K-Nearest Neighbours - Practical

In the last seminar we talked about how to implement K-nearest-neighbours manually. As with linear regression, when we have a small number of instances this can be calculated manually. However, this becomes time-consuming when dealing with large numbers of data. So what we will do today, is implement k-nearest-neighbours using an R package.

1. Install and Load Necessary Packages

Before starting, make sure you have R installed on your system. You'll also need to install and load the class package, which provides the knn() function for implementing the k-NN algorithm.

```
# Install the 'class' package if not already installed
# install.packages("class")

# Load the 'class' package
library(class)
```

2. Prepare Your Data

For this tutorial, we will import a dataset. On moodle, you will see a dataset called cancer.csv. You need to load this dataset into R and prepare it for training and testing.

```
# Load the dataset
data <- read.csv("C:/Users/joe/Downloads/breast+cancer+wisconsin+diagnostic/wdbc.data", he

# View the structure of the dataset
str(data)
```

```
'data.frame':   569 obs. of  32 variables:
 $ V1 : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844981 84501001
 $ V2 : chr  "M" "M" "M" "M" ...
 $ V3 : num  18 20.6 19.7 11.4 20.3 ...
 $ V4 : num  10.4 17.8 21.2 20.4 14.3 ...
 $ V5 : num  122.8 132.9 130 77.6 135.1 ...
 $ V6 : num  1001 1326 1203 386 1297 ...
 $ V7 : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ V8 : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
 $ V9 : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
 $ V10: num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
 $ V11: num  0.242 0.181 0.207 0.26 0.181 ...
 $ V12: num  0.0787 0.0567 0.06 0.0974 0.0588 ...
 $ V13: num  1.095 0.543 0.746 0.496 0.757 ...
 $ V14: num  0.905 0.734 0.787 1.156 0.781 ...
 $ V15: num  8.59 3.4 4.58 3.44 5.44 ...
 $ V16: num  153.4 74.1 94 27.2 94.4 ...
 $ V17: num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
 $ V18: num  0.049 0.0131 0.0401 0.0746 0.0246 ...
 $ V19: num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
 $ V20: num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
 $ V21: num  0.03 0.0139 0.0225 0.0596 0.0176 ...
 $ V22: num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
 $ V23: num  25.4 25 23.6 14.9 22.5 ...
 $ V24: num  17.3 23.4 25.5 26.5 16.7 ...
 $ V25: num  184.6 158.8 152.5 98.9 152.2 ...
 $ V26: num  2019 1956 1709 568 1575 ...
 $ V27: num  0.162 0.124 0.144 0.21 0.137 ...
 $ V28: num  0.666 0.187 0.424 0.866 0.205 ...
 $ V29: num  0.712 0.242 0.45 0.687 0.4 ...
 $ V30: num  0.265 0.186 0.243 0.258 0.163 ...
 $ V31: num  0.46 0.275 0.361 0.664 0.236 ...
 $ V32: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

Make sure your dataset contains both the predictor variables (features) and the target variable (the variable you want to predict).

3. Split the Data into Training and Testing Sets

Before applying the k-NN algorithm, it's essential to split your data into training and testing sets. This helps evaluate the model's performance.

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 70% training and 30% testing
train_index <- sample(1:nrow(data), 0.7 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]
```

4. Train the k-NN Model

Now, you can train the k-NN model using the knn() function from the class package.

```r
# Train the k-NN model
k <- 5  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
```

Make sure to replace predictors with the names of the predictor variables in your dataset.

5. Evaluate the Model

After training the model, you should evaluate its performance using appropriate metrics such as accuracy, precision, recall, or F1 score. We will talk about these more in our next seminar. For now, we will just use accuracy as a measure for evaluating our model.

Accuracy: Accuracy is the most straightforward metric, measuring the ratio of correctly predicted instances to the total number of instances. It is suitable when classes are balanced, but it may not be informative for imbalanced datasets.

```r
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 94.15205 %
```

6. Fine-Tune the Model (Optional)

You can fine-tune the model by experimenting with different values of k or by selecting different features (as in linear regression)

Conclusion

In this tutorial, you learned how to implement the k-Nearest Neighbors algorithm in R using the class package. Remember to preprocess your data, split it into training and testing sets (this has already been done for you for this dataset.), train the model, evaluate its performance, and fine-tune it as needed.

Practical Exercises

For these exercises, put your code into an R markdown document.

1. Fill in the following chart

k	accuracy
1	
2	
3	
4	
5	
9	
15	
21	
26	
31	
41	
51	

Describe what you see? Why do you think this is the case? Find the k, with the best accuracy. Plot all values of K in a graph.

```
# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.22807 %
```

```r
# Train the 2-NN model
k <- 2  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 92.39766 %
```

```r
# Train the 3-NN model
k <- 3  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 94.15205 %
```

```r
# Train the 5-NN model
k <- 5  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 94.15205 %
```

```r
# Train the 9-NN model
k <- 9  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 95.90643 %
```

```
# Train the 15-NN model
k <- 15  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 95.32164 %
```

```
# Train the 21-NN model
k <- 21  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 94.73684 %
```

```
# Train the 26-NN model
k <- 26  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 94.73684 %
```

```
# Train the 31-NN model
k <- 31  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
```

```r
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 92.39766 %
```

```r
# Train the 41-NN model
k <- 41  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.81287 %
```

```r
# Train the 51-NN model
k <- 51  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 90.64327 %
```

2. Fill in the following chart. Run knn with the value you found above but change the train/test values.

Train	Test	Accuracy
10	90	
20	80	
30	70	
40	60	
50	50	
60	40	
70	30	
80	20	

Train	Test	Accuracy
90	10	
95	5	
99	1	

```
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 10% training and 90% testing
train_index <- sample(1:nrow(data), 0.1 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.423 %
```

```
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 20% training and 80% testing
train_index <- sample(1:nrow(data), 0.1 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.423 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 30% training and 70% testing
train_index <- sample(1:nrow(data), 0.3 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.22807 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 40% training and 60% testing
train_index <- sample(1:nrow(data), 0.4 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.52047 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 50% training and 50% testing
train_index <- sample(1:nrow(data), 0.5 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.22807 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 60% training and 40% testing
train_index <- sample(1:nrow(data), 0.6 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.66667 %
```

```r
# Set a random seed for reproducibility
set.seed(123)
```

```r
# Split the data into 70% training and 30% testing
train_index <- sample(1:nrow(data), 0.7 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 91.22807 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 80% training and 20% testing
train_index <- sample(1:nrow(data), 0.8 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 92.10526 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 90% training and 10% testing
train_index <- sample(1:nrow(data), 0.9 * nrow(data))
train_data <- data[train_index, ]
```

```r
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 96.49123 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 95% training and 5% testing
train_index <- sample(1:nrow(data), 0.95 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 93.10345 %
```

```r
# Set a random seed for reproducibility
set.seed(123)

# Split the data into 99% training and 1% testing
train_index <- sample(1:nrow(data), 0.99 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Train the 1-NN model
```

```
k <- 1  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:32], test = test_data[, 3:32],
             cl = train_data$V2, k = k)
# Evaluate the model
accuracy <- mean(model == test_data$V2) * 100
cat("Accuracy:", accuracy, "%\n")
```

```
Accuracy: 100 %
```

3. Pick some combinations of variables in turn (V3 to V32) and run knn with the value you found above. For example, below I have selected columns 3-6. Refer to Week 1, subsetting for help with this. Find the best accuracy you can.

For example the code below runs the model for V3, V4, V5 and V6.

```
# Train the k-NN model
k <- 5  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, 3:6], test = test_data[, 3:6],
             cl = train_data$V2, k = k)
```

The code below runs the model for V4, V6, V9

```
# Train the k-NN model
k <- 5  # Specify the number of neighbors (you can choose any value)
model <- knn(train = train_data[, c(4, 6, 9)], test = test_data[, c(4, 6, 9)],
             cl = train_data$V2, k = k)
```