DA5020 – Asssigment 11

Rebecca Weiss

12/3/2021

Clear the workspace:

```
rm(list = ls())
```

```
#load all necessary libraries
library(tidyverse)
library(caret)
```

1. Load the diabetes dataset "diabetes.csv", inspect the data and gather any relevant summary statistics.

```
# load datasdet downloaded from kaggle
df <- read_csv("diabetes.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Pregnancies = col_double(),
##   Glucose = col_double(),
##   BloodPressure = col_double(),
##   SkinThickness = col_double(),
##   Insulin = col_double(),
##   BMI = col_double(),
##   DiabetesPedigreeFunction = col_double(),
##   Age = col_double(),
##   Outcome = col_double()
## )
```

```
# get summary stats/info
summary(df)
```

```
##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
##  Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
```

```
##  3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
##  1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
##  Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
##  Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
##  3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
##     Outcome
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.349
##  3rd Qu.:1.000
##  Max.   :1.000
```

```
str(df)
```

```
## spec_tbl_df [768 x 9] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Pregnancies             : num [1:768] 6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : num [1:768] 148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure           : num [1:768] 72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : num [1:768] 35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : num [1:768] 0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num [1:768] 33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num [1:768] 0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : num [1:768] 50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : num [1:768] 1 0 1 0 1 0 1 0 1 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Pregnancies = col_double(),
##   ..   Glucose = col_double(),
##   ..   BloodPressure = col_double(),
##   ..   SkinThickness = col_double(),
##   ..   Insulin = col_double(),
##   ..   BMI = col_double(),
##   ..   DiabetesPedigreeFunction = col_double(),
##   ..   Age = col_double(),
##   ..   Outcome = col_double()
##   .. )
```

From inspecting the data, we can see that there are no missing data and that all are coded with a 0 or 1 for the *Outcome*, and all explanatory variables are numeric which I will convert to a factor. From the summary statistics, the spread looks pretty good with all means/medians being similar, with the exception of insulin, which has a wide spread and is skewed to lower numbers. According to the data description in Kaggle, the value is a 2-hour serum insulin, so it is possible that this skew is expected. Because of this, for now I will leave the outliers impacting the distribution.

```
df$Outcome <- factor(df$Outcome, levels = c(0, 1))
str(df$Outcome)
```

```
##  Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 1 2 2 ...
```

2. Normalize the explanatory variables using min-max normalization.

```
# create a function
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x)))
}

# apply over all explanatory variables
df_norm <- cbind(normalize(df[-9]), df[9])
summary(df_norm)
```

```
##   Pregnancies          Glucose       BloodPressure     SkinThickness
##  Min.   :0.000000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.001182   1st Qu.:0.1170   1st Qu.:0.07329   1st Qu.:0.00000
##  Median :0.003546   Median :0.1383   Median :0.08511   Median :0.02719
##  Mean   :0.004545   Mean   :0.1429   Mean   :0.08168   Mean   :0.02427
##  3rd Qu.:0.007092   3rd Qu.:0.1658   3rd Qu.:0.09456   3rd Qu.:0.03783
##  Max.   :0.020095   Max.   :0.2352   Max.   :0.14421   Max.   :0.11702
##     Insulin             BMI          DiabetesPedigreeFunction      Age
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000922        Min.   :0.02482
##  1st Qu.:0.00000   1st Qu.:0.03227   1st Qu.:0.0002881        1st Qu.:0.02837
##  Median :0.03605   Median :0.03783   Median :0.0004403        Median :0.03428
##  Mean   :0.09433   Mean   :0.03782   Mean   :0.0005578        Mean   :0.03929
##  3rd Qu.:0.15041   3rd Qu.:0.04326   3rd Qu.:0.0007402        3rd Qu.:0.04846
##  Max.   :1.00000   Max.   :0.07931   Max.   :0.0028605        Max.   :0.09574
##  Outcome
##  0:500
##  1:268
##
##
##
##
```

From normalizing the original *df*, we can put all the explanatory variables between 0 and 1 to minimize the standard deviations. Here, we see that the insulin value mentioned above is still skewed, but that standard deviation being lower should decrease the impact of outliers in our model.

3. Split the data into a training set and a test set i.e. perform an 80/20 split; 80% of the data should be designated as the training data and 20% as the test data.

```
# create variable to split the index
set.seed(123)
index <- as.integer(nrow(df)*.8)


# create train and test from index
train <- df_norm[1:index,]
test <- df_norm[(index+1):nrow(df),]
```

```r
# make sure split is correct
dim(train)
```

```
## [1] 614   9
```

```r
dim(test)
```

```
## [1] 154   9
```

4. Create a function called knn_predict(). The function should accept the following as input: the training set, the test set and the value of k. For example knn_predict(train.data, test.data, k).

- Implement the logic for the k-nn algorithm from scratch (without using any libraries). There is an example in the lecture series on Canvas. The goal of your k-nn algorithm is to predict the Outcome (i.e. whether or not the patient has diabetes) using the explanatory variables. • The function should return a list/vector of predictions for all observations in the test set.

```r
# Creating the component functions that will be used in the custom k-NN implementation

# dist() - calculates the Euclidean distance between two vectors of equal size containing numeric eleme
dist <- function(p,q)
{
  p <- unlist(p)
  q <- unlist(q)
  dist <- sqrt(sum((p - q)^2))
  return (dist)
}


# neighbors() - get vector of distances between an object u and a dataframe of features
neighbors <- function(train, u) {

  newdf <- train %>%
    rowwise() %>%
    mutate(distance = dist(train[-9], u))

  train$distance <- newdf$distance

  return (train)
}


# k.closest - get smallest k values in a vector of values
k.closest <- function(neighbors,k)
{
  ordered.neighbors <- neighbors[order(neighbors$distance),]
  closest <- ordered.neighbors[1:k,]

  return(closest)
}
```

```r
# find mean of k closest neighbors
k.mean <- function(x)
{
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

```r
knn_predict <- function(train, test, k) {
  result = c()

  # Iterating through and classifying every row in the test data set
  for (i in 1:nrow(test)) {
    u <- test[i,]
    nb <- neighbors(train,u)
    k.nbrs <- k.closest(nb,k)

    # Finding the mean of the k closest values
    res <- k.mean(as.vector(k.nbrs$Outcome))
    result <- c(result, res)
  }

  return(result)
}
```

5. Demonstrate that the knn_predict() function works and use it to make predictions for the test set. You can determine a suitable value of k for your demonstration. After which, analyze the results that were returned from the function using a confusion matrix. Explain the results. Note: refer to the 'Useful Resources' section for more information on building a confusion matrix in R.

```r
# try K = 4, store into "predicted" variable
predicted <- knn_predict(train, test, 4)
# convert to factor
predicted <- factor(predicted, levels = c(0, 1))

# Printing the results as a table
table(predicted, test$Outcome)
```

```
##
## predicted  0  1
##         0  0  0
##         1 99 55
```

```r
# create a confusion matrix
cm <- confusionMatrix(predicted, test$Outcome, positive = '1')
```

From the output, we see that the accuracy is not great, which = 0.3571429 when K = 4. We also notice that the model only predicted patients as having diabetes, as we can see from there being no "0" in the outcome for predicted.

6 (bonus). Repeat question 5 and perform an experiment using different values of k. Ensure that you try at least 5 different values of k and display the confusion matrix from each attempt. Which value of k produced the most accurate predictions?

```
# create function
diff_ks <- function(i) {
  predicted <- knn_predict(train, test, i)
  # convert to factor
  predicted <- factor(predicted, levels = c(0, 1))

  print(paste0('Creating a confusion matrix with ', i,' value of K.'))
  # create a confusion matrix
  cm <- confusionMatrix(predicted, test$Outcome, positive = '1')
  print(paste0('Accuracy =', cm$overall[1],' when K = ', i))
  return(cm)
}
```

```
k_to_test = seq(5, 18, 3)
lapply(k_to_test, diff_ks)
```

```
## [1] "Creating a confusion matrix with 5 value of K."
## [1] "Accuracy =0.357142857142857 when K = 5"
## [1] "Creating a confusion matrix with 8 value of K."
## [1] "Accuracy =0.357142857142857 when K = 8"
## [1] "Creating a confusion matrix with 11 value of K."
## [1] "Accuracy =0.357142857142857 when K = 11"
## [1] "Creating a confusion matrix with 14 value of K."
## [1] "Accuracy =0.357142857142857 when K = 14"
## [1] "Creating a confusion matrix with 17 value of K."
## [1] "Accuracy =0.357142857142857 when K = 17"
```

```
## [[1]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  0  0
##          1 99 55
##
##                Accuracy : 0.3571
##                  95% CI : (0.2816, 0.4382)
##     No Information Rate : 0.6429
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.3571
```

```
##          Neg Pred Value :    NaN
##              Prevalence : 0.3571
##          Detection Rate : 0.3571
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
##
## [[2]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  0  0
##          1 99 55
##
##                Accuracy : 0.3571
##                  95% CI : (0.2816, 0.4382)
##     No Information Rate : 0.6429
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.3571
##          Neg Pred Value :    NaN
##              Prevalence : 0.3571
##          Detection Rate : 0.3571
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
##
## [[3]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  0  0
##          1 99 55
##
##                Accuracy : 0.3571
##                  95% CI : (0.2816, 0.4382)
##     No Information Rate : 0.6429
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
```

```
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.3571
##          Neg Pred Value :    NaN
##              Prevalence : 0.3571
##          Detection Rate : 0.3571
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
##
## [[4]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  0  0
##          1 99 55
##
##                Accuracy : 0.3571
##                  95% CI : (0.2816, 0.4382)
##     No Information Rate : 0.6429
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.3571
##          Neg Pred Value :    NaN
##              Prevalence : 0.3571
##          Detection Rate : 0.3571
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
##
## [[5]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  0  0
##          1 99 55
##
##                Accuracy : 0.3571
##                  95% CI : (0.2816, 0.4382)
##     No Information Rate : 0.6429
##     P-Value [Acc > NIR] : 1
```

```
##
##                  Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.3571
##          Neg Pred Value :    NaN
##              Prevalence : 0.3571
##          Detection Rate : 0.3571
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
```