Group Report Lab 1

Pedram Kasebzadeh, Mariano Maquieira, Ruben Munoz

November 26, 2018

Assignment 1. Spam classification with nearest neighbors

1.1 divide data into training and test sets

1.2 Train and Test

```
## 1.2
# Logistic regression model

logistic <- glm(Spam ~ .,data=train, family="binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Prediction using TRAIN data

p1 <- predict(logistic,train, type="response")

#We will consider values that are greater than 0.5 as 1, otherwise 0.

pred1<- ifelse(p1>0.5,1,0)

#Confusion matrix. Predicting TRAIN data.

tab1<- table(Predicted=pred1, Actual = train$Spam)

tab1
```

```
##          Actual
## Predicted   0   1
##         0 803  81
##         1 142 344
```

```
cat("misclassification error is", 1-sum(diag(tab1))/sum(tab1))
```

```
## misclassification error is 0.1627737
```

```
#the rate is 0.16, so far the best one! (and the worst)



#Prediction using TEST data

p2<- predict (logistic, test, type="response")

#We will consider values that are greater than 0.5 as 1, otherwise 0.

pred2<-ifelse(p2>0.5,1,0)

tab2<-table(predicted=pred2, Actual = test$Spam)
```

```
tab2
```

```
##          Actual
## predicted   0   1
##         0 791  97
##         1 146 336
```

```
cat("misclassification error is", 1-sum(diag(tab2))/sum(tab2))
```

```
## misclassification error is 0.1773723
```

```
# the rate is 0.17, as expected for the test data performs a little worse.
```

1.3 Logistic Regression

1.3

```
#1.3

#Misclassification error - train data 0.9

pred3<- ifelse(p1>0.9,1,0)

tab3<- table(Predicted=pred3, Actual = train$Spam)

tab3
```

```
##          Actual
## Predicted   0   1
##         0 944 419
##         1   1   6
```

```
cat("misclassification error is", 1-sum(diag(tab3))/sum(tab3))
```

```
## misclassification error is 0.3065693
```

```
#the rate is 0.3

#even though we have more hits on 0, overall it is worse... (rate 0.3 vs 0.16 before...)



pred4<-ifelse(p2>0.9,1,0)

tab4<-table(predicted=pred4, Actual = test$Spam)

tab4
```

```
##          Actual
## predicted   0   1
##         0 936 427
##         1   1   6
```

```
cat("misclassification error is", 1-sum(diag(tab4))/sum(tab4)) #0.31
```

```
## misclassification error is 0.3124088
```

```
# the rate is 0.31, as expected for the test data performs a little worse.

#The new rule introduced more errors.

#But, it has higher prediction on 0 (no spam),
#this could be a plus since, on the "business"
#side is much worse predicting non-spam as spam, than spam as non-spam.
```

1.4 KKNN Classifier

```
## 1.4
k<-kknn(Spam~., train, train, k = 30)

#Misclassification error - train data 0.5

pred5<- ifelse(k$fitted.values>0.5,1,0)

tab5<- table(Predicted=pred5, Actual = train$Spam)

tab5
```

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

```
cat("misclassification error is", 1-sum(diag(tab5))/sum(tab5))
```

```
## misclassification error is 0.1722628
```

```
#the rate is 0.17, glm() had 0.16... hence,  kknn with k=30 is worse




#Misclassification error - test data 0.5

k2<-kknn(Spam~., train, test, k = 30)

pred6<- ifelse(k2$fitted.values>0.5,1,0)

tab6<- table(Predicted=pred6, Actual = test$Spam)

tab6
```

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

```
cat("misclassification error is", 1-sum(diag(tab6))/sum(tab6))
```

```
## misclassification error is 0.329927
```

```
#the rate is 0.32, much worse.... glm() had 0.17, kknn with k=30 is definitely worse.

#there is a greater difference if we test the model against train and test
#data
```

1.5 Repeat step 4

```
k3<-kknn(Spam~., train, train, k = 1)

pred7<- ifelse(k3$fitted.values>0.5,1,0)

tab7<- table(Predicted=pred7, Actual = train$Spam)

tab7
```

```
##          Actual
## Predicted   0   1
##         0 945   0
##         1   0 425
```

```
cat("misclassification error is", 1-sum(diag(tab7))/sum(tab7))
```

```
## misclassification error is 0
```

```
#the rate is 0. the model is overfitted.



k4<-kknn(Spam~., train, test, k = 1)

pred8<- ifelse(k4$fitted.values>0.5,1,0)

tab8<- table(Predicted=pred8, Actual = test$Spam)

tab8
```

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

```
cat("misclassification error is", 1-sum(diag(tab8))/sum(tab8))
```

```
## misclassification error is 0.3459854
```

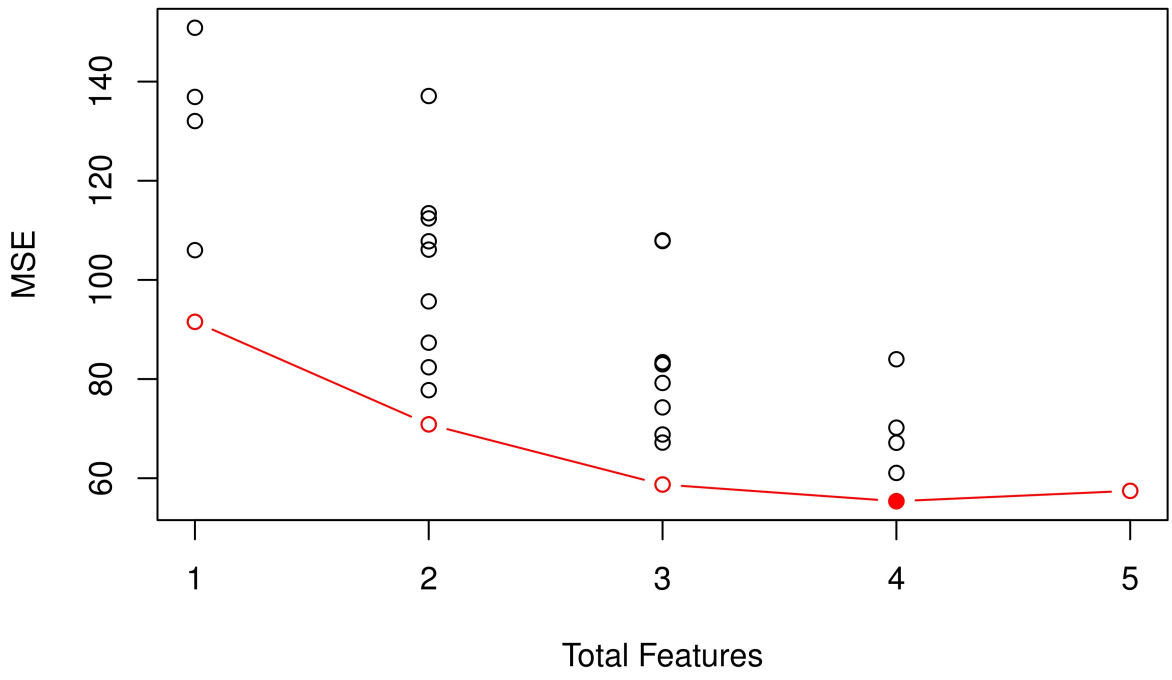
```
#the rate is 0.34

#here we are using an extremely small K... this makes the model
#really complex/flexible, hence overfitted to the training data
```

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

```
set.seed(12345)
```

```r
data_raw_A3 <- swiss

k = 5


data_mixed_A3 <- data_raw_A3[sample(nrow(data_raw_A3)), ]
folds <- cut(seq(1, nrow(data_mixed_A3)), breaks = k , labels = FALSE)
bin_combinations = expand.grid(rep(list(0:1), 5))
names_data = names(data_mixed_A3)
l <- c()
col_counter = c(rep(0, nrow(bin_combinations) - 1))

for (i in 2:(nrow(bin_combinations)))

{

x = c()

for (j in 1:(ncol(data_mixed_A3) - 1)) {

if (bin_combinations[i, j] == 1) {

x = c(x, names_data[j + 1])

col_counter[i - 1] = col_counter[i - 1] + 1
}
}

l <- c(l, paste(x, collapse = " + "))
}
cvk = c()
lm_hechizo = function(formula, data_train, data_test) {
a = model.matrix(formula, data_train)

dep_y = all.vars(formula)[1]

y = as.matrix(data_train[dep_y])

reg = solve((t(a) %*% a)) %*% t(a) %*% y
b = model.matrix(formula, data_test)
return(as.vector(b %*% reg))

}

for (j in 1:(nrow(bin_combinations) - 1)) {
mse = c()
b = l[j]
a =  "Fertility~"
c = as.formula(paste(c(a, b), collapse = ""))
# 5
for (i in 1:k)
{
trainIndexes <- which(folds != i, arr.ind = TRUE)
```

```r
trainData <- data_mixed_A3[trainIndexes, ]
testData = data_mixed_A3[-trainIndexes, ]
lm.fit <- lm_hechizo(c, trainData, testData)

mse = c(mse, mean((lm.fit - testData$Fertility) ^ 2))
}
cvk = c(cvk, mean(mse))
}

cvk_min = c()

for (i in 1:k) {

x = cvk[which(col_counter == i)]

y = which.min(x)

cvk_min = c(cvk_min, x[y])

}


plot(x = col_counter,
y = cvk,
xlab = "Total Features",
ylab = "MSE")
lines(cvk_min, type = "b", col = "red")
points(
x = which.min(cvk_min),

y = cvk_min[which.min(cvk_min)],

pch = 19,

col = "red"

)
```




```
min(cvk)
```

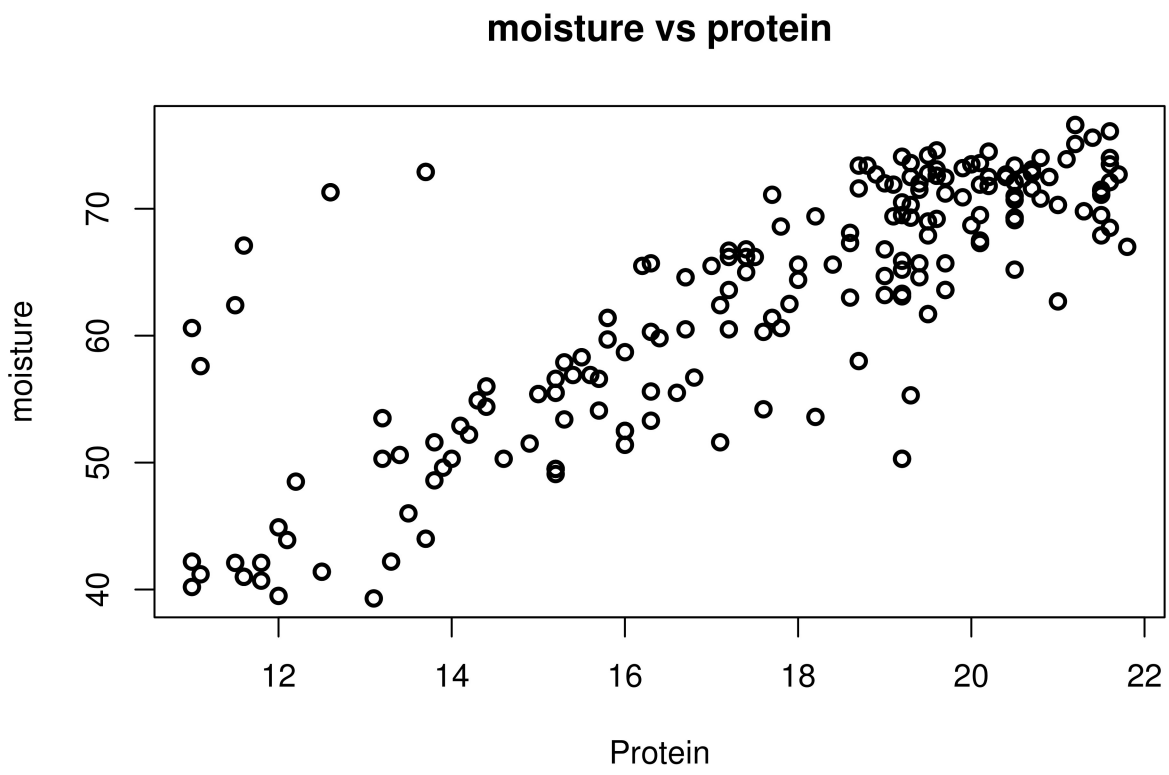
```
## [1] 55.36484
```

```
l[which(cvk %in% min(cvk))]
```

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

Assignment 4. Linear regression and regularization

4.1 Import data



There is a pronounced relationship between moisture and protein, but it seems slightly curved, almost linear.

4.2

We check the MSE of each M1, M2, M3... and the smallest error should be the optimal one (neither under nor over fitted)

4.3

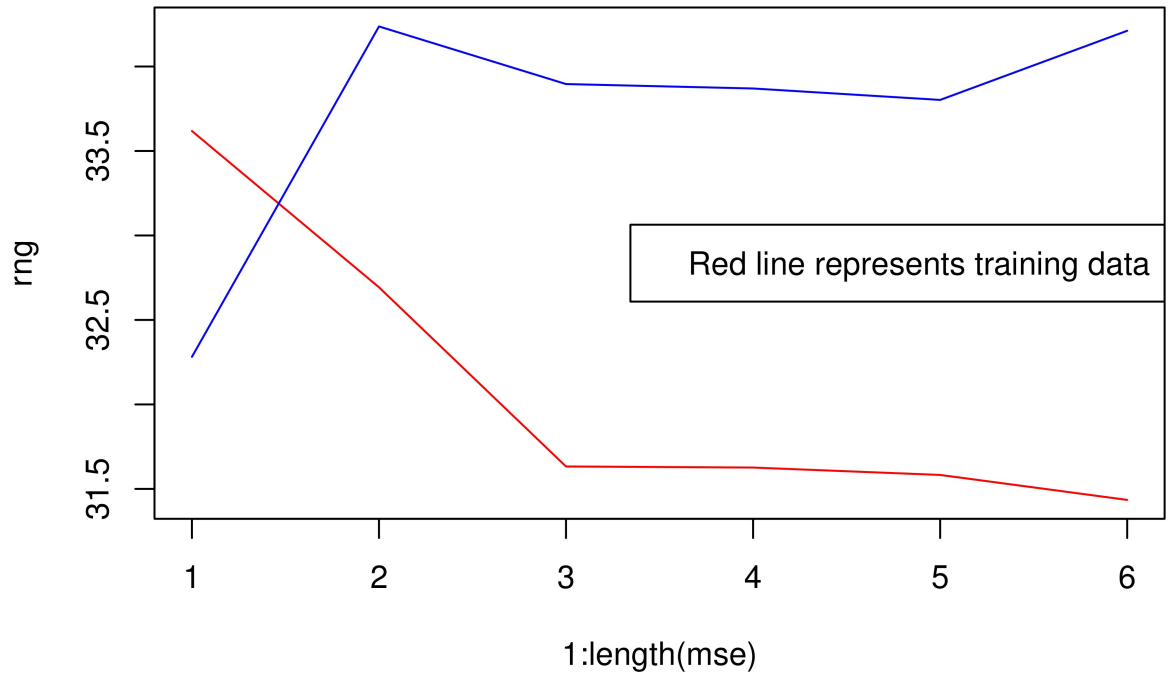
```
data <- read_excel("tecator.xlsx")

names(data)
```

```
##   [1] "Sample"     "Channel1"   "Channel2"   "Channel3"   "Channel4"
##   [6] "Channel5"   "Channel6"   "Channel7"   "Channel8"   "Channel9"
##  [11] "Channel10"  "Channel11"  "Channel12"  "Channel13"  "Channel14"
##  [16] "Channel15"  "Channel16"  "Channel17"  "Channel18"  "Channel19"
##  [21] "Channel20"  "Channel21"  "Channel22"  "Channel23"  "Channel24"
##  [26] "Channel25"  "Channel26"  "Channel27"  "Channel28"  "Channel29"
##  [31] "Channel30"  "Channel31"  "Channel32"  "Channel33"  "Channel34"
##  [36] "Channel35"  "Channel36"  "Channel37"  "Channel38"  "Channel39"
##  [41] "Channel40"  "Channel41"  "Channel42"  "Channel43"  "Channel44"
##  [46] "Channel45"  "Channel46"  "Channel47"  "Channel48"  "Channel49"
```

```
##  [51] "Channel50"  "Channel51"  "Channel52"  "Channel53"  "Channel54"
##  [56] "Channel55"  "Channel56"  "Channel57"  "Channel58"  "Channel59"
##  [61] "Channel60"  "Channel61"  "Channel62"  "Channel63"  "Channel64"
##  [66] "Channel65"  "Channel66"  "Channel67"  "Channel68"  "Channel69"
##  [71] "Channel70"  "Channel71"  "Channel72"  "Channel73"  "Channel74"
##  [76] "Channel75"  "Channel76"  "Channel77"  "Channel78"  "Channel79"
##  [81] "Channel80"  "Channel81"  "Channel82"  "Channel83"  "Channel84"
##  [86] "Channel85"  "Channel86"  "Channel87"  "Channel88"  "Channel89"
##  [91] "Channel90"  "Channel91"  "Channel92"  "Channel93"  "Channel94"
##  [96] "Channel95"  "Channel96"  "Channel97"  "Channel98"  "Channel99"
## [101] "Channel100" "Fat"        "Protein"    "Moisture"
```

```r
data<-data[,c("Protein","Moisture")]


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

mse_test<-c()

mse_train<-c()

for (i in 1:6){
  model <- lm(formula= Moisture~poly(Protein,i, raw = T), data = train)
  mse_test[i]<-mean((test$Moisture-predict(model,test))^2)

}



for (i in 1:6){
  model <- lm(formula= Moisture~poly(Protein,i, raw = T), data = train)
  mse_train[i]<-mean((train$Moisture-predict(model,train))^2)

}


mse=mse_test

mse2=mse_train

rng = mse

rng[1] = min(c(min(mse),min(mse2)))

rng[6] = max(c(max(mse),max(mse2)))

plot(
  x=1:length(mse),
  y=rng,
  col = "transparent"
```

```
)
lines(
  x=1:length(mse),
  y=mse2,
  col = "red",
  type = "l"
)

lines(x=1:length(mse2),
       y=mse,
        col = "blue",type="l")
legend('right',c("Red line represents training data"))
```



4.4

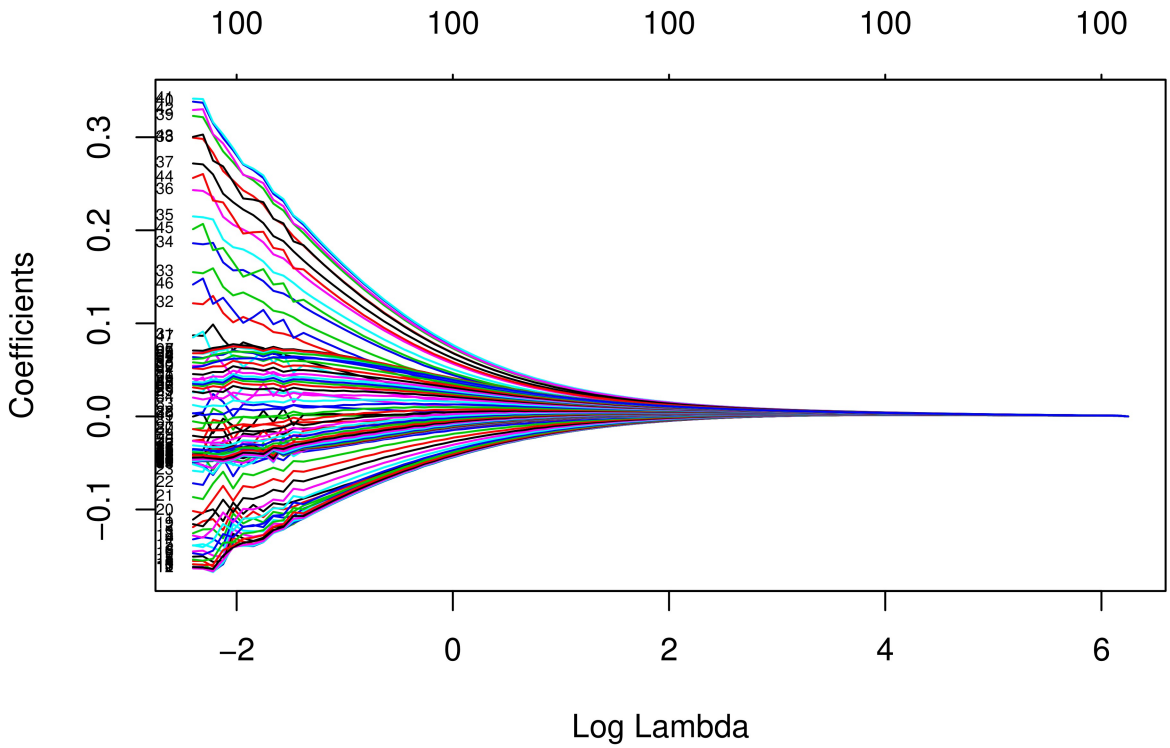
the best subset of predictors consists of: 63 predictors.

we used 'both' direction which specifies the stepwise search, it does both "backward", or "forward".

4.5

```
covariates=scale(data[,1:100])

response=scale(data[, 101])

model0=glmnet(as.matrix(covariates),response, alpha=0,family="gaussian")
```

```
plot(model0, xvar="lambda", label=TRUE)
```



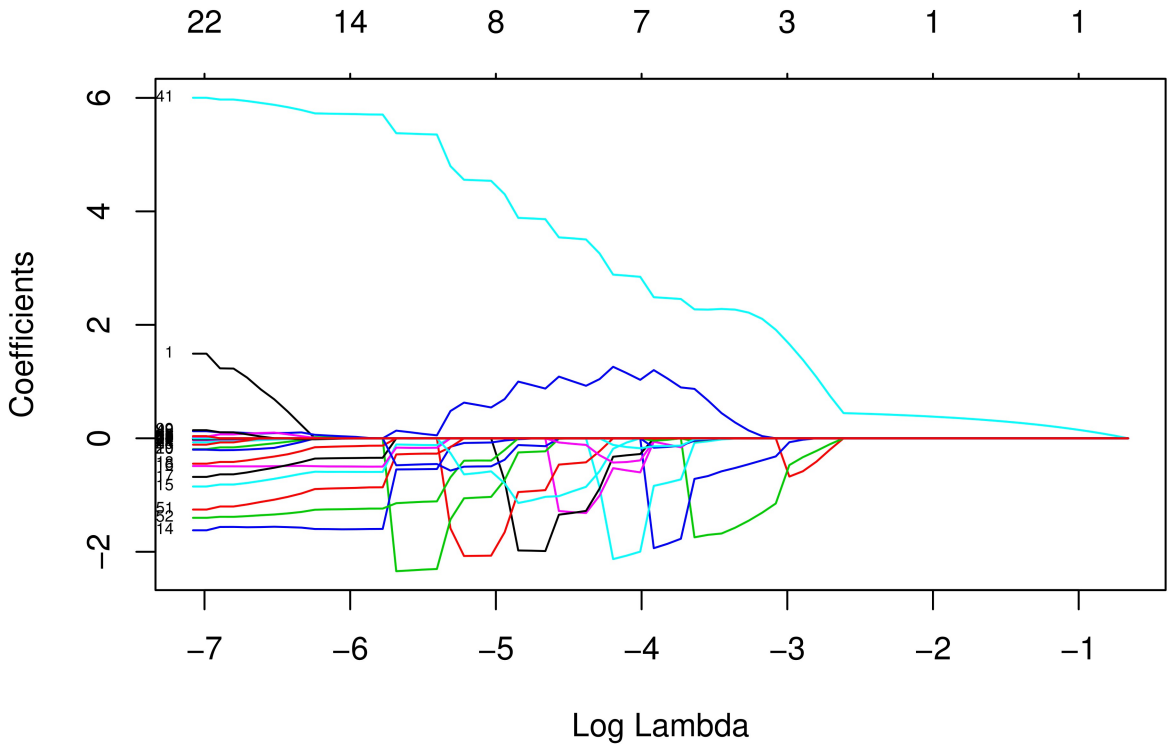

```
#The advantage of Ridge regression over least squares comes from
#the bias-variance trade-off... We are simplyfing the model a little bit to reduce overfitting.

#As LAMBDA increases, we simplify more and more our model...
#leading to increased bias, and lower variance.
```

The advantage of Ridge regression over least squares comes from the bias-variance trade-off... We are simplyfing the model a little bit to reduce overfitting.

As LAMBDA increases, we simplify more and more our model... leading to increased bias, and lower variance.

4.6

```
model0=glmnet(as.matrix(covariates),response, alpha=1,family="gaussian")

plot(model0, xvar="lambda", label=TRUE)
```




```
#the main difference is that ridge can only shrink it very close to 0, but no 0. Lasso can shrink it al

#Therefore lasso can completely exclude  USELESS VARIABLES, whereas
#ridge not. That is why lasso is a bit better at reducing variance than ridge.

#in contrast ridge regression tends to do a little better when most  variables are useful.
```

the main difference is that ridge can only shrink it very close to 0, but no 0. Lasso can shrink it all the way to 0 Therefore lasso can completely exclude USELESS VARIABLES, whereas ridge not. That is why lasso is a bit better at reducing variance than ridge. in contrast ridge regression tends to do a little better when most variables are useful.

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

LASSO is better than 'ridge' regression, in the sense that it produces simpler and more interpretable models that involve only a subset of the predictors.

In both LASSO and 'ridge' as ?? increases, the variance decreases and the bias increases.

None of them will universally dominate the other.

LASSO usually performs better in a setting where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or that equal zero.

Ridge usually performs better when the response is a function of many predictors, all with coefficients of roughly equal size.

Truth is, we never know beforehand the number of predictors that are related to the response.

We shall use cross-validation to determine which approach is better on a particular data set.

The most notably difference is that unlike 'ridge', LASSO performs variable selection, and hence results in models that are easier to interpret.

4.7

```
model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")

coef(model, s="lambda.min")
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  2.632947e-15
## Channel1     .
## Channel2     .
## Channel3     .
## Channel4     .
## Channel5     .
## Channel6     .
## Channel7     .
## Channel8     .
## Channel9     .
## Channel10    .
## Channel11    .
## Channel12    .
## Channel13    .
## Channel14   -1.602254e+00
## Channel15   -5.920294e-01
## Channel16   -4.978047e-01
## Channel17   -3.471019e-01
## Channel18   -1.377282e-01
## Channel19    .
## Channel20    .
## Channel21    .
## Channel22    .
## Channel23    .
## Channel24    .
## Channel25    .
## Channel26    .
## Channel27    .
## Channel28    .
## Channel29    .
## Channel30    .
## Channel31    .
## Channel32    .
## Channel33    .
## Channel34    .
## Channel35    .
## Channel36    .
## Channel37    .
## Channel38    .
## Channel39    5.881677e-03
## Channel40    2.533079e-02
```

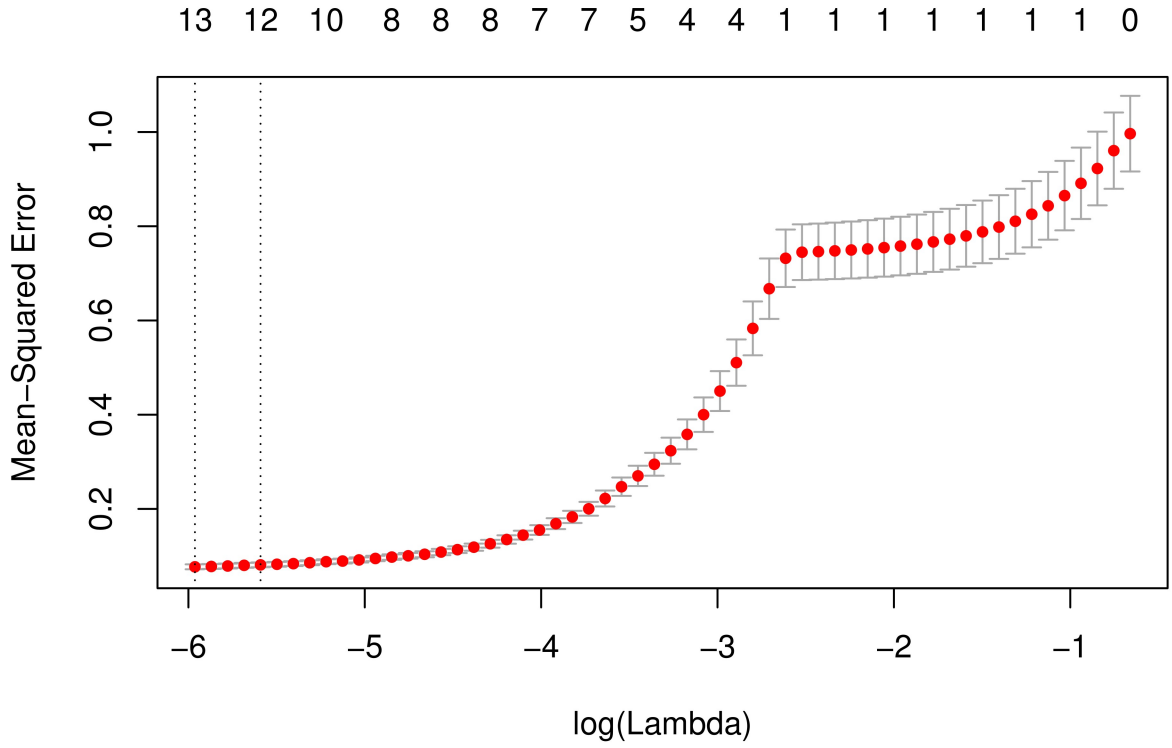
```
## Channel41    5.714424e+00
## Channel42    .
## Channel43    .
## Channel44    .
## Channel45    .
## Channel46    .
## Channel47    .
## Channel48    .
## Channel49   -2.466071e-05
## Channel50   -4.082507e-03
## Channel51   -8.735307e-01
## Channel52   -1.245545e+00
## Channel53   -1.069154e-03
## 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    .
```

```
model$lambda.min
```

```
## [1] 0.002571916
```

```
plot(model)
```



4.8

Appendix

```
library("readxl")

library(kknn)

library(ggplot2)

library(MASS)

library(glmnet)
```

```r
## 1.1
# xls files

data <- read_excel("spambase.xlsx")

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

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 model

logistic <- glm(Spam ~ .,data=train, family="binomial")

# Prediction using TRAIN data

p1 <- predict(logistic,train, type="response")

#We will consider values that are greater than 0.5 as 1, otherwise 0.

pred1<- ifelse(p1>0.5,1,0)

#Confusion matrix. Predicting TRAIN data.

tab1<- table(Predicted=pred1, Actual = train$Spam)

tab1

cat("misclassification error is", 1-sum(diag(tab1))/sum(tab1))

#the rate is 0.16, so far the best one! (and the worst)



#Prediction using TEST data

p2<- predict (logistic, test, type="response")

#We will consider values that are greater than 0.5 as 1, otherwise 0.

pred2<-ifelse(p2>0.5,1,0)

tab2<-table(predicted=pred2, Actual = test$Spam)

tab2
```

```r
cat("misclassification error is", 1-sum(diag(tab2))/sum(tab2))

# the rate is 0.17, as expected for the test data performs a little worse.


#1.3

#Misclassification error - train data 0.9

pred3<- ifelse(p1>0.9,1,0)

tab3<- table(Predicted=pred3, Actual = train$Spam)

tab3

cat("misclassification error is", 1-sum(diag(tab3))/sum(tab3))

#the rate is 0.3

#even though we have more hits on 0, overall it is worse... (rate 0.3 vs 0.16 before...)



pred4<-ifelse(p2>0.9,1,0)

tab4<-table(predicted=pred4, Actual = test$Spam)

tab4

cat("misclassification error is", 1-sum(diag(tab4))/sum(tab4)) #0.31

# the rate is 0.31, as expected for the test data performs a little worse.

#The new rule introduced more errors.

#But, it has higher prediction on 0 (no spam),
#this could be a plus since, on the "business"
#side is much worse predicting non-spam as spam, than spam as non-spam.



## 1.4
k<-kknn(Spam~., train, train, k = 30)

#Misclassification error - train data 0.5

pred5<- ifelse(k$fitted.values>0.5,1,0)

tab5<- table(Predicted=pred5, Actual = train$Spam)

tab5

cat("misclassification error is", 1-sum(diag(tab5))/sum(tab5))
```

```r
#the rate is 0.17, glm() had 0.16... hence,  kknn with k=30 is worse




#Misclassification error - test data 0.5

k2<-kknn(Spam~., train, test, k = 30)

pred6<- ifelse(k2$fitted.values>0.5,1,0)

tab6<- table(Predicted=pred6, Actual = test$Spam)

tab6

cat("misclassification error is", 1-sum(diag(tab6))/sum(tab6))

#the rate is 0.32, much worse.... glm() had 0.17, kknn with k=30 is definitely worse.

#there is a greater difference if we test the model against train and test
#data


k3<-kknn(Spam~., train, train, k = 1)

pred7<- ifelse(k3$fitted.values>0.5,1,0)

tab7<- table(Predicted=pred7, Actual = train$Spam)

tab7

cat("misclassification error is", 1-sum(diag(tab7))/sum(tab7))

#the rate is 0. the model is overfitted.



k4<-kknn(Spam~., train, test, k = 1)

pred8<- ifelse(k4$fitted.values>0.5,1,0)

tab8<- table(Predicted=pred8, Actual = test$Spam)

tab8

cat("misclassification error is", 1-sum(diag(tab8))/sum(tab8))

#the rate is 0.34

#here we are using an extremely small K... this makes the model
#really complex/flexible, hence overfitted to the training data
```

```r
set.seed(12345)

data_raw_A3 <- swiss

k = 5


data_mixed_A3 <- data_raw_A3[sample(nrow(data_raw_A3)), ]
folds <- cut(seq(1, nrow(data_mixed_A3)), breaks = k , labels = FALSE)
bin_combinations = expand.grid(rep(list(0:1), 5))
names_data = names(data_mixed_A3)
l <- c()
col_counter = c(rep(0, nrow(bin_combinations) - 1))

for (i in 2:(nrow(bin_combinations)))

{

x = c()

for (j in 1:(ncol(data_mixed_A3) - 1)) {

if (bin_combinations[i, j] == 1) {

x = c(x, names_data[j + 1])

col_counter[i - 1] = col_counter[i - 1] + 1
}
}

l <- c(l, paste(x, collapse = " + "))
}
cvk = c()
lm_hechizo = function(formula, data_train, data_test) {
a = model.matrix(formula, data_train)

dep_y = all.vars(formula)[1]

y = as.matrix(data_train[dep_y])

reg = solve((t(a) %*% a)) %*% t(a) %*% y
b = model.matrix(formula, data_test)
return(as.vector(b %*% reg))

}

for (j in 1:(nrow(bin_combinations) - 1)) {
mse = c()
b = l[j]
a =  "Fertility~"
c = as.formula(paste(c(a, b), collapse = ""))
# 5
for (i in 1:k)
```

```
{
trainIndexes <- which(folds != i, arr.ind = TRUE)
trainData <- data_mixed_A3[trainIndexes, ]
testData = data_mixed_A3[-trainIndexes, ]
lm.fit <- lm_hechizo(c, trainData, testData)

mse = c(mse, mean((lm.fit - testData$Fertility) ^ 2))
}
cvk = c(cvk, mean(mse))
}

cvk_min = c()

for (i in 1:k) {

x = cvk[which(col_counter == i)]

y = which.min(x)

cvk_min = c(cvk_min, x[y])

}


plot(x = col_counter,
y = cvk,
xlab = "Total Features",
ylab = "MSE")
lines(cvk_min, type = "b", col = "red")
points(
x = which.min(cvk_min),

y = cvk_min[which.min(cvk_min)],

pch = 19,

col = "red"

)
min(cvk)
l[which(cvk %in% min(cvk))]

## 4.1
library(MASS)

data <- read_excel("tecator.xlsx")

plot (data$Protein, data$Moisture,type='p',xlab="Protein", lwd=2, main = "moisture vs protein", ylab="m


data <- read_excel("tecator.xlsx")

names(data)
```

```r
data<-data[,c("Protein","Moisture")]


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

mse_test<-c()

mse_train<-c()

for (i in 1:6){
  model <- lm(formula= Moisture~poly(Protein,i, raw = T), data = train)
  mse_test[i]<-mean((test$Moisture-predict(model,test))^2)

}



for (i in 1:6){
  model <- lm(formula= Moisture~poly(Protein,i, raw = T), data = train)
  mse_train[i]<-mean((train$Moisture-predict(model,train))^2)

}


mse=mse_test

mse2=mse_train

rng = mse

rng[1] = min(c(min(mse),min(mse2)))

rng[6] = max(c(max(mse),max(mse2)))

plot(
  x=1:length(mse),
  y=rng,
  col = "transparent"
)
lines(
  x=1:length(mse),
  y=mse2,
  col = "red",
  type = "l"
)

lines(x=1:length(mse2),
       y=mse,
        col = "blue",type="l")
```

```r
legend('right',c("Red line represents training data"))


data <- read_excel("tecator.xlsx")

data<-data[,2:102]

names(data)

fit <- lm(Fat~.,data=data)

step <- stepAIC(fit, direction="both")

summary(step)

#the best subset of predictors consists of: 63 predictors.

#we used 'both' direction which specifies the stepwise search, it does both "backward", or "forward".


covariates=scale(data[,1:100])

response=scale(data[, 101])

model0=glmnet(as.matrix(covariates),response, alpha=0,family="gaussian")

plot(model0, xvar="lambda", label=TRUE)

#The advantage of Ridge regression over least squares comes from
#the bias-variance trade-off... We are simplyfing the model a little bit to reduce overfitting.

#As LAMBDA increases, we simplify more and more our model...
#leading to increased bias, and lower variance.


model0=glmnet(as.matrix(covariates),response, alpha=1,family="gaussian")

plot(model0, xvar="lambda", label=TRUE)

#the main difference is that ridge can only shrink it very close to 0, but no 0. Lasso can shrink it al

#Therefore lasso can completely exclude  USELESS VARIABLES, whereas
#ridge not. That is why lasso is a bit better at reducing variance than ridge.

#in contrast ridge regression tends to do a little better when most  variables are useful.


model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")

coef(model, s="lambda.min")

model$lambda.min
```

```
plot(model)
```