Data Science II, Homework 1

Will Simmons

1 March 2020

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_knit$set(dev.args = list(type = "cairo"))

library(tidyverse)
library(caret)
library(readr)
library(modelr)
#library(ModelMetrics)

train =
  read_csv('./data/solubility_train.csv') %>%
  as_tibble() %>%
  mutate_at(vars(starts_with("FP")),         ## converting FP variables to factor
            funs(factor))

test =
  read_csv('./data/solubility_test.csv') %>%
  as_tibble() %>%
  mutate_at(vars(starts_with("FP")),           ## converting FP variables to factor
          funs(factor))
```

Description

In this exercise, we will predict solubility of compounds using their chemical structures. The training data are in the "solubility_train.csv" and the test data are in "solubility_test.csv". Among the 228 predictors, 208 are binary variables that indicate the presence or absence of a particular chemical substructure, 16 are count features, such as the number of bonds or the number of bromine atoms, and 4 are continuous features, such as molecular weight or surface area. The response is in the column "Solubility" (the last column).

Checking the data

First, I'll check for missing data, zero/near-zero variance predictors, and linear dependencies.

```
## missing data?
sum(!complete.cases(train))
  ## 0 missing variable observations from training set

## zero var pred?
nearZeroVar(train)
  ## index 154, 199, 200 near zero var
  ## train[,c(154, 199, 200)]

## linear combinations?
findLinearCombos(train)
  ## none
```

There were no missing values and no linear dependencies, but there were three features that had near-zero variance: `FP154`, `FP199`, and `FP200`. We'll remove these columns from both the training and testing sets.

```
## removing near-zero variance columns from both training and testing sets

train =
    train[,-c(154, 199, 200)]

test =
    test[,-c(154, 199, 200)]
```

(a) Fit a linear model using least squares on the training data and calculate the mean square error using the test data.

Fitting model:

```
set.seed(1)

#ctrl_lm = trainControl(method = "none")

## trainControl function
ctrl_cv = trainControl(method = "repeatedcv", number = 10, repeats = 10)

lm_model =
  train(Solubility~.,
        data = train,
        method = "lm",
        trControl = ctrl_cv    ## using CV so can put in resample table for part D
  )
```

Calculating MSE on test data:

```
set.seed(1)

## using modelr
lm_mse =
  mse(lm_model, test)
## 0.5634133
```

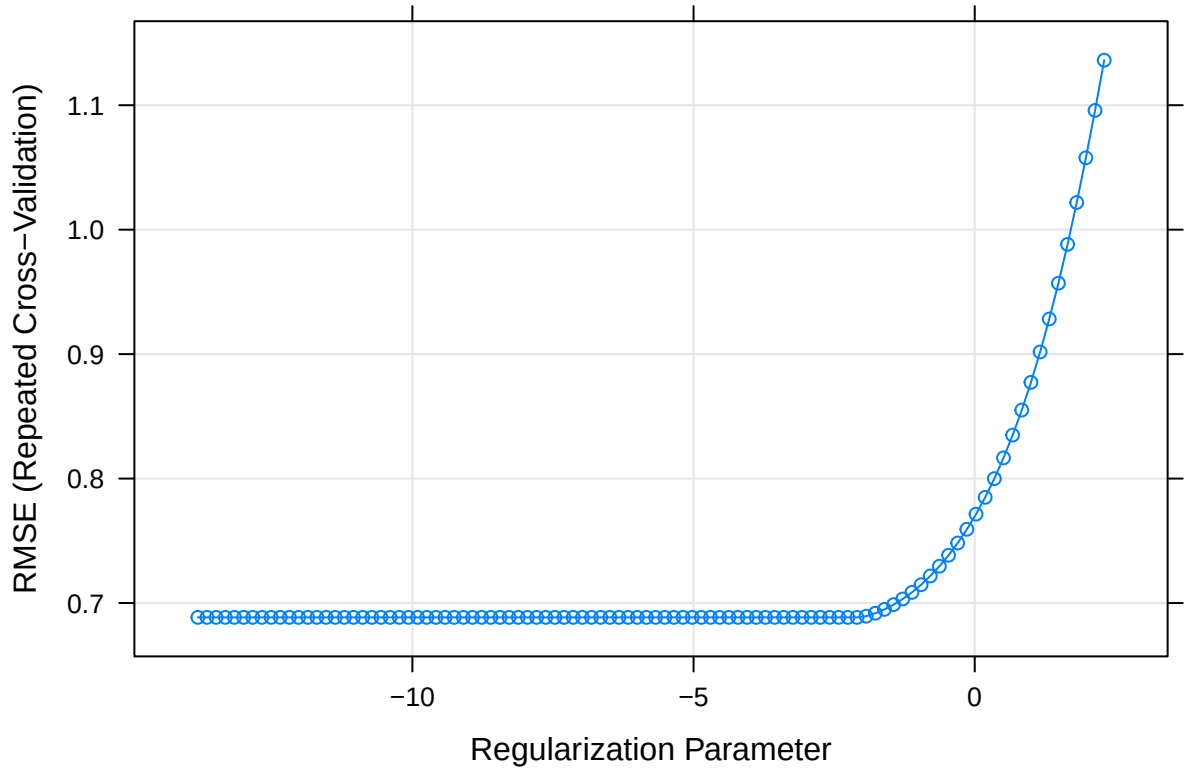
The mean square error for the linear model is **0.5634133**.

(b) Fit a ridge regression model on the training data, with lambda chosen by cross-validation. Report the test error.

Fitting model:

```
set.seed(1)

## test 1
```

```r
#lambda = 10^seq(-3, 0.5, length = 10)
## need smaller values

## test 2
lambda = 10^seq(-6, 1, length = 100)

ridge_model =
  train(Solubility~.,
        data = train,
        method = "glmnet",
        tuneGrid = expand.grid(alpha = 0,
                               lambda = lambda),
        trControl = ctrl_cv
        )

## check cross-validated lambda range in plot
plot(ridge_model, xTrans = function(x) log(x)) ## x value = log(Lambda)
```




```r
ridge_model$bestTune
```

```
##    alpha    lambda
## 73     0 0.1232847
```

```
## 0.1232847
```

Calculating MSE on test data:

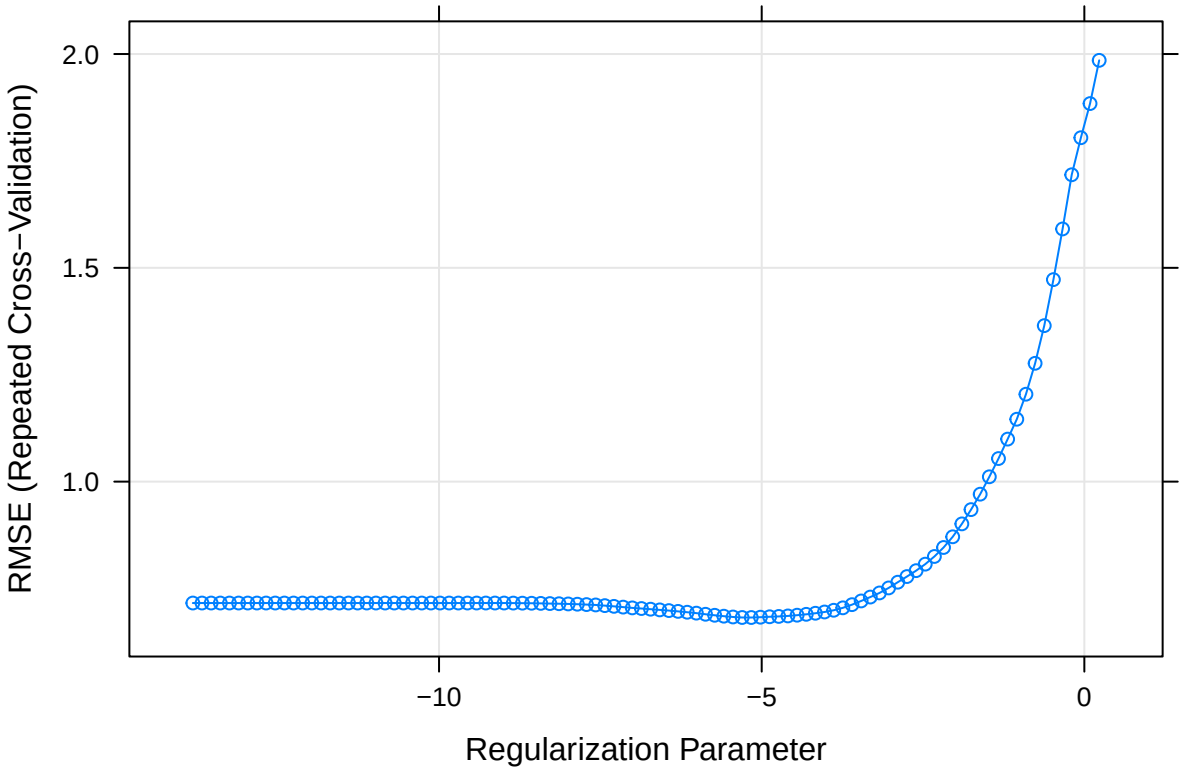
```
set.seed(1)

ridge_mse =
  mse(ridge_model, test)
## 0.5290609
```

The mean square error for the ridge model is **0.5290609**.

(c) Fit a lasso model on the training data, with lambda chosen by cross-validation. Report the test error, along with the number of non-zero coefficient estimates.

Fitting model:

```
set.seed(1)

## trainControl function - same as above
#ctrl_cv = trainControl(method = "repeatedcv", number = 10, repeats = 10)

## produced reasonable curve with plot below
lambda = 10^seq(-6, 0.1, length = 100)

lasso_model =
  train(Solubility~.,
        data = train,
        method = "glmnet",
        tuneGrid = expand.grid(alpha = 1,
                               lambda = lambda),
        trControl = ctrl_cv
        )

## check cross-validated lambda range in plot
plot(lasso_model, xTrans = function(x) log(x)) ## x value = log(Lambda)
```




```
lasso_model$bestTune
```

```
##    alpha      lambda
## 62     1 0.005735692
```

```
## 0.005735692
```

Calculating MSE on test data and investigating number of nonzero coefficient estimates:

```
set.seed(1)

## mse
lasso_mse =
  mse(lasso_model, test)
## 0.5094902

## coefficient estimates
non_zero =
  coef(lasso_model$finalModel, lasso_model$bestTune$lambda) %>%
  as.list() %>% unlist() %>%
  as_tibble() %>%
  filter(value != 0) %>%
  summarize(count = n())
```

```
## Warning: Calling `as_tibble()` on a vector is discouraged, because the behavior is likely to change i
## This warning is displayed once per session.
```

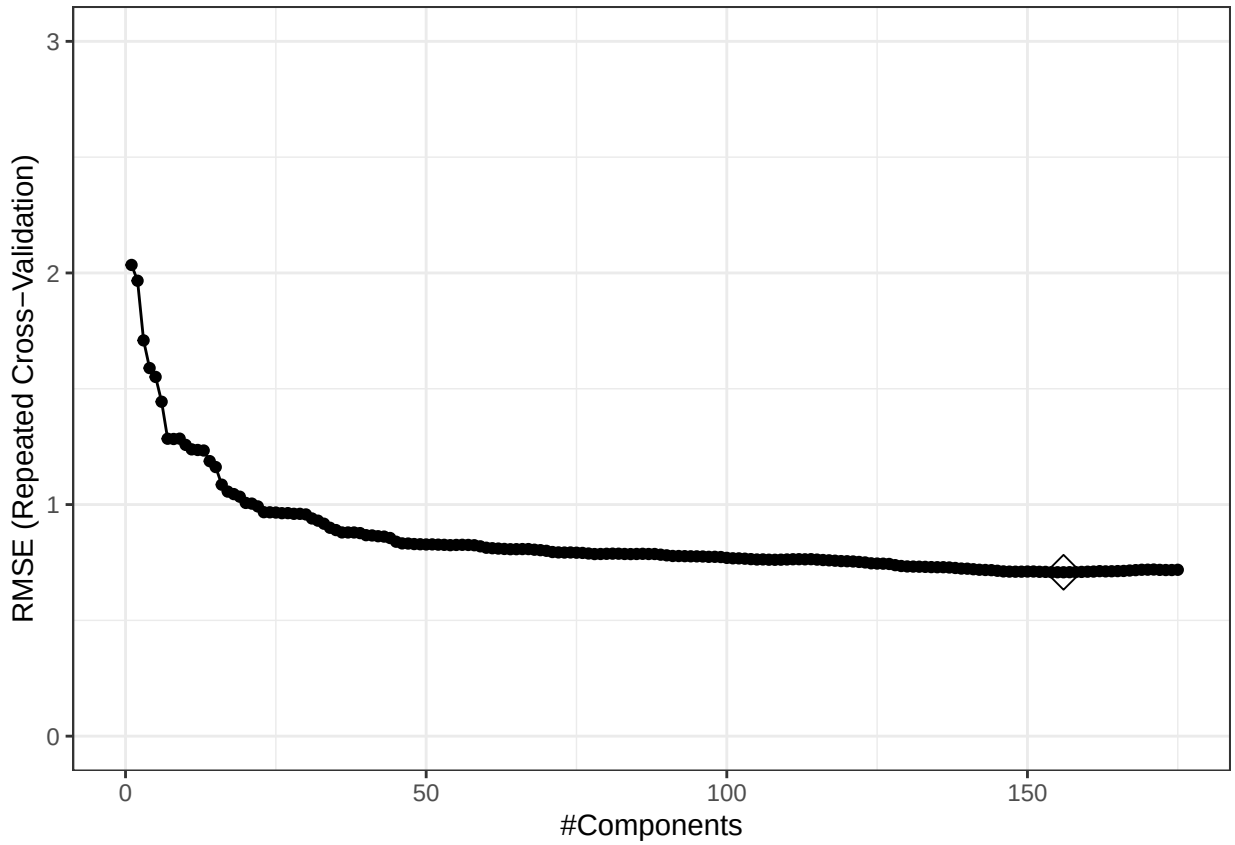
```
## 137
```

The mean square error for the LASSO model is **0.5094902**. This model has **137** non-zero coefficient estimates.

(d) Fit a principal component regression model on the training data, with M chosen by cross-validation. Report the test error, along with the value of M selected by cross-validation.

Fitting model:

```
set.seed(1)

pcr_model =
  train(Solubility~.,
        data = train,
        method = "pcr",
        trControl = ctrl_cv,
        tuneGrid = data.frame(ncomp = 1:(ncol(train)-1)),
        preProc = c("center", "scale"))

ggplot(pcr_model, highlight = TRUE) +
  xlim(0, 175) +
  ylim(0, 3) +
  theme_bw()
```



Calculating MSE on test data:

```
set.seed(1)

pcr_M =
  pcr_model$bestTune
## M = 156

pcr_mse =
  mse(pcr_model, test)
##0.5614959
```

The mean square error for the PCR is **0.5614959**, with $\mathbf{M = 156}$ chosen by cross-validation.

(e) Briefly discuss the results obtained in (a)~(d).

As we can see from our MSE values above, the Lasso model had the smallest test error.

```
#library(kableExtra)

tibble(
  Test = c(
    "Linear",
    "Ridge",
```

```
    "Lasso",
    "PCR"),
  MSE = c(
    lm_mse,
    ridge_mse,
    lasso_mse,
    pcr_mse)
  ) %>%
  arrange(MSE) %>%
  knitr::kable()
```

Test	MSE
Lasso	0.5094902
Ridge	0.5290609
PCR	0.5614959
Linear	0.5634133

Lasso reduced many feature coefficient estimates to 0 (total nonzero Lasso coefficient estimates = **137**), and PCR reduced the feature space to a cross-validated M = **156**. All non-OLS models outperformed the standard OLS model.

(f) Which model will you choose for predicting solubility?

I will choose to use the Lasso model to predict solubility, given its test error value is the smallest of the four models tested.