Exercise #6

Model Evaluation with R

Brian Schweigler; 16-102-071

27/04/2022

Preliminaries

Load the required libraries

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.5
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 4.0.5
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
library(FNN)
```

```
## Warning: package 'FNN' was built under R version 4.0.5
```

To normalize data, we define the following function:

```
normalize = function(x) {
    (x - min(x)) / (max(x) - min(x))
}
```

We define the best model as the one with the lowest MSE.

```
best_k_for_knn_reg = function(train,
                                                            train_labels,
                                                            test,
                                                            test_labels,
                                                            kStart,
                                                            kEnd) {
    best_mse = NA
    for (k in kStart:kEnd) {
```

```
        model = knn.reg(
            train = train,
            test = test,
            y = train_labels,
            k = k
        )

        mse = mean((model$pred - test_labels) ^ 2)

        if (is.na(best_mse) || mse < best_mse) {
            best_mse = mse
            best_k = k
        }
    }

    return(best_k)
}
```

Loading the computers dataset:

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

computers_df = read.csv("Computers.txt", header = TRUE, sep = "\t"  , comment.char = "#")
```

As we know from exercise 4, Computers.txt has no outliers discernible (without additional information).

Loading the cars dataset and cleaning NAs:

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

cars_df = read.csv("Cars.txt", header = TRUE, sep = "\t", comment.char = "#")
```

There are 6 NAs for horsepower that need to be removed -> delete the whole corresponding rows. These are also mentioned in Cars.pdf.

```
cars_df_cleaned <- cars_df[!is.na(cars_df$horsepower),]
```

Besides the NAs found for horspower, the data seems to be good (without additional information.)

1. Compare the two models (linear regression VS k-NN) for the Computers dataset (filename: Computers.txt) as it follows:

1a. Evaluate the quality of the fit (of the best model) between a single regression model of your choice and a multiple regression.

First exclude vendor, model and ERP, then normalize:

```
computers_df = computers_df[, 3:9]
computers_df_norm = as.data.frame(lapply(computers_df, normalize))
summary(computers_df_norm)
```

```
##       MYCT              MMIN              MMAX              CACH        
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000  
##  1st Qu.:0.02225   1st Qu.:0.02204   1st Qu.:0.06156   1st Qu.:0.00000  
##  Median :0.06271   Median :0.06062   Median :0.12412   Median :0.03125  
##  Mean   :0.12598   Mean   :0.08780   Mean   :0.18350   Mean   :0.09846  
##  3rd Qu.:0.14026   3rd Qu.:0.12325   3rd Qu.:0.24925   3rd Qu.:0.12500  
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000  
##      CGMIN             CHMAX              PRP         
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000  
##  1st Qu.:0.01923   1st Qu.:0.02841   1st Qu.:0.01836  
##  Median :0.03846   Median :0.04545   Median :0.03846  
##  Mean   :0.09036   Mean   :0.10380   Mean   :0.08708  
##  3rd Qu.:0.11538   3rd Qu.:0.13636   3rd Qu.:0.09353  
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.00000  
```

Now we create the two regression models:

```
lm_computers_single = lm(PRP ~ MMAX, data = computers_df_norm)
summary(lm_computers_single)
```

```
## 
## Call:
## lm(formula = PRP ~ MMAX, data = computers_df_norm)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.20171 -0.03153  0.00289  0.02564  0.37280 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -0.034302   0.006975  -4.918 1.78e-06 ***
## MMAX         0.661501   0.026915  24.577  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.0712 on 207 degrees of freedom
## Multiple R-squared:  0.7448, Adjusted R-squared:  0.7435 
## F-statistic: 604.1 on 1 and 207 DF,  p-value: < 2.2e-16
```

```
lm_computers_multiple = lm(PRP ~ MYCT + MMIN + MMAX + CACH + CGMIN + CHMAX, data = computers_df_norm)
summary(lm_computers_multiple)
```

```
## 
## Call:
## lm(formula = PRP ~ MYCT + MMIN + MMAX + CACH + CGMIN + CHMAX, 
```

```
##     data = computers_df_norm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.17118 -0.02200  0.00472  0.02318  0.33719
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.052210   0.006846  -7.626 9.32e-13 ***
## MYCT         0.063332   0.022711   2.789   0.0058 **
## MMIN         0.426909   0.050997   8.371 9.42e-15 ***
## MMAX         0.311374   0.035869   8.681 1.32e-15 ***
## CACH         0.143530   0.031231   4.596 7.59e-06 ***
## CGMIN       -0.012289   0.038894  -0.316   0.7524
## CHMAX        0.228073   0.033852   6.737 1.65e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05244 on 202 degrees of freedom
## Multiple R-squared:  0.8649, Adjusted R-squared:  0.8609
## F-statistic: 215.5 on 6 and 202 DF,  p-value: < 2.2e-16
```

Compare these by calculating MSE:

```
computers_predict_single = cbind(computers_df_norm,
                                                                 predict(lm_computers_single, interval =
computers_predict_multiple = cbind(computers_df_norm,
                                                                     predict(lm_computers_multiple, inte
mse_computers_single_linear = mean((computers_predict_single$PRP - computers_predict_single$fit) ^
                                                                        2)
mse_computers_multi_linear = mean((
    computers_predict_multiple$PRP - computers_predict_multiple$fit
) ^ 2)
print(sprintf("Single linear regression MSE = %f", mse_computers_single_linear))
```

```
## [1] "Single linear regression MSE = 0.005020"
```

```
print(sprintf(
    "Multiple linear regression MSE = %f",
    mse_computers_multi_linear
))
```

```
## [1] "Multiple linear regression MSE = 0.002658"
```

As expected, the multiple-regression model performs better (0.0027 vs 0.0050).

1b. Use the k-NN regression to build the second model, applying LOO or 10fold cross-validation

We will be using 10-fold cross-validation:

```
x = 10
n = nrow(computers_df_norm)
chunkSize = floor(n / x)
meanMSE = 0.0
indexRange = 1:n
permutation = sample(indexRange, n)
```

```r
startIndex = 1
for (i in 1:x) {
    stopIndex = startIndex + chunkSize - 1

    # Setting the indices for current fold
    test = permutation[startIndex:stopIndex]
    train = indexRange[-test]

    # Removing PRP from the training data
    computers_train = computers_df_norm[train, -7]
    computers_train_labels = computers_df_norm[train, 7]
    computers_test = computers_df_norm[test, -7]
    computers_test_labels = computers_df_norm[test, 7]

    best_k = best_k_for_knn_reg(
        computers_train,
        computers_train_labels,
        computers_test,
        computers_test_labels,
        kStart =  1,
        kEnd = 50
    )

    computers_knn =  knn.reg(
        train = computers_train,
        test = computers_test,
        y = computers_train_labels,
        k = best_k
    )
    mse = mean((computers_knn$pred - computers_test_labels) ^ 2)
    print(sprintf("Fold %d: Best k = %d with MSE = %f", i, best_k, mse))

    meanMSE = meanMSE + mse

    # Start index for next iteration
    startIndex = stopIndex + 1
}
```

```
## [1] "Fold 1: Best k = 1 with MSE = 0.000765"
## [1] "Fold 2: Best k = 1 with MSE = 0.011639"
## [1] "Fold 3: Best k = 2 with MSE = 0.001030"
## [1] "Fold 4: Best k = 10 with MSE = 0.000701"
## [1] "Fold 5: Best k = 33 with MSE = 0.000367"
## [1] "Fold 6: Best k = 5 with MSE = 0.001537"
## [1] "Fold 7: Best k = 5 with MSE = 0.000743"
## [1] "Fold 8: Best k = 28 with MSE = 0.000279"
## [1] "Fold 9: Best k = 1 with MSE = 0.004694"
## [1] "Fold 10: Best k = 3 with MSE = 0.000655"
```

```r
mse_computers_knn = meanMSE / x
print(sprintf("k-NN regression with %d-fold CV: MSE = %f", x, mse_computers_knn))
```

```
## [1] "k-NN regression with 10-fold CV: MSE = 0.002241"
```

1c. Compare the best model in (a) and the k-NN model you defined in (b). Which model do you prefer? Why? What is/are the advantage(s) of your choice? What about the drawbacks?

As can be seen, the k-NN model outperforms the linear regression model. If we have enough memory, a k-NN model is (slightly) better here, but if we are limited (or have a huge amount of training data), the linear regression model would be more sensible.

```
print(sprintf("Single linear regression MSE = %f", mse_computers_single_linear))
```

```
## [1] "Single linear regression MSE = 0.005020"
```

```
print(sprintf("Multiple linear regression: MSE = %f", mse_computers_multi_linear))
```

```
## [1] "Multiple linear regression: MSE = 0.002658"
```

```
print(sprintf("k-NN regression with 10-fold CV: MSE = %f", mse_computers_knn))
```

```
## [1] "k-NN regression with 10-fold CV: MSE = 0.002241"
```

2. Compare the two models (linear regression VS k-NN) for the Cars dataset (filename: Cars.txt) as it follows:

2d. Evaluate the quality of the fit (of the best model) between a single regression model of your choice and a multiple regression.

As we have already loaded the dataset (and removed the NAs), we will now exclude vendor, model, and ERP

```
cars_df_cleaned_short = cars_df_cleaned[, 1:7]
```

Then normalize the dataframe:

```
cars_df_norm = as.data.frame(lapply(cars_df_cleaned_short, normalize))
summary(cars_df_norm)
```

```
##       mpg            cylinders       displacement       horsepower    
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.2128   1st Qu.:0.2000   1st Qu.:0.09561   1st Qu.:0.1576  
##  Median :0.3657   Median :0.2000   Median :0.21447   Median :0.2582  
##  Mean   :0.3842   Mean   :0.4944   Mean   :0.32665   Mean   :0.3178  
##  3rd Qu.:0.5319   3rd Qu.:1.0000   3rd Qu.:0.53682   3rd Qu.:0.4348  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000  
##      weight        acceleration         year       
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.1736   1st Qu.:0.3438   1st Qu.:0.2500  
##  Median :0.3375   Median :0.4464   Median :0.5000  
##  Mean   :0.3869   Mean   :0.4489   Mean   :0.4983  
##  3rd Qu.:0.5676   3rd Qu.:0.5372   3rd Qu.:0.7500  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

Creating the linear regressions:

```
lm_cars_single = lm(mpg ~ weight, data = cars_df_norm)
summary(lm_cars_single)
```

```
## 
## Call:
## lm(formula = mpg ~ weight, data = cars_df_norm)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.31845 -0.07329 -0.00893  0.05686  0.43934 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.66174    0.01102   60.03   <2e-16 ***
## weight      -0.71735    0.02420  -29.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.1152 on 390 degrees of freedom
## Multiple R-squared:  0.6926, Adjusted R-squared:  0.6918 
## F-statistic: 878.8 on 1 and 390 DF,  p-value: < 2.2e-16
```

```
lm_cars_multiple = lm(mpg ~ cylinders + displacement + horsepower + weight + acceleration + year,
                                            data = cars_df_norm)
summary(lm_cars_multiple)
```

```
##
## Call:
## lm(formula = mpg ~ cylinders + displacement + horsepower + weight +
##     acceleration + year, data = cars_df_norm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.23119 -0.06347 -0.00213  0.05397  0.38193
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.490358   0.030972  15.832   <2e-16 ***
## cylinders    -0.043864   0.044163  -0.993    0.321
## displacement  0.079031   0.075730   1.044    0.297
## horsepower   -0.001915   0.067711  -0.028    0.977
## weight       -0.637357   0.062850 -10.141   <2e-16 ***
## acceleration  0.038101   0.045590   0.836    0.404
## year          0.240436   0.016793  14.318   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09136 on 385 degrees of freedom
## Multiple R-squared:  0.8093, Adjusted R-squared:  0.8063
## F-statistic: 272.2 on 6 and 385 DF,  p-value: < 2.2e-16
```

Comparing them through the MSEs:

```
cars_predict_single = cbind(cars_df_norm,
                                                        predict(lm_cars_single, interval = 'confidence'
cars_predict_multiple = cbind(cars_df_norm,
                                                            predict(lm_cars_multiple, interval = 'confi
mse_car_single_linear = mean((cars_predict_single$mpg - cars_predict_single$fit) ^
                                                    2)
mse_car_multi_linear = mean((cars_predict_multiple$mpg - cars_predict_multiple$fit) ^ 2)
print(sprintf("Single linear regression MSE = %f", mse_car_single_linear))
```

```
## [1] "Single linear regression MSE = 0.013211"
```

```
print(sprintf("Multiple linear regression MSE = %f", mse_car_multi_linear))
```

```
## [1] "Multiple linear regression MSE = 0.008198"
```

Once again, multiple linear regression performs better than the single regression.

2e. Use the k-NN regression to build the second model, applying LOO or 10fold cross-validation.

We will be using 10-fold cross-validation:

```
x = 10
n = nrow(computers_df_norm)
chunkSize = floor(n / x)
meanMSE = 0.0
indexRange = 1:n
permutation = sample(indexRange, n)
startIndex = 1
for (i in 1:x) {
```

```r
    stopIndex = startIndex + chunkSize - 1

    # Setting the indices for current fold
    test = permutation[startIndex:stopIndex]
    train = indexRange[-test]

    # Removing mpg (first column) from the training data
    cars_train = cars_df_norm[train, -1]
    cars_train_labels = cars_df_norm[train, 1]
    cars_test = cars_df_norm[test, -1]
    cars_test_labels = cars_df_norm[test, 1]

    best_k = best_k_for_knn_reg(
        cars_train,
        cars_train_labels,
        cars_test,
        cars_test_labels,
        kStart =  1,
        kEnd = 50
    )

    cars_knn =  knn.reg(
        train = cars_train,
        test = cars_test,
        y = cars_train_labels,
        k = best_k
    )
    mse = mean((cars_knn$pred - cars_test_labels) ^ 2)
    print(sprintf("Fold %d: Best k = %d with MSE = %f", i, best_k, mse))

    meanMSE = meanMSE + mse

    # Start index for next iteration
    startIndex = stopIndex + 1
}
```

```
## [1] "Fold 1: Best k = 6 with MSE = 0.002469"
## [1] "Fold 2: Best k = 4 with MSE = 0.000787"
## [1] "Fold 3: Best k = 1 with MSE = 0.003546"
## [1] "Fold 4: Best k = 1 with MSE = 0.002352"
## [1] "Fold 5: Best k = 9 with MSE = 0.002139"
## [1] "Fold 6: Best k = 1 with MSE = 0.002741"
## [1] "Fold 7: Best k = 1 with MSE = 0.003015"
## [1] "Fold 8: Best k = 5 with MSE = 0.002487"
## [1] "Fold 9: Best k = 1 with MSE = 0.005066"
## [1] "Fold 10: Best k = 5 with MSE = 0.001377"
```

```r
mse_cars_knn = meanMSE / x
print(sprintf("k-NN regression with %d-fold CV: MSE = %f", x, mse_cars_knn))
```

```
## [1] "k-NN regression with 10-fold CV: MSE = 0.002598"
```

2f. Compare the best model in (d) and the k-nn model you defined in (e). Which model do you prefer? Why? What is/are the advantage(s) of your choice? What about the drawbacks?

Once again, k-NN regression has a lower MSE than both the single and multiple linear regressions. If we have enough memory, a k-NN model is definitely better here (compared to the cars df), but if we are limited (or have a huge amount of training data), the linear regression model would be more sensible.

```
print(sprintf("Single linear regression MSE = %f", mse_car_single_linear))
```

```
## [1] "Single linear regression MSE = 0.013211"
```

```
print(sprintf("Multiple linear regression: MSE = %f", mse_car_multi_linear))
```

```
## [1] "Multiple linear regression: MSE = 0.008198"
```

```
print(sprintf("k-NN regression with 10-fold CV: MSE = %f", mse_cars_knn))
```

```
## [1] "k-NN regression with 10-fold CV: MSE = 0.002598"
```