Exercise #7

Logistic Regression with R

Brian Schweigler; 16-102-071

04/05/2022

Preliminaries

Load the required libraries

```
library(FNN)
```

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

Set a seed for later:

```
set.seed(1786397)
```

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 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.)

Loading the cancer dataset:

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

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

Having a look at the cancer_df:

```
summary(cancer_df)
```

```
##        ID              Diagnostic            Radius          Texture
##  Min.   :     8670   Length:569         Min.   : 6.981   Min.   : 9.71
##  1st Qu.:   869218   Class :character   1st Qu.:11.700   1st Qu.:16.17
##  Median :   906024   Mode  :character   Median :13.370   Median :18.84
##  Mean   : 30371831                      Mean   :14.127   Mean   :19.29
##  3rd Qu.:  8813129                      3rd Qu.:15.780   3rd Qu.:21.80
##  Max.   :911320502                      Max.   :28.110   Max.   :39.28
##    Perimeter           Area            Smooth           Compact
##  Min.   : 43.79   Min.   : 143.5   Min.   :0.05263   Min.   :0.01938
##  1st Qu.: 75.17   1st Qu.: 420.3   1st Qu.:0.08637   1st Qu.:0.06492
##  Median : 86.24   Median : 551.1   Median :0.09587   Median :0.09263
##  Mean   : 91.97   Mean   : 654.9   Mean   :0.09636   Mean   :0.10434
##  3rd Qu.:104.10   3rd Qu.: 782.7   3rd Qu.:0.10530   3rd Qu.:0.13040
##  Max.   :188.50   Max.   :2501.0   Max.   :0.16340   Max.   :0.34540
##    Concavity          Concave           Symmetry         Fractal
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.1060   Min.   :0.04996
##  1st Qu.:0.02956   1st Qu.:0.02031   1st Qu.:0.1619   1st Qu.:0.05770
##  Median :0.06154   Median :0.03350   Median :0.1792   Median :0.06154
##  Mean   :0.08880   Mean   :0.04892   Mean   :0.1812   Mean   :0.06280
##  3rd Qu.:0.13070   3rd Qu.:0.07400   3rd Qu.:0.1957   3rd Qu.:0.06612
##  Max.   :0.42680   Max.   :0.20120   Max.   :0.3040   Max.   :0.09744
##     RadiusSE        TextureSE       PerimeterSE         AreaSE
##  Min.   :0.1115   Min.   :0.3602   Min.   : 0.757   Min.   :  6.802
##  1st Qu.:0.2324   1st Qu.:0.8339   1st Qu.: 1.606   1st Qu.: 17.850
##  Median :0.3242   Median :1.1080   Median : 2.287   Median : 24.530
##  Mean   :0.4052   Mean   :1.2169   Mean   : 2.866   Mean   : 40.337
##  3rd Qu.:0.4789   3rd Qu.:1.4740   3rd Qu.: 3.357   3rd Qu.: 45.190
##  Max.   :2.8730   Max.   :4.8850   Max.   :21.980   Max.   :542.200
##     SmoothSE          CompactSE         ConcavitySE        ConcaveSE
##  Min.   :0.001713   Min.   :0.002252   Min.   :0.00000   Min.   :0.000000
##  1st Qu.:0.005169   1st Qu.:0.013080   1st Qu.:0.01509   1st Qu.:0.007638
##  Median :0.006380   Median :0.020450   Median :0.02589   Median :0.010930
```

```
##  Mean   :0.007041   Mean   :0.025478   Mean   :0.03189   Mean   :0.011796
##  3rd Qu.:0.008146   3rd Qu.:0.032450   3rd Qu.:0.04205   3rd Qu.:0.014710
##  Max.   :0.031130   Max.   :0.135400   Max.   :0.39600   Max.   :0.052790
##    SymmetrySE         FractalSE           RadiusMax       TextureMax
##  Min.   :0.007882   Min.   :0.0008948   Min.   : 7.93   Min.   :12.02
##  1st Qu.:0.015160   1st Qu.:0.0022480   1st Qu.:13.01   1st Qu.:21.08
##  Median :0.018730   Median :0.0031870   Median :14.97   Median :25.41
##  Mean   :0.020542   Mean   :0.0037949   Mean   :16.27   Mean   :25.68
##  3rd Qu.:0.023480   3rd Qu.:0.0045580   3rd Qu.:18.79   3rd Qu.:29.72
##  Max.   :0.078950   Max.   :0.0298400   Max.   :36.04   Max.   :49.54
##   PerimeterMax       AreaMax         SmoothMax         CompactMax
##  Min.   : 50.41   Min.   : 185.2   Min.   :0.07117   Min.   :0.02729
##  1st Qu.: 84.11   1st Qu.: 515.3   1st Qu.:0.11660   1st Qu.:0.14720
##  Median : 97.66   Median : 686.5   Median :0.13130   Median :0.21190
##  Mean   :107.26   Mean   : 880.6   Mean   :0.13237   Mean   :0.25427
##  3rd Qu.:125.40   3rd Qu.:1084.0   3rd Qu.:0.14600   3rd Qu.:0.33910
##  Max.   :251.20   Max.   :4254.0   Max.   :0.22260   Max.   :1.05800
##   ConcavityMax      ConcaveMax       SymmetryMax       FractalMax
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.1565   Min.   :0.05504
##  1st Qu.:0.1145   1st Qu.:0.06493   1st Qu.:0.2504   1st Qu.:0.07146
##  Median :0.2267   Median :0.09993   Median :0.2822   Median :0.08004
##  Mean   :0.2722   Mean   :0.11461   Mean   :0.2901   Mean   :0.08395
##  3rd Qu.:0.3829   3rd Qu.:0.16140   3rd Qu.:0.3179   3rd Qu.:0.09208
##  Max.   :1.2520   Max.   :0.29100   Max.   :0.6638   Max.   :0.20750
```

As is mentioned in the PDF, no NAs are within the data. The range of the values can't be gauged without further information.

1. Consider the Cars dataset (filename: Cars.txt).

1a. Build three different (generalized) linear regression models to predict mpg (at least one of them must be a multiple regression model).

First we will only select the values we are interested in:

```
cars_df_cleaned_short = cars_df_cleaned[, 1:6]
```

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
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.1736   1st Qu.:0.3438
##  Median :0.3375   Median :0.4464
##  Mean   :0.3869   Mean   :0.4489
##  3rd Qu.:0.5676   3rd Qu.:0.5372
##  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_single_alternate = lm(mpg ~ acceleration, data = cars_df_norm)
summary(lm_cars_single_alternate)
```

```
##
## Call:
```

```
## lm(formula = mpg ~ acceleration, data = cars_df_norm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.47844 -0.14935 -0.03188  0.12769  0.61807
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.14400    0.02771   5.196 3.29e-07 ***
## acceleration  0.53511    0.05799   9.228  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1883 on 390 degrees of freedom
## Multiple R-squared:  0.1792, Adjusted R-squared:  0.1771
## F-statistic: 85.15 on 1 and 390 DF,  p-value: < 2.2e-16
```

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

```
##
## Call:
## lm(formula = mpg ~ cylinders + displacement + horsepower + weight +
##     acceleration, data = cars_df_norm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.30802 -0.07611 -0.00905  0.05968  0.43462
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6750986  0.0348099  19.394   <2e-16 ***
## cylinders    -0.0529160  0.0545937  -0.969   0.3330
## displacement -0.0008556  0.0933718  -0.009   0.9927
## horsepower   -0.2214708  0.0815369  -2.716   0.0069 **
## weight       -0.4865494  0.0766040  -6.351    6e-10 ***
## acceleration -0.0130042  0.0561912  -0.231   0.8171
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.113 on 386 degrees of freedom
## Multiple R-squared:  0.7077, Adjusted R-squared:  0.7039
## F-statistic: 186.9 on 5 and 386 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_single_alt = cbind(cars_df_norm,
                                                        predict(lm_cars_single_alternate, interv
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_single_linear_alt = mean((cars_predict_single_alt$mpg - cars_predict_single_alt$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("Alternate Single linear regression MSE = %f", mse_car_single_linear_alt))
```

```
## [1] "Alternate Single linear regression MSE = 0.035277"
```

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

```
## [1] "Multiple linear regression MSE = 0.012563"
```

Multiple linear regression performs slightly better. Also, weight is better suited than acceleration for single linear regression.

1b. Perform 10-fold cross validation to estimate the test error of the models you built in a).

For this we can define a function:

```
generalized_linear_cv = function(df, formula, x) {
    n = nrow(df)
    chunkSize = floor(n / x)
    mse.list = c()

    indexRange = 1:n
    permutation = sample(indexRange, n)

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

        # Indices for current fold
        test = permutation[startIndex:stopIndex]
        train = indexRange[-test]

        df.train = df[train, ]
        df.test = df[test, ]

        df.glm = glm(formula, data = df.train)
        df.predict = predict.glm(df.glm, newdata = df.test, type = "response")
        mse = mean((df.predict - df.test$mpg) ^ 2)

        mse.list = append(mse.list, mse)

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

    meanMSE = mean(mse.list)
    return(list("mean" = meanMSE, "mse" = mse.list))
}
```

```
cars_single_results = generalized_linear_cv(cars_df_norm, lm_cars_single, 10)
cars_single_alt_results = generalized_linear_cv(cars_df_norm, lm_cars_single_alternate, 10)
cars_multi_results = generalized_linear_cv(cars_df_norm, lm_cars_multiple, 10)
```

```
print(sprintf("Single linear regression MSE = %f", cars_single_results$mean))
```

```
## [1] "Single linear regression MSE = 0.013372"
```

```
print(sprintf("Alternate Single linear regression MSE = %f", cars_single_alt_results$mean))
```

```
## [1] "Alternate Single linear regression MSE = 0.035667"
```

```
print(sprintf("Multiple linear regression MSE = %f", cars_multi_results$mean))
```

```
## [1] "Multiple linear regression MSE = 0.012957"
```

Using the 10-fold cross validation we still see that the multiple regression variant is slightly better.

1c. Compare the schemes in a) performing a t-test.

For the t-test, we will require the vector of the MSEs:

We first compare the single with the multiple linear regression.

```
t.test(cars_single_results$mse, cars_multi_results$mse, paired=TRUE, alternative="two.sided")
```

```
##
##  Paired t-test
##
## data:  cars_single_results$mse and cars_multi_results$mse
## t = 0.2308, df = 9, p-value = 0.8226
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.003658501  0.004489850
## sample estimates:
## mean of the differences
##            0.0004156748
```

The p-value is too high to say that the models are vastly different (as seen in the close MSEs of the two).

Comparing the alternative variant of the single regression model to both the other models, we see that this one is significantly different (p value < 0.05 for both).

```
t.test(cars_single_results$mse, cars_single_alt_results$mse, paired=TRUE, alternative="two.sided")
```

```
##
##  Paired t-test
##
## data:  cars_single_results$mse and cars_single_alt_results$mse
## t = -7.4615, df = 9, p-value = 3.845e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.02905396 -0.01553547
## sample estimates:
## mean of the differences
##             -0.02229472
```

```
t.test(cars_multi_results$mse, cars_single_alt_results$mse, paired=TRUE, alternative="two.sided")
```

```
##
##  Paired t-test
##
```

```
## data:  cars_multi_results$mse and cars_single_alt_results$mse
## t = -10.533, df = 9, p-value = 2.318e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.02758768 -0.01783310
## sample estimates:
## mean of the differences
##             -0.02271039
```

2. Apply the logistic regression to predict the category diagnosis and interpret the most important values of the model that you obtained with R. Can you estimate the error rate of your model?

As a preliminary step, we will remove the id and then encode diagnosis as a factor:

```
cancer.df.short <- cancer_df[,-1]
cancer.df.short$Diagnostic <-
    factor(
        cancer.df.short$Diagnostic,
        levels = c("B", "M"),
        labels = c("Benign", "Malignant")
    )
```

Now we can create the training and test sets:

```
train.size = round(nrow(cancer.df.short) * 0.7)
train.index = sample(c(1:nrow(cancer.df.short)), train.size)
cancer.df.train = cancer.df.short[train.index, ]
cancer.df.test = cancer.df.short[-train.index, ]
cancer.df.train.label <- cancer.df.train[, 1]
cancer.df.test.label <- cancer.df.test[, 1]
```

The logistic model can be created as follows:

```
cancer.df.classifier <-
    glm(Diagnostic ~ .,
            data = cancer.df.train,
            family = binomial(link = "logit"))
```

```
## Warning: glm.fit: algorithm did not converge
```

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

```
cancer.df.classifier
```

```
##
## Call:  glm(formula = Diagnostic ~ ., family = binomial(link = "logit"),
##     data = cancer.df.train)
##
## Coefficients:
##  (Intercept)        Radius       Texture     Perimeter          Area
##   -1.156e+03    -2.290e+02     1.728e+01     1.579e+01     1.589e+00
##       Smooth       Compact     Concavity       Concave      Symmetry
##   -2.347e+03    -5.330e+03     1.786e+03     2.471e+03    -1.448e+03
##      Fractal      RadiusSE     TextureSE   PerimeterSE        AreaSE
##    1.630e+04     4.859e+02    -5.058e+01     7.531e+01    -5.221e+00
##     SmoothSE     CompactSE   ConcavitySE     ConcaveSE    SymmetrySE
##   -6.241e+03     3.613e+03    -4.460e+03     2.227e+04    -1.164e+04
##    FractalSE     RadiusMax    TextureMax  PerimeterMax       AreaMax
##   -2.396e+04     1.847e+02     2.475e-01    -1.691e+01    -6.448e-01
##    SmoothMax    CompactMax  ConcavityMax    ConcaveMax   SymmetryMax
##    2.353e+03     6.567e+02     3.356e+02    -3.348e+02     2.122e+03
##   FractalMax
##   -5.514e+03
##
## Degrees of Freedom: 397 Total (i.e. Null);  367 Residual
## Null Deviance:       527.4
```

```
## Residual Deviance: 1.877e-07     AIC: 62
```

The algorithm did not converge, but we have a low AIC value.

Making a prediction on test data:

```
cancer.df.pred <-
    predict(cancer.df.classifier, cancer.df.test, type = "response")
threshold <- 0.5
cancer.df.pred.results <-
    as.factor(ifelse(cancer.df.pred < threshold, "Benign", "Malignant"))
```

And evaluating this model:

```
correct.predictions = sum (cancer.df.test.label == cancer.df.pred.results)
print(sprintf("Number of correct predictions = %d", correct.predictions))
```

```
## [1] "Number of correct predictions = 153"
```

```
accuracy = correct.predictions / nrow(cancer.df.test)
print(sprintf("Accuracy = %f", accuracy))
```

```
## [1] "Accuracy = 0.894737"
```

```
table(cancer.df.test$Diagnostic)
```

```
##
##    Benign Malignant
##       109        62
```

Predicting the error rate

We could use precision and recall to estimate the error rate:

```
confusion.mat <-
    table(cancer.df.test.label, cancer.df.pred.results)[2:1, 2:1]
confusion.mat
```

```
##                     cancer.df.pred.results
## cancer.df.test.label Malignant Benign
##            Malignant        54      8
##            Benign           10     99
```

```
TP = confusion.mat[1]
TN = confusion.mat[4]
FP = confusion.mat[2]
FN = confusion.mat[3]

precision = TP / (TP + FP)
print(sprintf("Precision = %f", precision))
```

```
## [1] "Precision = 0.843750"
```

```
recall = TP / (TP + FN)
print(sprintf("Recall = %f", recall))
```

```
## [1] "Recall = 0.870968"
```