Classification & Regression

Rakhshanda Jabeen

03/08/2020

Contents

Summary

This group report is about implementation of one of the machine learning classes, supervised learning. In every task we will give fixed input data and out expectation. By using fixed input data we will train our model to calculate the expected output. We preprocessed the data to select the best features by using cross validation technique. For the categorical data we will use classification methods e.g. K-nn and Logistic regression and for the quantative data we will use regression methods e.g. Ridge and Linear regression etc. Every task contains the graphical representation of the final results.

1. Spam Qualification with nearest neighors

This assignment is based on binomialy distributed data. We will implement two different models to analyse our data. As the data type is categorical we will implement logistic regression and K-nn methods, then we will compare their confusion matrices to analyse the errors of each model. We will consider the model with lowest error.

1.1 Data Import

We have divided data in two halves using sample built-in function of R. The first half of the data we are using as training data and second half is used as test data.

1.2 Logistic Regression Model (Classification Principle > 0.5)

In this task we have implemented logistic regression to create our model. By using our trained model we have predicted our expected output. In logistic regression we analyze our model by using confusion matrix.

$$\hat{Y} = 1 \; if \; p(Y = 1|X) > 0.5, otherwise \; \hat{Y} = 0$$

Confusion Matrix of Spam for train data

	0	1
0	803	81
1	142	344

Confusion Matrix of Spam for test data

	0	1
0	791	97
1	146	336

Missclassification Rate

	MisclassificationRate
Train data	16.28
Test Data	17.74

According to the above mentioned confusion matrix missclassifcation rate of our training data is less than our test data. Hence, our model is not optimal enough for new data other than training data.

1.3 Logistic Regression Model (Classification Principle > 0.8)

$$\hat{Y} = 1\ if\ p(Y = 1|X) > 0.8, otherwise\ \hat{Y} = 0$$

Confusion Matrix of Spam for train data

	0	1
0	941	335
1	4	90

Confusion Matrix of Spam for test data

	0	1
0	926	367
1	11	66

Missclassification Rate

	MisclassificationRate
Train data	24.74
Test Data	27.59

	F1Score
F1 Score 80%	0.2588235
F1 Score 50%	0.7344262

By comparing task *1.2* and *1.3* for different thresholds 50% and 80%, we have concluded that by increasing the classification principle of our model has become more restricted to classify 1's.

We have used F1 score to check the predictive power of our model for test data. F1 score uses the recall and precision rate of the test data to check the model's accuracy. F1 score, 1 means the perfect precision and recall while 0 is the worst.

1.4 KNN (k=30)

In *1.4* and *1.5* we are using K nearest nabour to create the model. We have repeated our K-NN model for two different K values to evaluate the model by itself. By decreasing the K our model has become overfit.

Confusion Matrix of Spam for train data

	0	1
0	854	86
1	91	339

Confusion Matrix of Spam for test data

	0	1
0	772	159
1	165	274

Missclassification Rate

	MisclassificationRate
Train data	12.92
Test Data	23.65

By comparing the result with *1.2*, K-NN has the higher error rate for test data as compare to logistic regression. Hence, we have concluded that logistic regression is better than the K-NN

1.5 KNN (k=1)

Confusion Matrix of Spam for train data

	0	1
0	945	0
1	0	425

Confusion Matrix of Spam for test data

	0	1
0	708	166
1	229	267

Missclassification Rate

	MisclassificationRate
Train data	0.00
Test Data	28.83

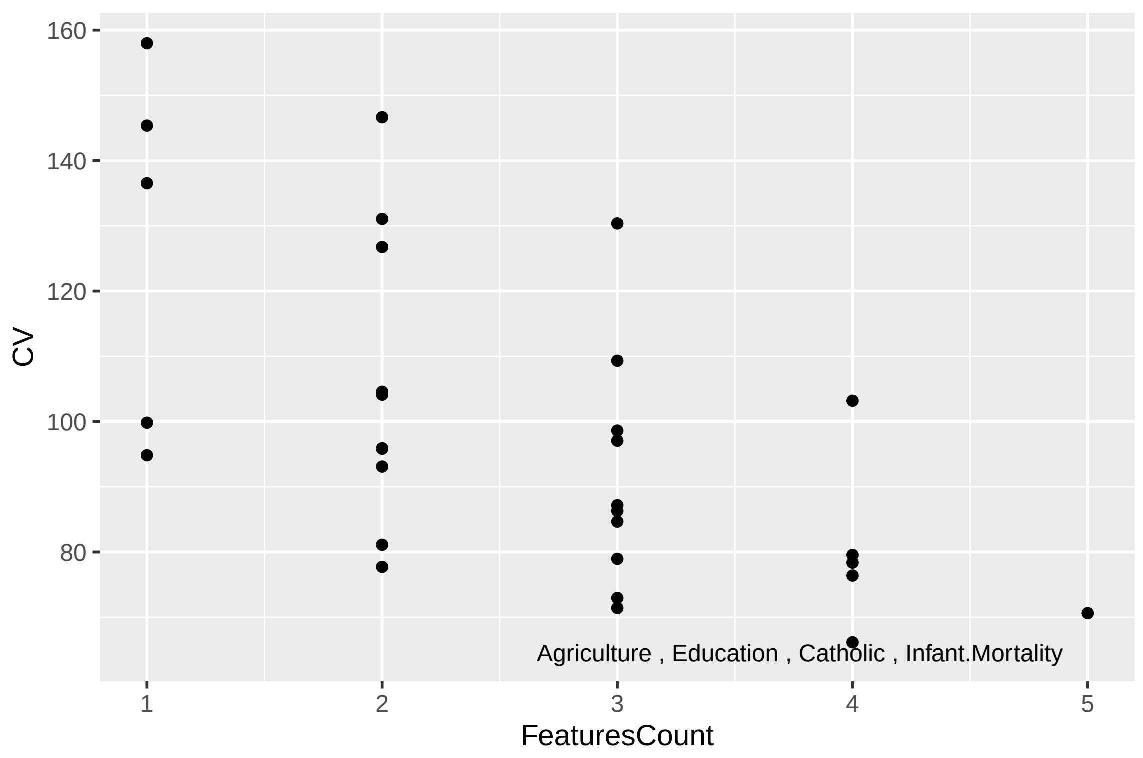
By comparing the results of *1.4* and *1.5*, when K=1, K-NN has created boundries for each data point so for the training data our model has error rate zero. Hence, we have concluded that by decreasing the K=1 our model become overfit.

2. Feature selection by cross-validation in a linear model

In this task we will process the data to choose the most important feature. We have used the cross validation to choose the features which has high impact. We have used the manual linear regression formula as our model to calculate the SSE(sum of squared error). We will consider the feature with minimum SSE error as our best feature.

```
$minCV
[1] "66.15368"

$Feature
[1] "Agriculture , Education , Catholic , Infant.Mortality"
```

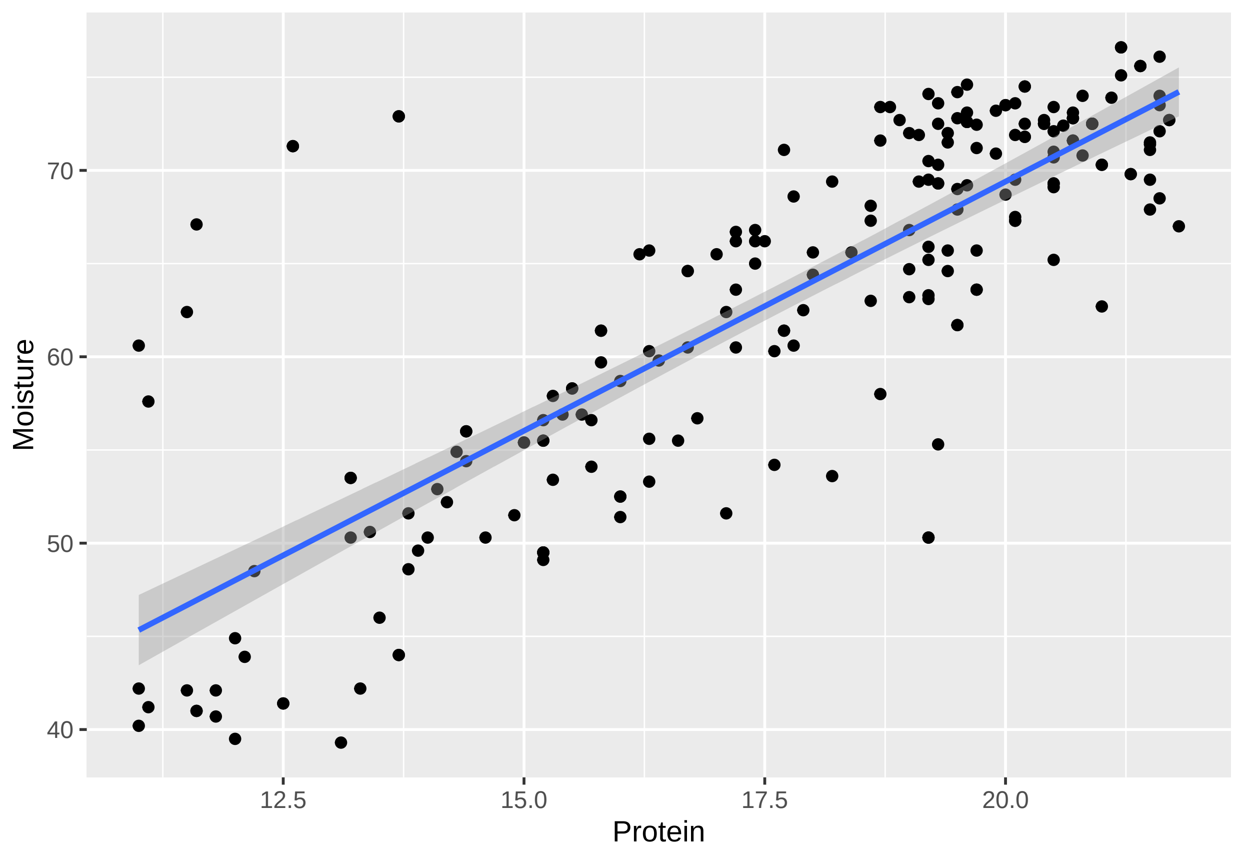


3. Linear regression and regularization

We are given data of tecator. In this task we have to check the complexity of different models and checking how we can decrease the complexity of model. We will analyze if our model is going to fit the data. Also we will use different methods for feature selection e.g Ridge Regression, Lasso and setpAIC.

3.1 Import Data

We have checked the dependency of both paramters Moisture and Protein. We have concluded that both Moisture and Protein are linearly dependent on each other and well defined by linear model.



3.2 Report a probabilistic model that describes M_i

$$p(M_i|x, w) = N(w_0 + \sum_{j=1}^{i} w_j x, \sigma_i^2)$$

$$where\ E(M_i|x, w) = w_0 + \sum_{j=1}^{i} w_j x$$

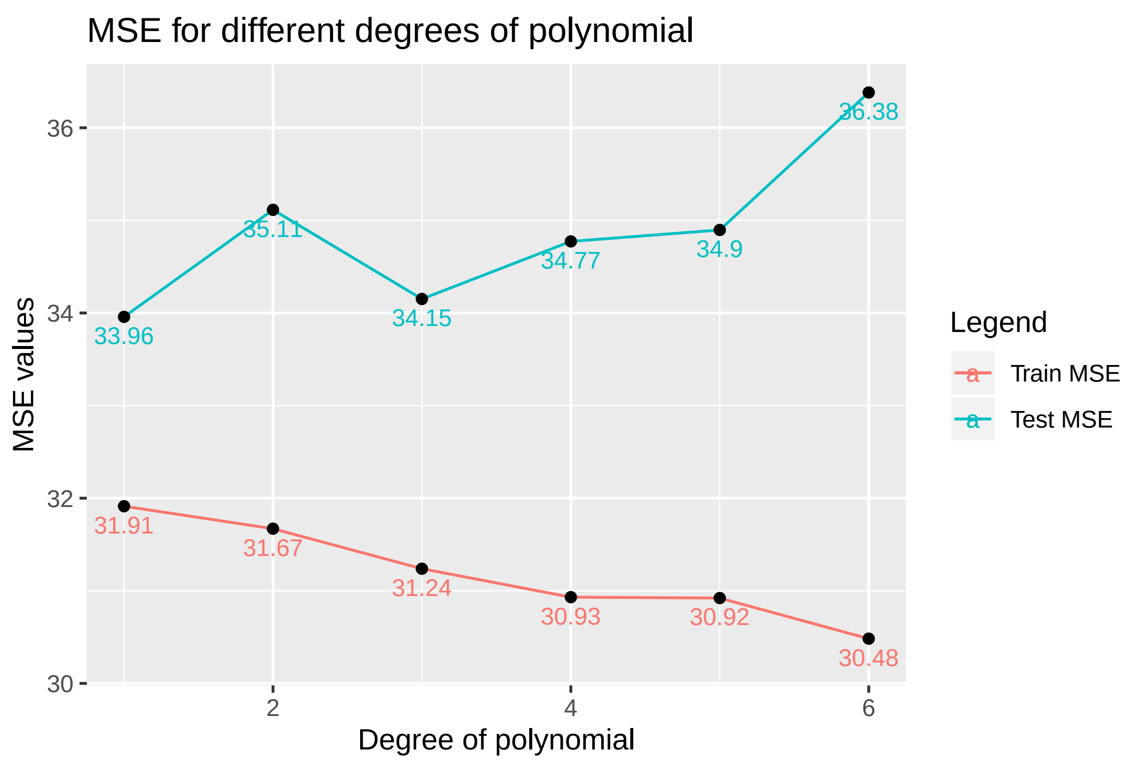
$$\sigma_i^2\ is\ the\ variance.$$

$$i = degree\ of polynomial$$

Why is it appropriate to use MSE criterion when fitting this model to a training data? Our data is normally distributed. We use SSE to analyze our model and check if our model fits the training data or not. As SSE is dependendent on size of data that's why it is difficult to directly analyze the values for different size of data. So to avoid this problem we take mean of our SSE and then it is called MSE. So it is approapriate to use MSE criterion to fit our training data.

3.3 M_6 polynomial

Following graph shows the training and testing data MSE according to the degree of Polynomial.



Above displayed graph shows that the graph is increasing for training data according to the degree of ploynomial and decreasing for the test data. We select model which has least MSE. According to the graph our M1 model is the best model, as much as we increase degree of polynomial, MSE is increasing as well. As the degree of polynomial increasing our model is becoming overfit. M1 is flexible enough to well generalize our model for new data.

Bias-variance tradeoff is the set of models. High complex models have low bias and high variance means overfitting the model while low complex models have high bias and low variance which underfits the model. Mediam complex model has mediam bias and median variance. As we have to choose the model which is neither overfit nor underfit, in our case M1 has the mediam bias and mediam variance. Hence our best choice of model selection according to Bias-variance tradeoff is M1.

3.4 StepAIC

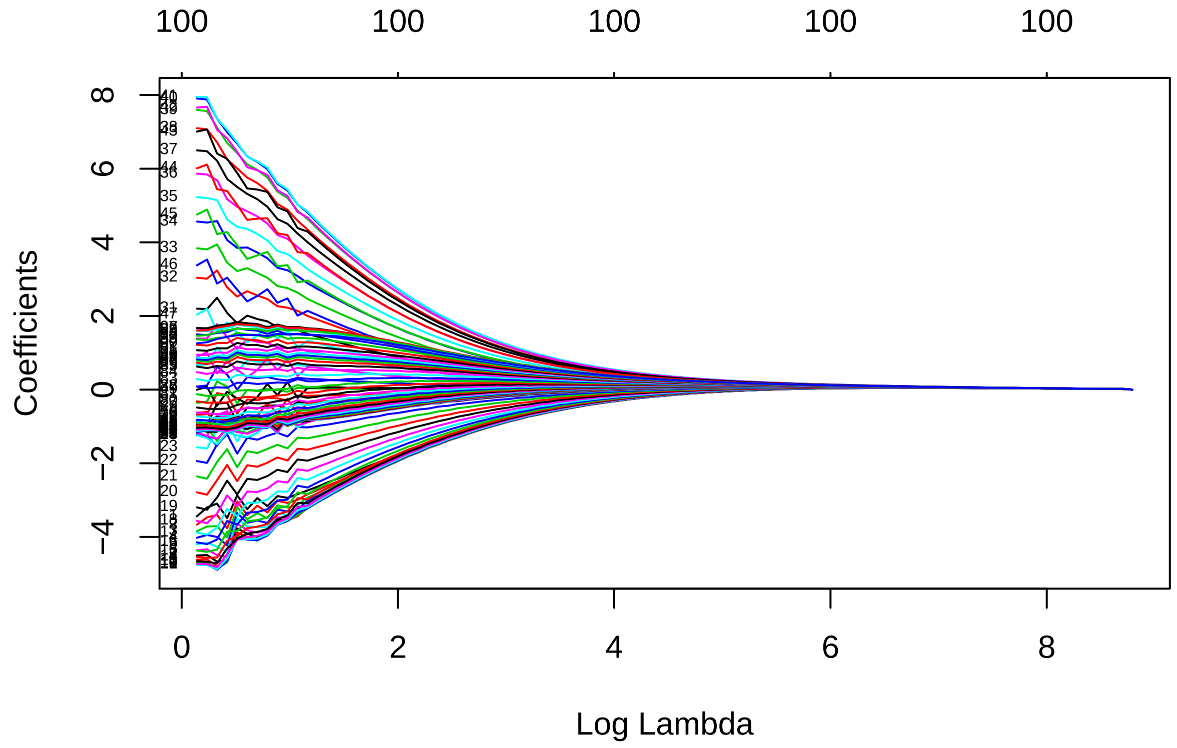
In given data we are given big list of features. To process our data we have to selct the best features to decrease the complexity of our model by using setpAIC. We are creating linear model before applying stepAIC, where we have used Fat as our response feature and channels as our predictive features.

```
## [1] "Select 63 Coefficients out of 100."
```

3.5 Ridge regression

In this task we have created the ridge regression using Fat as response and channels as our predictive feature.

This graph depicts the dependency of coefficients on $log(\lambda)$ by using Ridge Regression



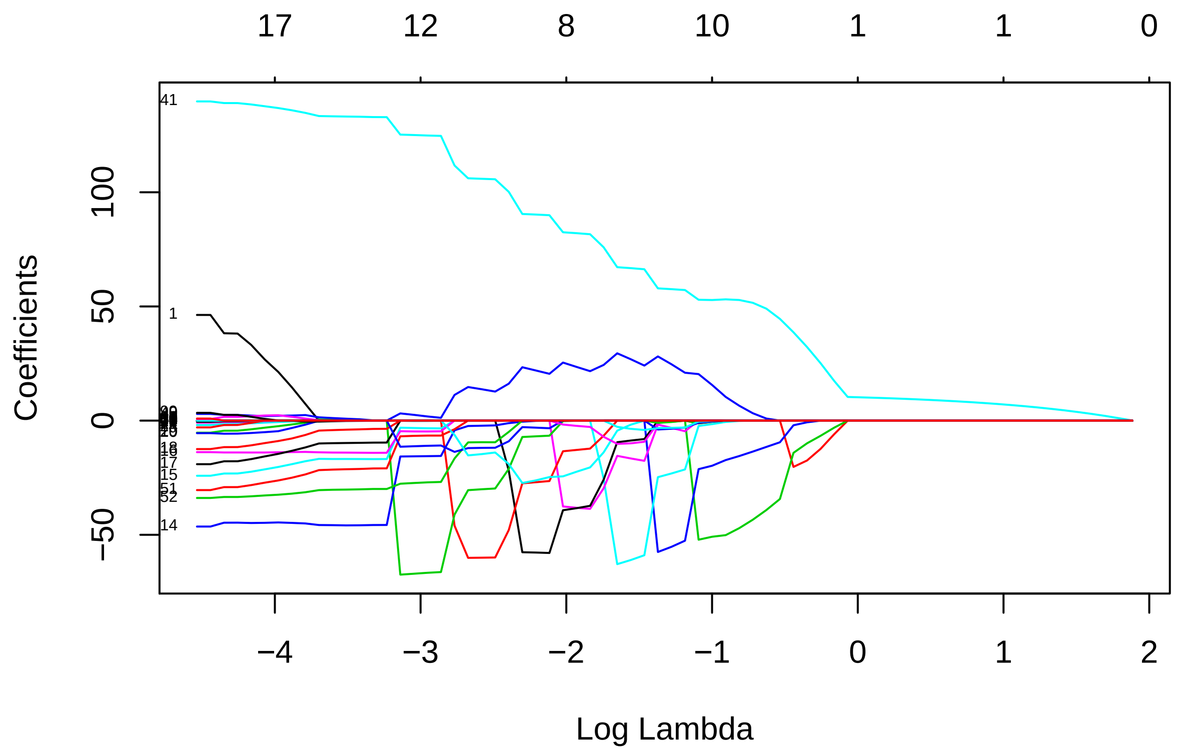
we are using rdige regression to decrease the model complexity by performing L2 regularization. In L2 regularization, we add penality which is equal to the squared of the magnitudes of coefficients and try to minimize them.

All the features which are far from the true value, we enforce them to be small. By implementing ridge regression on our data we rare decreasing the model complxity while keeping all the features in the model.

3.6 Lasso

In this task we have used the same response and predictor as we have used in task *3.4* and *3.5*.

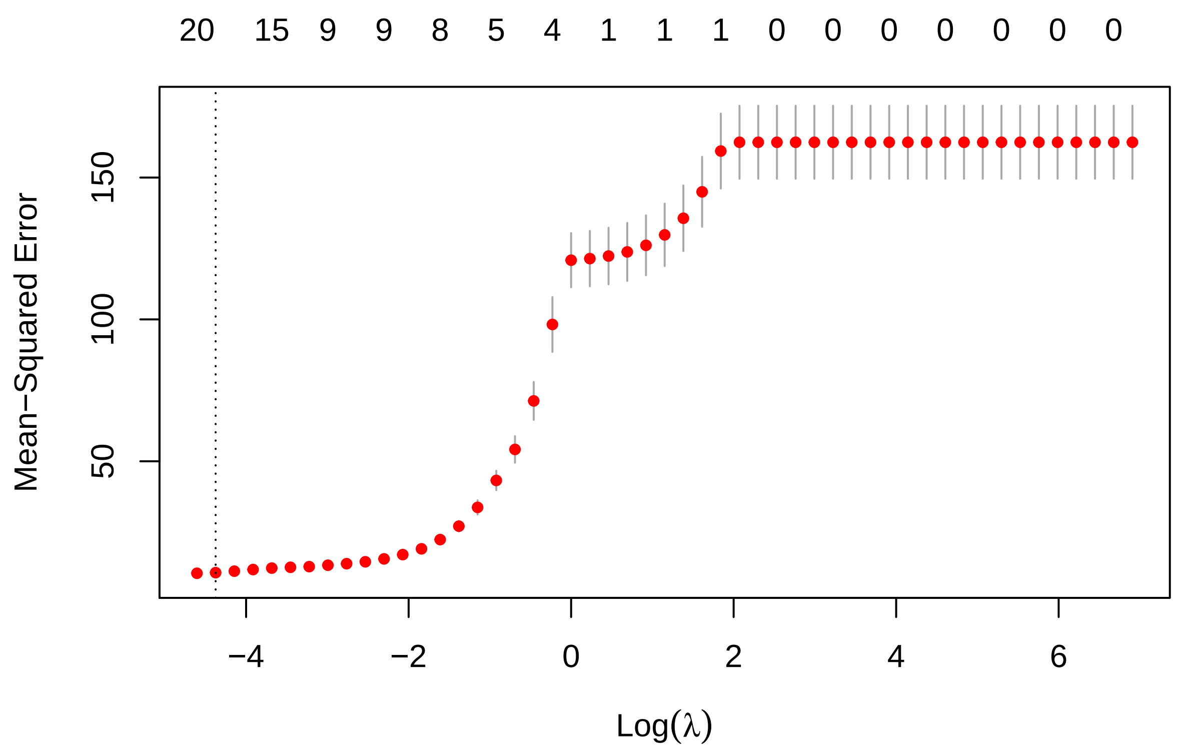
Below graph shows the dependendency of coefficients on $log(\lambda)$ by using Lasso Regression.



We have implemented Lasso Regression to decrease the model complexity by performing L1 regularization. L1 regularization adds the penality equal to the absolute of the magnitude of the coeeficients.

Lasso regression overcomes the disadvantages of ridge regression by not only pushing the high values of the coefficients but acutally setting irrelevant variables to zero

3.7 CV Lasso



```
## [1] "optimal lambda 0"
```

```
## [1] "Select 100 Coefficients out of 100."
```

Cross validation in Lasso is used to choose the optimal lambda. We choose optimal lambda to avoid the overfitting and underfiting of the model. From the graph we can see that after $log(\lambda) = 2$ means squared error has become constant, which means all the coefficients of the predictors have become 0 and only the constant term exist.

3.8 Comparison of task 4 and task 7

In task *3.4* we have used stepAIC for feature selection and in result we got 63 predictors with high impact. However, in task *3.7* we have used lasso which has returned all the predictors. In our case, value of λ is equal to zero so model will give us back the least squares fit.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE)
library(readxl)
library(kknn)
library(kableExtra)
library(MASS)
library(knitr)
library(kknn)
library(ggplot2)
library(MASS)
library(tidyverse)
library(glmnet)
RNGkind(sample.kind = "Rounding")
set.seed(12345)
data =  read_excel("E:/Machine Learning/lab_01Block01/data/spambase.xlsx")
data$Spam <- as.factor(data$Spam)
n = nrow(data)
id_50 = sample(1:n,floor(n*0.5))
test_spam_data = data[-id_50,]
train_spam_data = data[id_50,]
model = glm(formula = Spam ~. ,data = train_spam_data,family = binomial)
train_spam_data$prediction_prob = predict(model ,type = "response")
test_spam_data$prediction_prob = predict(model, newdata = test_spam_data , type = "response")
train_predict = ifelse(train_spam_data$prediction_prob > 0.5, 1L, 0L)
test_predict = ifelse(test_spam_data$prediction_prob > 0.5, 1L, 0L)
#confusion matrix
confusionTrain_matrix = table(train_predict,train_spam_data$Spam)
confusionTest_matrix = table(test_predict,test_spam_data$Spam)
#missclassification rate
missrate_train = mean(train_predict != train_spam_data$Spam)
missrate_test = mean(test_predict != test_spam_data$Spam)
kable(confusionTrain_matrix) %>% kable_styling(latex_option = "striped")
kable(confusionTest_matrix) %>% kable_styling(latex_option = "striped")
missrate_train = missrate_train * 100
missrate_test = missrate_test * 100
missrate_train = format(missrate_train,digits = 2,nsmall=2)
missrate_test = format(missrate_test,digits = 2,nsmall=2)
missRate = data.frame(MisclassificationRate = c(missrate_train,missrate_test))
rownames(missRate) = c("Train data","Test Data")
kable(missRate) %>% kable_styling(latex_option = "striped")
# classification principle = 80%
train_predict_80 = ifelse(train_spam_data$prediction_prob > 0.8, 1L, 0L)
test_predict_80 = ifelse(test_spam_data$prediction_prob > 0.8, 1L, 0L)
#confusion matrix
confusionTrain_matrix_80 = table(train_predict_80, train_spam_data$Spam)
confusionTest_matrix_80 = table(test_predict_80, test_spam_data$Spam)
#missclassification rate
missrate_train_80 = mean(train_predict_80 != train_spam_data$Spam)
missrate_test_80 = mean(test_predict_80 != test_spam_data$Spam)
kable(confusionTrain_matrix_80) %>% kable_styling(latex_option = "striped")
kable(confusionTest_matrix_80) %>% kable_styling(latex_option = "striped")
missrate_train_80 = missrate_train_80 * 100
```

```r
missrate_test_80 = missrate_test_80 * 100
missrate_train_80 = format(missrate_train_80,digits = 2,nsmall=2)
missrate_test_80 = format(missrate_test_80,digits = 2,nsmall=2)
missRate = data.frame(MisclassificationRate = c( missrate_train_80,missrate_test_80))
rownames(missRate) = c("Train data","Test Data")
kable(missRate) %>% kable_styling(latex_option = "striped")
#
#                Actual
#                 0   1
# prediction  0  TN  FN
#             1  FP  TP
#
#
#
# Recall = TP/(TP+FN)
# Precision = TP/(TP+FP)
TP_test_80 = confusionTest_matrix_80[2,2]
FN_test_80 = confusionTest_matrix_80[1,2]
FP_test_80 = confusionTest_matrix_80[2,1]
Recall_test_80 = TP_test_80/(TP_test_80 + FN_test_80)
Precision_test_80 = TP_test_80/(TP_test_80 + FP_test_80)
F1_test_80 = 2 *Recall_test_80*Precision_test_80/(Recall_test_80 + Precision_test_80)
TP_test = confusionTest_matrix[2,2]
FN_test = confusionTest_matrix[1,2]
FP_test = confusionTest_matrix[2,1]
Recall_test = TP_test/(TP_test + FN_test)
Precision_test = TP_test/(TP_test + FP_test)
F1_test = 2* Recall_test * Precision_test/(Recall_test + Precision_test)
F1Score = data.frame(F1Score = c( F1_test_80,F1_test))
rownames(F1Score) = c("F1 Score 80%","F1 Score 50%")
kable(F1Score) %>% kable_styling(latex_option = "striped")
knn_model_train = kknn(formula = Spam~.,train = train_spam_data,test = train_spam_data,k = 30)
knn_model_test = kknn(formula = Spam~.,train = train_spam_data,test = test_spam_data,k = 30)
confusionMatrix_train_knn = table(knn_model_train$fitted.values,train_spam_data$Spam)
error_train_knn = mean(knn_model_train$fitted.values != train_spam_data$Spam)
confusionMatrix_test_knn  =  table(knn_model_test$fitted.values,test_spam_data$Spam)
error_test_knn = mean(knn_model_test$fitted.values != test_spam_data$Spam)
kable(confusionMatrix_train_knn) %>% kable_styling(latex_option = "striped")
kable(confusionMatrix_test_knn) %>% kable_styling(latex_option = "striped")
error_train_knn = error_train_knn * 100
error_test_knn = error_test_knn * 100
error_train_knn = format(error_train_knn,digits = 2,nsmall=2)
error_test_knn = format(error_test_knn,digits = 2,nsmall=2)
missRate = data.frame(MisclassificationRate = c( error_train_knn,error_test_knn))
rownames(missRate) = c("Train data","Test Data")
kable(missRate) %>% kable_styling(latex_option = "striped")
knn_model_train_1 = kknn(formula = Spam~.,train = train_spam_data,test = train_spam_data,k = 1)
knn_model_test_1 = kknn(formula = Spam~.,train = train_spam_data,test = test_spam_data,k = 1)
confusionMatrix_train_knn_1 = table(knn_model_train_1$fitted.values,train_spam_data$Spam)
error_train_knn_1 = mean(knn_model_train_1$fitted.values != train_spam_data$Spam)
confusionMatrix_test_knn_1  =  table(knn_model_test_1$fitted.values,test_spam_data$Spam)
error_test_knn_1 = mean(knn_model_test_1$fitted.values != test_spam_data$Spam)
kable(confusionMatrix_train_knn_1) %>% kable_styling(latex_option = "striped")
kable(confusionMatrix_test_knn_1) %>% kable_styling(latex_option = "striped")
error_train_knn_1 = error_train_knn_1 * 100
error_test_knn_1 = error_test_knn_1 * 100
error_train_knn_1 = format(error_train_knn_1,digits = 2,nsmall=2)
error_test_knn_1 = format(error_test_knn_1,digits = 2,nsmall=2)
missRate = data.frame(MisclassificationRate = c( error_train_knn_1,error_test_knn_1))
rownames(missRate) = c("Train data","Test Data")
kable(missRate) %>% kable_styling(latex_option = "striped")
mylinReg = function(X,Y)
{
  beta = solve(t(X) %*% X) %*% t(X) %*% Y
  return (beta)
}
myCV = function(X,Y,Nfolds)
{
  columnCount = ncol(X)
  n = nrow(X)
  MSE=numeric(2^columnCount-1)

  FeaturesCount=numeric(2^columnCount-1)
  Features=character(2^columnCount-1)

  set.seed(12345)

  random_fold  = sample(folds,size = n,replace = TRUE)
  curr = 0
  for(k in 1:columnCount){
    combs = combn(1:columnCount, k)
    for(j in 1:ncol(combs)){
      columnsIndex = combs[, j]
      XSub =  as.matrix(X[, columnsIndex])
      XSub = cbind(1,XSub)
      SSE = 0
      for(fold in c(1:Nfolds))
      {
        foldIndex= which(random_fold == fold)
        # train Data
        XSubTrain = XSub[-foldIndex,]
        YTrain = Y[-foldIndex]

        #Test Data
        XSubTest = XSub[foldIndex,]
        YTest = Y[foldIndex]

        beta = mylinReg(XSubTrain,YTrain)

        predY = XSubTest %*% beta
        SSE = SSE + sum((predY - YTest)^2)

      }
      curr = curr + 1

      MSE[curr] = SSE/n
      Features[curr]=paste(colnames(X)[columnsIndex],collapse = " , ")
      FeaturesCount[curr] = length(colnames(X)[columnsIndex])
    }
  }
  i=which.min(MSE)
  return(list(CV=MSE, Features=Features,optIndex = i,FeaturesCount = FeaturesCount))
}
X = as.matrix(swiss[,2:6])
Y = swiss[[1]]
folds = 5
CVScore = myCV(X,Y,folds)
list(minCV= format(CVScore$CV[CVScore$optIndex]),Feature = CVScore$Features[CVScore$optIndex])
df = data.frame(CVScore)
ggplot(df,aes(x=FeaturesCount ,y=CV)) + geom_point()+ geom_text(aes(label=ifelse(CV==min(df$CV),as.chara
tecator_data = read_excel("E:/Machine Learning/lab_01Block01/data/tecator.xlsx")
ggplot(tecator_data,aes(x=Protein,y =Moisture )) +geom_point() +geom_smooth(method = "lm")
set.seed(12345)
n = nrow(tecator_data)
id_50 = sample(1:n,floor(n*0.5))
test_tecator_data = tecator_data[id_50,]
train_tecator_data = tecator_data[-id_50,]
poly = 6
#newTest_data = matrix(rep(1, nrow(test_tecator_data)), ncol = 1)
#newTrain_data = matrix(rep(1,nrow(train_tecator_data)),ncol = 1)
newTest_data = matrix(nrow = nrow(test_tecator_data), ncol=0)
newTrain_data = matrix(ncol=0,nrow = nrow(train_tecator_data))
for(m in 1:poly)
{
  newTest_data = cbind(newTest_data,as.matrix(test_tecator_data$Protein)^m)
  newTrain_data = cbind(newTrain_data,as.matrix(train_tecator_data$Protein)^m)
}
colnames(newTest_data) = c("pow_1","pow_2","pow_3","pow_4","pow_5","pow_6")
colnames(newTrain_data) = c("pow_1","pow_2","pow_3","pow_4","pow_5","pow_6")
polyErrors = data.frame()
for(m in 1:(poly))
{
  colNames = colnames(newTest_data)[1:m]
  formula = paste(colNames,collapse = '+')
  formula = paste("Moisture","~",formula)
  formula = as.formula(formula)


  pow_1 = as.data.frame(newTrain_data)[,1:m]
  names(pow_1) = colnames(newTrain_data)[1:m]
  newSubTrain_data = cbind(pow_1,train_tecator_data['Moisture'])
  newSubTrain_data = as.data.frame(newSubTrain_data)

  lin_model = lm(formula,data = newSubTrain_data)

  pow_1 = as.data.frame(newTest_data)[,1:m]
  names(pow_1) = colnames(newTest_data)[1:m]
  newSubTest_data = cbind(pow_1,test_tecator_data$Moisture)
  newSubTest_data = as.data.frame(newSubTest_data)
  Y_pred = predict(lin_model,newdata=newSubTest_data)
  SSE_pred =  mean((Y_pred - test_tecator_data$Moisture)^2)

  Y_pred_train = predict(lin_model,type = "response")
  SSE_pred_train =  mean((Y_pred_train - train_tecator_data$Moisture)^2)


  polyErrors = rbind(polyErrors,data.frame(Polynomial = m,trainSSE = SSE_pred_train,testSSE = SSE_pred)
}
ggplot(polyErrors) + geom_line(aes(x=Polynomial ,y=trainSSE,color="1"))+ geom_line(aes(x=Polynomial ,y=
  geom_point(aes(x=Polynomial ,y=trainSSE)) + geom_point(aes(x=Polynomial ,y=testSSE))+
  geom_text(aes(x=Polynomial, y=trainSSE, label=round(trainSSE, 2), color="1"),
            nudge_y =-0.2, size =3) +
  geom_text(aes(x=Polynomial, y=testSSE, label=round(testSSE, 2), color="2"),
            nudge_y =-0.2, size =3) +
  xlab("Degree of polynomial") + ylab("MSE values")+
  scale_color_discrete(name = "Legend", labels = c("Train MSE", "Test MSE")) +
  ggtitle("MSE for different degrees of polynomial")

stepData  = tecator_data[,-c(1,104,103)]
stepModel = lm(Fat ~ ., data = stepData)
step = stepAIC (stepModel,trace = FALSE,direction = "both")
paste("Select",length(step$coefficients)-1,"Coefficients out of 100.")
yData = tecator_data %>% select(Fat) %>% data.matrix()
xData = tecator_data %>% select(-c(Fat,Protein,Moisture,Sample)) %>% data.matrix()
RigModel =  glmnet(xData, yData, alpha = 0)
plot(RigModel, xvar="lambda", label=TRUE)
lassoModel =  glmnet(xData, yData, alpha = 1)
plot(lassoModel, xvar="lambda", label=TRUE)
lambdas = 10^seq(3, -2, by = -.1)
lambdas[length(lambdas)+1] = 0
lassoModel.cv =  cv.glmnet(xData, yData, alpha = 1,type.measure = "mse",lambda = lambdas)
plot(lassoModel.cv)
optLambda = lassoModel.cv$lambda.min
paste("optimal lambda",optLambda)
optLasso =  glmnet(xData, yData, alpha = 1,lambda = optLambda,standardize = TRUE)
paste("Select",length(coef(optLasso))-1,"Coefficients out of 100.")
```