**INTRODUCTION TO MACHINE LEARNING AND K NEAREST NEIGHBOURS**

**by Simon Moss**

|  |
| --- |
| **Introduction** |

**Classification and regression**

K nearest neighbours, or KNN, is a simple algorithm that can be used to classify objects, such as people, or to predict outcomes. To illustrate, the following table outlines some research questions that KNN can address

|  |  |
| --- | --- |
| Research question | Type of research question |
| From information about the previous books people have enjoyed, can we identify which books these individuals are likely to enjoy in the future | Classification |
| From information about their previous grades and interests, can we predict which research candidates are likely to enjoy statistics | Classification |
| From information about their previous grades and interests, can we predict when research candidates will submit their thesis | Regression |

In the previous table, some of the research questions are designed to classify individuals into categories, such as whether or not they enjoy statistics. This goal is called classification. Other research questions are designed to predict some numerical outcome, such as the number of months candidates need before they submit their thesis. This goal is called regression. KNN can be used to support both classification and regression.

**Machine learning**

KNN is a very simple example of machine learning—in particular a kind called supervisory learning. Consequently, this document not only outlines KNN but also introduces the fundamentals of machine learning. If you have already developed basic knowledge about machine learning, you can disregard the sections about machine learning.

|  |
| --- |
| **Introduction to machine learning** |

So, what exactly is machine learning? Scholars have yet to agree on a single definition of machine learning. But, in general, machine learning entails the following features

* machine learning comprises various sequences of computer operations, called algorithms, that tend to improve automatically—without human intervention—over time.
* to develop these algorithms, the computer often utilises a subset of data, called training data, to develop a model—a model that represents or summarises the data as closely as possible
* then, the computer uses the remaining data, called testing data, to evaluate this model.

|  |
| --- |
| **Introduction to machine learning: Cross validation** |

Besides KNN, scholars have developed hundreds of models or algorithms to enable machines to learn, to classify, and to predict outcomes. Common examples include

* neural networks
* decision trees and random forests
* support vector machines
* Bayesian networks
* deep graphs, and
* genetic algorithms.

But, how can researchers decide which models to utilise? How can researchers decide, for example, whether to use KNN, decision trees, or other approaches? One answer to this question revolves around cross-validation. Cross validation is a technique that can be used to decide which model classifies or predicts outcomes most effectively.

**Example**

Imagine a researcher who wants to develop an algorithm or app that can predict which research candidates are likely to complete their thesis on time. Specifically, the researcher collates information on 1000 candidates who had enrolled at least 8 years ago and thus should have completed their thesis. An extract of data appears in the following table. Each row corresponds to one individual. The columns represent

* whether the candidates completed their thesis on time
* the grade point average of this candidate during their undergraduate studies—as a percentage of the maximum GPA
* age
* IQ, and
* EQ—or emotional intelligence—as measured by a battery of tests on a 10 point scale

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Complete on time | GPA | Age | IQ | EQ |
| Yes | 85% | 34 | 113 | 7 |
| No | 90% | 27 | 104 | 6 |
| No | 70% | 56 | 142 | 8 |
| Yes | 65% | 71 | 107 | 4 |
| … | … | … | … |  |

**Training data**

The researcher could then subject these data to one of several machine learning methods, such as KNN. In particular, the researcher could

* randomly select 75% of the candidates; the characteristics of these candidates will be called the training data
* these training data could then be subjected to the machine learning method, such as KNN, to develop a model—a model that represents the data as closely as possible
* the following figure schematically depicts this sequence of activities in which the training data are subjected to a machine learning method, such as KNN, to create a model



**Testing data**

To evaluate this model, the researcher then subjects the remaining data, called the testing data, to this model. In particular, the researcher can assess how often the model classified or predicted the testing data correctly. In this example, the model will predict whether each candidate is likely to complete the thesis on time or not. The following figure schematically depicts how the training data are subjected to the model to derive predictions.



**Testing these predictions and the confusion matrix**

To assess whether this model is accurate, the researcher then needs to compare the predicted outcomes with the actual outcomes. To illustrate, the following table displays an extract of both the original data and the predicted completion on time. As this table shows

* the first individual, in the first column, actually completed on time and was also predicted to complete on time; hence, in this instance, the model was correct
* the second individual did not actually complete on time and was also predicted not to complete on time; again, in this instance, the model was correct
* however, the third individual did not actually complete on time but was predicted to complete on time; therefore, in this instance, the model was incorrect

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Actually  completed on time | GPA | Age | IQ | EQ | Predicted completion on time |
| Yes | 85% | 34 | 113 | 7 | Yes |
| No | 90% | 27 | 104 | 6 | No |
| No | 70% | 56 | 142 | 8 | Yes |
| Yes | 65% | 71 | 107 | 4 | No |
| … | … | … | … |  | … |

The actual and predicted outcomes can be summarised in a table, sometimes called a confusion matrix or matching matrix. The following table illustrates this matrix. According to this matrix

* 420 of the candidates who were predicted to complete actually completed; these predictions were correct or true
* 130 of the candidates who were predicted to complete did not actually complete; these predictions were thus false
* 100 of the candidates who were predicted not to complete actually completed; these predictions were thus false as well
* 450 of the candidates who were predicted not to complete did not actually complete; these predictions were correct or true.

|  |  |  |
| --- | --- | --- |
|  | Actually completed | Actually did not complete |
| Predicted to complete | 420 true positives | 130 false positives |
| Predicted not to complete | 100 false negatives | 450 true negatives |

In this instance, the outcome of interest was completion of the thesis. Thus, we refer to completion as positives. Hence

* the 420 individuals who were predicted to complete and actually completed are called true positives
* in contrast, the 130 individuals who were predicted to complete but did not actually complete are called false positives: the model falsely predicted these individuals would be positives—that is, they would complete.

But, how should you interpret these numbers. In this instance

* we can calculate the percentage of accurate predictions
* in this instance, 870 of the 1000 predictions were accurate; 230 of the 1000 predictions were inaccurate
* thus, 87% of the predictions were accurate.

But, is 87% high? What conclusions can we derive from these numbers? Actually, we cannot really derive valid information from these numbers alone. Instead, we need to compare the confusion matrices that various machine learning methods would have generated. The following three tables present the confusion matrices that three separate methods would have generated. If you examine these tables closely, you will conclude that

* KNN is appreciably more accurate than support vector machines in this example
* yet, KNN and random forests seem to be almost equally accurate
* this finding shows that KNN or random forests, rather than support vector machines, should be used to predict which candidates will complete their thesis

|  |  |  |
| --- | --- | --- |
| KNN  87% accurate | Actually completed | Actually did not complete |
| Predicted to complete | 420 true positives | 130 false positives |
| Predicted not to complete | 100 false negatives | 450 true negatives |

|  |  |  |
| --- | --- | --- |
| Support vector machine  47% accurate | Actually completed | Actually did not complete |
| Predicted to complete | 220 true positives | 330 false positives |
| Predicted not to complete | 300 false negatives | 250 true negatives |

|  |  |  |
| --- | --- | --- |
| Random forest  86% accurate | Actually completed | Actually did not complete |
| Predicted to complete | 400 true positives | 120 false positives |
| Predicted not to complete | 120 false negatives | 460 true negatives |

**Sensitivity versus specificity: two outcomes**

In the previous section, the accuracy of KNN and random forests was similar. So, how can researchers decide which of these two methods to utilise? That is, what other metrics could researchers calculate to guide this decision? Two helpful metrics are called sensitivity and specificity. To illustrate, consider the following table. To calculate sensitivity, note that

* sensitivity = true positives / (true positives + false negatives)
* in this example, sensitivity = 420 /(420 + 100) = .81
* as this number implies, of the people who were actually positive—who actually completed the thesis—the method correctly identified .81 or 81% of these individuals.

|  |  |  |
| --- | --- | --- |
| KNN | Actually completed | Actually did not complete |
| Predicted to complete | 420 true positives | 130 false positives |
| Predicted not to complete | 100 false negatives | 450 true negatives |

In contrast, to calculate specificity, note that

* specificity = true negatives / (true negatives + false positives)
* in this example, specificity = 450 / (450 + 130) = .78
* as this number implies, of the people who were actually negative—who did not actually complete the thesis—the method correctly identified .78 or 78% of these individuals

After you inspect this example a couple of times, you will realise that…

|  |
| --- |
| * sensitivity, in essence, indicates how well the method detects positive cases |
| * specificity, in essence, indicates how well the method correctly overlooks negative cases |

How can you utilise these metrics? What can you conclude from this information. To illustrate, consider the following table. This table specifies the sensitivity and specificity of two methods: KNN and random forests. As this table shows

* KNN is slightly more sensitive than random forests—and thus detects positive cases better
* random forests is slightly higher in specificity than KNN—and thus overlooks negative cases better.

|  |  |  |
| --- | --- | --- |
|  | Sensitivity | Specificity |
| KNN | 81% | 78% |
| Random forests | 77% | 79% |

After you scan this table, you might be tempted to choose KNN over random forests. That is, relative to random forests, KNN is 4% higher in sensitivity but only 1% lower in specificity. But, this conclusion is premature because

* sometimes, you might be more concerned about specificity than sensitivity
* to illustrate in this example, suppose you incorrectly predict that someone will complete a thesis
* this candidate may be granted a scholarship but might never complete a thesis—squandering huge amounts of money
* thus, you want to guarantee the method will usually overlook or reject negative cases—that is, people who will not complete
* therefore, specificity might be more important than sensitivity in this circumstance
* you might thus choose the random forest instead of KNN.

**Other metrics**

One limitation is that measures of sensitivity and specificity vary across samples and populations. If you repeated this procedure, but with another population—such as Masters of Coursework candidates instead of research candidates

* the results might differ appreciably
* your sensitivity and specificity values might be significantly greater or smaller.

The reason is that sensitivity and specificity measures depend appreciably on the proportion of positive cases—in this instance, the percentage of research candidates who completed their thesis. If you shifted to a population in which completion is lower or higher, the sensitivity and specificity values might change dramatically. Therefore, researchers instead often calculate two other metrics the positive likelihood ratio and the negative likelihood ratio value. To illustrate

* the positive likelihood ratio = sensitivity / (1 - specificity)
* for KNN therefore, the positive likelihood ratio = .81 / .22 = 3.68.

So, how do you interpret this positive likelihood ratio of 3.68? This 3.68 indicates that positive predictions are 3.68 times more likely to be positive cases than negative cases. In this example, predictions that someone will complete indicate the person is 3.68 times as likely to actually complete than not complete. Conversely

* the negative likelihood ratio = (1-sensitivity) / specificity
* for KNN therefore, the positive likelihood ratio = .19 / .78 = .24.

So, how do you interpret this negative likelihood ratio of .24? This .24 indicates negative predictions are .24 times as likely to be positive cases than negative cases. In this example, predictions that someone will not complete indicate the person is .24 times as likely to complete than not complete. At first glance, this interpretation might seem confusing. But, in essence,

|  |
| --- |
| * positive likelihood ratios represent the extent to which positive predictions indicate the outcome will be positive |
| * negative likelihood rations represent the extent to which negative predictions indicate the outcome will be positive |

Beside likelihood ratios, researchers sometimes compute more flexible indices, such as area under the curve. But, these indices are not discussed in this document.

**Sensitivity versus specificity: more than two outcomes**

The previous examples have revolved around two possible outcomes: candidates could either complete or not complete their thesis. But sometimes, the number of outcomes or classifications exceeds two. For example

* the researcher might want to predict whether candidates will complete on time, complete late, or never complete
* the following table illustrates the confusion matrix this circumstance might elicit.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Actually  completed on time | Actually  completed late | Actually  did not complete |
| Predicted to  complete on time | 220 | 78 | 19 |
| Predicted to  complete late | 85 | 104 | 65 |
| Predicted not to complete | 13 | 69 | 187 |

How can you calculate sensitivity, specificity, and likelihood ratios in this circumstance? Do you need entirely different formulas? No. You can actually use the same formula but

* you examine each outcome separately
* for each outcome, you collapse the other outcomes, so the matrix again comprises two rows and two columns of numbers

To illustrate, you might first examine the outcome *completed on time*. Therefore, you would collapse the other outcomes. That is, you would designate *completed late* and *did not complete* as one outcome or classification, generating the following table.

|  |  |  |
| --- | --- | --- |
|  | Actually  completed on time | Actually  completed late or on time |
| Predicted to complete on time | 220 | 97 |
| Predicted to complete late or not at all | 98 | 425 |

Once you construct a matrix that comprises two rows and two columns of data, you can

* calculate the sensitivity and specificity of completing on time, using the same formulas as before
* repeat with the two other outcomes: complete late and not complete at all—to generate three sets of sensitivity and specificity values

**How to choose the testing sample**

In the previous example, 75% of the data were utilised to develop the model, and 25% of the data were utilised to test the model. One question, however, is which 25% of the data should be used to test the model. For example, if the dataset comprised 100 rows of data, such as 100 individuals

* researchers could use the first 25 rows to test the data
* or researchers could use rows 26 to 50 to test the data
* or researchers could use rows 51 to 75 to test the data
* or finally researchers could use rows 76 to 100 to test the data

In practice, researchers sometimes apply all four approaches and then average the results. This procedure is called a four-fold cross validation. Alternatively

* rather than test 25% of cases at a time, researchers often test 10% of cases at a time and repeat this procedure 10 times—called a ten-fold cross validation
* many other practices can also be adopted.

|  |
| --- |
| **K nearest neighbours: The underlying rationale** |

You are now ready to learn about K nearest neighbours or KNN. KNN is a simple algorithm that can be used to classify individuals or to predict outcomes. To illustrate, consider the following scatterplot. On this scatterplot

* the y axis represents IQ
* the x axis represents EQ
* each circle represents one candidate
* the green circles represent candidates who completed their thesis; the red circles represent candidates who did not complete their thesis

As this figure shows, the red circles, the candidates who did not complete their thesis, tend to coincide with low EQ. The green circles, the candidates who completed their thesis, tend to coincide with high EQ.



Now suppose you want to predict whether a candidate who has just started will complete the thesis. How would you classify the candidate who appears in the black circle in the following scatterplot? Would you classify the person as red, and thus unlikely to complete, or green, and thus likely to complete, based on the EQ and IQ of this individual.



To reach this decision, the KNN, or K nearest neighbours, algorithm simply determines which class or outcome is closest to this individual. To illustrate,

* if the researcher sets K to 1, the algorithm will identify the one data point that is closest to this individual
* in this instance, as revealed in the following scatterplot, a green circle is closest to the candidate who corresponds to the black circle
* so, this candidate is classified as someone who will complete the thesis



In contrast, if the researcher sets K to 5, the algorithm will identify the five data points that are closest to this individual. In this instance, as revealed in the following scatterplot, the closest five data points include two green circles and three red circles. Red is more common. So, the candidate should be classified as red—as someone who will not complete the thesis.



Consequently, one of the complications with KNN, is the classification will often depend on the value of K. So, what value of K should you utilise? How can you decide which value to use? The answer is that

* no one single value is appropriate
* but researchers tend to choose a value that equals the square root of the number of rows or participants in the training data.
* for example, if the training sample comprised 25 candidates, K would be set to 5

Lower values of K are too sensitive to outliers. Higher values of K often disregard rare categories.

**Number of variables**

The previous example comprised only two variables: IQ and EQ. If the data comprised three variables, you would need to draw a 3D graph, but the same principle would apply. If the data comprised more than four variables, you would need to draw a 4D graph. Obviously

* you cannot actually draw a 4D graph
* but the same principle applies: you want to identify the circles that are closest to your new individual
* but, to calculate distance, you cannot actually use a ruler; but, instead, you use the formula that is like measuring distance with a ruler—but somehow measures this distance in more than three dimensions.

So, can you measure distance in more than three dimensions? If so, how? You can actually use a variety of formulas, such as a measure called Euclidean distance. To illustrate this measure, consider the following graph. Suppose you wanted to measure the distance between the two points at the start and end of the broken arrow. The first point is located at 5.5 and 75—corresponding to the EQ and IQ of this person. The second point is located at 6.0 and 100.



To calculate the Euclidean distance

* first compute the difference between these points on each variable; that is, the difference between 6.0 and 5.5 is 0.5; the difference between 100 and 75 is 25
* now square these differences, generating 0.25 and 625 respectively
* then sum these numbers, to generate 625.25
* finally, square root this answer; the answer, 25.005, is called the Euclidean distance between these points.

The same formula can be applied if your data comprised four variables. That is, the computer could still

* calculate the difference between the two points on each variable
* square these differences and sum the answers
* square root this answer
* use this formula to identify the closest points.

In other words, although the example referred to only two variables, the same principles can apply when the data comprises three, four, or even more variables.

|  |
| --- |
| **K nearest neighbours. Step 1: Install and use R** |

You can use a variety of statistical packages to apply this technique. This document will show you how to conduct this technique in software called R. If you have not used R before, you can download and install this software at no cost. To achieve this goal

* visit <https://www.cdu1prdweb1.cdu.edu.au/files/2020-08/Introduction%20to%20R.docx> to download an introduction to R
* read the section called Download R and R studio
* although not essential, you could also skim a few of the other sections of this document to familiarize yourself with R.

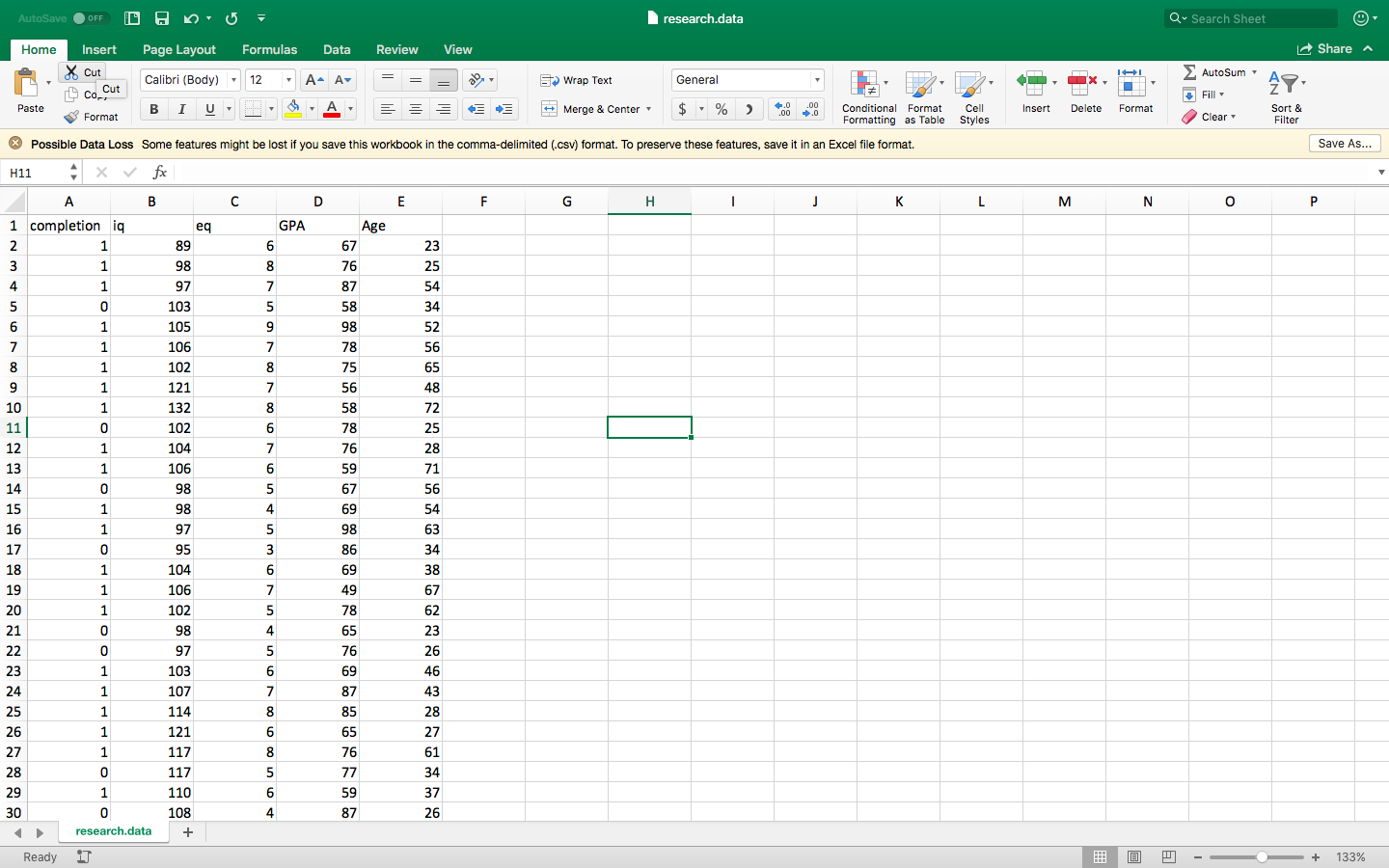
|  |
| --- |
| **K nearest neighbours. Step 2: Upload the data file** |

Your next step is to upload the data into R. To achieve this goal

* open Microsoft Excel
* enter your data into Excel; you might need to copy your data from another format. Or your data might already have been entered into Excel

In particular, as the following example shows

* each column should correspond to one variable
* ensure the first column corresponds to the key outcome—in this instance, whether participants have completed the thesis or not
* each row should correspond to one unit—such as one person, one animal, one specimen, and so forth
* the first row labels the variables
* to prevent complications, use labels that comprise only lowercase letters—although you could end the label with a number, such as age3



To convert this file into a csv file—such as a file called research.data—and then to upload this file into R studio

* visit <https://www.cdu1prdweb1.cdu.edu.au/files/2020-08/Introduction%20to%20R.docx> to download the introduction to R—unless you have already downloaded this document
* read the section called “Upload some data”

|  |
| --- |
| **K nearest neighbours. Step 3: Enter the code and interpret the results** |

To apply KNN, you need to enter some code. The code might resemble the following display. At first glance, this code looks absolutely terrifying. But actually this code is straightforward once explained.

|  |
| --- |
| normalize <- function(x) {  return (x-min(x)/(max(x)-min(x)))  }  research.data.norm <- as.data.frame(lapply(research.data[, 2:5], normalize))  set.seed(123)  training.rows <-sample(1:nrow(research.data.norm), size=nrow(research.data.norm)\*0.7, replace = FALSE)  train.data <- research.data[training.rows, ]  testing.data <- research.data[-training.rows, ]  train.data.outcomes <- research.data[training.rows, 1]  testing.data.outcomes <- research.data[-training.rows, 1]  install.packages("class ")  library(class)  NROW(testing.data.outcomes)  knn.3 <- knn(train= train.data, test=testing.data, cl= train.data.outcomes, k = 3)  knn.3  ACC.3 <-100 \* sum(testing.data.outcomes ==knn.3)/NROW(testing.data.outcomes)  install.packages("caret")  install.packages("e1071")  library(caret)  library(e1071)  confusionMatrix(table(knn.3, testing.data.outcomes)) |

To enter code, you could write one row, called a command, at a time in the Console. But, if you want to enter code more efficiently,

* in R studio, choose the File menu and then “New File” as well as “R script”
* in the file that opens, paste the code that appears in the left column of the following table
* to execute this code, highlight all the instructions and press the “Run” button—a button that appears at the top of this file

You should not change the bold characters in the left column. You might change the other characters, depending on the name of your data file, the number of variables, and so forth. The right column of the following table explains this code. You do not, however, need to understand all the code.

|  |  |
| --- | --- |
| Code to enter | Explanation or clarification |
| **normalize <- function(x) {**  **return (x-min(x)/(max(x)-min(x)))**  **}** | * In the data file, some of the variables, such as IQ, comprise high numbers * Other variables, such as EQ, comprise lower numbers * KNN is more effective whenever all variables correspond to a similar scale or range. * This code merely establishes a function or formula that achieves this goal—called normalisation. |
| research.data.norm **<- as.data.frame(lapply(**research.data[, 2:5], **normalize))** | * This code normalizes all the variables except the first column—the outcome. * research.data[, 2:5] refers to columns 2 to 5 in the data file—all variables except the outcome * the remainder of this code applies the normalization formula to columns 2 to 5 * the results are stored in another data file, called research.data.norm   If you enter *research.data.norm* into the Console, the data file is displayed. This data file will resemble the original data file, except the variables are normed. |
| **set.seed(123)** | * Later, the computer will be asked to identify some random numbers * This code, however, instructs the computer to begin these random numbers at position 123 * Consequently, you could, if you wanted, identify the same random numbers again |
| training.rows **<-sample(1:nrow**(research.data.norm), **size=nrow**(research.data.norm)\*0.7, **replace = FALSE)** | * The command *sample* identifies a series of random integers * Note than *nrow(research.data.norm)* is simply the number of rows or participants in the data file, such as 1000 * Thus 1 : nrow(research.data.norm) merely instructs the computer to randomly distill integers between 1 and 1000 in this example * Similarly, *nrow(research.data.norm)\*0.7* equals 0.7 times the number of rows or participants, such as 700 * Thus size=nrow(research.data.norm)\*0.7 actually instructs the computer to randomly identify 700 or so random numbers * replace = FALSE tells the computer not to repeat these numbers   Ultimately, this convoluted set of codes merely instructs the computer to generate a series of random integers, such as 10 26 13 27 28. The number of random integers equals 70% of the total sample. These integers will be stored in a container called training.rows. To check, simply enter training.rows into the Console. |
| train.data <- research.data[training.rows, ]  testing.data <- research.data[-training.rows, ] | * The first line of code creates the training data * To illustrate, suppose the random numbers, generated in the previous step were 10 26 13 27 28. * This code would extract rows 10 26 13 27 28 from the data file research.data * These rows would be stored in a container called train.data and are hence the training data * The remaining rows are stored in a container called testing.data. In particular, the – before training.rows refers to all the rows that are not 1 0 26 13 27 28 and are hence the test data   If you want to check these containers, simply enter train.data or testing,data into the Console |
| train.data.outcomes <- research.data[training.rows, 1]  testing.data.outcomes <- research.data[-training.rows, 1] | * The first line of code here extracts the first column—the outcomes—from the training data * The other line of code extracts the first column from the test data |
| **install.packages("class ")**  **library(class)** | * *class* refers to a set of formulas or procedures, called a package, that can be used to conduct KNN * install.packages merely installs this package onto the computer * library then activates this package * the quotation marks should perhaps be written in R rather than Word; the reason is that R recognises this simple format— " —but not the more elaborate format that often appears in Word, such as “ or ”. * In addition, sometimes you need to restart R after you install packages; otherwise, you might receive some error messages |
| **NROW**(testing.data.outcomes) | * This code determines the number of rows in your testing data * You should then square root the answer to estimate an appropriate level of K * In the following example, we assume that K is 3—but you might need to utilise a larger value of k in the following code |
| knn.3 <- **knn(train=** train.data, **test=**testing.data**, cl=** train.data.outcomes, **k =** 3) | * This code actually completes the KNN. * In essence, you merely need to specify the name you assigned to the training data, the testing data, the outcomes of your training data, and the level of K * The output is simply the predicted outcomes for each participant in the training data * This output is stored in a container called knn.3 * For example, if you entered knn.3 into your console, you would receive   [1] 1 1 1 1 1 1 1 1 1 1   * This output shows the computer predicts that every participant in the testing data file is predicted to complete the thesis * This prediction is unusual and probably indicates the sample size was too low |
| Proportion.3 **<- 100 \* sum**(testing.data.outcomes **==**knn.3)/**NROW**(testing.data.outcomes) | * This code essentially calculates the percentage of predictions that were correct * The numerator identifies the number of outcomes in the test data—the actual outcomes—that are equivalent to the predictions from KNN * The denominator is the number of rows, or participants, in the test data   If you entered Proportion.3, you would receive a percentage. If the percentage is 60, for example, you would conclude that 60% of the predicted outcomes were correct. |
| **install.packages("caret")**  **install.packages("e1071")**  **library(caret)**  **library(e1071)** | * This code installs and activates other packages that can be used to uncover and analyse the confusion matrix |
| **confusionMatrix(table**(knn.3, testing.data.outcomes)) | * This code presents the confusion matrix and calculates other relevant statistics * For example, in addition to the confusion matrix, this code presents the accuracy—or proportion of correct predictions—as well as the confidence interval of this proportion * The output also presents the sensitivity and specificity   The confusion matrix might initially look confusing. Just note   * The top left hand number is simply the k value * The rest of the first row and column indicate the possible outcomes: 0 and 1 * So, the actual data appears outside the first column and row—and, in this instance, is 0 0 4 and 6   ­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Confusion Matrix and Statistics  testing.data.outcomes  knn. 3 0 1  0 0 0  1 4 6    Accuracy : 0.6  95% CI : (0.2624, 0.8784)  No Information Rate : 0.6  P-Value [Acc > NIR] : 0.6331    Kappa : 0    Mcnemar's Test P-Value : 0.1336    Sensitivity : 0.0  Specificity : 1.0  Pos Pred Value : NaN  Neg Pred Value : 0.6  Prevalence : 0.4  Detection Rate : 0.0  Detection Prevalence : 0.0  Balanced Accuracy : 0.5 |

|  |
| --- |
| **K nearest neighbours: Other considerations** |

After you apply the previous code to implement KNN, you may be interested in some additional clarifications about the algorithm as well as possible variations. This section presents some clarifications and variations.

**Features of KNN**

KNN is a simple but popular algorithm. The following table outlines some of the key features or characteristics of KNN.

|  |  |
| --- | --- |
| Feature | Explanation or clarification |
| KNN is non-parametric | * That is, unlike linear regression KNN does not impose assumptions about the model * For example, KNN does not assume the variables or residuals are normally distributed |
| KNN is what is called a lazy algorithm | * That is, KNN does not actually develop a model from the training data * Instead, KNN uses the training data to generate predictions about each test case * To illustrate, if you return to the previous graphs, the red and green dots were derived from the training data. The black dots represent the test data |
| KNN can be used to both classify individuals, as illustrated in the previous example, or to predict numerical outcomes | * To predict numerical outcomes, KNN uses a similar rationale as shown previously * To illustrate, consider the following graph. * Rather than green circles and red circles, each point is represented by a number—representing the number of months needed to complete a thesis * To predict the outcome of the black circle, the KNN algorithm will then usually average the nearby numbers * Rather than construct a confusion matrix, the researcher could examine the correlation between the predicted and actual outcomes |



**Number of variables**

KNN is especially effective when the number of variables is modest, such as fewer than 20. If the number of variables is excessive, KNN might not be as effective. Instead, you could use another method instead or first reduce the number of variables. To reduce the number of variables, you might

* include only the most important variables
* subject the data to a principal components analysis or factor analysis; these techniques can reduce many variables to 2 to 10 main variables

**Variations**

When researchers implement KNN in R, the software calculates Euclidean distances. Other software may calculate different measures of distance, such as Manhattan, Minkowski, cosine, Hamming, Jaccard, and Mahalanobis. Euclidean distances are sufficient, however, provided the variables are normalised first and numerical rather than categorical or binary.

|  |
| --- |
| **References** |

Ripley, B. D. (1996) Pattern Recognition and Neural Networks. Cambridge.

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.