Block1_Lab1

Roshni Sundaramurthy

23 November 2018

Assignment 1. Spam classification with nearest neighbors

1.1

```r
# Assignment 1. Spam classification with nearest neighbors
# 1.load data and divide it into training and test sets (50%/50%)
library(xlsx)
spambase <- read.xlsx("spambase.xlsx",1)
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=spambase[id,]
test=spambase[-id,]
```

1.2

```
## Confusion Matrix for test data :
```

```
##    classify
##     notspam spam
##   0     791  146
##   1      97  336
```

```
## [1] "Misclassification rate of test data :  0.1774"
```

```
## Confusion Matrix for train data :
```

```
##    classify_train
##     notspam spam
##   0     803  142
##   1      81  344
```

```
## [1] "Misclassification rate of train data :  0.1628"
```

Analysis:

A logistic regression model (learned using maximum likelihood) is fitted between spam and all other features of the given data using the glm function. The confusion matrix for this predicted data gives the true negative values (Y=0,Yhat=0) & positive (Y=1,Yhat=1) values. This denotes that 336 objects of the data have been classified correctly and for test data but 344 objects of the data have been classified correctly and for train data. The misclassification rate of test data is higher than train data since we fit the model using train data. The prediction error should be less for good model.

1.3

```
## Confusion Matrix for test data :
```

```
##    classify_3
##     notspam spam
##   0     936    1
##   1     427    6

## [1] "Misclassification rate of test data :  0.3124"

## Confusion Matrix for train data :

##    classify_train_3
##     notspam spam
##   0     944    1
##   1     419    6

## [1] "Misclassification rate of train data :  0.3066"
```

Analysis:

Now the same procedure is repeated but the classification principle is given as if predicted data is greater than 0.9, it is classified as spam otherwise it is not spam. The misclassification error for test data is 0.3124 and the misclassification error for train data is 0.3066. When compared to the previous prediction, the misclassification error for train data remains lower than the test data. Now only 6 objects are classified correctly. So, it is obvious that the previous classification principle can yield good results.

1.4

```
## [1] "Misclassification rate of test data :  0.3299"

## [1] "Misclassification rate of train data :  0.1723"
```

Analysis:

The same data is used to fit the model using kknn(). The resulted misclassification rate for test is 0.3299 and for train it is 0.1723. When compared to the logistic regression (step2), the error rate seems to be high. This is due to that the classification principle is mentioned in logistic regression. But in kknn, it does not show any important predictors and also no table of coefficients with p-values is obtained.

1.5

```
## [1] "Misclassification rate of test data :  0.346"

## [1] "Misclassification rate of train data :  0"
```

Analysis:

When K value is reduced from 30 to 1, the misclassification rate for train is decreased. When K decreases (K=1), the granularity or resolution is too fine, which is overfit and therefore leads to high variance. The complexity will be high for low values of K.

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

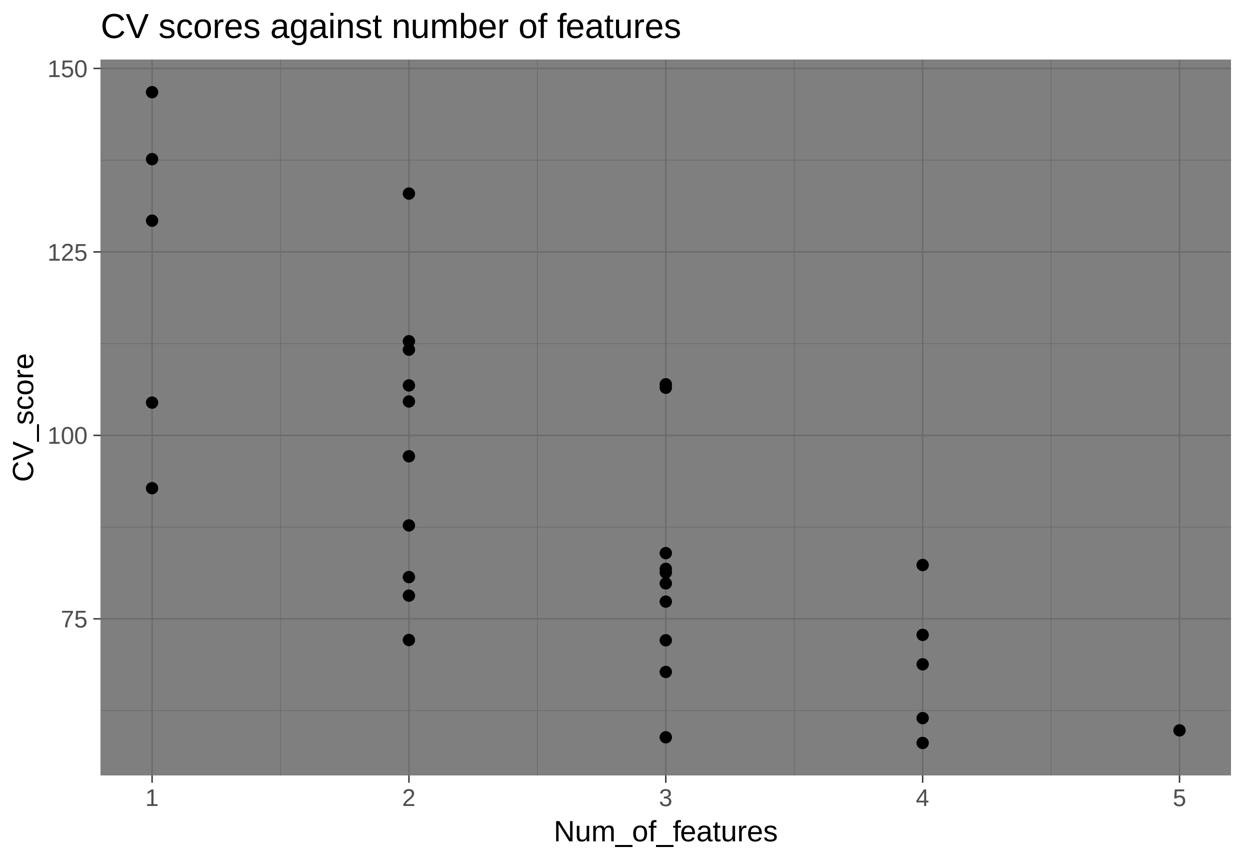
3.1

Function that performs feature selection (best subset selection) [code in appendix]

3.2

```
set.seed(12345)
swiss_data <- swiss[sample(nrow(swiss)),]
swiss_model <- best_subset(swiss_data[,2:6], swiss_data[,1, drop = FALSE], 5)
```

```
## Optimal Subset of Features:  1, 3, 4, 5    CV Score:  58.0877264293113
```



The best_subset function is tested with Swiss data. As we see the plot of cv scores against number of features, the maximum values of CV scores gradually decreases with increase in number of features.

Optimal subset of features: “Agriculture”, “Education”, “Catholic” and “Infant.Mortality” as observed from their CV score and these features have the largest impact on “Fertility”.

Agriculture : This variable represents the percentage of the male population involved in agriculture (swiss data). The agricultural policy have an impact on natural fertility rates primarily through improved nutrition.

Education : It denotes percentage of education beyond primary school for draftees. Education and fertility probably reflects the indirect effect of education through the economic factors of income and value of a couple’s time. Research consistently shows that women who are empowered through education tend to have fewer children.

Catholic : It denotes the percentage of ‘catholic’ (as opposed to ‘protestant’). Catholic natural family planning is a way to avoid or achieve pregnancy based on awareness of a woman fertility.

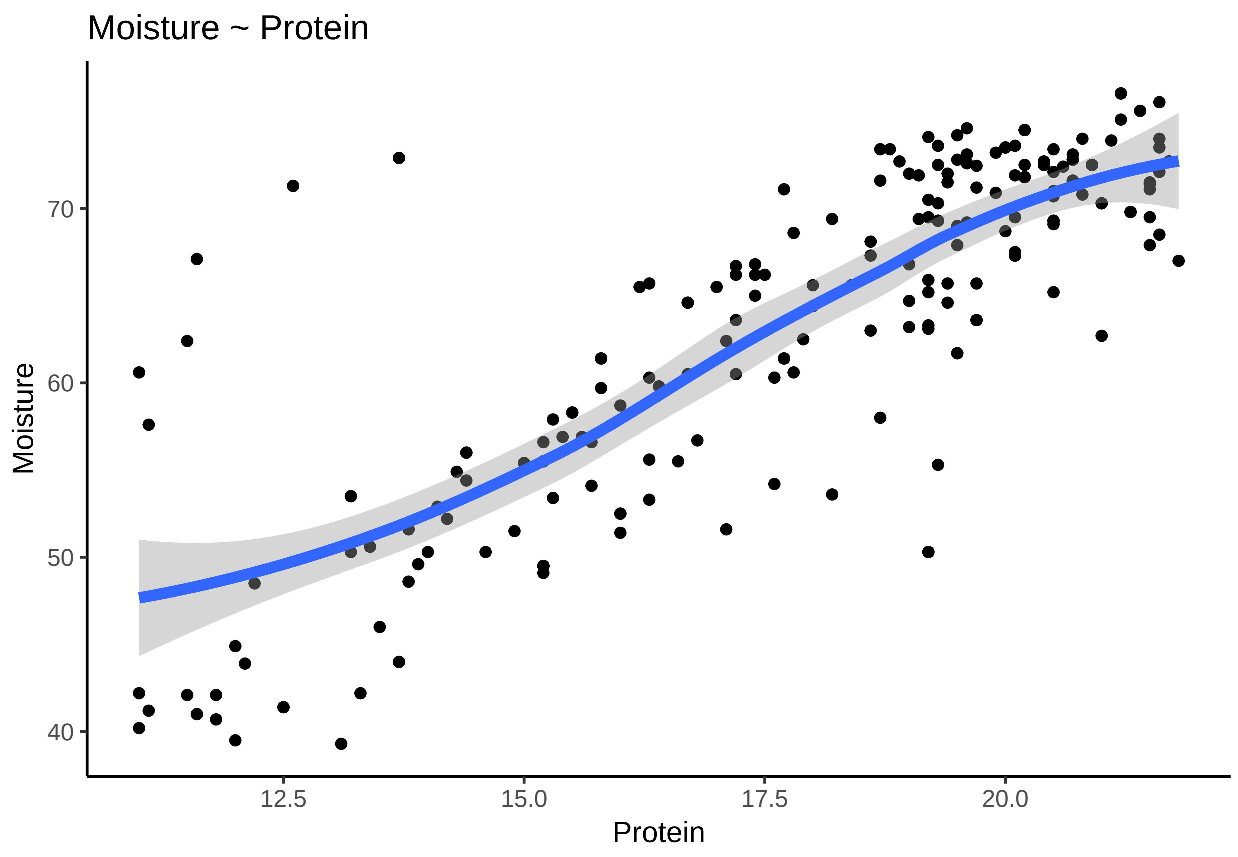
Infant.Mortality : Percentage of live births who live less than 1 year. According to theory, fertility and infant mortality should affect each other simultaneously, and these effects should be positive. Studies state that when mortality rates decline quickly but fertility rates fail to follow, countries can find it harder to reduce poverty. The association of changes in infant mortality with changes in marital fertility is more appropriate for examining secular fertility decline and most relevant for policy issues.

Assignment 4. Linear regression and regularization

4.1

```r
# Assignment 4. Linear regression and regularization
# 1. load data
tecator <- read.xlsx("tecator.xlsx",1)

#plot of Moisture versus Protein
library(ggplot2)
ggplot(tecator)+geom_point(aes(Protein,Moisture))+
  geom_smooth(aes(Protein,Moisture),method="loess", size=2)+
  ggtitle("Moisture ~ Protein")+xlab("Protein")+ylab("Moisture")+
  theme_classic()
```



Analysis:

A scatter plot is used to visualize the linear relationship between the dependent (response) variable and independent (predictor) variables. The scatter plot along with the smoothing line above suggests a monotonically increasing relationship between the 'Protein' and 'Moisture' variables. This is good because, one of the underlying assumptions in linear regression is that the relationship should be linear and additive. So, these data are well described by a linear model.

4.2

Probabilistic Model that describes M_i which is normally distributed.

$$y \sim w_0 + w_i x + e$$
$$e \sim N(0, \sigma^2)$$

as Moisture is normally distributed

$$P(y|x,w) = N(w_0 + w_1 x, \sigma^2) = N(w^T x, \sigma^2)$$

MSE criterion should be used when fitting this model to a training data as the different test MSE values for model M_i (for different polynomial order) are compared to select the model with very less error rate to achieve the best fit. And also it has both variance and bias.

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{Var}_{\hat{\theta}}(\hat{\theta}) + \mathrm{Bias}(\hat{\theta}, \theta)^2$$

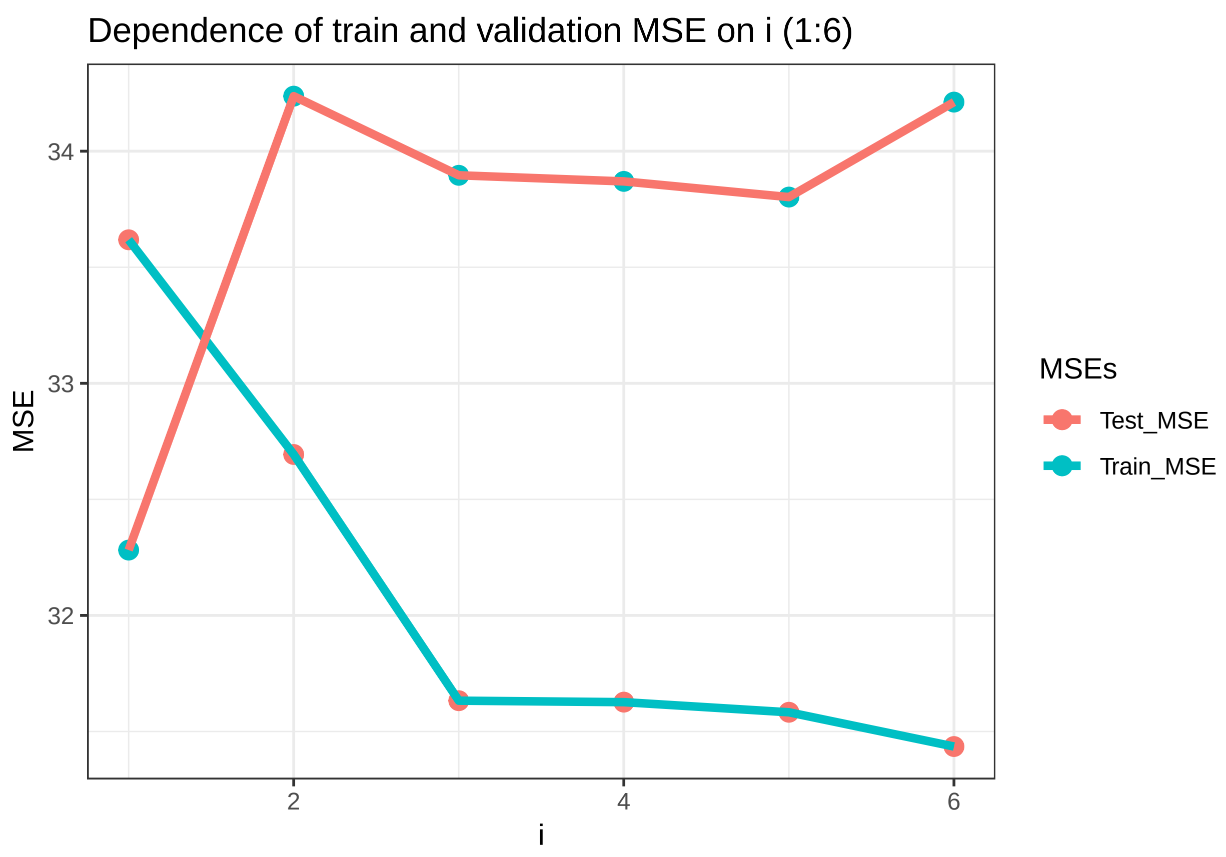
The best model will have both low bias and low variance. The models polynomial order "i" increases and this makes model to be complex. MSE value should be nearer to "0" for best fit. Since our model is normally distributed and as the polynomial order i increases,the functions M_i(x) are able to capture increasingly complex behavior. MSE is the best unbiased estimator minimizing the sample variance giving MSE value nearer to zero, which is the best. As linear regression overfits data for higher polynomial, ridge regression would be an appropriate probabilistic model to implement.

4.3

```r
# 3. Divide the data into training and validation sets( 50%/50%) and fit models M_i=1:6
n=dim(tecator)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
tecator_train=tecator[id,]
tecator_test=tecator[-id,]

train_mse <- c()
test_mse <- c()
for(i in 1:6) {
  model <- lm(formula=Moisture~poly(Protein,i,raw=T), data=tecator_train)
  train_mse[i] <- mean(model$residuals^2)
  test_mse[i] <- mean((tecator_test$Moisture - predict(model,tecator_test))^2)
}
# MSEs
list("Train MSEs"=train_mse,"Test MSEs"=test_mse)
```

```
## $`Train MSEs`
## [1] 33.61836 32.69342 31.63266 31.62641 31.58273 31.43513
##
## $`Test MSEs`
## [1] 32.28154 34.23708 33.89615 33.86992 33.80234 34.21152
```



Analysis:

The six models are fitted (as i=1:6) and the corresponding train and test MSEs are reported. The test MSE for linear model is 32.281 and it is lesser than other models. The test MSE initially increases as we increase the flexibility of the model but eventually starts to decrease and increases again after we introduce a lot of flexibility. The train MSE (given by the blue curve) decreases monotonically as the flexibility of the model increases. This is due to the reason that the polynomial fit can become as flexible as we need it to in order to minimise the difference between its values. So, we may conclude that linear model is best according to the plot.

We have different MSE values because of bias-variance tradeoff. As flexibility is increased the bias will tend to drop quickly (faster than the variance can increase) and so we see a drop in test MSE. However, as flexibility increases further, there is less reduction in bias (because the flexibility of the model can fit the training data easily) and instead the variance rapidly increases, due to the model being overfit.

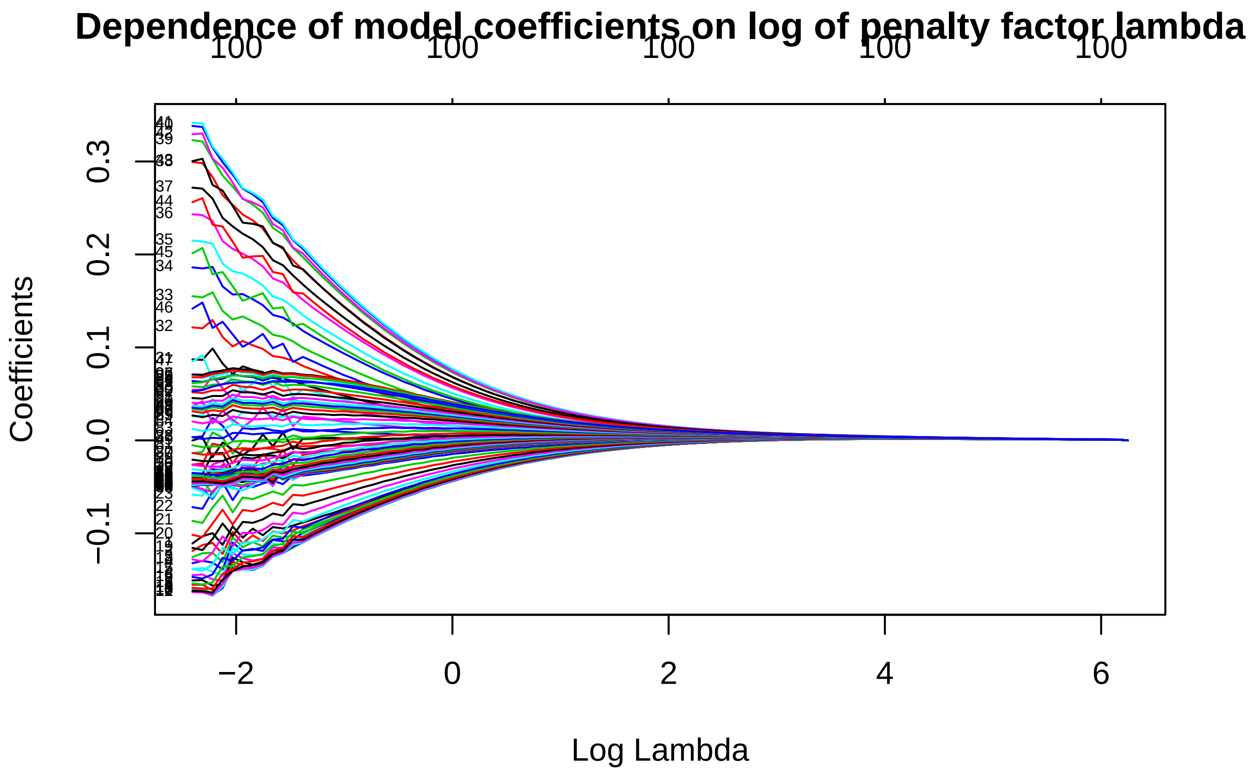
4.4

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## Fat ~ Channel1 + Channel2 + Channel3 + Channel4 + Channel5 +
##     Channel6 + Channel7 + Channel8 + Channel9 + Channel10 + Channel11 +
##     Channel12 + Channel13 + Channel14 + Channel15 + Channel16 +
##     Channel17 + Channel18 + Channel19 + Channel20 + Channel21 +
##     Channel22 + Channel23 + Channel24 + Channel25 + Channel26 +
##     Channel27 + Channel28 + Channel29 + Channel30 + Channel31 +
##     Channel32 + Channel33 + Channel34 + Channel35 + Channel36 +
##     Channel37 + Channel38 + Channel39 + Channel40 + Channel41 +
##     Channel42 + Channel43 + Channel44 + Channel45 + Channel46 +
##     Channel47 + Channel48 + Channel49 + Channel50 + Channel51 +
##     Channel52 + Channel53 + Channel54 + Channel55 + Channel56 +
##     Channel57 + Channel58 + Channel59 + Channel60 + Channel61 +
##     Channel62 + Channel63 + Channel64 + Channel65 + Channel66 +
##     Channel67 + Channel68 + Channel69 + Channel70 + Channel71 +
##     Channel72 + Channel73 + Channel74 + Channel75 + Channel76 +
##     Channel77 + Channel78 + Channel79 + Channel80 + Channel81 +
##     Channel82 + Channel83 + Channel84 + Channel85 + Channel86 +
##     Channel87 + Channel88 + Channel89 + Channel90 + Channel91 +
##     Channel92 + Channel93 + Channel94 + Channel95 + Channel96 +
##     Channel97 + Channel98 + Channel99 + Channel100
##
## Final Model:
## Fat ~ Channel1 + Channel2 + Channel4 + Channel5 + Channel7 +
##     Channel8 + Channel11 + Channel12 + Channel13 + Channel14 +
##     Channel15 + Channel17 + Channel19 + Channel20 + Channel22 +
##     Channel24 + Channel25 + Channel26 + Channel28 + Channel29 +
##     Channel30 + Channel32 + Channel34 + Channel36 + Channel37 +
##     Channel39 + Channel40 + Channel41 + Channel42 + Channel45 +
##     Channel46 + Channel47 + Channel48 + Channel50 + Channel51 +
##     Channel52 + Channel54 + Channel55 + Channel56 + Channel59 +
##     Channel60 + Channel61 + Channel63 + Channel64 + Channel65 +
##     Channel67 + Channel68 + Channel69 + Channel71 + Channel73 +
##     Channel74 + Channel78 + Channel79 + Channel80 + Channel81 +
##     Channel84 + Channel85 + Channel87 + Channel88 + Channel92 +
##     Channel94 + Channel98 + Channel99
##
##
##            Step Df     Deviance Resid. Df Resid. Dev       AIC
## 1                                    114   169.8123 151.27203
## 2   - Channel70  1 5.580758e-05       115   169.8124 149.27210
## 3   - Channel89  1 6.338934e-04       116   169.8130 147.27290
## 4   - Channel66  1 4.350148e-04       117   169.8135 145.27345
## 5  - Channel100  1 9.526559e-04       118   169.8144 143.27466
## 6   - Channel57  1 1.512331e-03       119   169.8159 141.27657
## 7   - Channel38  1 4.235150e-03       120   169.8202 139.28193
## 8   - Channel58  1 7.141818e-03       121   169.8273 137.29098
## 9   - Channel53  1 2.509829e-02       122   169.8524 135.32275
## 10   - Channel9  1 3.771904e-02       123   169.8901 133.37049
## 11  - Channel91  1 3.178511e-02       124   169.9219 131.41071
## 12  - Channel77  1 5.501288e-02       125   169.9769 129.48030
## 13  - Channel49  1 9.282875e-02       126   170.0698 127.59769
## 14  - Channel33  1 1.137405e-01       127   170.1835 125.74143
## 15  - Channel96  1 1.838591e-01       128   170.3674 123.97358
## 16  - Channel93  1 1.204802e-01       129   170.4878 122.12557
## 17  - Channel82  1 2.012906e-01       130   170.6891 120.37927
## 18  - Channel86  1 2.608049e-01       131   170.9499 118.70753
## 19  - Channel72  1 3.340581e-01       132   171.2840 117.12725
## 20  - Channel35  1 4.539629e-01       133   171.7380 115.69633
## 21  - Channel43  1 3.667681e-01       134   172.1047 114.15500
## 22  - Channel44  1 3.686336e-01       135   172.4734 112.61502
## 23  - Channel90  1 4.430432e-01       136   172.9164 111.16659
## 24  - Channel83  1 4.636039e-01       137   173.3800 109.74225
## 25   - Channel3  1 4.495464e-01       138   173.8295 108.29899
## 26  - Channel23  1 4.393963e-01       139   174.2689 106.84177
## 27   - Channel6  1 6.745513e-01       140   174.9435 105.67238
## 28  - Channel62  1 6.873639e-01       141   175.6309 104.51547
## 29  - Channel10  1 6.770690e-01       142   176.3079 103.34272
## 30  - Channel18  1 5.551316e-01       143   176.8631 102.01861
## 31  - Channel27  1 8.012085e-01       144   177.6643 100.99038
## 32  - Channel16  1 8.124404e-01       145   178.4767  99.97132
## 33  - Channel21  1 9.726859e-01       146   179.4494  99.13987
## 34  - Channel95  1 8.809590e-01       147   180.3304  98.19277
## 35  - Channel97  1 6.630855e-01       148   180.9934  96.98189
## 36  - Channel76  1 1.451145e+00       149   182.4446  96.69882
## 37  - Channel75  1 7.506552e-01       150   183.1952  95.58160
## 38  - Channel31  1 1.682931e+00       151   184.8782  95.54769
```

Analysis:

Using stepAIC, 63 variables were selected.

4.5

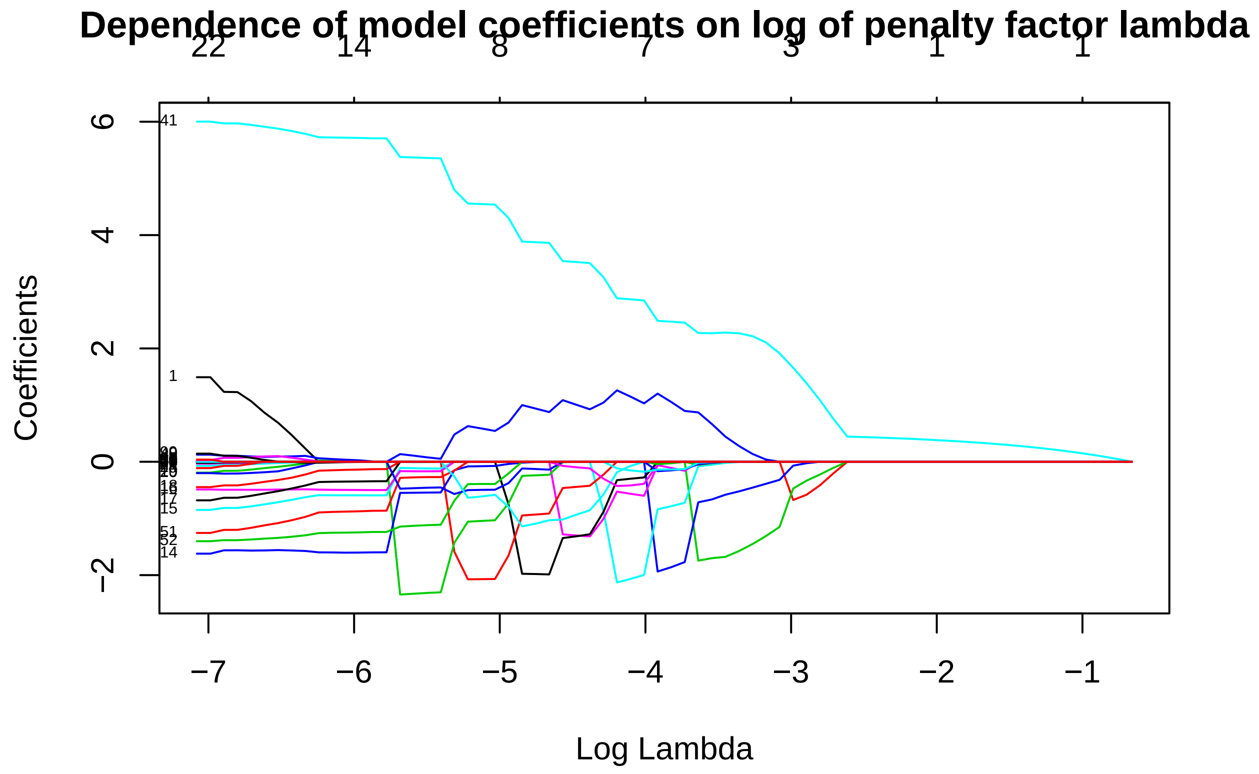


```
## integer(0)
```

Analysis:

In ridge regression, only the beta coefficients are minimized (towards zero) by lambda (shrinkage term) but variable selection is not done. As lambda increases, the coefficients are shrinked. The sum of squares of the coefficients is increasing until it reaches a point where lambda is effectively zero. Here almost all variables are selected (say 100 in plot). The faster a coefficient is shrinking the less important it is in prediction. The optimal lambda that best minimised the error in cross-validation is 0.0903.

4.6



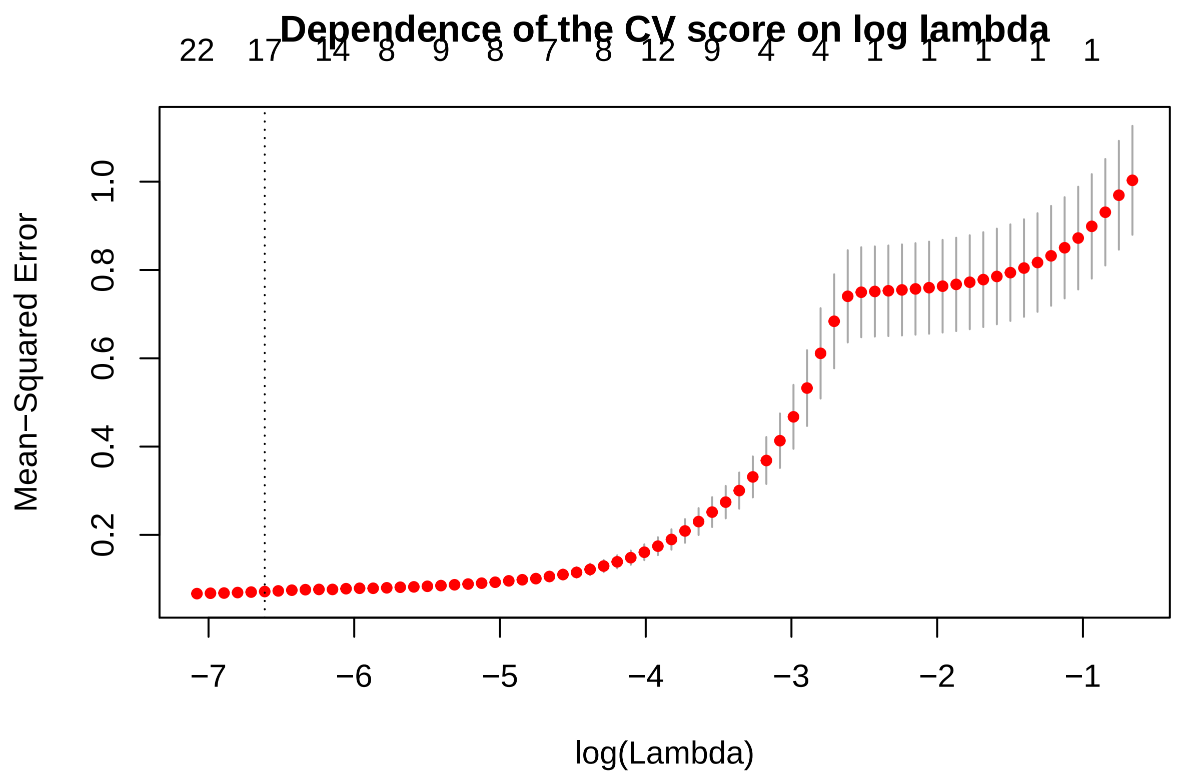
```
## integer(0)
```

Analysis:

In lasso regression, the shrinkage and also the variable selection is done. Each colored line represents the value (feature) taken by a different coefficient in model. The variable that most influence the model is 41, because it enters the model first, and steadily positively affect the response variable. Furthermore we can select other important variables, looking at their trends, such as 14, 1, and some other variables. Unlike Ridge regression, lasso shows the clear trend of coefficients for log lambda values.

4.7

```
## Optimal lambda :  0
```



```
## integer(0)

## 101 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  2.056770e-15
## Channel1     3.558856e+00
## Channel2    -1.002702e-01
## Channel3    -9.873364e-02
## Channel4    -1.037070e-01
## Channel5    -1.059657e-01
## Channel6    -1.042730e-01
## Channel7    -1.023191e-01
## Channel8    -9.723732e-02
## Channel9    -1.035566e-01
## Channel10   -1.306390e-01
## Channel11   -1.038289e-01
## Channel12   -1.054070e-01
## Channel13   -1.101994e-01
## Channel14   -1.852875e+00
## Channel15   -9.232649e-01
## Channel16   -4.317551e-01
## Channel17   -8.116618e-01
## Channel18   -2.760541e-01
## Channel19   -3.241280e-01
## Channel20   -2.285526e-02
## Channel21   -1.904541e-01
## Channel22   -1.365259e-01
## Channel23   -9.882698e-02
## Channel24   -3.379959e-01
## Channel25    9.767010e-06
## Channel26    1.564449e-05
## Channel27    1.177612e-05
## Channel28    4.168151e-06
## Channel29   -2.192310e-06
## Channel30   -1.419133e-05
## Channel31   -2.929112e-05
## Channel32   -3.585348e-05
## Channel33   -2.923911e-05
## Channel34   -4.393845e-01
## Channel35    5.078211e-02
## Channel36    8.741396e-02
## Channel37    1.170235e-01
## Channel38    1.256048e-01
## Channel39    1.022836e-01
## Channel40    1.937027e-01
## Channel41    6.046320e+00
## Channel42    1.920580e-05
## Channel43    3.072625e-05
## Channel44    2.810021e-05
## Channel45    1.329839e-05
## Channel46   -6.508176e-06
## Channel47   -1.927326e-05
## Channel48   -1.142689e-01
## Channel49   -1.640467e-01
## Channel50   -1.520684e-01
## Channel51   -1.439607e+00
## Channel52   -1.450626e+00
## Channel53    1.173403e-01
## Channel54   -2.544180e-02
## Channel55    2.379556e-01
## Channel56    1.786641e-01
## Channel57    1.088342e-01
## Channel58    3.827317e-02
## Channel59   -3.337061e-02
## Channel60   -4.678122e-02
## Channel61   -7.660996e-02
## Channel62   -9.232219e-02
## Channel63   -1.211865e-01
## Channel64   -1.434675e-01
## Channel65   -9.603867e-05
## Channel66   -9.476279e-05
## Channel67   -8.313058e-05
## Channel68   -7.924006e-05
## Channel69   -9.077200e-05
## Channel70   -9.834521e-05
## Channel71   -8.793983e-05
## Channel72   -7.111501e-05
## Channel73   -7.348592e-05
## Channel74   -8.008236e-05
## Channel75   -6.421790e-05
## Channel76   -3.573192e-05
## Channel77   -3.109089e-05
## Channel78   -2.433485e-05
## Channel79   -1.915311e-05
## Channel80   -7.919000e-06
## Channel81   -4.173118e-06
## Channel82    3.299737e-07
## Channel83    2.585382e-06
## Channel84    4.304522e-06
## Channel85    1.452775e-05
## Channel86    2.351431e-05
## Channel87    3.420146e-05
## Channel88    3.989331e-05
## Channel89    4.368505e-05
## Channel90    4.188634e-05
## Channel91    4.235487e-05
## Channel92    4.708440e-05
## Channel93    4.907099e-05
## Channel94    5.064515e-05
## Channel95    5.160440e-05
## Channel96    5.168853e-05
## Channel97    5.446463e-05
## Channel98   -1.726322e-01
## Channel99    3.267044e-01
## Channel100   3.117440e-01
```

Analysis:

Using cross validation, the optimal λ value is *0* and all variables (Channels) have been choosen by the model. The Lasso method extracts different lambda values like lambda_min (first dotted line in plot) which that gives minimum mean cross-validated error and lambda_1se (second dotted line) which gives a model such that error is within one standard error of the minimum. The upper part of the plot shows the number of coefficients for a given log lambda. And as it is obviously shown in the plot, the MSE increases when the value of log λ increases, shrinkage occurs so that variables that are at zero can be thrown away.

4.8

Comparing steps 4 and 7:

The variable selection differs between stepAIC and Lasso fit. The stepwise algorithm (stepAIC) accounts for the full model uncertainty. Here it selects some variables. But the lasso fit (with cross validation) selects all features (Channels corresponding to Fat variable) based on the lambda values (as we specify lambda values from 0). The stepAIC method seems to be better when compared with Lassofit in this case.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
# Assignment 1. Spam classification with nearest neighbors
# 1.load data and divide it into training and test sets (50%/50%)
library(xlsx)
spambase <- read.xlsx("spambase.xlsx",1)
n=dim(spambase)[1]
```

```r
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=spambase[id,]
test=spambase[-id,]
# 2. Use logistic regression (glm(), predict()) to classify the training and test data
set.seed(12345)
spam_model <- glm(Spam ~.,family=binomial(link='logit'), data=train)

##Y=1 if p(Y)>0.5, Y=0
# prediction for test
predicted_test <- predict(spam_model,test, type="response")
classify <- ifelse(predicted_test > 0.5, "spam", "notspam")

# confusion matrix for test
test_matrix <- table(test$Spam,classify)
cat(paste("Confusion Matrix for test data : "))
test_matrix

# misclassification rates for test data
# true positive value=270, error % = 270/2740= 0.09854014598
misclassification_rate_test <- round(1 - (sum(diag(test_matrix))/sum(test_matrix)),4)
paste("Misclassification rate of test data : ", misclassification_rate_test)
# prediction for  train
prediction_train <- predict(spam_model,train,type="response")
classify_train <- ifelse(prediction_train > 0.5, "spam", "notspam")

# confusion matrix for train
train_matrix <- table(train$Spam,classify_train)
cat(paste("Confusion Matrix for train data : "))
train_matrix

# misclassification rates  for train data
misclassification_rate_train <- round(1 - (sum(diag(train_matrix))/sum(train_matrix)),4)
paste("Misclassification rate of train data : ", misclassification_rate_train)
# 3. Use logistic regression (functions glm(), predict()) to classify the training and test data
#Y=1 if p(Y)>0.9, Y=0

# prediction for test
predicted_data_3 <- predict(spam_model,test,type="response")
classify_3 <- ifelse(predicted_data_3 > 0.9, "spam", "notspam")

# confusion matrix for test
matrix_3 <- table(test$Spam,classify_3)
cat(paste("Confusion Matrix for test data : "))
matrix_3

# misclassification rates for test
misclass_rate_test <- round(1 - (sum(diag(matrix_3))/sum(matrix_3)),4)
paste("Misclassification rate of test data : ", misclass_rate_test)

# prediction for  train
pred_train_3 <- predict(spam_model,train,type="response")
classify_train_3 <- ifelse(pred_train_3 > 0.9, "spam", "notspam")
```

```r
# confusion matrix for train
train_matrix_3 <- table(train$Spam,classify_train_3)
cat(paste("Confusion Matrix for train data : "))
train_matrix_3

# misclassification rates for train
misclass_rate_train <- round(1 - (sum(diag(train_matrix_3))/sum(train_matrix_3)),4)
paste("Misclassification rate of train data : ", misclass_rate_train)
# 4. Use standard classifier kknn() with K=30 from package kknn
library(kknn)
#for test
kknn_model1 <- kknn(factor(Spam) ~., train, test, k=30, kernel = "optimal")

fit <- fitted(kknn_model1)  # to know the fitted values by comparing probabilities

# confusion matrix for test
kknn_matrix1 <- table(fit, test$Spam)
# misclassification rate
kknn_misclass_rate1 <- round(1 - (sum(diag(kknn_matrix1))/sum(kknn_matrix1)),4)
paste("Misclassification rate of test data : ", kknn_misclass_rate1)

#for train
kknn_model1_train <- kknn(factor(Spam) ~ ., train, train, k = 30,   kernel = "optimal")
fit_train <- fitted(kknn_model1_train)
# confusion matrix for train
kknn_matrix1_train <- table(fit_train, train$Spam)
kknn_misclass_rate1_train <- round(1 - (sum(diag(kknn_matrix1_train))/sum(kknn_matrix1_train)),4)
paste("Misclassification rate of train data : ", kknn_misclass_rate1_train)

# 5. Repeat step 4 for K=1
#for test
kknn_model2 <- kknn(factor(Spam) ~., train, test, k=1,  kernel = "optimal")
fit_k_1 <- fitted(kknn_model2)
# confusion matrix for test
kknn_matrix2 <- table(fit_k_1, test$Spam)
kknn_misclass_rate2 <- round(1 - (sum(diag(kknn_matrix2))/sum(kknn_matrix2)),4)
paste("Misclassification rate of test data : ", kknn_misclass_rate2)

#for train
kknn_model2_train <- kknn(factor(Spam) ~ ., train, train, k = 1,    kernel = "optimal")
fit1_train <- fitted(kknn_model2_train)

# confusion matrix for train
kknn_matrix2_train <- table(fit1_train, train$Spam)
kknn_misclass_rate2_train <- round(1 - (sum(diag(kknn_matrix2_train))/sum(kknn_matrix2_train)),4)
paste("Misclassification rate of train data : ", kknn_misclass_rate2_train)

# Assignment 3. Feature selection by cross-validation in a linear model
best_subset <- function(X,Y,Nfolds){
  data <- data.frame()
 # gen random numbers for combinations
  for (i in 1:ncol(X)){
    comb <- combn(ncol(X), i)
    cv_vec <- c()
  #select required variable
    for (j in 1:ncol(comb)){
      sel_var <- comb[,j]
      sel_X <- X[,sel_var, drop = FALSE]
  #kfold creating
      kfold <- split(1:nrow(sel_X), sort((1:nrow(sel_X))%% Nfolds))
    #for each fold, model data
      cv <- 0
      for(k in 1:Nfolds) {
        fold <- kfold[[k]]
        train_data <- as.matrix(sel_X[-fold, ])
        train_data_y <- as.matrix(Y[-fold, ])
        train_data <- cbind(train_data, rep(1,nrow(train_data)))

        test_data  <- as.matrix(sel_X[fold,])
        test_data_y <- as.matrix(Y[fold,])
        test_data <- cbind(test_data, rep(1, nrow(test_data)))
       #fit and prediction formulas
        beta <- solve(t(train_data) %*% train_data) %*% t(train_data) %*% train_data_y
        prediction <- test_data %*% beta
        cv <- mean(sum(cv, mean((test_data_y-prediction)^2)))
      }
      cv <- cv/Nfolds
      cv_vec <- c(cv_vec, cv)
      combinations <- paste(as.character(as.vector(sel_var)), collapse=", ")
      data_comb_CV <- data.frame(Combinations = combinations,CV_score = cv, Num_of_features = length(se
      data <- rbind(data, data_comb_CV)
    }
  }
  opt_feature <- data$Combinations[which(data$CV_score == min(data$CV_score))]
  opt_cv <- data$CV_score[which(data$CV_score == min(data$CV_score))]
  cat(paste("Optimal Subset of Features: ",opt_feature,"   CV Score: ", opt_cv))
  # Plot of CV scores against number of features
  library(ggplot2)
  p <- ggplot(data) + geom_point(aes(x=Num_of_features, y=CV_score)) +
       ggtitle("CV scores against number of features") + theme_dark()

  print(p)
  return(data)
}

set.seed(12345)
swiss_data <- swiss[sample(nrow(swiss)),]
swiss_model <- best_subset(swiss_data[,2:6], swiss_data[,1, drop = FALSE], 5)
# Assignment 4. Linear regression and regularization
# 1. load data
tecator <- read.xlsx("tecator.xlsx",1)

#plot of Moisture versus Protein
library(ggplot2)
ggplot(tecator)+geom_point(aes(Protein,Moisture))+
  geom_smooth(aes(Protein,Moisture),method="loess", size=2)+
  ggtitle("Moisture ~ Protein")+xlab("Protein")+ylab("Moisture")+
  theme_classic()
knitr::include_graphics("MSE.PNG")
# 3. Divide the data into training and validation sets( 50%/50%) and fit models M_i=1:6
n=dim(tecator)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
tecator_train=tecator[id,]
tecator_test=tecator[-id,]

train_mse <- c()
test_mse <- c()
for(i in 1:6) {
  model <- lm(formula=Moisture~poly(Protein,i,raw=T), data=tecator_train)
  train_mse[i] <- mean(model$residuals^2)
  test_mse[i] <- mean((tecator_test$Moisture - predict(model,tecator_test))^2)
}
# MSEs
list("Train MSEs"=train_mse,"Test MSEs"=test_mse)
#visualize MSE
mse_p <- data.frame(train_mse, test_mse)
library(ggplot2)
ggplot()+geom_point(data=mse_p, aes(x=1:6, y=train_mse,color="blue"),size=3)+
  geom_point(data=mse_p, aes(x=1:6, y=test_mse,color="green"),size=3)+
  geom_line(data=mse_p, aes(x=1:6, y=train_mse,color="green"),size=1.5)+
  geom_line(data=mse_p, aes(x=1:6, y=test_mse,color="blue"),size=1.5)+
  ggtitle("Dependence of train and validation MSE on i (1:6)")+xlab("i")+ ylab("MSE")+
  scale_color_discrete(name = "MSEs", labels = c("Test_MSE", "Train_MSE"))+theme_bw()
# 4. Perform variable selection of a linear model in which Fat is response and
#Channel1-Channel100 are predictors by using stepAIC
set.seed(12345)
AIC_data <- tecator[,c(2:102)]
model <- lm(Fat~., data = AIC_data) # Fat is dependent variable

# stepwise selection procedure, use stepAIC()
library(MASS)
step <- stepAIC(model,direction = "both", trace = FALSE)
step$anova
# Fit a Ridge regression model
library(glmnet)
set.seed(12345)
Fat_response <- scale(tecator[c("Fat")])
Channel_covariate <- scale(tecator[,2:101])
Ridge.model <- glmnet(as.matrix(Channel_covariate),Fat_response, alpha=0,family="gaussian")
plot(Ridge.model, xvar="lambda", label=TRUE)+
  title("Dependence of model coefficients on log of penalty factor lambda")
#min(Ridge.model$lambda)
# Fit a Lasso regression model
set.seed(12345)
Lasso.model <- glmnet(as.matrix(Channel_covariate),Fat_response, alpha=1,family="gaussian")
lambda_value <- Lasso.model$lambda
plot(Lasso.model, xvar="lambda", label=TRUE)+
  title("Dependence of model coefficients on log of penalty factor lambda")
```

```r
#Use cross-validation to find the optimal LASSO model
set.seed(12345)
Lasso_cvmodel <- cv.glmnet(as.matrix(Channel_covariate), Fat_response, alpha=1,
                           family="gaussian", lambda = c(0,lambda_value))
opt_lambda <- Lasso_cvmodel$lambda.min
cat(paste("Optimal lambda : ",opt_lambda))
plot(Lasso_cvmodel, xvar="lambda", label=TRUE)+
  title("Dependence of the CV score on log lambda")
coef(Lasso_cvmodel, s="lambda.min")
```