Computer lab 1 - Helpfile

Student

15 6 2019

Contents


```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

Assignemnt 1

Spam classification with nearest neighbors

1.1 Import data

```
# read the data
data = read_xlsx("spambase.xlsx")
```

```
## readxl works best with a newer version of the tibble package.
## You currently have tibble v1.4.2.
## Falling back to column name repair from tibble <= v1.4.2.
## Message displays once per session.
```

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

1.2 Logistic regression

```
# function to create prediction of spam classification
Spam_predclass_confusionmatrix_misclass = function(data_train_test, classification_rate){


  # create the model
  glm_fit = glm(Spam ~ .,
                family = binomial, # for 0,1 case - binomial
                data = train)

    # predict data on train or test data
    # make a prediction, of probability
    glm_pred = predict(object = glm_fit,
                       type = "response", # removes the dependent var
                       newdata = data_train_test)


  # classify by given classification principle
  # Y_hat = p(Y=1 | X) > 0.5 - 1 = spam, 0 = no spam
  glm_classi = ifelse(glm_pred > classification_rate, 1,0)

  # compute confusion matrix
  confustion_matrix = table(data_train_test$Spam, glm_classi)

  # missclassification
  missclassificationrate = round(1-sum(diag(confustion_matrix))/sum(confustion_matrix),2)

  result = list("classification" = glm_classi,
                "confusion_matrix" = confustion_matrix,
```

```
                 "missclassification_rate" = missclassificationrate)
  return(result)
}
```

Case :

- Training set with classification probability 0.5

```
results_1.2_train_0.5 = Spam_predclass_confusionmatrix_misclass(train, 0.5)
```

Confusion Matrix

```
results_1.2_train_0.5$confusion_matrix
```

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

Misclassifiation rate

```
## [1] 0.16
```

Case :

- Test set with classification probability 0.5

```
results_1.2_test_0.5 = Spam_predclass_confusionmatrix_misclass(test, 0.5)
```

Confusion Matrix

```
results_1.2_test_0.5$confusion_matrix
```

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

Misclassifiation rate

```
## [1] 0.18
```

1.3 Logistic regression

Case:

- Training set with classification probability 0.9

```
results_1.2_train_0.9 = Spam_predclass_confusionmatrix_misclass(train, 0.9)
```

Confusion Matrix

```
results_1.2_train_0.9$confusion_matrix
```

```
##    glm_classi
##       0   1
##   0 944   1
##   1 419   6
```

Misclassifiation rate

```r
results_1.2_train_0.9$missclassification_rate
```

```
## [1] 0.31
```

Case :

- Test set with classification probability 0.5

```r
results_1.2_test_0.9 = Spam_predclass_confusionmatrix_misclass(test, 0.9)
```

Confusion Matrix

```r
results_1.2_test_0.9$confusion_matrix
```

```
##    glm_classi
##       0   1
##   0 936   1
##   1 427   6
```

Misclassifiation rate

```
## [1] 0.31
```

1.4 Nearest neighbor classifier K=30

```r
# use classifier kknn()
Spam_pred_knn_func = function(data_train_test, k){

  # pred
  Spam_pred_knn = kknn(formula = as.factor(Spam)~. ,
                     k = k,
                     train = train,
                     test = data_train_test)

  # confusion matrix
  cm_knn = table(data_train_test$Spam, Spam_pred_knn$fitted.values)

  # missclassification
  missclassificationrate = round(1-sum(diag(cm_knn))/sum(cm_knn),2)

  result = list("classification" = Spam_pred_knn,
                "confusion_matrix" = cm_knn,
                "missclassification_rate" = missclassificationrate)
  return(result)
}
```

Case:

Training set - K = 30

```r
nearest_neighboar_K30_train = Spam_pred_knn_func(train, 30)
```

Confusion Matrix:

```r
nearest_neighboar_K30_train$confusion_matrix
```

```
##
##       0   1
```

```
##    0 807 138
##    1  98 327
```

Misclassification:

```r
nearest_neighboar_K30_train$missclassification_rate
```

```
## [1] 0.17
```

Test set - K = 30

Confusion Matrix:

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

Misclassification:

```
## [1] 0.33
```

1.5 Nearest neighbor classifier K=1

Case:

- Training set K = 1

Confusion Matrix:

```
##
##        0   1
##    0 945   0
##    1   0 425
```

Misclassification:

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

- Test set K = 1

Confusion Matrix:

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

Misclassification:

```
## [1] 0.35
```

Assignment 2

Inference about lifetime of machines

```r
# clean environment
rm(list = ls())
```

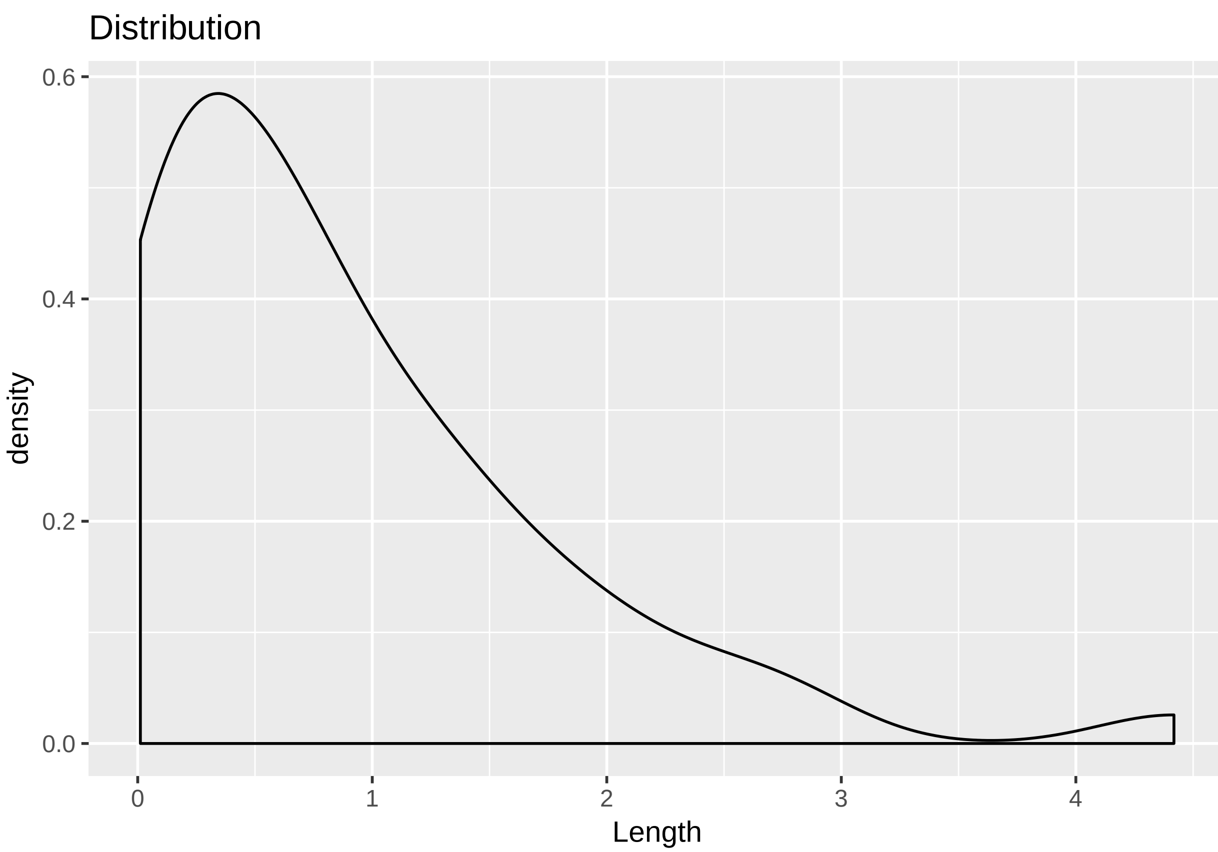
2.1 Import the data

```
# read the data
data = read_xlsx("machines.xlsx")
```

2.2, 2.3 Log likelihood

Distribution of x:

```
# What is the distribution type of x
ggplot(data) +
  #geom_histogram(aes(x = Length, color = "hist")) +
  geom_density(aes(x = Length)) +
  ggtitle("Distribution")
```



Implement the log likelihood:

```
log_likelihood = function(theta,x){
  n = length(x)
  ll_func = n*log(theta) - theta*sum(x)
  return(ll_func)
}
```

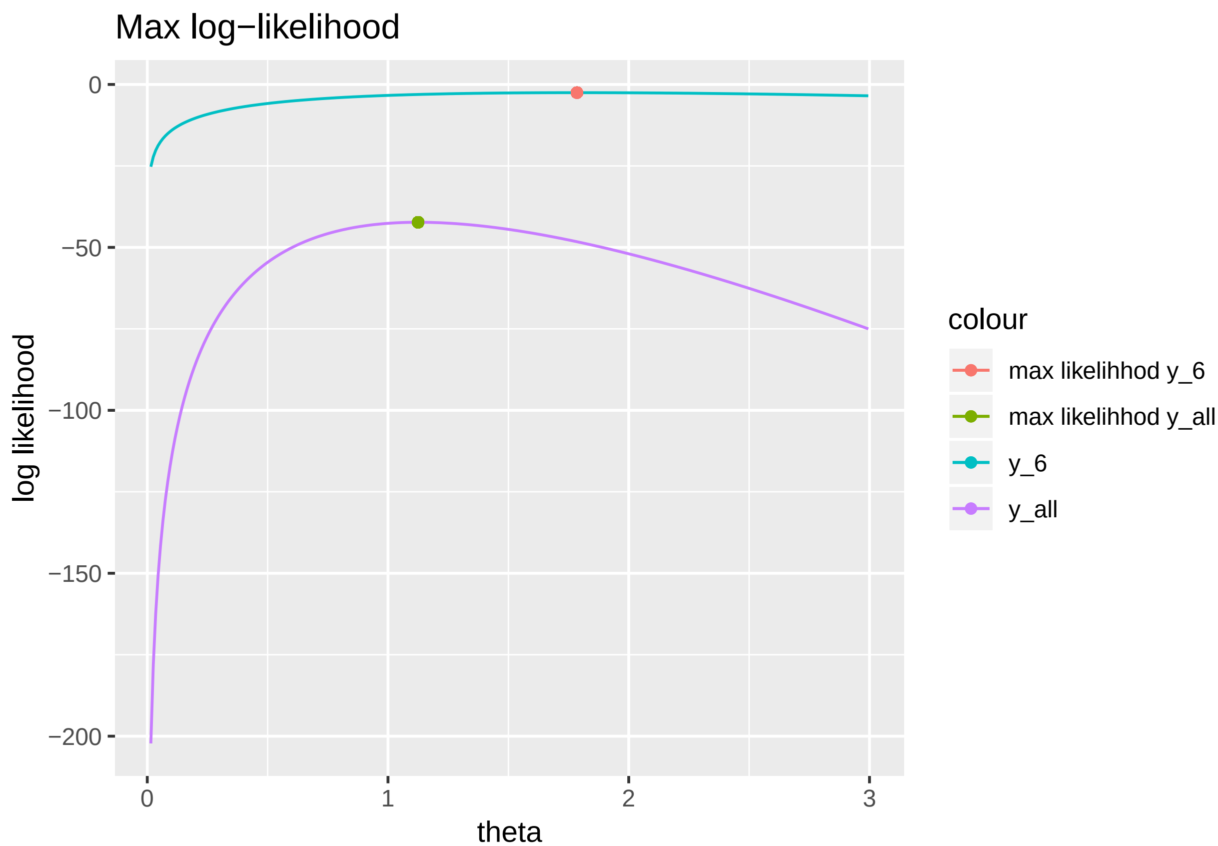
log-likelihood for all observations and for the first 6 observations

```
theta = seq(0.015, 3, 0.01)
y_all = log_likelihood(theta, data$Length)
y_6 = log_likelihood(theta, data$Length[1:6])
```

```r
plot_data = data.frame("theta" = theta,
                       "y_all" = y_all,
                       "y_6" = y_6)

theta_hat_all = theta[which.max(y_all)]
theta_hat_6 = theta[which.max(y_6)]

ggplot(plot_data) +
  geom_line(aes(x = theta, y = y_all, color = "y_all")) +
  geom_line(aes(x = theta, y = y_6, color = "y_6")) +
  ggtitle("Max log-likelihood") + ylab("log likelihood")+
  geom_point(aes(x = theta_hat_all ,
                 y = max(y_all), color = "max likelihhod y_all"))+
  geom_point(aes(x = theta_hat_6,
                 y = max(y_6), color = "max likelihhod y_6"))
```




```r
  #+ ylim(c(-100,0))

print("The maximum likelihood of theta for all is:")
```

```
## [1] "The maximum likelihood of theta for all is:"
```

```r
print(theta_hat_all)
```

```
## [1] 1.125
```

```r
print("The maximum likelihood of theta for y_6 is:")
```

```
## [1] "The maximum likelihood of theta for y_6 is:"
```

```r
print(theta_hat_6)
```

```
## [1] 1.785
```

2.4 Bayesian model

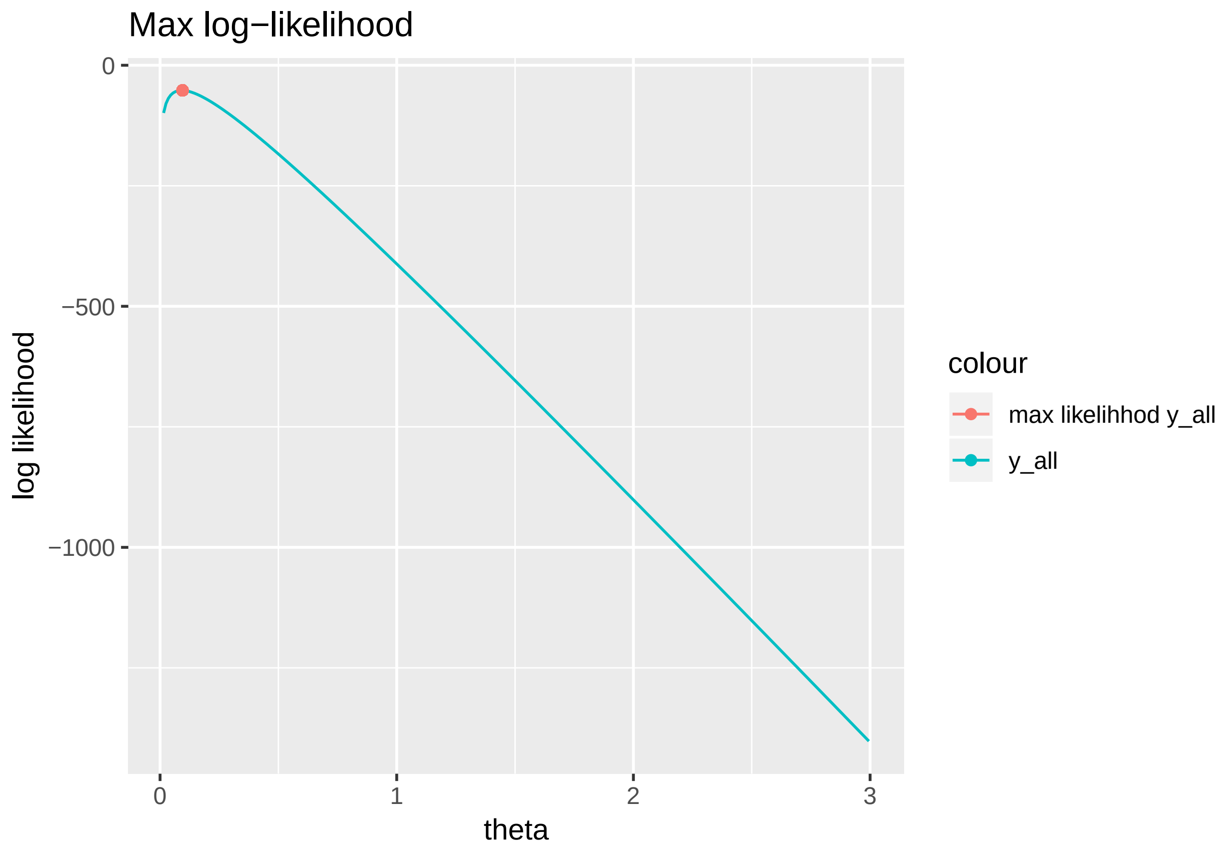
```r
#bayesian model
log_posterior = function(theta, x){
  n = length(x)
  lambda = 10
  l_post = n*log(lambda) + n*log(theta) - theta*(sum(x)+lambda*n)
  return(l_post)
}
```

```r
theta = seq(0.015, 3, 0.01)
y_log_post = log_posterior(theta, data$Length)

plot_data = data.frame("theta" = theta,
                       "y_log_post" = y_log_post)

theta_hat_log_post = theta[which.max(y_log_post)]

ggplot(plot_data) +
  geom_line(aes(x = theta, y = y_log_post, color = "y_all")) +
  ggtitle("Max log-likelihood") + ylab("log likelihood")+
  geom_point(aes(x = theta_hat_log_post,
                 y = max(y_log_post), color = "max likelihhod y_all"))
```




```
  #+ ylim(c(-100,0))

print("The maximum likelihood of theta for all is:")
```

```
## [1] "The maximum likelihood of theta for all is:"
```

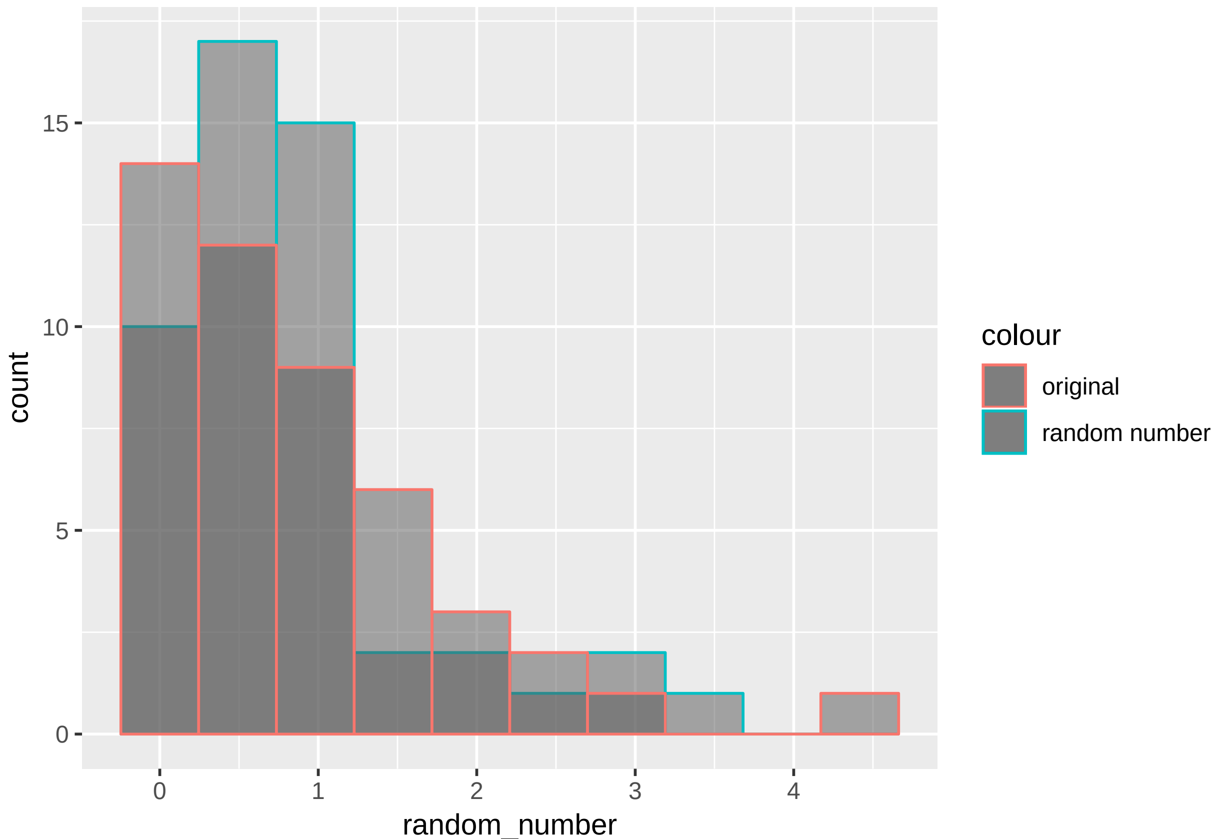
```
print(theta_hat_log_post)
```

```
## [1] 0.095
```

2.5 Use theta from step 2.2 (theta_hat_all)

```
# create random number
random_number = rexp(50,rate = theta_hat_all)
```

```
# create 2 hist
ggplot() +
  geom_histogram(aes(x = random_number, color = "random number"),
                 alpha = 0.5,
                 bins = 10) +
  geom_histogram(aes(x = data$Length, color = "original"),
                 alpha = 0.5,
                 bins = 10)
```



Assignemnt 3

Feature selection by cross-validation in a linear model.

```
data("swiss")
set.seed(12345)
X<-as.matrix(swiss[,-1])

Y<-as.matrix(swiss[,1])

test<-function(beta, X, Y,df){
  #Test calculates the MSE.
  X<-as.matrix(X)
  n<-dim(X)[1]
  intercept<-rep(1,n)# Adding intercept. X<-cbind(intercept,X)
  #SSE<-t(Y)%*%Y-t(beta)%*%t(X)%*%Y
  YtXb<-Y-X%*%beta
  SSE<-t(YtXb)%*%YtXb
  #Returns the SSE.
  SSE<-sum(SSE)
  MSE<-SSE/(df)#Returns MSE.
  return(MSE)
}
```

```r
beta<-function(X,Y){
  #This function estimates the betas.
  X<-as.matrix(X)
  n<-dim(X)[1]
  intercept<-rep(1,n)#Adding the intercept. X<-cbind(intercept,X)
  corX<-t(X)%*%X#Covariance matrix.
  Y<-as.matrix(Y)
  tXY<-t(X)%*%Y
  invtXX<-solve(corX)#Inverse of covariance matrix.
  bet<-invtXX%*%tXY
  #beta values. #SSE<-t(Y)%*%Y-t(bet)%*%tXY #MSE<-SSE/(dim(X)[1]-dim(X)[2])
  #bet[length(bet)+1]<-MSE
  return(bet)
}

folder<-function(X,Y,k=5){
  #This function shuffles the indexes and then runs the folds.
  #It also calls the functionstest for mse and beta for beta
  #estimation
  set.seed(12345)

  n<-dim(as.matrix(X))[1]; folds<-n/k-1; folds_vec<-seq(1,n,ceiling(folds))
  folds_vec[length(folds_vec)]<-n # Taking the mesurment of all the inputs.
  shuffled<-sample(1:n, n, replace = F)

  X<-as.matrix(X, drop=FALSE)
  X<- X[shuffled,,drop=FALSE] ; Y<-Y[shuffled] ## Shuffles the order of the observations.
  # Making a loop.
  # Prepering containers for the values.
  results<-c()
  nloops<-length(folds_vec)-1#
  testfold<-(folds_vec[1]):(folds_vec[2])
  X<-as.matrix(X,drop=FALSE)
  results<-beta(X[-testfold,,drop=FALSE],Y[-testfold])
  MSE<-c()

  #The first estimation is done outside so that no numbers will used twice.
  df<-dim(X[-testfold,,drop=FALSE])[1]-dim(X[-testfold,,drop=FALSE])[2]
  MSE[1]<-test(results,X[-testfold,,drop=FALSE],Y[-testfold],df)

  #Fold loop.
  for(i in 2:nloops){
    testfold<-(folds_vec[i]+1):(folds_vec[i+1])
    results1<-beta(X[-testfold,,drop=FALSE],Y[-testfold])
    results<-cbind(results,results1)
    df<-dim(X[-testfold,,drop=FALSE])[1]-dim(X[-testfold,,drop=FALSE])[2]
    MSE[i]<-test(results1,X[-testfold,,drop=FALSE],Y[-testfold],df)
  }

  results<-rbind(results,MSE)
  return(results)
}
```

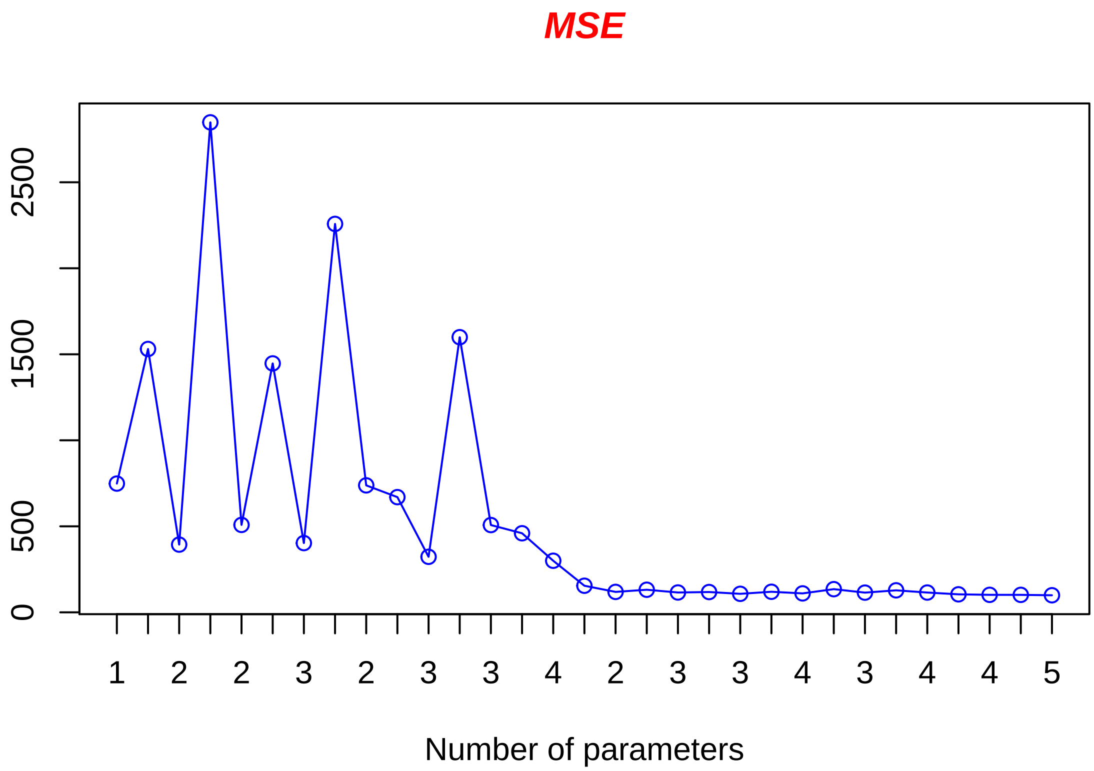
```r
Nfold<-function(x,y,k=5){
  #Nfold creates all the combinations that of the variables that are possible.
  leng<-dim(x)[2]
  result_list<-list()

  for(i in 1:(2^leng-1)){
    variable_test<-intToBits(i)[1:leng]#Using a binary form to make sure all varibles are
    variable_test<-which(variable_test==01)#Adds the variables.
    x1<-as.matrix(x,drop=FALSE)
    x1<-x[,variable_test]
    result_list[[i]]<-folder(x1,y,k)# Stores the results.
  }

  best<-unlist(lapply(result_list, function(x){mean(x["MSE",])}))#Takes out the mse values
  best1<-which.min(best)# Gets the index of the lowest value.
  label<-c()

  for(i in 1:(2^leng-1)){
    feture<-intToBits(i)[1:5]
    feture<-which(feture==01); feture<-length(feture)
    label[i]<-feture}
    plot(best, type="o", col="blue", xaxt = "n" ,ann=FALSE)
    title(main="MSE", xlab = "Number of parameters",
          col.main="red",
          font.main=4)
    axis(1,at=1:31, labels=label )
    best2<-rowMeans(result_list[[best1]])#Calculates the mean.
    return(best2)
  }

Nfold(X,Y,5)
```



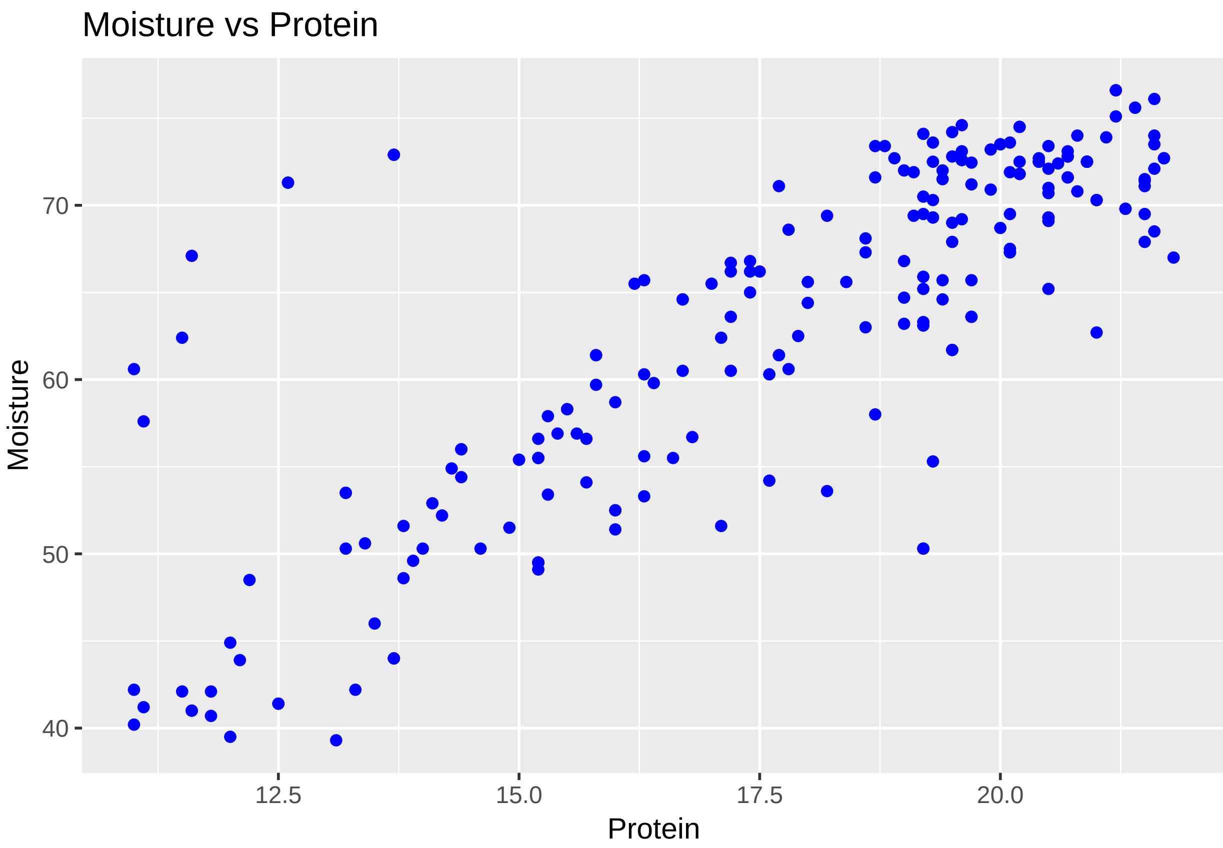

```
##      Agriculture     Examination       Education        Catholic
##        0.1073760       0.4378462      -0.6996480       0.1175659
## Infant.Mortality             MSE
##        2.9967636      99.2722954
```

Assignment 4

Linear regression and regularization

4.1 Import & plot data

```r
# Import the data
data = read_xlsx("tecator.xlsx")
```



4.2 Probabilistic model

which describes M

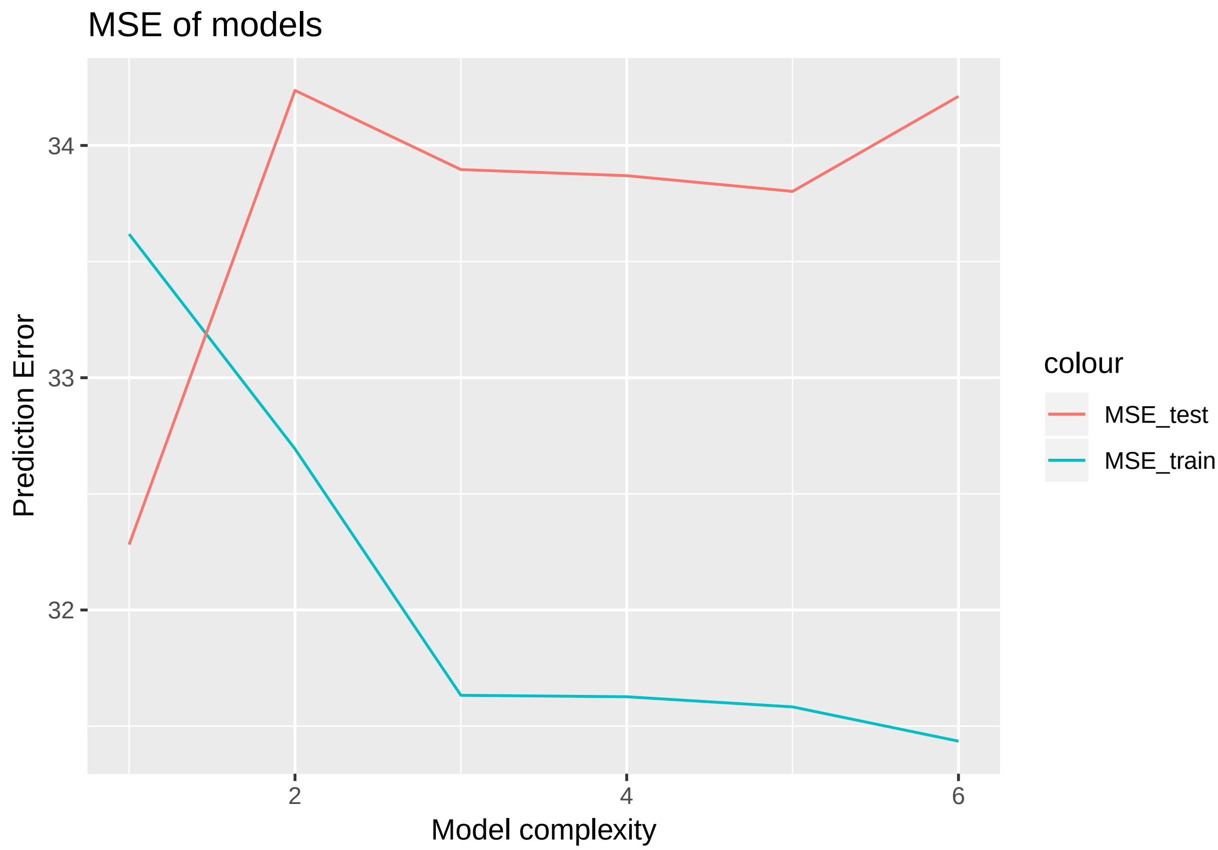
Theorie question - Lecture 1d $M_i\tilde{}\ N(prot_i x, \sigma^2)$ or $p(M|x, prot) = N(prot_i x, \sigma^2)$

4.3 Fit models

```
# fit the model
# create 6 model

# want to save the MSE, create empty variable for loop
models = seq(from = 1, to =6)
MSE_train = numeric(length = length(models))
MSE_validation = numeric(length = length(models))

for (i in models) {
  fit_train = lm(Moisture ~ poly(Protein, degree = i, raw=TRUE), data = train)
  MSE_train[i] = mean(fit_train$residuals^2)
  prediction =predict(fit_train,
                      newdata = test)
  MSE_validation[i] = mean((prediction-test$Moisture)^2)
}
```



4.4 Variable selection

```
####Step 4####
subset = data[,2:101] #only channel1-channel100
data_lm = lm(data$Fat ~ ., subset)
stepAIC = stepAIC(data_lm, trace=FALSE)
```

Length of selected variables:

```
length(stepAIC$coefficients)
```

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

4.5 Ridge regression

```
# fit ridge regression

# predictor and response variables for the model
response = scale(data[,"Fat"]) # just take the response var
covariates = scale(data[,2:101]) # remove sample column
#remove colnames
colnames(response) = NULL
colnames(covariates) = NULL

# create fit of ridge regression
```

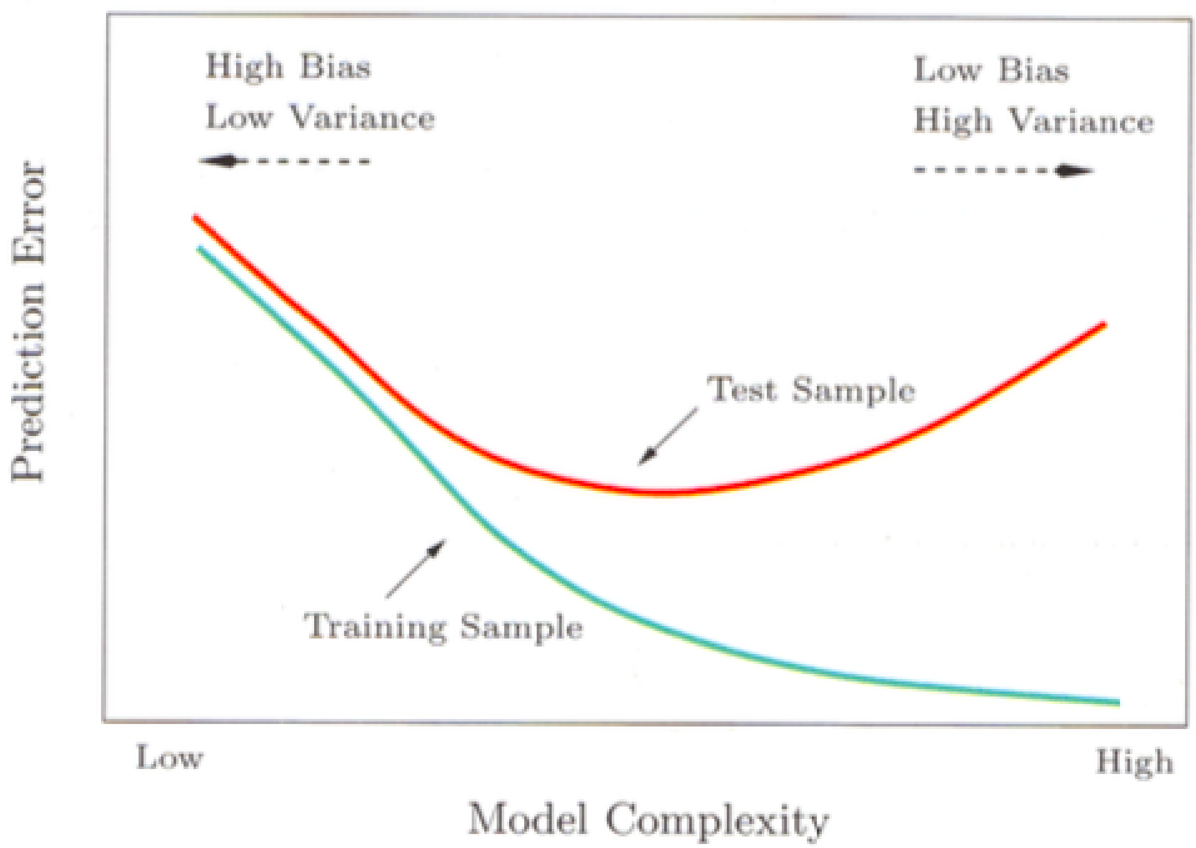
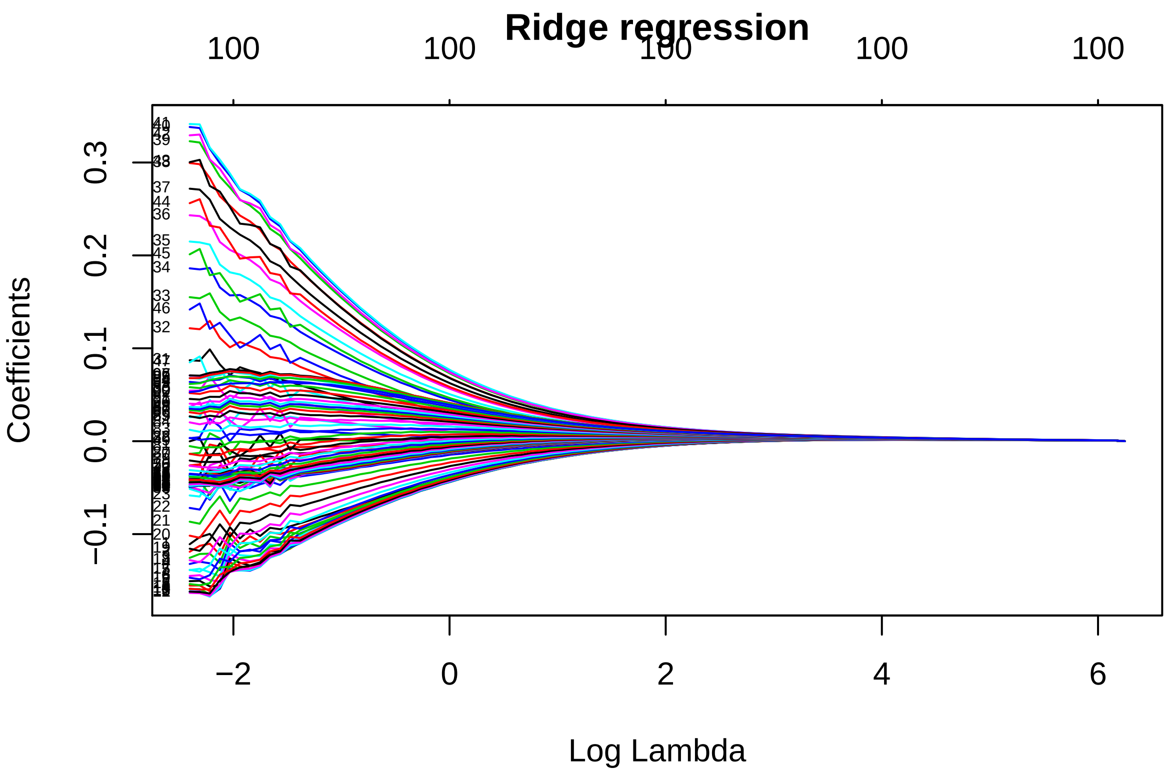


Figure 1: Bais-variance trade off

```r
ridge_reg_fit = glmnet(x = as.matrix(covariates),
                      y = response ,
                      alpha = 0 , # alpha = 0 - ridge penalty
                      family = "gaussian")
```

```r
# create plot
plot(ridge_reg_fit, xvar="lambda", label=TRUE, main="Ridge regression")
```

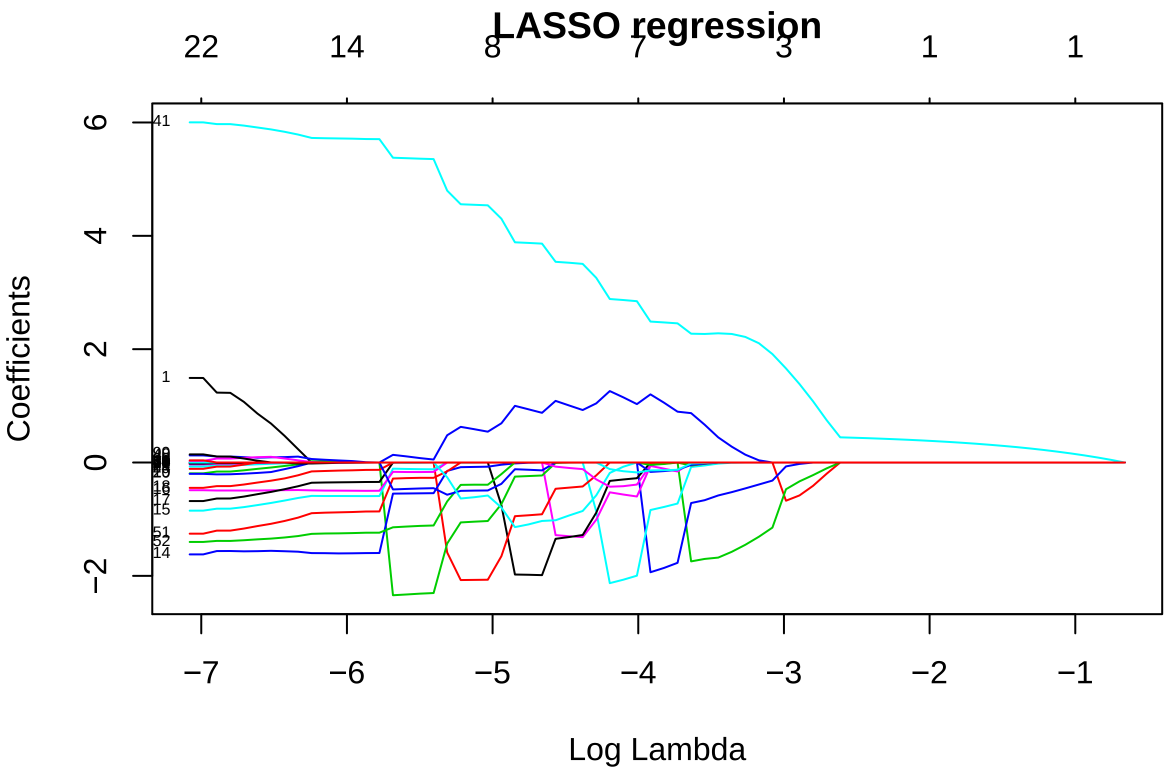


4.6 LASSO regression

```r
# fit lasso regression

# create fit lasso regression
lasso_reg_fit = glmnet(x = as.matrix(covariates),
                      y = response ,
                      alpha = 1 , # alpha = 1 - lasso penalty
                      family = "gaussian")
```

```r
# create plot
plot(lasso_reg_fit, xvar="lambda", label=TRUE, main="LASSO regression")
```

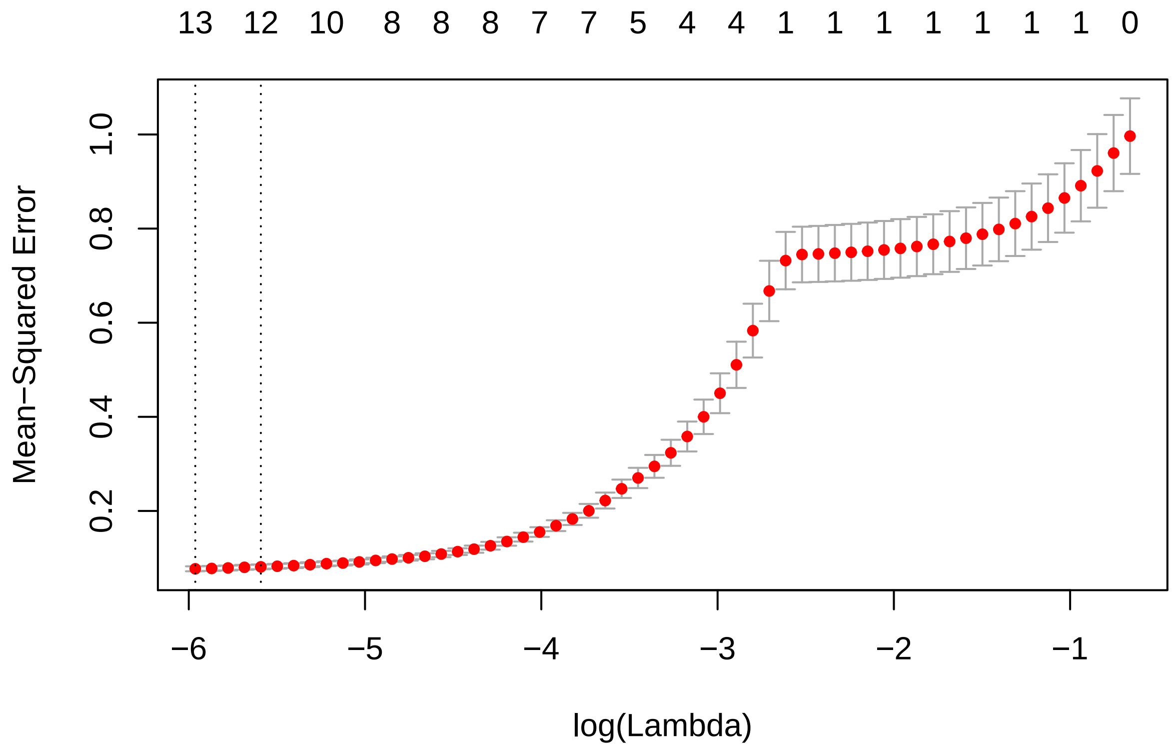


4.7 Optimal LASSO model via cross-validation

```r
# cross validation
lasso_reg_cv <- cv.glmnet(x = as.matrix(covariates),
                          y = response ,
                          alpha=1 ,
                          family="gaussian")
# optimal lambda
# lasso_reg_cv$lambda.min
# remove commend to print
```

```r
# show which coefficients where used
#coef(lasso_reg_cv, s="lambda.min")
# remove commend to print

# show how many coefficients where used
plot(lasso_reg_cv)
```


13 12 10 8 8 8 7 7 5 4 4 1 1 1 1 1 1 1 0
Mean−Squared Error
0.2
0.4
0.6
0.8
1.0
−6
−5
−4
−3
−2
−1
log(Lambda)