Machine Learning Lab 1

mimte666 - Mim Kemal Tekin

25/11/2018

Contents

Assignment 1: Spam classification with nearest neighbors

The data file spambase.xlsx contains information about the frequency of various words, characters etc for a total of 2740 e-mails. Furthermore, these e-mails have been manually classified as spams (spam = 1) or regular e-mails (spam = 0).

Task 1.1

Import the data into R and divide it into training and test sets (50%/50%)

```r
####################
##### TASK 1.1 #####
####################
library(readxl)

# import data
df_spambase = read_xlsx("../spambase.xlsx")

# divide data into training and test sets
n = dim(df_spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
# training data
train = df_spambase[id,]
# test data
test = df_spambase[-id,]

# head(train)
# head(test)
```

Task 1.2

Use logistic regression (functions glm(), predict()) to classify the training and test data by the classification principle

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

and report the confusion matrices (use table()) and the misclassification rates for training and test data. Analyse the obtained results.

Table 1: Test with Train Data

	Predict (-)	Predict (+)
Actual (-)	803	142
Actual (+)	81	344

Table 2: Test with Test Data

	Predict (-)	Predict (+)
Actual (-)	791	146
Actual (+)	97	336

Table 3: Misclassification Rates

	Train Data	Test Data
Misclassification	0.1627737	0.1773723
Accuracy	0.8372263	0.8226277

Logistic regression is used for predicting variables which follow binomial distribution. Binomial distribution is defined as $P(X=k) = \binom{n}{p} p^k (1-p)^{n-k} \sim B(n,p)$. In binomial distribution, each values have one out of two possible values. In logistic regression, we try to predict by our features the probability of being which one of two different values. Linear regression can be used for this prediction, but this prediction should be satisfy the requirements of probability which is being positive and being lower than 1. This probability prediction is defined as following formula:

$$P(Y=1|X) = \frac{e^{(\beta_0+\beta_1 x)}}{e^{(\beta_0+\beta_1 x)}+1}$$

After this, prediction of target variables can be done satisfying a threshold condition for each probability. We set positive value if probability is greater than the threshold, else set negative value.

In this logistic regression model, we used 0.5 as threshold value. Results of test with train and test data are quite similar. In table 3, we can see misclassification rates, they are close each other and misclassification rate is around 16-17%.

Task 1.3

Use logistic regression to classify the test data by the classification principle

$$\hat{Y} = 1 \text{ if } P(Y=1|X) > 0.9 \text{ , otherwise } \hat{Y} = 0$$

and report the confusion matrices (use table()) and the misclassification rates for training and test data. Compare the results. What effect did the new rule have?

Table 4: Test with Train Data

	Predict (-)	Predict (+)
Actual (-)	944	1
Actual (+)	419	6

Table 5: Test with Test Data

	Predict (-)	Predict (+)
Actual (-)	936	1
Actual (+)	427	6

Table 6: Misclassification Rates

	Train Data	Test Data
Misclassification	0.3065693	0.3124088
Accuracy	0.6934307	0.6875912

Threshold value is changed to 0.9. In Table 4 and Table 5, we can see Actual (-) predictions have almost 100% accuracy, but Actual (+) predictions are totally incorrect in tests with both test and train data. When we set too high threshold, we are changing all proportion of prediction and the model becomes more biased to the side of negativity. According to the model, we can say being non-spam is much more likely than being spam because of high threshold. In table 6, misclassification rates are close each other and it seems like it is low, but the reason of this is majority of correctly predicted values in negatives (Actual (-)). The model only predicts non-spam mails as non-spam mails correctly, and it predicts spam mails as non-spam mails which is totally wrong. The reason of "correct" predictions in Predict (-)s id model cannot predict properly and when threshold is too high, the model is biased to negative predicts.

Task 1.4

Use standard classifier kknn() with K=30 from package kknn, report the the misclassification rates for the training and test data and compare the results with step 2.

Table 7: Test with Train Data

	Predict (-)	Predict (+)
Actual (-)	807	138
Actual (+)	98	327

Table 8: Test with Test Data

	Predict (-)	Predict (+)
Actual (-)	672	265
Actual (+)	187	246

Table 9: Misclassification Rates

	Train Data	Test Data
Misclassification	0.1722628	0.329927
Accuracy	0.8277372	0.670073

k-Nearest Neighbor Classifier calculates distances between observations in train and test data. In this example, distances are calculated by Minkowski Distance Algorithm by default. This algorithm is a lazy learning method which means it does not create a model to predict, it stores the train data and it runs the algorithm in every fit. After calculating the distances or similarities between test data and train data, it takes the k observations which are the closest to current observation of test data and it predicts the target variable by majority of the results in train data.

In this task, we divide the data to 2 equal size data as train and test, k = 30. After fitting this test observations, we applied threshold to get our exact results.
We can see the success of this algorithm by testing with train data and test data separately in above.
Misclassification rates are 0.329 with test data and 0.172 with train data. Misclassification rate of test with test data is about double of test with train data. This means our model is quite good fit for train data, but when we test with another data it cannot predict well. The model is biased to training data.

When we compare these results with the results in task 1.3, we can see the model in task 1.3 is more consistent. Misclassification and accuracy rates are so close each other. The model can still predict using different test data with close accuracy, but we cannot say the same thing for the model in this task.

Task 1.5

Repeat step 4 for K=1 and compare the results with step 4. What effect does the decrease of K lead to and why?

Table 10: Test with Train Data

	Predict (-)	Predict (+)
Actual (-)	945	0
Actual (+)	0	425

Table 11: Test with Test Data

	Predict (-)	Predict (+)
Actual (-)	640	297
Actual (+)	177	256

Table 12: Misclassification Rates

	Train Data	Test Data
Misclassification	0	0.3459854
Accuracy	1	0.6540146

If K=1 like in this example, the model will be overfitted. Because this means, algorithm will search for only the closest observation in the training data, and it will take its result as a prediction by 100% probability. In Table 10, we can see result of this situation clearly. Test with train data is perfectly matched, and there is no misclassification (Table 12). When we try to test with a different test data model gives us 0.345 as misclassification rate and 0.634 as accuracy rate which are not really good. This model may be seemed like acceptable by only looking to rates of test results with test data, but it is not a good model because of overfitting to training data.
As a conclusion, decrease of K lead to overfit in model and when we decrease k slowly we can see variety of probabilities which are calculated by algorithm, will be decreased.

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

Task 3.1

Implement an R function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation without using any specialized function like lm() (use only basic R functions). Your function should depend on:
- X: matrix containing X measurements.
- Y: vector containing Y measurements.
- Nfolds: number of folds in the cross-validation.
You may assume in your code that matrix X has 5 columns. The function should plot the CV scores computed for various feature subsets against the number of features, and it should also return the optimal subset of

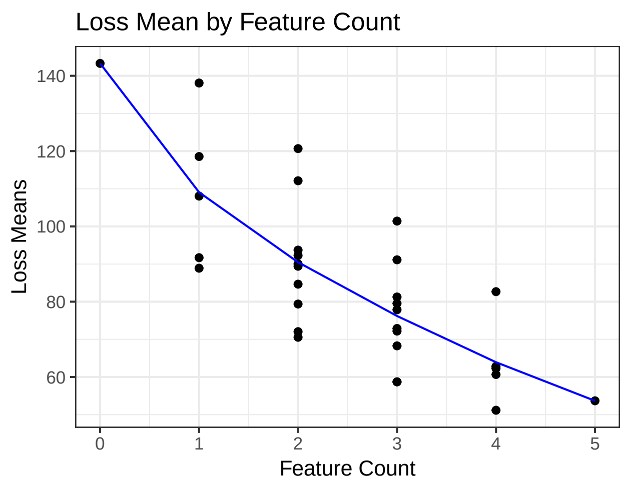
features and the corresponding cross-validation (CV) score. Before splitting into folds, the data should be permuted, and the seed 12345 should be used for that purpose.

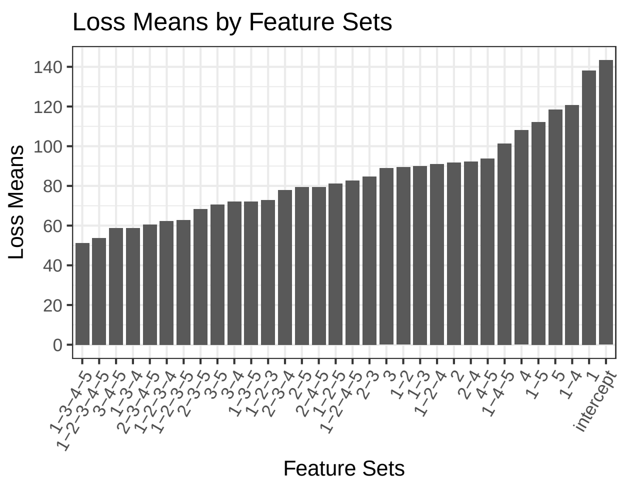
Cross-validation algorithm is a model selection algorithm. It splits whole data to k pieces. It uses all pieces as a test data once while other rest of the data as a train data. By this way, it collects model prediction errors for k combinations of data by using specific loss function for model and return their mean.
In this task, we try to choose optimal features for our regression model and we use MSE as a loss function. So we will create different models with all combinations of the features, after this we will calculate loss means of them to find the best model.
Function does respectively shuffling data, iteration of all feature combinations and in this loop it uses k-fold cross-validation algorithm to get the mean of losses. In k-fold cross-validation algorithm, we set a seed in order to get always same indexes to split data. By this, we are able to compare errors of the models which use the same portion for all folds.

The code of CV function is at appendix.

Task 3.2

Test your function on data set swiss available in the standard R repository:
- Fertility should be Y
- All other variables should be X
- Nfolds should be 5
Report the resulting plot and interpret it. Report the optimal subset of features and comment whether it is reasonable that these specific features have largest impact on the target.





```
## [1] "Optimal Features:"
```

```
## [1] "Agriculture"      "Education"        "Catholic"
## [4] "Infant.Mortality"
```

In this task we used "swiss" dataset which is inside R. This dataset about Standardized fertility measure and socio-economic indicators for each of 47 French-speaking provinces of Switzerland at about 1888. Dataset has 5 predictor which are respectively "Agriculture" (1), "Examination" (2), "Education" (3), "Catholic" (4), "Infant.Mortality" (5).

The plots above, we can see loss mean changes by feature count in left one and loss means of all combinations in right one. We can consider by examining left plot, generally increasing feature count decreases loss means, but we catch the optimal value (the lowest loss mean) in the models which have 4 feature.
We can see this optimal combination is "1-3-4-5" columns in right plot. And the feature names are specified as "Agriculture", "Education", "Catholic" and "Infant.Mortality".

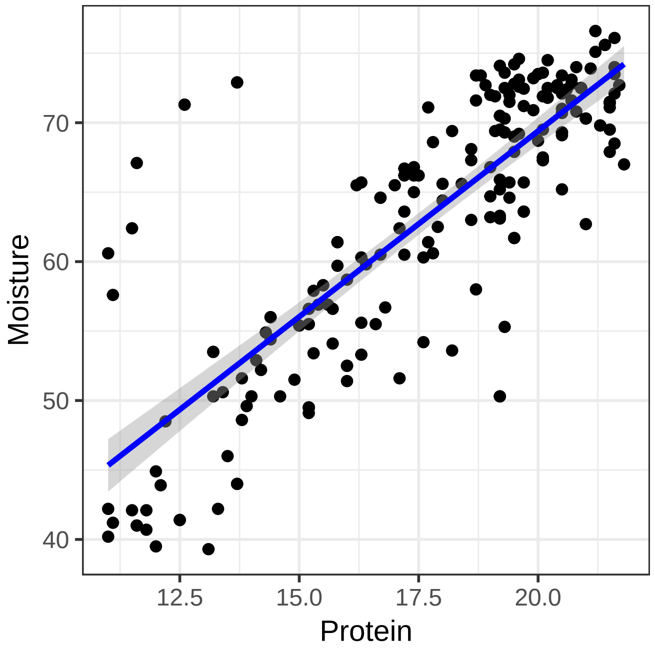
When we think about this results and examine closely right plot, we can see lonely “Education” feature is also gives better predictions than the others and it is also in the best pairs which are 3-5 and 3-4. When we think about real life, “Education” is really effective variable for fertility measurement. Another fact that we can see in this plot, while “Agriculture” has the worst loss mean when we use it alone, we can see it is almost inside all the feature groups of lowest loss mean. So we can conclude this as “Agriculture” is also an effective feature when we use it other effective features.

Assignment 4: Linear regression and regularization

The Excel file tecator.xlsx contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry.

Task 4.1

Import data to R and create a plot of Moisture versus Protein. Do you think that these data are described well by a linear model?



Liear model can fit this data. These 2 variables also have a positive correleation. The cluster around 55-75 at Moisture and 8-14 at Protein is an outlier cluster. Other rest of data is in an increase movement and does not have big variance, so we can draw a linear regression. The problem is the outlier cluster create a bias for our model, because they increase the intercept. Data has many observations at the area of left below corner and the model predicts them much higher than they are. This problem occurs since observations of the outlier cluster have much higher value than others.

Task 4.2

Consider model M_i in which Moisture is normally distributed, and the polynomial terms up to power i (i.e M_1 is a linear model, M_2 is a quadratic model and so on). Report a probabilistic model that describes M_i. Why is it appropriate to use MSE criterion when fitting this model to a training data.

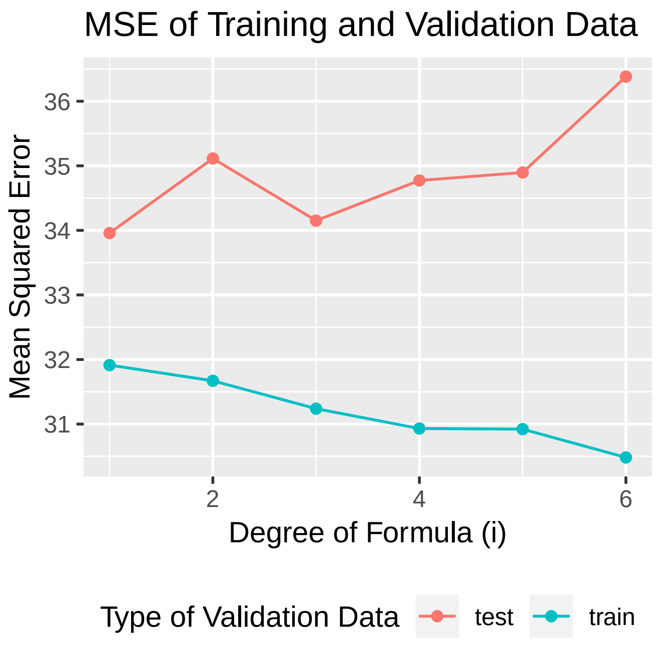
Probabilistic Model:

$$M_i = P(Y|x) = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_k x^k = \sum_{i=0}^{k} \beta_i x^i \sim N(\mu, \sigma^2)$$

MSE criterion is the average squared difference between the predicted values and actual values, so it is a helpful measurement to compare models and see their prediction performances.

Task 4.3

Divide the data into training and validation sets (50% / 50%) and fit models M_i, $i = 1...6$. For each model, record the training and the validation MSE and present a plot showing how training and validation MSE depend on i (write some R code to make this plot). Which model is best according to the plot? How do the MSE values change and why? Interpret this picture in terms of bias-variance tradeoff.



The plot above shows Mean Squared Errors of the $M_1, M_2, M_3, M_4, M_5, M_6$ models by testing with training and test data.
Tests which are made by using train data shows us that MSE is decreased while complexity of model is increasing. This leads overfitting to the train data and it provides more accuracy on tests with training data, but in the other hand it increases the mispredictions on tests with validation data. We can observe in the 6th degree of the formula, we have huge variance between tests.
It is possible to say that the best model in this plot is M_1 because it has the lowest MSE of test with test data.

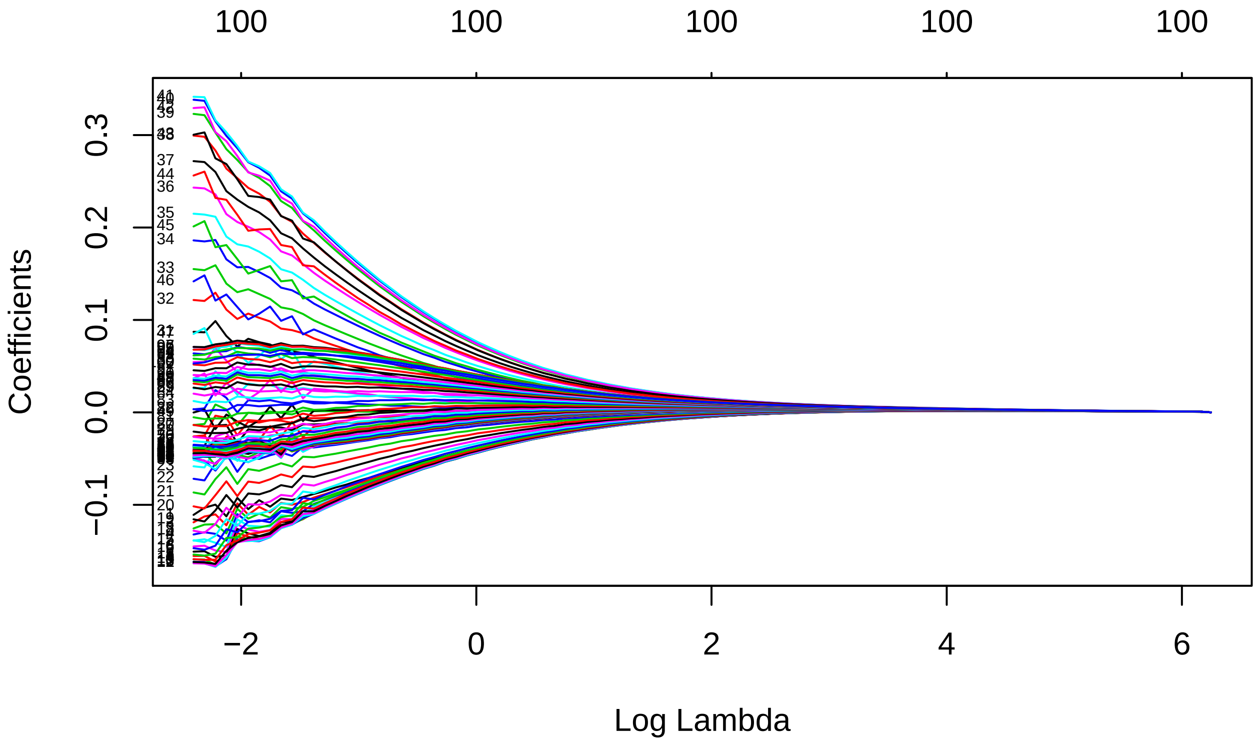
Task 4.4

Perform variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors by using stepAIC. Comment on how many variables were selected.

```
## [1] "Selected Features:"

##  [1] "Channel1"  "Channel2"  "Channel4"  "Channel5"  "Channel7"
##  [6] "Channel8"  "Channel11" "Channel12" "Channel13" "Channel14"
## [11] "Channel15" "Channel17" "Channel19" "Channel20" "Channel22"
## [16] "Channel24" "Channel25" "Channel26" "Channel28" "Channel29"
## [21] "Channel30" "Channel32" "Channel34" "Channel36" "Channel37"
## [26] "Channel39" "Channel40" "Channel41" "Channel42" "Channel45"
## [31] "Channel46" "Channel47" "Channel48" "Channel50" "Channel51"
## [36] "Channel52" "Channel54" "Channel55" "Channel56" "Channel59"
## [41] "Channel60" "Channel61" "Channel63" "Channel64" "Channel65"
## [46] "Channel67" "Channel68" "Channel69" "Channel71" "Channel73"
## [51] "Channel74" "Channel78" "Channel79" "Channel80" "Channel81"
## [56] "Channel84" "Channel85" "Channel87" "Channel88" "Channel92"
## [61] "Channel94" "Channel98" "Channel99"

## [1] "Selected Feature Quantity:"

## [1] 63
```

Task 4.5

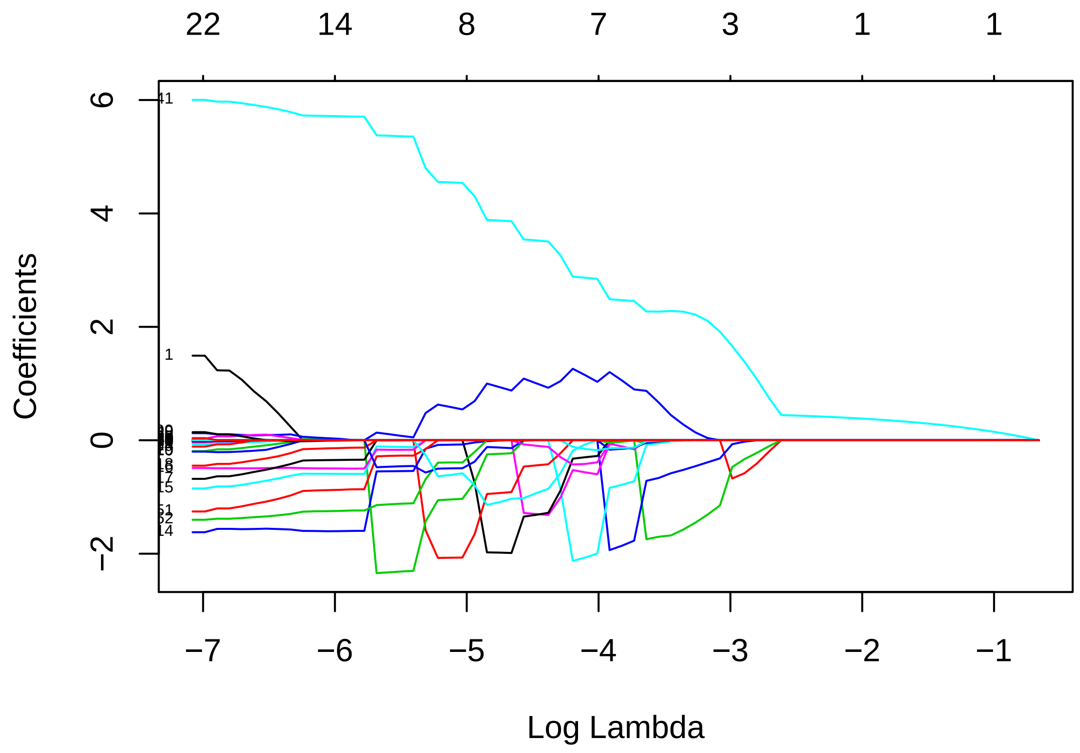
Fit a Ridge regression model with the same predictor and response variables. Present a plot showing how model coefficients depend on the log of the penalty factor λ and report how the coefficients change with λ.



The ridge is a shrinkage method. In this task we apply ridge regression model to reduce the complexity of the model that we create. The part of providing reduce complexity in ridge formula is $\lambda \sum_{j=0}^{p} \beta_j^2$. If we increase the λ, the minimizer of ridge model will decrease β_j values of the model. By this way, the complexity of model will decrease. While $\lambda \to \infty$, Coefficients will be close to zero, but this case will not be happened.

Task 4.6

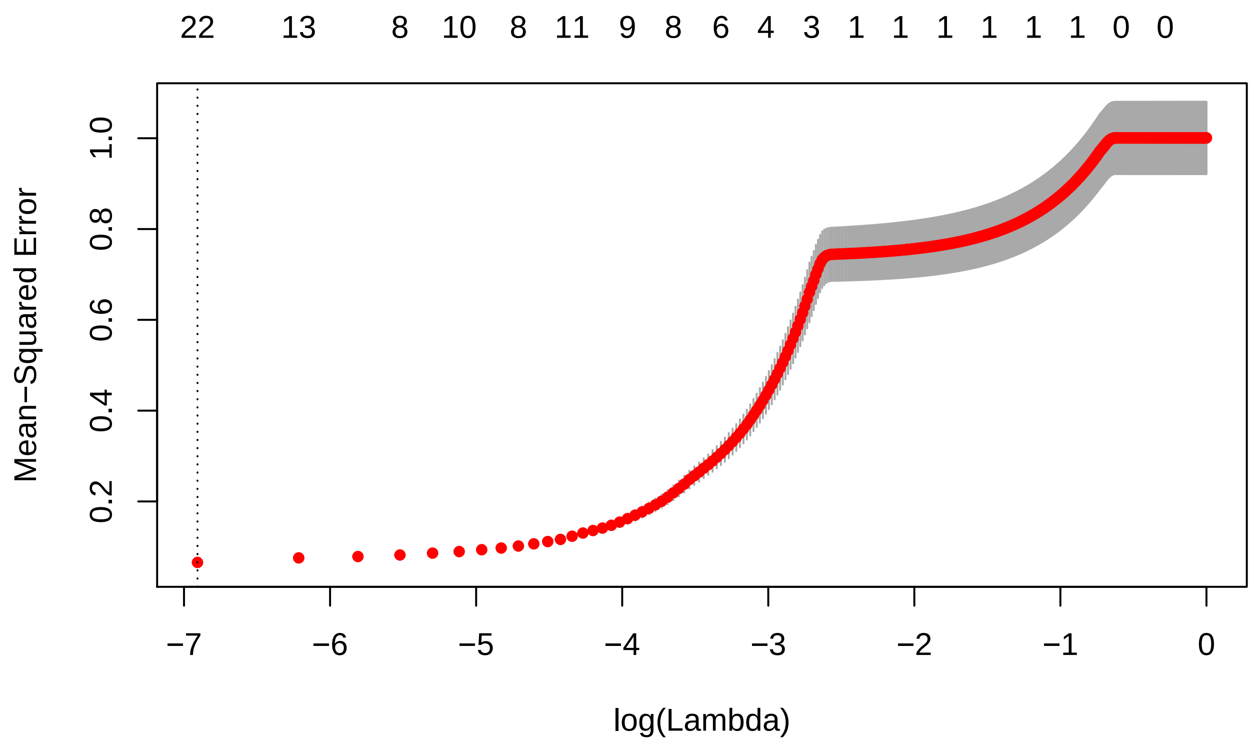
Repeat step 5 but fit LASSO instead of the ridge regression and compare the plots from step 5 and 6. Conclusions?



The LASSO is also a shrinkage method. In this task we apply LASSO regression instead of ridge regression to reduce the complexity of the model that we create. The part of providing reduce complexity in ridge formula is $\lambda \sum_{j=1}^{p} |\beta_j|$. We can see that while λ increases, effects of coefficients change, and some of them are eliminated. Model starts with many variables, then quantity of coefficients are decreased by increasing of λ. In the end, all coefficients are eliminated.
The difference between ridge and LASSO is that while LASSO can eliminate coefficients, ridge cannot. Ridge reduces the complexity of model by decreasing the impact of coefficients to model while LASSO reduces it by both impact of coefficients and count of coefficients.

Task 4.7

Use cross-validation to find the optimal LASSO model (make sure that case $\lambda = 0$ is also considered by the procedure), report the optimal λ and how many variables were chosen by the model and make conclusions. Present also a plot showing the dependence of the CV score and comment how the CV score changes with λ.



```
## [1] "Optimal Lambda:"

## [1] 0

## [1] "Coefficient Count:"

## [1] 101
```

100 Feature sent to cross-validation algorithm and we have 101 coefficient, one of them is intercept. This means model chose 100/100 features inside the optimal model. And our optimal λ is chosen as 0. This means λ does not effect to coefficients, that is why our it is zero.
When we examine to plot, we can see MSE increases while lambda increases. Until -5 at log(Lambda), MSE increases slowly, and after that it starts to increase fastly. After -1 to 0, MSE is the same, because there is no more coefficient to penalize.
Additionally this plot starts with 22 coefficients and the value of log(Lambda) is almost -7. In this point MSE=0.06567. But the lowest MSE is 0.06017 with Optimal Lambda is 0 and coefficient count is 101. In this case the error difference is not too much. So if we want to reach minimum MSE, we can use 101 coefficient and $\lambda = 0$, but if we want a less complex model it is acceptable to choose $\lambda = 0.001$, since the MSE values are close each other.

Task 4.8

Compare the results from step 4 and 7.

In the step 4 we used AIC (Akaike information criterion). AIC is defined with this formula $AIC = 2k - 2ln(\hat{L})$ and this algorithm focuses to minimize this criterion by minimizing complexity of model and also maximizing the likelihood (goodness of fit) of the model. But in cross-validation with LASSO, we compare the MSE by changing the penalize coefficient (λ). This provides to regularize the model. We can say while AIC tries to find best fitable predictors for the model, LASSO tries to find the best predictors for prediction. According to these knowledge, AIC function found 63 features which provide best fit capability for the model, and LASSO found it as 101 features which are gives lowest prediction error (MSE).

Appendix

```
knitr::opts_chunk$set(echo = FALSE, fig.width=3.5, fig.height=3.5, fig.align = "center", message = F, wa
#####################
##### TASK 1.1 #####
#####################
library(readxl)

# import data
df_spambase = read_xlsx("../spambase.xlsx")

# divide data into training and test sets
n = dim(df_spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
# training data
train = df_spambase[id,]
# test data
test = df_spambase[-id,]

# head(train)
# head(test)
#####################
##### TASK 1.2 #####
#####################
# fitting the model
model_glm = glm(Spam ~ ., family=binomial(link = 'logit'), data = train)

conf_matrix = function(real_data, predicted_data){
  ct = table(real_data, predicted_data)
  rownames(ct) = c("Actual (-)", "Actual (+)")
  colnames(ct) = c("Predict (-)", "Predict (+)")
  return(ct)
}

calculate_mc_rate = function(confision_matrix, n){
  return(1-sum(diag(confision_matrix))/n)
}

# predict values
```

```r
predicts_train_data = predict(object = model_glm, newdata = train, type = "response")
predicts_test_data = predict(object = model_glm, newdata = test, type = "response")

# apply threshold 0.5
predicts_train_05 = ifelse(predicts_train_data > 0.5, 1, 0)
predicts_test_05 = ifelse(predicts_test_data > 0.5, 1, 0)

# get original values
org_spam_train = train$Spam
org_spam_test = test$Spam

# draw confusion matrix
cm_train_05 = conf_matrix(org_spam_train, predicts_train_05)
cm_test_05 = conf_matrix(org_spam_test, predicts_test_05)

# calculate misclassification rate
mc_rate_train = calculate_mc_rate(cm_train_05, length(org_spam_train))
mc_rate_test = calculate_mc_rate(cm_test_05, length(org_spam_train))
accuracy_train = 1 - mc_rate_train
accuracy_test = 1 - mc_rate_test

mc_rates = data.frame(train = c(mc_rate_train, accuracy_train),
           test = c(mc_rate_test, accuracy_test))
colnames(mc_rates) = c("Train Data", "Test Data")
rownames(mc_rates) = c("Misclassification", "Accuracy")

knitr::kable(cm_train_05, caption = "Test with Train Data")
knitr::kable(cm_test_05, caption = "Test with Test Data")

knitr::kable(mc_rates, caption = "Misclassification Rates")
####################
##### TASK 1.3 #####
####################
# apply threshold 0.5
predicts_train_09 = ifelse(predicts_train_data > 0.9, 1, 0)
predicts_test_09 = ifelse(predicts_test_data > 0.9, 1, 0)

# draw confusion matrix
cm_train_09 = conf_matrix(org_spam_train, predicts_train_09)
cm_test_09 = conf_matrix(org_spam_test, predicts_test_09)

# calculate misclassification rate
mc_rate_train = calculate_mc_rate(cm_train_09, length(org_spam_train))
mc_rate_test = calculate_mc_rate(cm_test_09, length(org_spam_train))
accuracy_train = 1 - mc_rate_train
accuracy_test = 1 - mc_rate_test

mc_rates = data.frame(train = c(mc_rate_train, accuracy_train),
           test = c(mc_rate_test, accuracy_test))
colnames(mc_rates) = c("Train Data", "Test Data")
rownames(mc_rates) = c("Misclassification", "Accuracy")

knitr::kable(cm_train_09, caption = "Test with Train Data")
```

```r
knitr::kable(cm_test_09, caption = "Test with Test Data")

knitr::kable(mc_rates, caption = "Misclassification Rates")
####################
##### TASK 1.4 #####
####################
library(kknn)

# train k-nearest neighbor
model_kknn_train = kknn(Spam ~ ., train = train, test = train, k = 30)
model_kknn_test = kknn(Spam ~ ., train = train, test = test, k = 30)

# apply threshold 0.5
predicts_kknn_train_05 = ifelse(model_kknn_train$fitted.values > 0.5, 1, 0)
predicts_kknn_test_05 = ifelse(model_kknn_test$fitted.values > 0.5, 1, 0)

# draw confusion matrix
cm_train_kknn_05 = conf_matrix(org_spam_train, predicts_kknn_train_05)
cm_test_kknn_05 = conf_matrix(org_spam_test, predicts_kknn_test_05)

# calculate misclassification rate
mc_rate_train = calculate_mc_rate(cm_train_kknn_05, length(org_spam_train))
mc_rate_test = calculate_mc_rate(cm_test_kknn_05, length(org_spam_train))
accuracy_train = 1 - mc_rate_train
accuracy_test = 1 - mc_rate_test

mc_rates = data.frame(train = c(mc_rate_train, accuracy_train),
           test = c(mc_rate_test, accuracy_test))
colnames(mc_rates) = c("Train Data", "Test Data")
rownames(mc_rates) = c("Misclassification", "Accuracy")

knitr::kable(cm_train_kknn_05, caption = "Test with Train Data")
knitr::kable(cm_test_kknn_05, caption = "Test with Test Data")

knitr::kable(mc_rates, caption = "Misclassification Rates")
####################
##### TASK 1.5 #####
####################
# train k-nearest neighbor
model_kknn_train = kknn(Spam ~ ., train = train, test = train, k = 1)
model_kknn_test = kknn(Spam ~ ., train = train, test = test, k = 1)

# apply threshold 0.5
predicts_kknn_train_05 = ifelse(model_kknn_train$fitted.values > 0.5, 1, 0)
predicts_kknn_test_05 = ifelse(model_kknn_test$fitted.values > 0.5, 1, 0)

# draw confusion matrix
cm_train_kknn_05 = conf_matrix(org_spam_train, predicts_kknn_train_05)
cm_test_kknn_05 = conf_matrix(org_spam_test, predicts_kknn_test_05)

# calculate misclassification rate
mc_rate_train = calculate_mc_rate(cm_train_kknn_05, length(org_spam_train))
mc_rate_test = calculate_mc_rate(cm_test_kknn_05, length(org_spam_train))
```

```r
accuracy_train = 1 - mc_rate_train
accuracy_test = 1 - mc_rate_test

mc_rates = data.frame(train = c(mc_rate_train, accuracy_train),
           test = c(mc_rate_test, accuracy_test))
colnames(mc_rates) = c("Train Data", "Test Data")
rownames(mc_rates) = c("Misclassification", "Accuracy")

knitr::kable(cm_train_kknn_05, caption = "Test with Train Data")
knitr::kable(cm_test_kknn_05, caption = "Test with Test Data")

knitr::kable(mc_rates, caption = "Misclassification Rates")
####################
##### TASK 3.1 #####
####################
library(ggplot2)
library(dplyr)
# function that calculates coefficients
calculate_coefficients = function(Y, X){
  X = as.data.frame(X)
  Xx = cbind(data.frame(intercept = rep(1, nrow(X))), X)
  Xx = as.matrix(Xx)
  coef_matrix = solve(t(Xx) %*% Xx) %*% t(Xx) %*% Y
  return(coef_matrix)
}
# function that predict values by coef_matrix and test values
predict_values = function(coef_matrix, X){
  X = as.data.frame(X)
  Xx = cbind(data.frame(intercept = rep(1, nrow(X))), X)
  Xx = as.matrix(Xx)
  Y_hat = Xx %*% coef_matrix
  return(Y_hat)
}
# function that calculate loss
calculate_loss = function(Y_actual, Y_estimated){
  loss = mean((Y_actual - Y_estimated)^2)
  return(loss)
}
CV = function(X, Y, Nfolds){
  # k-fold function
  k_fold = function(X, Y, N){
    # When we select only one feature in X (column),
    # R convert df to vector automatically
    # We have to convert it df again (this also used in other funcs)
    X = as.data.frame(X)
    # store length of our data
    n = length(Y)
    # get indexes of our data
    all_indexes = 1:n
    # create temporary variables
    loss_values = c()
    set.seed(12345)
    # iterate all combinations
    for(i in 1:N){
      # divide data into training and test sets
      # find the test ids, other ids will be training observations
      id = sample(all_indexes, floor(n/N))
      all_indexes = all_indexes[!all_indexes %in% id]
      test = X[id,]
      train = X[-id,]

      # get actual values
      test_actual = Y[id]
      train_actual = Y[-id]

      # train our linear regression
      coef_mat = calculate_coefficients(train_actual, train)

      # predict values
      predicts = predict_values(coef_mat, test)

      # calculate loss
      loss = calculate_loss(Y_actual = test_actual, Y_estimated = predicts)

      # store loss
      loss_values = c(loss_values, loss)
    }
    # print(loss_values[1])
    return(mean(loss_values))
    # return(loss_values)
  }

  # store length of our data
  n = length(Y)
  # get indexes of our data
  all_indexes = 1:n

  # shuffle data
  set.seed(12345)
  id = sample(all_indexes, n)
  X = as.data.frame(X[id,])
  Y = Y[id]

  # create temporary variables
  loss_values = c()
  feature_sets = c()
  counter = 1

  # in this assignment feature_qty is case specific.
  # we know how many feature we have
  feature_qty = 5
  feature_indexes = 1:feature_qty

  # iterate all combinations of features
  for(i in 0:(2^feature_qty-1)){
    # find the combination as binary string
    cmb = intToBits(i)
    # select features
    selected_features = feature_indexes[cmb[1:feature_qty]==01]
    # run k-fold cross validation for selected features
    loss = k_fold(as.data.frame(X[,selected_features]), Y, Nfolds)
    # store loss value and feature sets
    loss_values = c(loss_values, loss)
    if(i==0)
      feature_sets = c(feature_sets, "intercept")
    else
      feature_sets = c(feature_sets, paste(selected_features, collapse = "-"))
  }
  # feature dataframe
  feat_count = sapply(strsplit(feature_sets, "-"), length)
  feat_count[1] = 0
  fsets = data.frame(feat_count = feat_count,
                     set = feature_sets,
                     loss_mean = loss_values, stringsAsFactors = FALSE)
  # plot 1
  loss_means = fsets %>%
    group_by(feat_count) %>%
    summarise(mean = mean(loss_mean))
  plot1 = ggplot(fsets, aes(x = feat_count, y = loss_mean)) +
    geom_point() +
    geom_line(data = loss_means, aes(x = feat_count, y = mean), color = "blue") +
    labs(x="Feature Count", y="Loss Means", title="Loss Mean by Feature Count") +
    theme_bw() +
    scale_y_continuous(breaks = round(seq(0, max(fsets$loss_mean)+20, by = 20), 1))
  # plot 2
  plot2 = ggplot(fsets, aes(x = reorder(set, +loss_mean) , y = loss_mean)) +
    geom_bar(stat = "identity", width=0.8) +
    scale_y_continuous(breaks = round(seq(0, max(fsets$loss_mean)+20, by = 20), 1)) +
    labs(x="Feature Sets", y="Loss Means", title="Loss Means by Feature Sets") +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 60, hjust = 1))
  # get best features
  bfeats_index = strsplit(fsets[which.min(fsets$loss_mean),2],split = "-")[[1]]
  bfeats = colnames(X)[as.numeric(bfeats_index)]
  response = list(fsets = fsets,
                  plot1 = plot1,
                  plot2 = plot2,
                  best_features = bfeats)
  return(response)
}

####################
##### TASK 3.2 #####
####################
library(gridExtra)
# import data
data = swiss
# select dataw
```

```r
X = data[,2:6]
# actual values
Y = data[,1]

kfcv_result = CV(X, Y, 5)

grid.arrange(grobs=list(kfcv_result$plot1, kfcv_result$plot2), ncol=2)
print("Optimal Features:")
kfcv_result$best_features
# kfcv_result$fsets
####################
##### TASK 4.1 #####
####################
# import data
library(readxl)
df_tecator = read_xlsx("../tecator.xlsx")

plot_t_4_1 = ggplot(df_tecator, aes(x = Protein, y = Moisture)) +
  geom_point() +
  geom_smooth(aes(y = Moisture, x = Protein),
              method = "lm",
              colour="#0000FF",
              formula="y ~ x") +
  theme_bw()
plot_t_4_1
####################
##### TASK 4.2 #####
####################
# fit models
m1 = lm(formula = Moisture ~ Protein, data = df_tecator)
m2 = lm(formula = Moisture ~ Protein + I(Protein^2), data = df_tecator)

MSE_1 = mean(m1$residuals^2)
MSE_2 = mean(m2$residuals^2)
####################
##### TASK 4.3 #####
####################
# divide data
set.seed(12345)
n = nrow(df_tecator)
id = sample(1:n, floor(n*0.5))
test = df_tecator[id,]
train = df_tecator[-id,]

# function that creates formulas
create_formula = function(X,Y,i){
  return(as.formula(formula_str(X,Y,i)))
}

formula_str = function(X,Y,i){
  part = c()
  for(x in 1:i){
    part = c(part, "I(",X,"^",as.character(x),")", "+")
  }
  part = c(Y,"~",part)
  return(paste(part[-length(part)], collapse = ""))
}

# create temporary variables
formulas = c()
train_mse = c()
test_mse = c()
models = list()
plots = list()
for(i in 1:6){
  # create formula, and store it in case create a plot
  formulas[[i]] = create_formula("Protein", "Moisture", i)
  # fit models with train data
  curr_model = lm(formula = formulas[[i]], data = train)
  models[[paste(c("m",i), collapse = "")]] = curr_model
  # predict values
  predicts_test = predict(curr_model, newdata = test)
  predicts_train = predict(curr_model)
  # calculate mse values
  curr_test_mse = calculate_loss(Y_actual = test$Moisture, Y_estimated = predicts_test)
  curr_train_mse = calculate_loss(Y_actual = train$Moisture, Y_estimated = predict(curr_model))
  # curr_train_mse = mean(curr_model$residuals^2)
  test_mse = c(test_mse, curr_test_mse)
  train_mse = c(train_mse, curr_train_mse)
}

plot_data = data.frame(i = rep(1:6, 2),
                       mse = c(test_mse, train_mse),
                       data_type = c(rep("test", 6), rep("train", 6)))

plot_mse = ggplot(data = plot_data, aes(x = i, y = mse, color = data_type)) +
  geom_line() +
  geom_point() +
  labs(title = "MSE of Training and Validation Data",
       x = "Degree of Formula (i)", y = "Mean Squared Error",
       color = "Type of Validation Data") +
  scale_y_continuous(breaks = round(seq(30, 37, by = 1), 1)) +
  theme(legend.position = "bottom")

# # This code creates all plots of regression models
# all_plots = list()
# counter = 1
# for(i in 1:6){
#   plot_m = ggplot(df_tecator, aes(x = Protein, y = Moisture)) +
#     geom_point(alpha=0.5) +
#     geom_smooth(aes(x = Protein, y = Moisture),
#                 method = "lm",
#                 colour="#0000FF",
#                 formula=formula_str("x","y",i)) +
#     theme_bw()
#   all_plots[[counter]] = plot_m
#   counter = counter + 1
# }
# grid.arrange(grobs = all_plots, ncol=2)
plot_mse
####################
##### TASK 4.4 #####
####################
library(MASS)
model_fat = lm(formula = Fat ~ ., data = df_tecator[,-c(1, 103, 104)])
# Search best features
stepAIC_result = stepAIC(model_fat, trace = 0)
# selected features quantity is 63
selected_features = colnames(stepAIC_result$model)[-1]
selected_feat_qty = length(selected_features)
print("Selected Features:")
selected_features
print("Selected Feature Quantity:")
selected_feat_qty
####################
##### TASK 4.5 #####
####################
library(glmnet)
# scale data
covariates = scale(df_tecator[,2:101])
response = scale(df_tecator[,102])
# fit ridge model
model_ridge = glmnet(as.matrix(covariates), response, alpha = 0, family = "gaussian")
plot(model_ridge, xvar="lambda", label=TRUE)
####################
##### TASK 4.6 #####
####################
# fit lasso model
model_lasso = glmnet(as.matrix(covariates), response, alpha = 1, family = "gaussian")
plot(model_lasso, xvar="lambda", label=TRUE)
####################
##### TASK 4.7 #####
####################
cv_lasso = cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian", lambda=seq(0,1,0.001))
plot(cv_lasso)
print("Optimal Lambda:")
cv_lasso$lambda.min
print("Coefficient Count:")
length(coef(cv_lasso, s="lambda.min"))
```