Machine Learning Lab 01 (Special)

Maximilian Pfundstein (maxpf364)

24 November 2018

Contents

Special Task 1

a) Implementation

This section includes the function with the k-nearest-neighbour implementation.

```
knearest = function(data, K = 5, newdata) {

  # Internal Function to calculate D
  d = function(X, Y) {

    # Make sure the input in the matrix format
    X = as.matrix(X)
    Y = as.matrix(Y)

    # Calculate D as described in the exercise
    X_hat = X/sqrt(rowSums(X^2))
    Y_hat = Y/sqrt(rowSums(Y^2))
    C = X_hat %*% t(Y_hat)
    D = 1 - C

    return(D)
  }

  # Get D (trim classification column)
  D = d(data[,-49], newdata[-49])

  classificationRate = c()
  classification = c()
  classifiedCorrectly = c()

  # For each data entry
  for (i in 1:nrow(D)) {
    # Sort the distances and also get their index
    sortedRow = sort(D[,i], index.return = TRUE)
```

```r
    # Take the K best guys and save their indexed
    indexesKnn = sortedRow$ix[1:K]

    # Lookup if they're classified as 0 or 1
    classificationRates = data$Spam[indexesKnn]

    # Add the classification
    classificationRate = c(classificationRate,
                           (sum(data$Spam[indexesKnn] / K)))
    temp_classification = sum(data$Spam[indexesKnn]) > (K/2)
    classification = c(classification, temp_classification)
    classifiedCorrectly =
      c(classifiedCorrectly, temp_classification != as.logical(newdata$Spam[i]))
  }

  returnDataFrame = cbind(newdata, classificationRate)
  returnDataFrame = cbind(returnDataFrame, classification)
  returnDataFrame = cbind(returnDataFrame, classifiedCorrectly)

  return(returnDataFrame)

}
```

b) Classification

In the following the misqualification rates for training and test are shown. The function adds three columns to the data.frame, *classificationRate*, *classification* and *classifiedCorrectly*:

- *classificationRate*: Shows the percentage of neighbours that got classified as valid mail.
- *classification*: Simply checks if $classificationRate > 0.5$ to show the classification.
- *classifiedCorrectly*: Compares the original y and predicted y and tells, if it got classified correctly.

Misqualification rate for the training data:

```
## [1] 0.2627737
```

Table 1: Head of Training Classification

Spam	classificationRate	classification	classifiedCorrectly
1	0.2666667	FALSE	TRUE
1	0.2333333	FALSE	TRUE
1	0.6000000	TRUE	FALSE
0	0.5000000	FALSE	FALSE
0	0.0000000	FALSE	FALSE
0	0.2333333	FALSE	FALSE

Misqualification rate for the test data:

```
## [1] 0.3094891
```

Table 2: Head of Test Classification

Spam	classificationRate	classification	classifiedCorrectly
0	0.2333333	FALSE	FALSE
1	0.3000000	FALSE	TRUE
0	0.0333333	FALSE	FALSE
0	0.0000000	FALSE	FALSE
0	0.1666667	FALSE	FALSE
1	0.2666667	FALSE	TRUE

In 1.4) the misclassification rate for the training data set is 17.226% and for the test data set it's 32.993%.

We can see that the custom implementation based on the cosine similarity performs better on the training data compared the the R implemented function. The gap closes when we compare the classification rates on the test dataset, the custom implementation is a little bit worse. This gives the impression, that the custom implementation based on the cosine similarity slightly overfits compared to 1.4.

Special Task 2

This is the function for the density estimation at point X, K and a given dataset (vector).

```
density_estimation = function(data, K = 6, X) {

  N = length(data)
  S = data

  # V needs to be calculated, first get all distances
  distances = abs(X - S)

  # Sort them the get the K nearest and get their indexes
  sorted_distances = sort(distances, index.return = TRUE)
  idx_knearest = sorted_distances$ix[1:K]

  # We take the point which is furthest away and take it as the "radius" to get
  # the V (linear)
  V = 2 * max(abs(data[idx_knearest[K]] - X))

  # Returns based on formula from slides
  return(K/(N*V))
}
```

Shown is the requested plot, the yellow line shows the density estimation function with 1000 points between min and max of `cars$speed`. In addition to that the histogram is shown.

Important Note: The density estimation itself is straight forward as it basically is just the calculation of $K/(N * V)$. However the correct calculation of V is not that clear. The slides `Lecture 1a.pdf` state that Δ is the "length of the interval containing K neighbors". This statement is correct, but doesn't describe unambiguously the correct calculation. The following questions are left unanswered:

- Does it have to be the *minimal* length that barely captures all K neighbours?
- If not, how is it aligned? Means, is it aligned to the closest or furthest neighbor and on which site is it aligned, left or right?
- Is it the distance from the furthest to the closest neighbor?

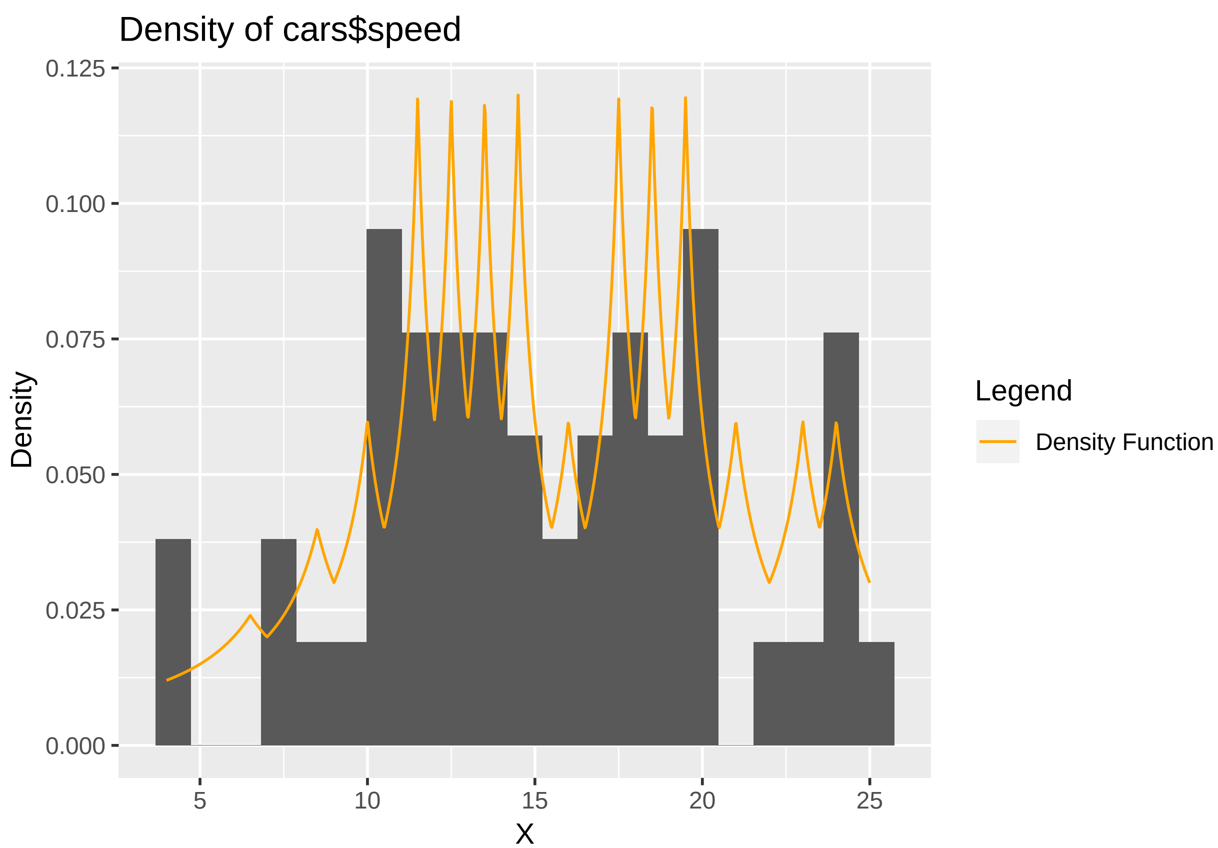
Therefore I looked this topic up in (Bishop 2006) chapter *2.5.2 Nearest-neighbour methods*. There it says:

We therefore return to our general result (2.246) for local density estimation, and instead of fixing V and determining the value of K from the data, we consider a fixed value of K and use the data to find an appropriate value for V. To do this, we consider a small sphere centred on the point x at which we wish to estimate the density p(x), and we allow the radius of the sphere to grow until it contains precisely K data points. The estimate of the density p(x) is then given by (2.246) with V set to the volume of the resulting sphere. This technique is known asK nearest neighbours and is illustrated in Figure 2.26, for various choices of the parameter K, using the same data set as used in Figure 2.24 and Figure 2.25. We see that the value of K now governs the degree of smoothing and that again there is an optimum choice for K that is neither too large nor too small. Note that the model produced by K nearest 2.61 neighbours is not a true density model because the integral over all space diverges.

If we project the sphere into one dimensional space we have a line. The length of the line is the radius times two, as is goes into both directions. The radius is the smallest radius that captures K neighbors. So we take the neighbor, that is farthest away and double the distance to get Δ. This statement matches the statements made on the slides following the above mentioned one. It also states, that this model is **not** a true density model.

Adding this makes the density function more curvy and not that straight as seen on the slides. **End of Note.**

It can be seen, that the estimated density function has some errors in density, but is correlated to the histogram. The sum of density is higher than allowed for a desnity function (>1 in total which is possible as it is just an esimation and not a real density function).



Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
library(readxl)
library(ggplot2)
library(knitr)
set.seed(12345)

knearest = function(data, K = 5, newdata) {

  # Internal Function to calculate D
  d = function(X, Y) {

    # Make sure the input in the matrix format
    X = as.matrix(X)
    Y = as.matrix(Y)

    # Calculate D as described in the exercise
    X_hat = X/sqrt(rowSums(X^2))
    Y_hat = Y/sqrt(rowSums(Y^2))
    C = X_hat %*% t(Y_hat)
    D = 1 - C

    return(D)
  }

  # Get D (trim classification column)
  D = d(data[,-49], newdata[-49])

  classificationRate = c()
  classification = c()
  classifiedCorrectly = c()

  # For each data entry
  for (i in 1:nrow(D)) {
    # Sort the distances and also get their index
    sortedRow = sort(D[,i], index.return = TRUE)

    # Take the K best guys and save their indexed
    indexesKnn = sortedRow$ix[1:K]

    # Lookup if they're classified as 0 or 1
    classificationRates = data$Spam[indexesKnn]

    # Add the classification
    classificationRate = c(classificationRate,
                           (sum(data$Spam[indexesKnn] / K)))
    temp_classification = sum(data$Spam[indexesKnn]) > (K/2)
    classification = c(classification, temp_classification)
    classifiedCorrectly =
      c(classifiedCorrectly, temp_classification != as.logical(newdata$Spam[i]))
  }
```

```
  returnDataFrame = cbind(newdata, classificationRate)
  returnDataFrame = cbind(returnDataFrame, classification)
  returnDataFrame = cbind(returnDataFrame, classifiedCorrectly)

  return(returnDataFrame)

}

# Set seed and import data from excel
set.seed(12345)
spambase = read_excel("spambase.xlsx")

# Shuffle the data
n = dim(spambase)[1]
id = sample(1:n, floor(n*0.5))
c_data = spambase[id,]
c_newdata = spambase[-id,]

# Get the missqualification rates for training and  test
training_classification = knearest(c_data, K = 30, c_data)
test_classification = knearest(c_data, K = 30, c_newdata)

# Get the classification rate
training_rate = sum(training_classification$classifiedCorrectly/
  nrow(training_classification))
test_rate = sum(test_classification$classifiedCorrectly/
  nrow(test_classification))


print(training_rate)
kable(head(training_classification[,49:52]), caption = "Head of Training Classification")


print(test_rate)
kable(head(test_classification[,49:52]), caption = "Head of Test Classification")


density_estimation = function(data, K = 6, X) {

  N = length(data)
  S = data

  # V needs to be calculated, first get all distances
  distances = abs(X - S)

  # Sort them the get the K nearest and get their indexes
  sorted_distances = sort(distances, index.return = TRUE)
  idx_knearest = sorted_distances$ix[1:K]

  # We take the point which is furthest away and take it as the "radius" to get
  # the V (linear)
  V = 2 * max(abs(data[idx_knearest[K]] - X))
```

```r
  # Returns based on formula from slides
  return(K/(N*V))
}


# Get the min and max from the dataset and define a stepsize
min_speed = min(cars$speed)
max_speed = max(cars$speed)
steps = 1000

# X-Values are a sequence from min_speed to max_speed
x_values =   seq(min_speed, max_speed, by = (max_speed - min_speed)/steps)

# Y-Values are the density estimation at each point
y_values = unlist(lapply(x_values,
                         function(x) density_estimation(cars$speed, K = 6, x)))

# Put them into a dataframe
density_data_frame = data.frame(x_values, y_values)
colnames(density_data_frame) = c("X", "Density")

# Plot the data
print(ggplot(density_data_frame) +
  geom_histogram(data = cars, bins = 21) + aes(x = speed, y = ..density..) +
  geom_line(aes(x = X, y = Density, colour = "Density Function")) +
  labs(title="Density of cars$speed", y="Density", x="X", color = "Legend") +
  scale_color_manual(values = c("orange")))
```

Bibliography

Bishop, Christopher M. 2006. "Pattern Recognition and Machine Learning." Springer Science; Business Media, LLC.