Machine Learning, Spring 2021
Homework 6

Daniil Bulat, Jonas Josef Huwyler, Haochen Li,Giovanni Magagnin

4/29/2021

Exercise 1

Consider the WDBC data with the two features perimeter_mean and concaveP_mean. Load the *library(class)* and use the *knn()*-command for the K-nearest neighbor algorithm.

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

Question 1

Split the data into trainning and testing sets in the ratio 70%-30%.

```
# Shuffle and Split Data (70-30)
set.seed(123)
Data = Data[sample(nrow(Data)),]

# Create data frame for data to be used
df = data.frame(id = Data$id,
                diagnosis = Data$diagnosis,
                perimeter_mean = Data$perimeter_mean,
                concaveP_mean = Data$concaveP_mean)
ntrain = floor(nrow(df)*0.7)
df.Train = df[1:ntrain,]
df.Test = df[(ntrain+1):nrow(Data),]
```

Question 2

Run the K-NN algorithm for K = 1,10,25 with *knn(. . . ,prob=TRUE)*. What type of predictions do you get, what do they mean?

```
xvars = c("perimeter_mean","concaveP_mean")
yvars = "diagnosis"

knn_T_1 = knn(train = df.Train[xvars],
              test = df.Test[xvars],
              cl = as.matrix(df.Train[yvars]),
              k = 1,
              prob = TRUE )

knn_T_10 = knn(train = df.Train[xvars],
               test = df.Test[xvars],
               cl = as.matrix(df.Train[yvars]),
               k = 10,
```

```
               prob = TRUE )

knn_T_25 = knn(train = df.Train[xvars],
               test = df.Test[xvars],
               cl = as.matrix(df.Train[yvars]),
               k = 25,
               prob = TRUE )

print(knn_T_1)
```

```
##   [1] B B B B M B M B M M M M B B B M M M B B M B B B M B B M B B B B M M B B B
##  [38] M B B B M M B M B B M B B B M B B B B B B B B M M B B M M B B B B M M M
##  [75] B B B B M M M B B B B M B B B B B M M B M M B B B M B B M B B M M B M B B
## [112] B B B B B M M M M M B B B M B B M M M M M B M M B B B B M M B B M B B B M
## [149] B M B M B B B B B M M M M B B B M M B B M B M
## attr(,"prob")
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## Levels: B M
```

```
print(class(knn_T_1))
```

```
## [1] "factor"
```

We have a training data set on which we train the KNN algorithm. This means that the algorithm learns the location of every training data point and its type. Then the algorithm takes the location of every single test data points, looks at the type of its k (i.e. 1, 10 or 25) nearest neighbors and predicts the class of the test data point according to a majority vote.

The probability which is in this case displayed for the KNN algorithm shows us the percentage (or how close) the majority vote was. If our predicted class was M and the probability 0.75, this means that 75% of the k neighbors of that point "voted" for the M-class.

Question 3

Run the K-NN algorithm for K = 1,10,25 with *knn(. . . ,prob=FALSE)* and compute the (test) error rate.

```
knn_F_1 = knn(train = df.Train[xvars],
              test = df.Test[xvars],
              cl = as.matrix(df.Train[yvars]),
              k = 1,
              prob = FALSE )

knn_F_10 = knn(train = df.Train[xvars],
               test = df.Test[xvars],
               cl = as.matrix(df.Train[yvars]),
               k = 10,
               prob = FALSE )

knn_F_25 = knn(train = df.Train[xvars],
               test = df.Test[xvars],
               cl = as.matrix(df.Train[yvars]),
```

```
               k = 25,
               prob = FALSE )

err_F_1 = round(sum(knn_F_1 != as.matrix(df.Test[yvars]))/length(knn_F_1)* 100,2)
err_F_10 = round(sum(knn_F_10 != as.matrix(df.Test[yvars]))/length(knn_F_10)* 100,2)
err_F_25 = round(sum(knn_F_25 != as.matrix(df.Test[yvars]))/length(knn_F_25)* 100,2)

K = c(1,10,25)
errors = c(err_F_1,err_F_10,err_F_25)

for (i in 1:3){
  cat("The error rate of K-NN algorithm when K = ", K[i], "is", errors[i], "%\n")
}
```

```
## The error rate of K-NN algorithm when K =  1 is 12.87 %
## The error rate of K-NN algorithm when K =  10 is 12.28 %
## The error rate of K-NN algorithm when K =  25 is 12.87 %
```

Question 4

Run logistic regression on the same data, and predict "malign" when the probability is at least 0.5 and predict "benign" otherwise. Compute the (test) error rate.

```
glm.fit = glm(diagnosis~perimeter_mean + concaveP_mean,data = df.Train,family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = diagnosis ~ perimeter_mean + concaveP_mean, family = binomial,
##     data = df.Train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.45821  -0.26302  -0.10627   0.06101   2.59877
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -13.28007    1.77748  -7.471 7.94e-14 ***
## perimeter_mean   0.09716    0.01927   5.042 4.62e-07 ***
## concaveP_mean   77.06571   11.42843   6.743 1.55e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 515.00  on 397  degrees of freedom
## Residual deviance: 149.32  on 395  degrees of freedom
## AIC: 155.32
##
## Number of Fisher Scoring iterations: 7
```

```
yhat = predict(glm.fit, type = "response",newdata = df.Test)
yhat = ifelse(yhat<0.5,"B","M")

err_logit = round(sum(yhat != as.matrix(df.Test[yvars]))/length(yhat) * 100,2)
```

```r
cat("The error rate of logistic algorithm is", err_logit, "%\n")
```

```
## The error rate of logistic algorithm is 12.28 %
```

Question 5

Compare the four results (three times K-NN and logistic regression).

```r
errors = data.frame(KNN1 = err_F_1,
                    KNN10 = err_F_10,
                    KNN25 = err_F_25,
                    Logistic = err_logit,
                     row.names = "Error Rate")
print(errors)
```

```
##              KNN1 KNN10 KNN25 Logistic
## Error Rate 12.87 12.28 12.87    12.28
```

First, let's take a look at the three times K-NN algorithms. When K = 1 and K = 25, the error rates are equal to 12.87. When K = 10, the error rate is smaller than the former two models, which suggests that the error rates of KNN regressions perform a U-shape, and the algorithm with K = 10 performs better than the other two.

Second, the logistic regression's error rate is identical to the KNN algorithm with K = 10. As we know, if we have a linear relationship, the linear model would perform better than the non-parametric model. However, in this case, we are not sure whether the true model is linear or not. The condition that the two models share same error rate could also suggest that they define similar separating hyperplanes and have same predictions of classification.

Exercise 2

Consider an observation $x = (1, 1)$ and the following 7 nearest neighbors:

$$\mathcal{N}_x = \{(0, 2), (0, -1), (3, -1), (2, 3), (-3, 1), (0, 5), (5, 4)\}$$

with the corresponding class labels (in the same order)

$$\{1, 1, 2, 3, 3, 2, 2\}$$

First compute the distances from the neighbors to x. Then compute the probabilites for x to be in class 1,2,3 (respectively) with weighted 7-NN for the following weights:

Question 1

$$w = (\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7})$$

This corresponds to normal 7-NN. First, let's calculate the euclidean distances (with the same order) of $\mathcal{N}_x$ and x,

$$d = \{\sqrt{2}, \sqrt{10}, 2\sqrt{2}, \sqrt{5}, 4, \sqrt{17}, 5\}$$

Then the probabilites for x to be in class 1,2,3 will be

$$P(y = 1|x) = \frac{2}{7}$$

$$P(y = 2|x) = \frac{3}{7}$$

$$P(y = 3|x) = \frac{2}{7}$$

Question 2

$$w = (\frac{9}{34}, \frac{9}{34}, \frac{6}{34}, \frac{4}{34}, \frac{3}{34}, \frac{2}{34}, \frac{1}{34})$$

This is a weighted 7-NN.(The weighted vector follows the order of distance from nearest to farest) First, let's calculate the euclidean distances (with the same order) of $\mathcal{N}_x$ and x,

$$d = \{\sqrt{2}, \sqrt{10}, 2\sqrt{2}, \sqrt{5}, 4, \sqrt{17}, 5\}$$

Then the probabilites for x to be in class 1,2,3 will be

$$P(y = 1|x) = \frac{9+4}{34} = \frac{13}{34}$$

$$P(y = 2|x) = \frac{6+2+1}{34} = \frac{9}{34}$$

$$P(y = 3|x) = \frac{9+3}{34} = \frac{12}{34} = \frac{6}{17}$$

Exercise 3

Question 1

Write your own 2-NN regression algorithm for the Olympics data.

```
# Prepare for data
L.olympics=t(data.frame(c(1896,12),c(1900,11),c(1904,11),c(1906,11.2),c(1908,10.8),
                        c(1912,10.8),c(1920,10.8),c(1924,10.6),c(1928,10.8),c(1932,10.3),
                        c(1936,10.3),c(1948,10.3),c(1952,10.4),c(1956,10.5),c(1960,10.2),
                        c(1964,10),c(1968,9.95),c(1972,10.14),c(1980,10.25),c(1984,9.99),
                        c(1988,9.92),c(1992, 9.96),c(1996, 9.84),c(2000, 9.87),c(2004,9.85),
                        c(2008,9.69),c(2012, 9.61),c(2016, 9.83)))

colnames(L.olympics)=c("year","time")
rownames(L.olympics)=1:nrow(L.olympics)
data = L.olympics

# Write the function
my_2NN = function(x.train,x.test,y){
  # initializing predictions
  predictions = c()
  # Loops to find position of test data and compute the predictions
  for (i in c(1:length(x.test))){
    for (j in c(1:length(x.train))){
      # condition set to find the position of testing data
      if ((x.test[i]>x.train[j]) & (x.test[i]<x.train[j+1])){

        # prediction is the average of neighbor years
        # There are two cases:
        # 1. two outcomes are identical (majority vote)
        # 2. two outcomes are different (with same probability = 1/2)
        pred = sum(y[j],y[j+1])/2

        # store the results
        predictions[i] = pred
```

```
      }
    }

  }

  result = data.frame(year = x.test, predictions = predictions)

  return(result)
}
```

The 2-NN algorithms can be explained as 2 steps:

1. Find the two neighbor years, i.e. the closest year before the target year and the following year of the target year. This procedure is identical to finding the position of the target year in the sequence of Olympic years.
2. Compute the average number of the neighbor years' outcome. There are two cases for the outcomes of the neighbor years:
   - first, they are identical, then the prediction is identical to the outcomes due to majority vote;
   - second, the outcomes are different, then the prediction is the average since the probability of each outcome is equal to 0.5.

   Thus, the computation of predictions is equivalent to computing the average of neighbor years' outcome.

Question 2

Use the data for the years 1906,1932,1968,2000 as testing data (the rest are training data) and compute the test-RSS for your 2-NN regression algorithm.

```
# Define training and testing dataset
train <- L.olympics[-c(4, 10, 17, 24), ]
test  <- L.olympics[c(4, 10, 17, 24), ]

# subtract features and outcomes
x.train = train[1:nrow(train),1]
x.test = test[1:nrow(test),1]
y = train[1:nrow(train),2]

# predict use own function
pred_2NN = my_2NN(x.train,x.test,y)

# compute the test-RSS
errors = test[1:nrow(test),2] - pred_2NN[1:nrow(test),2]
RSS = sum(errors**2)

# result

# predictions for testing data
print(pred_2NN)
```

```
##    year predictions
## 4  1906      10.900
## 10 1932      10.550
## 17 1968      10.070
## 24 2000       9.845
```

```
# Test RSS
print(RSS)
```

```
## [1] 0.167525
```