You have mistakes in Q1 Logistic Regression and Q4 polynomial models. Otherwise, good report!

Computer lab 1 block 1

Vasileia Kampouraki & Laura Julià Melis & Mengxin Liu

11/24/2019

Assignment 1. Spam classification with nearest neighbors.

1. Import the data into R and divide it into training and test sets (50%/50%) by using the following code:

To read xlsx data, we implement read_xlsx function from readxl package. Then we divide the data into training and testing data.

See Appendix.

implement means that you created it. Probably, use or call would be more suitable words here. :)

2. Use logistic regression (functions glm(), predict()) to classify 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.

- Confusion matrices.

```
##
## fitted_response_test   0   1
##                      0 616 281
##                      1 335 138
```

Normal convention for confusion matrices - true labels on the rows and predicted labels on the columns. Good to mention if you are switching them.

```
##
## fitted_response_train   0   1
##                       0 804  93
##                       1 127 346
```

fitted_prob_test <- predict(fit1, data=test, type ="response")
Here, the parameter name should be "newdata" and not "data".
That's why you have wrong predictions :)

- Misclassification rates.

For the test data:

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

Test misclassification rate is wrong!

For the training data:

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

Looking at the confusion matrices, it's obvious that the train data are better classified than the test data and this is verified by the misclassification rates where for the training data the rate is 0.1605839 whereas for the test data it is 0.449635, which is much bigger.

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

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

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?

- Confusion matrices.

```
##
## fitted_response_test   0   1
##                      0 876 378
##                      1  75  41
```

```
##
## fitted_response_train   0   1
##                     0 921 333
##                     1  10 106
```

- Misclassification rates.

For the test data:

```
## [1] 0.3306569
```

Your confusion matrices and misclassification rates are wrong because of issue mentioned earlier.

For the training data:

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

In this occasion, it's hard to say in which data set we have better classification by looking just at the confusion matrix. That is logical as the misclassification rates are not significantly different. This happened because we changed the classification principle to a higher probability. That led to less spams been classified as non-spams but also now more non-spams get classified as spams. Now the misclassification rate for the test data has been dicreased and that's because we have higher classification accuracy (80%).

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.

- Misclassification rate for the test data:

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

- Misclassification rate for the training data:

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

If we compare these misclassification rates with the ones obtained in question 2 we can observe that the rate values for the training set are almost the same (0.161 and 0.167) while the rate for the test set is 0.14 points lower when using 30-nn model than the linear regression model.

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?

- Misclassification rate for the test data:

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

- Misclassification rate for the training data:

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

~~When choosing K=1, what we are doing is choosing only one point~~ (the closest one), so when making predictions the model will always classify the observation to that point (because it will have probability 1) irrespective of the dataset. For this reason, the missclassification rate for the training set is 0 but the rate for the test set is greater than the one using K=30.

Overfitting

When we apply the kNN algorithm for the test data set we get a misclassification rate of around 36%.

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

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.

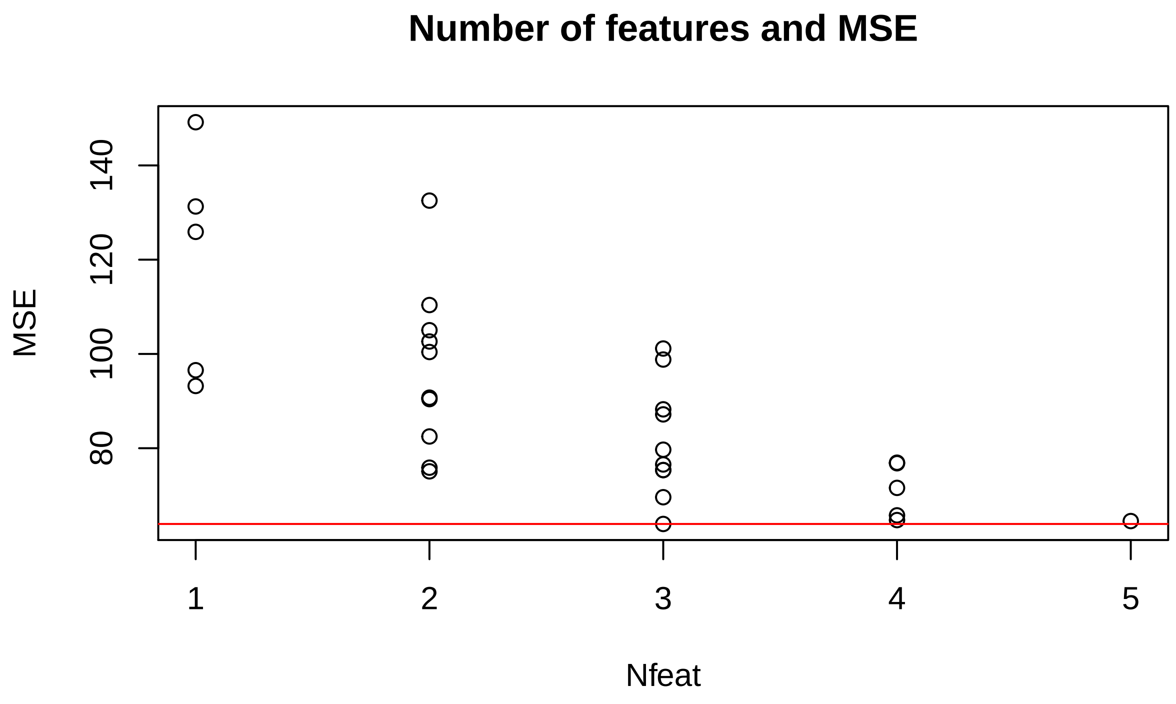
We build the code of cross validation score function upon the helping material, the following block is the code of the function to perform linear regression and calculate the CV score.

See Appendix. You are not using the randomized order of observations in your code.

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.



```
## $CV
## [1] 63.92262
##
## $Features
## [1] 0 0 1 1 1
```

In the plot we can observe all the Mean Squared Errors associated to all the different possibilities of fitting a model (in our case, $p = 5$ covariates so we have 31 points graphed). The red line indicates the minimum MSE so we can see that the number of features included in the fitted model with the lowest MSE is 3.

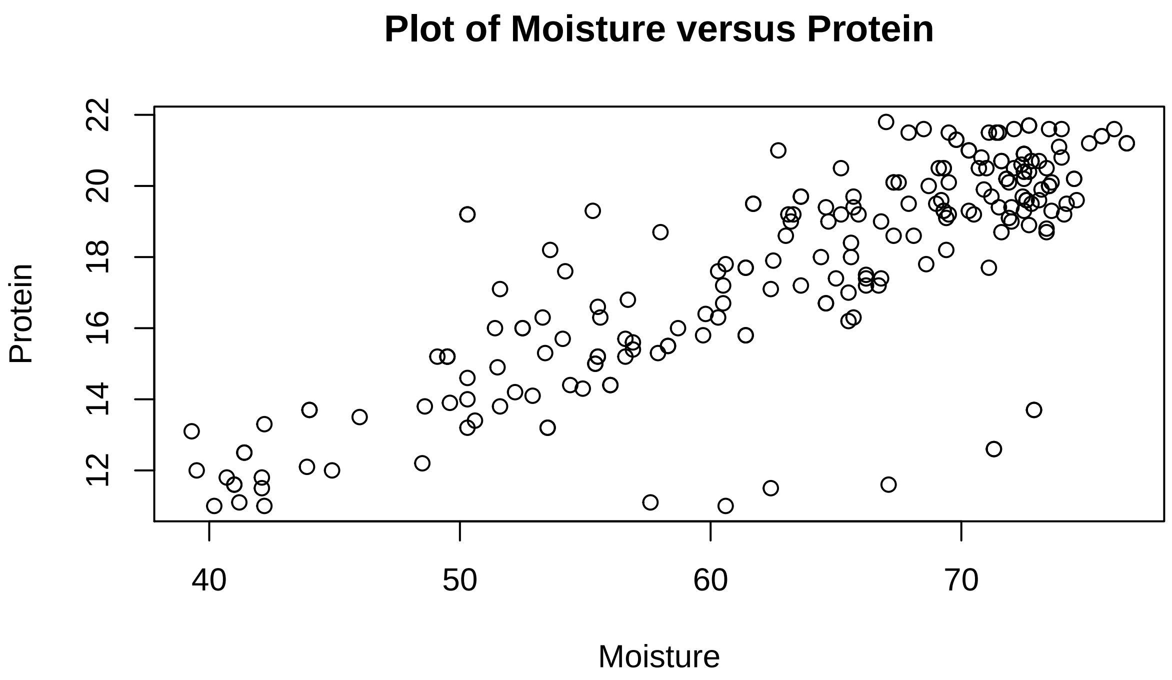
Also, from the list output we can observe that the optimal subset of features is (0 0 1 1 1) so we can write the optimal model as:

$$\text{Fertility} = \beta_0 + \beta_1 \cdot Education + \beta_2 \cdot Catholic + \beta_3 \cdot Infant.Mortality$$

It is reasonable that Education, Religion or Infant mortality are variables that have a largest impact on fertility than working as a farmer or having a good grade in the army exam that seem totally irrelevant to fertility. For instance, some beliefs in religion are directly connected to a high fertility.

Assignment 4. Linear regression and regularization.

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?



As can be seen from the plot, for most of the points, as Protein values increase the Moisture values do too. And this increasing relationship is linear, with some outliers though in the down right side, so a linear model whould probably fit the data but not so well.

Yes, good.

2. Consider model M_i in which Moisture is normally distributed, and the expected Moisture is a polynomial function of Protein including 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?

A probabilistic model that describes M_i would the following:

This expression does not show a probabilistic model. You need to mention the distribution and it's parameters.

$$\text{Moisture}_i = \beta_0 + \beta_1 Protein^1 + \cdots + \beta_i Protein^i$$

The mean squared error indicates how close a fitted regression model is to the data by taking the distances from the points (Y_i) to the fitted points $(\hat{Y}_i)$ and squareing them:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

But with a polynomial function of the independent variable, the model can get very complex and overfit (the model would be too close to the training points because it would also capture the noise) meaning that the

You mention what MSE is but not why it is appropriate here. Mean absolute error is also an error measure that can be used. But, here MSE is appropriate because : for a normally distributed model minimizing MSE results in the MLE.

distance between the test data points and the predicted points would increase (and that would be bad, we don't want that). Consequently, we can measure this modeling error calculating the MSE, having in mind that the MSE should be low in order to prevent over-fitting.

3. Divide the data into training and validation sets (50%/50%) and fit models $M_i = 1...6$. 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.

- MSE table for the training and the test datasets.

```
##   i mse_test mse_train
## 1 1 152.4494  26.44494
## 2 2 151.6767  26.36590
## 3 3 151.4196  25.07017
## 4 4 152.6582  24.82475
## 5 5 154.2534  24.41987
## 6 6 156.1621  24.00218
```

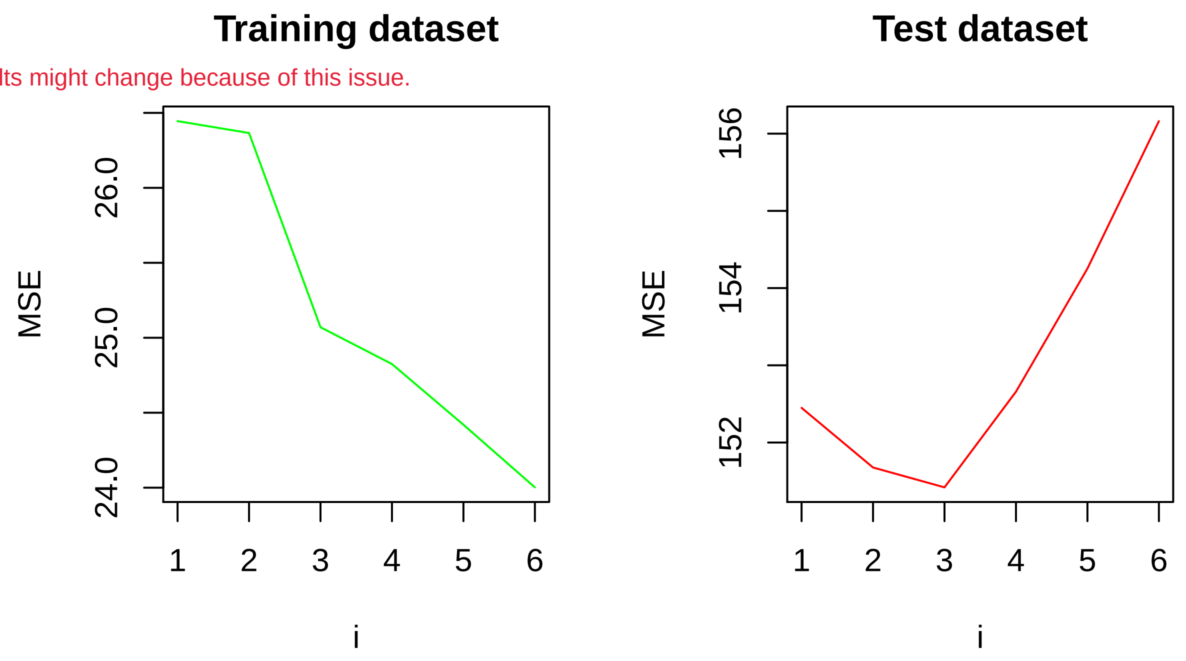
Your MSE values for both train and test are wrong!
fit <- lm(train$Moisture~poly(train$Protein, i, raw=T), data=train)

Here, raw=T is not appropriate. Not sure why you used it.
Check what this parameter actually does.
You probably ignored some warnings. In your formula, you should have used only the variable names.

For comparison, it is always better to show both of them on the same plot.

- Plots:

Plots and results might change because of this issue.



According to the plots, the best model is the one with $i = 3$:

$$\text{Moisture} = \beta_0 + \beta_1 Protein + \beta_2 Protein^2 + \beta_3 Protein^3$$

We see this from the *Test dataset* plot: the lowest MSE (151.4196) is the one we get when $i = 3$.

Looking at the Training MSE plot,we see that the MSE at first is big but the higher the order of polynomial gets, the lowest the MSE and that's because it starts fitting the model better.

However, we see an opposite image at the validation MSE plot, where at the beginning the MSE decreases and has the lowest value for $i = 3$ but then it increases exponentially and that is actually an indication of overfitting because then we have bad predictions, because the model starts fitting to "patterns" in the training data.

In terms of bias-variance tradeoff, we can interpret the combination of both plots by observing that when the model is less complex (when i is small), the bias (the difference between the expected fitted values and the real values) is high and the variance is "low" while when i is greater, the variance is higher and the bias lower. So the optimal model complexity is the one obtained by reducing variance at the cost of introducing some bias, and this happens when $i = 3$. Good explanation!

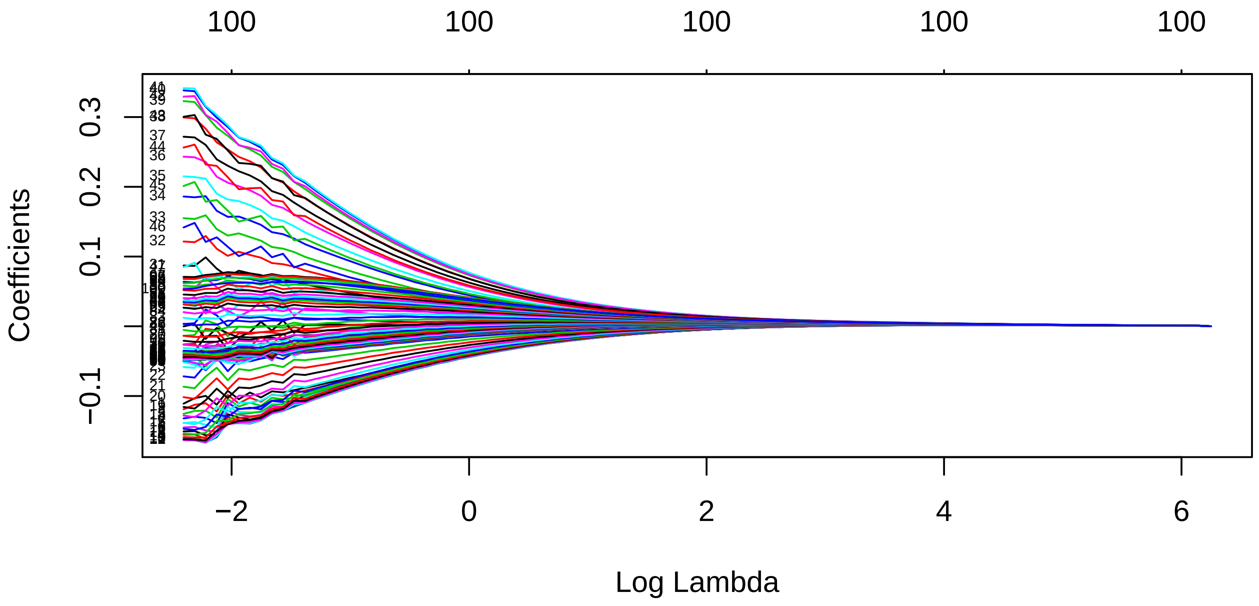
4. Perform variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors by using stepAIC. Comment on how many variables were selected.

We used the *direction* argument to indicate that the we want the stepwise method to combine backward and forward selection.

```
## [1] 64
```

From `length(fit$coefficients)` we can observe that $64 - 1 = 63$ (we get out the intecept) variables have been selected.

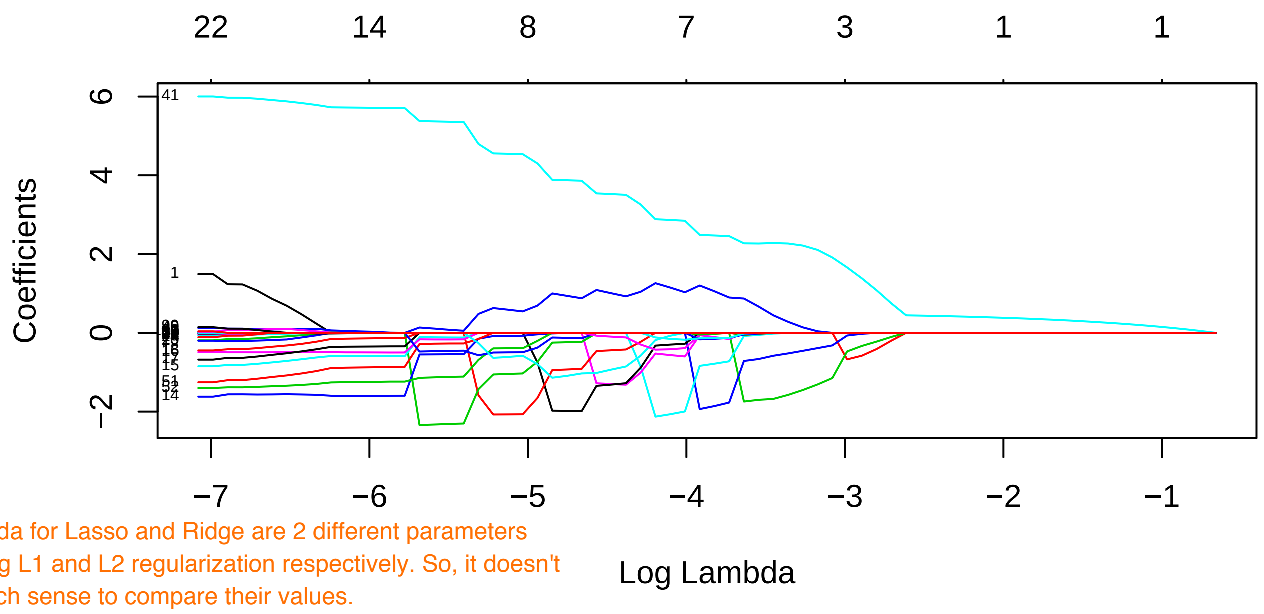
5. Fit a Ridge regression model with the same predictor and response variables. Present a plot showing how model coefficients depend on the log of the penalty factor λ and report how he coefficients change with λ.



Lambda is a tuning parameter that tries to penalize the coefficients in order to get only the important variables and not overfit. Thus, it shrinks coefficients towards zero for non-important variables.

In the plot we see different combinations of lambdas on 100 variables. Ridge regression does not remove any variables, it just shrinks their coefficients. We see that as the lambda increases, all the coefficients converge to zero and that's why we don't choose very big lambdas because then we will have underfit and the coefficients will be smaller than they ought to be to get the best predictive accuracy.

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

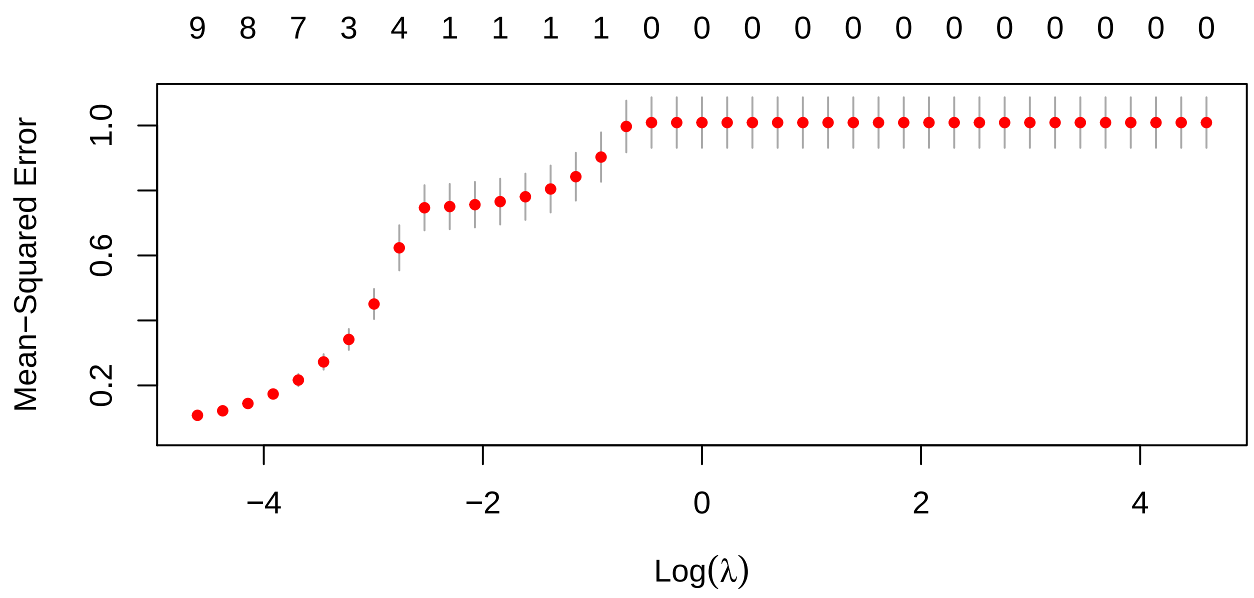


The lambda for Lasso and Ridge are 2 different parameters performing L1 and L2 regularization respectively. So, it doesn't make much sense to compare their values.

The main difference between LASSO and ridge regression is that in LASSO some coefficients can become zero and eliminated from the model, whereas in ridge regression there is no elimination of coefficients, just shrinkage. Larger penalties result in coefficient values closer to zero, which is the ideal for producing simpler models. We observe that with LASSO the coefficients converge to zero faster compared to Ridge regression. LASSO produces sparse solutions as can be seen from the plot.

If we choose $log\lambda = 0$, for example, almost all the coefficients in the Ridge Regression plot are non-zero while all the coefficients in the LASSO plot are 0.

7. Use cross-validation to find the optimal LASSO model (make sure that case $\lambda = 0$ is also considered by the procedure).



In the plot, the numbers in the top represent the number of nonzero coefficients. The lowest point in the curve indicates the optimal lambda and this exact value here is:

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

You could also consider lambda.1se which eliminates some variables while still achieving comparable CV MSE.

But $log(0)$ is undefined and goes to $-\infty$ when λ equals 0. So we can't see the lowest point of MSE in the plot but we know that in the model using the optimal lambda ($\lambda = 0$) we will have all the variables. Let's check it:

```
## [1] 101
```

So, 101-1=100 variables were chosen by the model.

8. Compare the results from steps 4 and 7.

In question 4 we used `stepAIC()` to perform the stepwise method with both backward and forward types of selection variables and as result we obtained a model with 63 variables while in question 7 the `cv.glmnet()` function has been used to perform cross-validation and find the optimal LASSO model, and as a result a model with $\lambda = 0$ and all 100 variables has been obtained.

In this case, we would probably choose the model obtained with the stepAIC method because it's simpler and it's unlikely to overfit.

Comparing AIC metrics could also help in choosing the better model.

Appendix

Assignment 1. Spam classification with nearest neighbors.

Question 1.

```r
# Importing the data:
library(readxl)
data <- read_excel("spambase.xlsx")

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

Question 2.

```r
# Logistic regression model:
fit1 <- glm(Spam~., data= train, family = binomial)

# Predictions:
fitted_prob_test <- predict(fit1, data=test,  type ="response") # type response to get probabilities.
fitted_prob_train<- predict(fit1, data=train,  type ="response")

fitted_response_train <- ifelse(fitted_prob_train > 0.5, 1, 0)
fitted_response_test <- ifelse(fitted_prob_test > 0.5, 1, 0)

# Confusion matrices.
(confusion_matrix_test <- table(fitted_response_test, test$Spam))
(confusion_matrix_train <- table(fitted_response_train, train$Spam))

# Misclassification rates.
mean(fitted_response_test != test$Spam)
mean(fitted_response_train != train$Spam)
```

Question 3.

```r
fitted_response_train <- ifelse(fitted_prob_train > 0.8, 1, 0)
fitted_response_test <- ifelse(fitted_prob_test > 0.8, 1, 0)
```

```r
# Confusion matrices.
(confusion_matrix_test <- table(fitted_response_test, test$Spam))
(confusion_matrix_train <- table(fitted_response_train, train$Spam))

# Misclassification rates
mean(fitted_response_test != test$Spam)
mean(fitted_response_train != train$Spam)
```

Question 4.

```r
library(kknn)
knn_test_30 <- kknn(Spam~., train, test, k=30)
knn_train_30 <- kknn(Spam~., train, train, k=30)

knn_response_train_30 <- ifelse(knn_train_30$fitted.values > 0.5, 1, 0)
knn_response_test_30 <- ifelse(knn_test_30$fitted.values > 0.5, 1, 0)

# Misclassification rates
mean(knn_response_test_30 != test$Spam) # for the test data
mean(knn_response_train_30 != train$Spam) # for the training data
```

Question 5.

```r
library(kknn)
knn_test_1 <- kknn(Spam~., train, test, k=1)
knn_train_1 <- kknn(Spam~., train, train, k=1)

knn_response_train_1 <- ifelse(knn_train_1$fitted.values > 0.5, 1, 0)
knn_response_test_1 <- ifelse(knn_test_1$fitted.values > 0.5, 1, 0)

# Misclassification rates
mean(knn_response_test_1 != test$Spam) # for the test data
mean(knn_response_train_1 != train$Spam) # for the training data
```

Assignment 3.

Question 1.

```r
mylin=function(X,Y, Xpred){
  Xpred1 = cbind(1,Xpred)
  X1 = cbind(1, X)
  beta = solve( t(X1) %*% X1) %*% t(X1) %*% Y # obtained using the "training" data matrix X
  Res = Xpred1%*%beta # y_hat for the "test" data
  return(Res)
}

myCV=function(X,Y,Nfolds){
  n=length(Y) # number of observations (rows)
  p=ncol(X) # number of covariates (variables or columns)
  set.seed(12345)
  ind=sample(n,n) # indexes are randomized
  X1=X[ind,] # randomize the order of the observations
  Y1=Y[ind]
  sF=floor(n/Nfolds) # number of observations inside each fold
  MSE=numeric(2^p-1) # vector of the length of 2^p-1 combinations
  Nfeat=numeric(2^p-1)
  Features=list() # features that will be selected
  curr=0 # current
  folds_obs <- cut(1:n,breaks=Nfolds,labels=FALSE)

  #we assume 5 features.

  for (f1 in 0:1)
    for (f2 in 0:1)
      for(f3 in 0:1)
        for(f4 in 0:1)
          for(f5 in 0:1){
            model= c(f1,f2,f3,f4,f5)
            if (sum(model)==0) next()
            SSE=0


            for (k in 1:Nfolds){
#MISSING:compute which indices should belong to current fold
            indices <- ind[which(folds_obs==k)] #indeces of the observations in fold k

#MISSING:implement cross-validation for model with features in "model" and iteration i.
            X_mylin <- X[-indices,which(model==1)]
            XPred_mylin <- X[indices,which(model==1)]
            Y_mylin <- Y[-indices]

#MISSING:Get the predicted values for fold k, Ypred, and the original values for fold 'k', Yp.
              Ypred <- mylin(X_mylin, Y_mylin, XPred_mylin)
              Yp <- Y[indices]

              SSE=SSE+sum((Ypred-Yp)^2)
            }
            curr=curr+1
            MSE[curr]=SSE/n
            Nfeat[curr]=sum(model)
            Features[[curr]]=model

          }

  plot(Nfeat, MSE) #MISSING: plot MSE against number of features
  abline(h=MSE[which.min(MSE)], col="red")

  i=which.min(MSE)
  return(list(CV=MSE[i], Features=Features[[i]]))
}
```

Question 2.

```r
data("swiss")
myCV(as.matrix(swiss[,2:6]), swiss[[1]], 5)
```

Assignment 4.

Question 1.

```r
# Importing the data:
data <- read_excel("tecator.xlsx")
plot(data$Moisture, data$Protein, main="Plot of Moisture versus Protein",
     xlab="Moisture", ylab="Protein")
```

Question 3.

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

mse_test <- vector()
mse_train <- vector()
for( i in 1:6){
  fit <- lm(train$Moisture~poly(train$Protein, i, raw=T),  data=train)
  mse_test[i] <- mean((test$Moisture - predict(fit, data=test))^2)
  mse_train[i] <- mean((train$Moisture - predict(fit, data=train))^2)
}
# MSE table for the training and the test datasets
(df <- as.data.frame(cbind(i=1:6,mse_test, mse_train)))

# MSE in a plot
par(mfrow=c(1,2))
plot(1:6, mse_train, type= "l",col="green", xlab="i", ylab="MSE", main="Training dataset")
plot(1:6, mse_test, type= "l", col="red", xlab="i", ylab="MSE", main="Test dataset")
```

Question 4.

```r
library(MASS)
df = data[-c(1, 103, 104)]
model<- lm(Fat ~ . , data = df)
fit <- stepAIC(model, direction = "both", trace = F)
length(fit$coefficients)
AIC(fit)
```

Question 5.

```r
# First we have to scale de variables.
response <- scale(df[,101])
covariates <- scale(df[,1:100])

# Now we can fit the ridge regression model.
library(glmnet)
ridge <- glmnet(as.matrix(covariates), response, alpha=0, family="gaussian")
plot(ridge, xvar="lambda", label=TRUE)
```

Question 6.

```r
lasso <- glmnet(as.matrix(covariates), response, alpha=1,
                family="gaussian")
plot(lasso, xvar="lambda", label=TRUE)
```

Question 7.

```r
lambdas_to_try <- 10^seq(-2, 2, by = 0.1) # we want to consider lambda=0
lambdas_to_try<- append(lambdas_to_try,0)

lasso_cv <- cv.glmnet(as.matrix(covariates), response, alpha = 1, lambda = lambdas_to_try,
                      family="gaussian")
plot(lasso_cv)

lasso_cv$lambda.min #  value of lambda that gives minimum mean cross-validated error.

log(lasso_cv$lambda.min)

x <- coef(lasso_cv, s="lambda.min")
length(x)-length(which(x==0))
```

Question 8.

```r
# R-squared of the stepAIC() optimal model obtained:
rsq_stepaic <- summary(fit)$r.squared

# LASSO model fitted with optimal lambda value
fit_lasso_cv <- glmnet(as.matrix(covariates), response, lambda = lasso_cv$lambda.min)
y_hat_cv <- predict(fit_lasso_cv, as.matrix(covariates))
ssr_cv <- t(response - y_hat_cv) %*% (response - y_hat_cv)
rsq_lasso_cv <- cor(response, y_hat_cv)^2 # R-squared

# COMPARISON TABLE:
rsq <- cbind("R-squared" = c(rsq_stepaic, rsq_lasso_cv))
rownames(rsq) <- c("step AIC", "lasso cross_validated")
print(rsq)
```