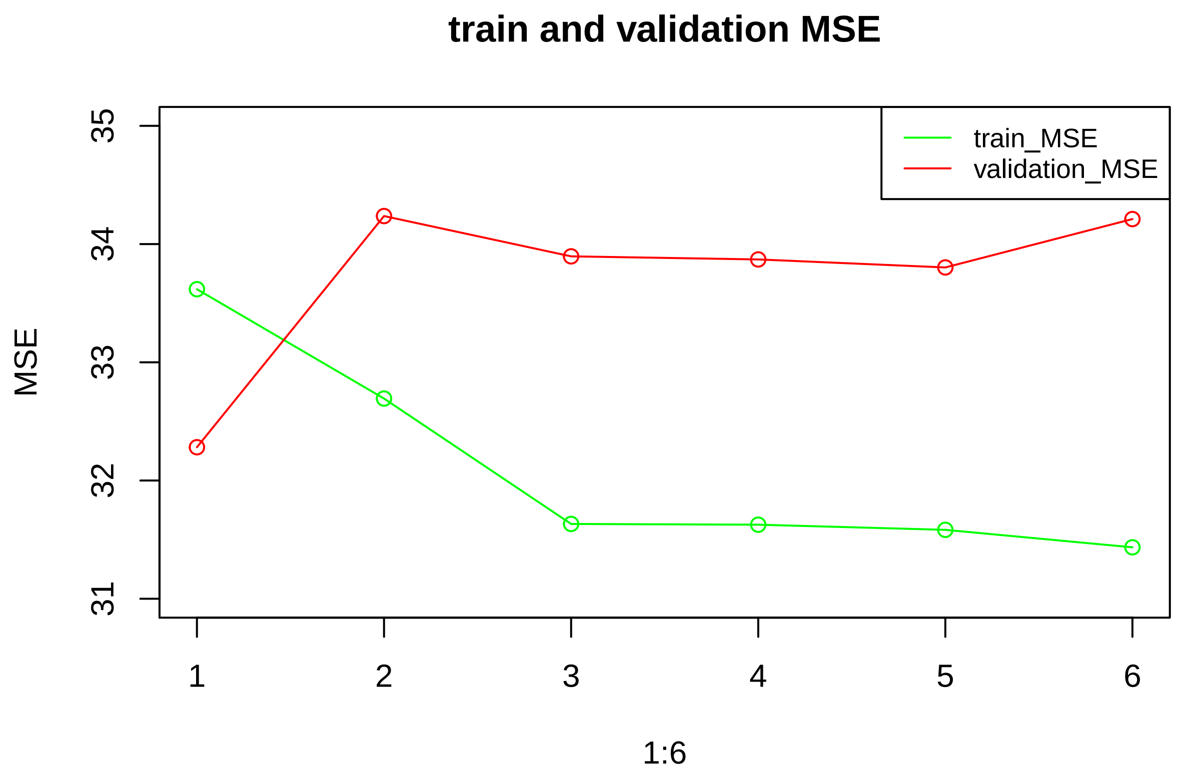
The mean squared error incorporates both the variance and bias, where, there is a trade-off between bias and variance that comes with model complexity: models that are too complex will have high variance and low bias; models that are too simple will have high bias and low variance. The best model will have both low bias and low variance. Since our model is normally distributed and as the polynomial order i increases,the functions M_i(x) are able to capture increasingly complex behavior. MSE criterion is the most appropriate to use when fitting the model to training data as MSE is the best unbiased estimator minimizing the sample variance giving MSE value nearer to zero which is the best. Linear regression over fits data for higher polynomial. In such case the ridge regression would be an appropriate probabilistic model to implement.

Part 4.3

Dividing the data into training and validation sets and fitting models Mi, recording the training and validation MSE and presenting a plot showing how training and validation MSE depend on i:

```
## [1] 33.61836 32.69342 31.63266 31.62641 31.58273 31.43513
```

```
## [1] 32.28154 34.23708 33.89615 33.86992 33.80234 34.21152
```



The graph shows the train data MSE (the green line) and the validation data MSE (the red line) for the polynomial degree, where i represents how much flexibility is in the model, with a higher power allowing the model to hit as many data points as possible and with lower power the model will be less flexible, for that the train data MSE is high at i= 1, and it's decreasing with low variance when the value of i increases and this's due to the model over fitting the training data and this over fitting will also affect the validation data MSE as the value of i increases, bias decreases, the variance increases and the validation data MSE increases. The best model seems to be the simplest model with i = 1 where the validation MSE is low.

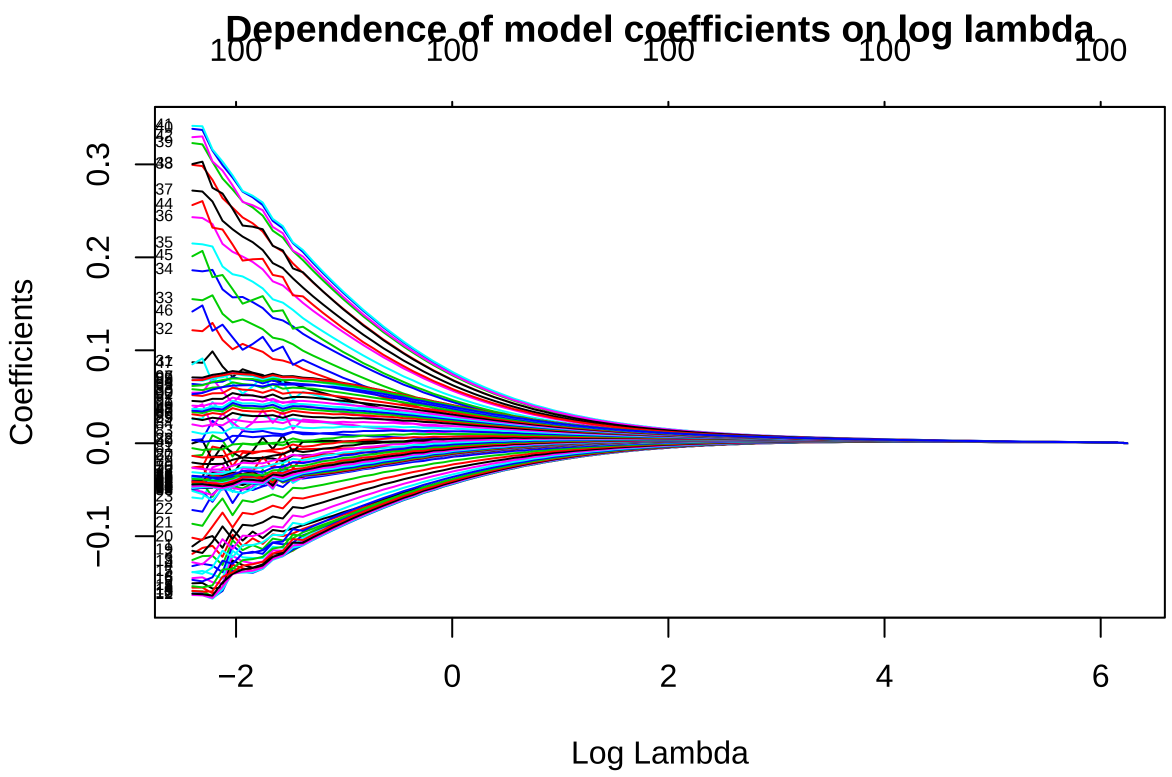
Part 4.4

variable selection of a linear model using stepAIC:

Number of variables that have been selected for the model were 63 variables.

Part 4.5

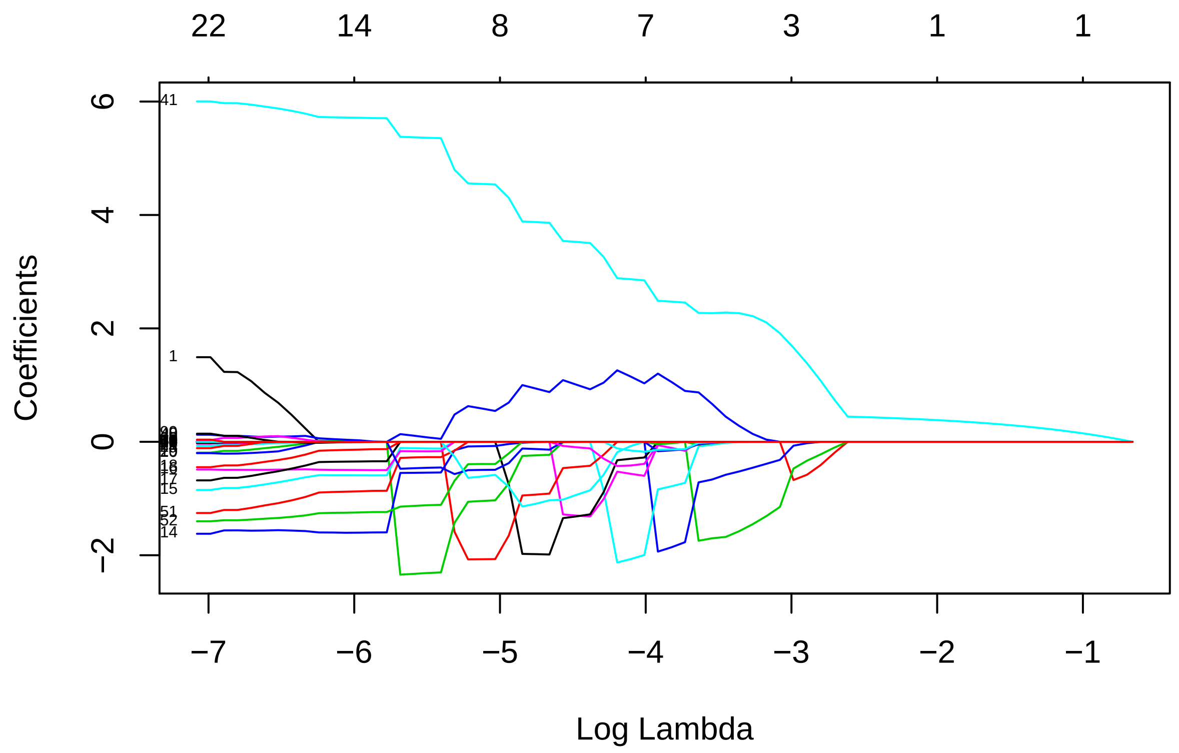
Fitting a Ridge regression model:



The idea of regularization is to prevent overfitting and make model less complex by shrinking coefficients, and we can achieve that by adding penalty factor λ , *Ridge Regression* model with variables Fat as response and channel1 to channel100 as predictors presented in the following plot which shows how our model coefficients' values depend on the log of the penalty factor λ, where the magnitude of the coefficients decreases and approaches zero as the value of log lambda increases and this will effect our model, where the flexibility will decrease.

Part 4.6

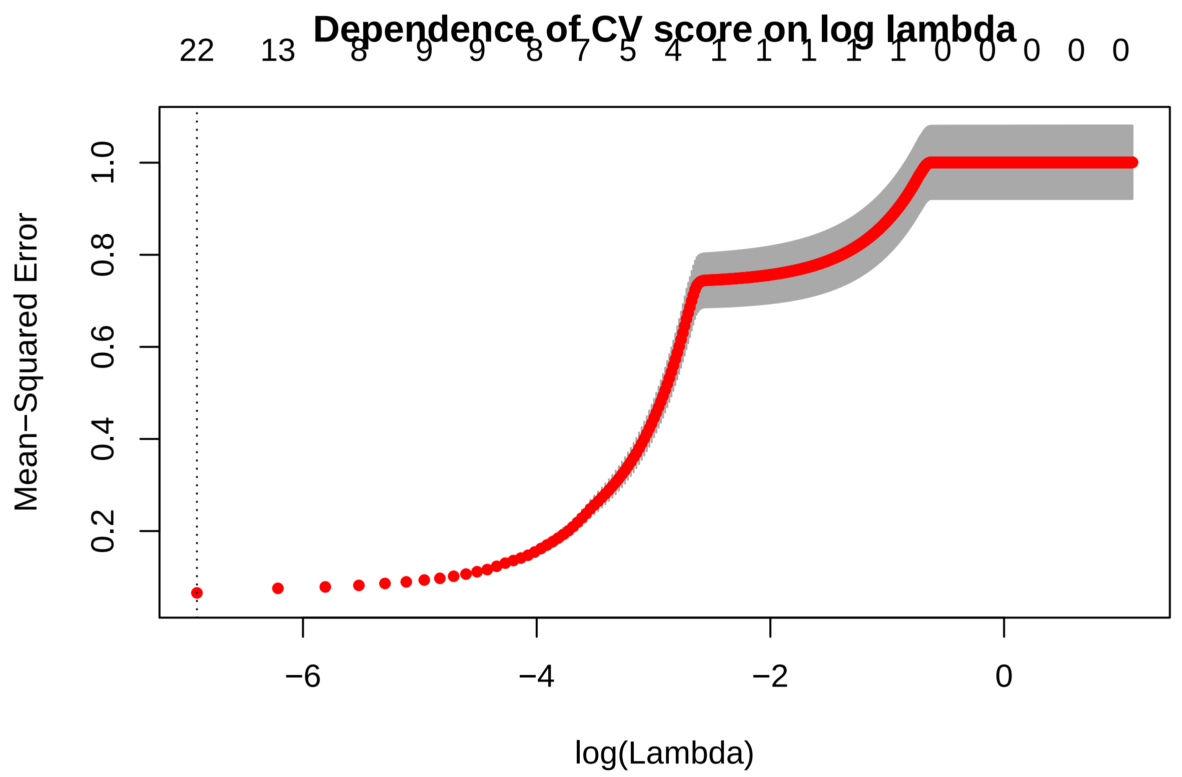
Fitting a Lasso regression model:



In ridge regression, we added a penalty parameter λ to make the fit model small. The shrinkage penalty is λ times the sum of squares of the coefficients, so coefficients that get too large are penalized. As we can see when λ gets larger, the bias is increased but the variance drops. Comparing the plots of *Ridge* and *Lasso*, we noticed that the *Ridge* didn't select variables. It included all of the variables in the final model (i.e it didn't get rid of the irrelevant features but rather minimize their impact on the trained model). In lasso, the penalty was the sum of the absolute values of the coefficients. And as we can see *Lasso* method over comes the disadvantage of *Ridge* regression by not only shrinking the coefficient estimates towards zero (where those coefficient are not significant or important), but actually had the effect of setting variables exactly equal to zero when λ is large enough. as we can see in our Lasso plot there are many significant variables that have been selected for our model, and the variable that most influence the model and more significant is the variable 41 and its positively affecting the response variable.

Part 4.7

Using cross-validation and finding the optimal LASSO model:



The cross validation technique have been used to find the optimal *Lasso* model, and as shown in the plot, the red dotted line represents the algorithm process for finding the most significant subset features, number of variables that have been chosen were all 100 variables that found as the most significant ones for our model, with major advantage of lasso for which it is a combination of both shrinkage and selection of variables, the tuning parameter λ is chosen by cross validation in cases with very large number of features, lasso allow us to efficiently find the sparse model that involve a small subset of the features. And as it is obviously shown in the plot, the MSE increases when the value of λ increases, shrinkage occurs so that variables that are at zero can be thrown away. And the optimal λ found here is 0.

To find the optimal LASSO model we use cross-validation. The optimal lambda found is 0.001341 and the dotted line in the plot below tells us that the best model has 17 non zero coefficients. As we can see from the red dots the MSE increases when lambda increases and the number of features decreases. The CV scores have low standard errors for small lambda and higher for a small number of features and bigger lambda.

Part 4.8

Comparing the results obtained by using both cross-validation to find the optimal LASSO model and stepAIC for variables selection, there was a notable difference : the model obtained from the Stephanie() function has many more variables compared to the one obtained by Cross validation and Lasso regression. Lasso regression is the best solution in this case for variables selection because we have many variables in our original data and we want to extract the most relevant ones to predict the response, the ones which have a strong correlation with the variable Fat. And also using too many variables may cause overfitting so we prefer Lasso model with cross validation.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
# Required libraries
library(readxl)
library(kknn)
library(ggplot2)
library(MASS)
library(glmnet)
# Importing data
spambase <- read_xlsx("spambase.xlsx")
tecator <- read_xlsx("tecator.xlsx")
# Assignment 1
# Part 1.1

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

# Part 1.2
# Modeling

model_glm <- glm(Spam~., family = "binomial", data = train)

# Prediction for test
test_data <- ifelse(predict(model_glm, test, type= "response")  > 0.5, 1, 0)

# confusion matrix for test
Con_mat_test <- table(test$Spam, test_data)

# misclasification rate for test
mis_rate_test <- 1- sum(diag(Con_mat_test))/sum(Con_mat_test)

# Prediction for train
train_data <- ifelse(predict(model_glm, train, type="response")  > 0.5, 1, 0)

#  confusion matrix for train
Con_mat_train <- table(train$Spam, train_data)

# misclassification rate for train
mis_rate_train <- 1- sum(diag(Con_mat_train))/sum(Con_mat_train)

Con_mat_test
mis_rate_test
Con_mat_train
mis_rate_train
# Part 1.3
# Prediction for test
test_data09 <- ifelse(predict(model_glm, test, type= "response")  > 0.9, 1, 0)

# confusion matrix for test
```

```r
Con_mat_test09 <- table(test$Spam, test_data09)

# misclasification rate for test
mis_rate_test09 <- 1- sum(diag(Con_mat_test09))/sum(Con_mat_test09)

# Prediction for train
train_data09 <- ifelse(predict(model_glm, train, type= "response")  > 0.9, 1, 0)

# # confusion matrix for train
Con_mat_train09 <- table(train$Spam, train_data09)

# misclassification rate for train
mis_rate_train09 <- 1- sum(diag(Con_mat_train09))/sum(Con_mat_train09)
Con_mat_test09
mis_rate_test09
Con_mat_train09
mis_rate_train09
# Part 1.4

model_kknn30 <- kknn(factor(Spam) ~., train, test, k = 30)

# confusion matrix for test kknn
Con_mat_test_kknn30 <- table(test$Spam, model_kknn30$fitted.values)

# misclasification rate for test
mis_rate_test_kknn30 <- 1- sum(diag(Con_mat_test_kknn30))/sum(Con_mat_test_kknn30)

# confusion matrix for train kknn
Con_mat_train_kknn30 <- table(train$Spam, model_kknn30$fitted.values)

# misclasification rate for train
mis_rate_train_kknn30 <- 1- sum(diag(Con_mat_train_kknn30))/sum(Con_mat_train_kknn30)
Con_mat_train_kknn30
mis_rate_train_kknn30
Con_mat_test_kknn30
mis_rate_test_kknn30
# Part 1.5
model_kknn_k1 <- kknn(factor(Spam) ~., train, test, k = 1)

# confusion matrix for test kknn
Con_mat_test_kknn_k1 <- table(test$Spam, model_kknn_k1$fitted.values)

# misclasification rate for test
mis_rate_test_kknn_k1 <- 1- sum(diag(Con_mat_test_kknn_k1))/sum(Con_mat_test_kknn_k1)

# confusion matrix for train kknn
Con_mat_train_kknn_k1 <- table(train$Spam, model_kknn_k1$fitted.values)

# misclasification rate for train
mis_rate_train_kknn_k1 <- 1- sum(diag(Con_mat_train_kknn_k1))/sum(Con_mat_train_kknn_k1)


Con_mat_train_kknn_k1
```

```r
mis_rate_train_kknn_k1
Con_mat_test_kknn_k1
mis_rate_test_kknn_k1
# Assignment 3

cross_validation <- function(X,Y,Nfolds){

  df <- data.frame()
  cv_vec <- vector()
  selectedFeatures_vec <- vector()

  # feature selection
  for (i in 1:NCOL(X)){

    combinationforfeatures <- combn(ncol(X), i)

    for (j in 1:ncol(combinationforfeatures)){

      selectedFeature <- combinationforfeatures[,j]

      selected_X <- X[,selectedFeature, drop = FALSE]

      k_fold <- split(1:nrow(selected_X), sort((1:nrow(selected_X))%% Nfolds))

      # initializing a vector

      C_V <- 0

      for(k in 1:Nfolds) {

        # extracting rows
        fold <- k_fold[[k]]

        # actual split of the data
        data.train <- as.matrix(selected_X[-fold, , drop = FALSE])
        data.train_y <- as.matrix(Y[-fold, , drop = FALSE])

        data.test  <- as.matrix(selected_X[fold, , drop = FALSE])
        data.test_y <- as.matrix(Y[fold, , drop = FALSE])

        data.train <- cbind(data.train, rep(1,nrow(data.train)))
        data.test <- cbind(data.test, rep(1, nrow(data.test)))

        # train and test your model with data.train and data.test
        coeff_w_hat <- solve(t(data.train) %*% data.train) %*% (t(data.train) %*% data.train_y)

        # prediction
        prediction <- data.test %*% coeff_w_hat

        # calculate cross validaition score for each selected feature
        C_V <- C_V + mean((data.test_y-prediction)^2)
      }
      C_V <- C_V/Nfolds
      # rbinding to get the data frame
      numberoffeatures <- length(selectedFeature)
      cv_vec <- c(cv_vec, C_V)
      selectedFeatures_vec <- as.vector(selectedFeature)
      combinations <- paste(as.character(selectedFeatures_vec), collapse=", ")
      df_comb_CV <- data.frame(Combinations = combinations,CV = C_V,
                               Count_of_features =numberoffeatures)
      df <- rbind(df, df_comb_CV)
    }

  }

  # Optimal subset of features
  best_feature <- df$Combinations[which(df$CV == min(df$CV))]
  best_cv <- df$CV[which(df$CV == min(df$CV))]
  cat(paste("Best Subset: ",best_feature,"   CV: ", best_cv))

  plot_df <- ggplot(df) + geom_point(aes(x=Count_of_features, y=CV))+
    ggtitle("CV scores against number of features")
  print(plot_df)
  return(df)
}

set.seed(12345)
swiss <- swiss[sample(nrow(swiss)), ]
c <- cross_validation(swiss[,2:6], swiss[,1, drop = FALSE], 5)

# Assignment 4
# Part 4.1
ggplot(tecator, aes(x = Moisture, y = Protein)) + geom_point()+ ggtitle("Moisture versus Protein")

# Part 4.3
n=dim(tecator)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train_tecator=tecator[id,]
validation_tecator=tecator[-id,]

train_MSE <- c()
validation_MSE <- c()
for (i in 1:6){
model <- lm(formula= Moisture~poly(Protein, degree = i, raw = T), data = train_tecator)
train_MSE[i] <- mean(model$residuals^2)

# prediction and calculating MSE for validation = mean(y_hat-y_test)^2
validation_MSE[i] <- mean((predict(model, validation_tecator)-validation_tecator$Moisture)^2)
}
train_MSE
validation_MSE

ylim = c(31, 35)

plot(x=1:6,y=train_MSE, col= "green", ylim= ylim , ylab = "MSE",main ="train and validation MSE")
```

```r
lines(x=1:6,y=train_MSE, col= "green")
points(x=1:6,y=validation_MSE , col= "red" )
lines(x=1:6,y=validation_MSE, col= "red")
legend("topright", legend = c("train_MSE","validation_MSE"), col=c("green","red"), lty = 1, cex=0.8)

# Part 4.4
a <- tecator[,2:102]
fit <- lm(Fat~.,data = a)

#The "maximal" model is used as input for the stepAIC() function:
step <- stepAIC(fit, direction="both",trace = FALSE)
coeff_aics = step$coefficients
n_coeff_aics = length(coeff_aics)
# Part 4.5
covariates=scale(tecator[,2:101])
response=scale(tecator$Fat)
Ridge.model=glmnet(as.matrix(covariates), response, alpha=0,family="gaussian")
plot(Ridge.model, xvar="lambda", label=TRUE, main = "Dependence of model coefficients on log lambda")

# Part 4.6
Lasso.model=glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")

plot(Lasso.model, xvar="lambda", label=TRUE)
# Part 4.7
c.v_Lasso.model=cv.glmnet(as.matrix(covariates), response , lambda = seq(0,3,0.001) ,alpha=1,family="ga
plot(c.v_Lasso.model, xvar="lambda", label=TRUE, main = "Dependence of CV score on log lambda")
coef(c.v_Lasso.model, s="lambda.min")
c.v_Lasso.model$lambda.min
```