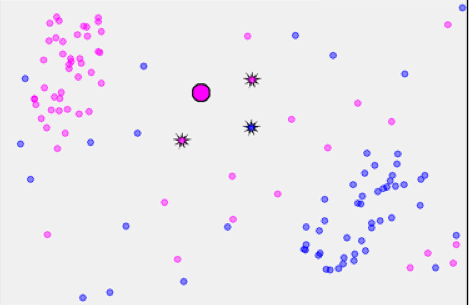
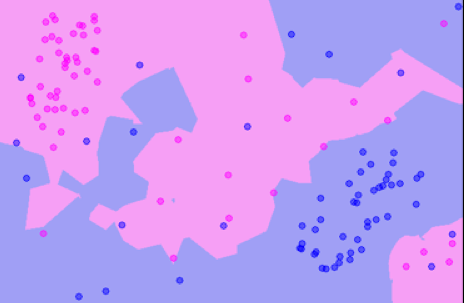
**An Introduction To The k Nearest Neighbour Algorithm:**

David Goody

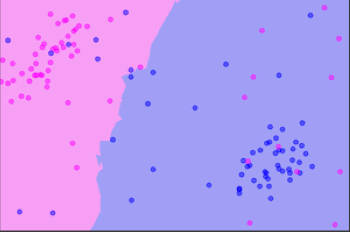
1. The k nearest neighbour algorithm is a way of classifying data points based on their similarity to other points in the data (their nearest neighbour). Imagine we have details of the locations of thousands of schools and which local authority they are in, but we don’t have details of where the local authority boundaries are. If we are given the location of a new school somewhere on the south coast we can estimate which local authority it is based on the local authority of the nearest existing schools.
   1. k=1 approach: Find the nearest existing school our new school. We then assume our new school is in same local authority as this school.
   2. k=3 approach: Find the 3 nearest existing schools to our new school. If 2 or more of these existing are in the same local authority, then assume our new school is in that local authority.
   3. k=10 approach: Find the 10 nearest existing schools to our new school. Work is the most common local authority within these 10 schools (for example 6 might be in Hampshire). Assume our new school is also in that local authority.
2. The local authority is a simplified one as there are neat borders between LAs that mean our schools are nicely clustered. For most clustering analysis our data is messier. In the image below[[1]](#footnote--1) there is a big cluster of purple dots in the top left and a bit cluster of blue dots in the bottom right but in between there is a mixture of purple and blue. If we add a new point (the big circle) we can then look at the three nearest points (k=3). Two of these are purple and one is blue so we assume our new point is purple.



1. By doing the same calculation that we’ve done for an individual point all over the map we can shade in the areas where a k=3 nearest neighbour approach will give an answer of purple and where it will give an answer of blue. Here we see a bit purple area in the top left and a big blue area in the bottom right but a mixture of colours in between.



1. If we’d set a much higher value of k – for example k=17 – then our map becomes much cleaner with a single clear border between purple and blue. This is because individual outlier points (such as blue dots in the top left) have less of an impact as they are outnumbered by the more common purple dots.



1. The algorithm doesn’t need to be limited to the 2 dimensions of a map. You can use as many dimensions as you’d like. For example you could make a crude prediction of an institutions Ofsted grade by taking 5 different measures of school performance and calculating the distance[[2]](#footnote-0) between the school in question and others where we already know their Ofsted grade. When we are working with many variables we can’t visualise our data in the neat way we can for the previous map style example.
2. The k nearest neighbour algorithm is dependent on the user saying how many nearest schools to look at (this “how many” is the k in the title). Depending on which value of k you use, you may get different answers. It is good practice to test your data with differing values of k to get a feel for how sensitive you data is. This can be tested through cross-validation. This is where we pick a subset of our data to “train” our model and then feed in the rest of our data points and see how many the data points our model classifies correctly. Once you’ve done this a number of times you should be able to see which value of k is most accurate.
3. The k nearest neighbour algorithm is well suited to problems where we have a good sized training data set that has already been categorised and is good at excluding ignoring the effects of outliers. The algorithm can be very sensitive to local factors, an issue which can be partly mitigated by using larger values of k or by making small variations to case you are trying to classify to see if you get a different result (sensitivity analysis).
4. Where you have a binary classifier (i.e. you only have two outcomes – say blue dots and purple dots ) it is best to use odd values of k as this will ensure you get an answer. If you picked k=4 you might get 2 blue dots and 2 purple dots in which case the basic algorithm cannot suggest how to classify your new point.
5. The algorithm is useful for identifying similar cases – for example which 10 schools are most similar to this one based on performance table data.
6. Ways in which the basic algorithm can be varied include:
   1. Weighting the contribution of each of the neighbours so the nearest neighbours have a greater contribution that those further away can help to refine the outcome
   2. If your existing dataset has uneven numbers of results in each category this could skew your results. To mitigate for this you can use under-sampling. So with our earlier example if we had 100 purple dots but only 50 blue dots we could get randomly pick just 50 of the purple dots to give us a balanced dataset.
   3. Instead of using the basic Euclidean distance you can use metrics different distance calculations such as Spherical Distance which can be useful for comparisons in multi-dimensional space
   4. Applying this to discrete variables, such as text fields, by calculating the number of differences or similarities (Hamming Distance).
   5. Where you have a large number of dimensions the algorithm can struggle with all points being a similar distance from each other (“the curse of dimensionality”). This can be mitigated by reducing the number of dimensions (variables) using a process such as Principle Component Analysis

**Further Reading**

* <http://www.saedsayad.com/k_nearest_neighbors.htm>
* <http://www.statsoft.com/textbook/k-nearest-neighbors>
* <http://www.math.le.ac.uk/people/ag153/homepage/KNN/OliverKNN_Talk.pdf>
* <http://www.cs.upc.edu/~bejar/apren/docum/trans/03d-algind-knn-eng.pdf>
* <http://classes.engr.oregonstate.edu/eecs/winter2011/cs434/notes/knn-4.pdf>
* <http://blog.webagesolutions.com/archives/1164>

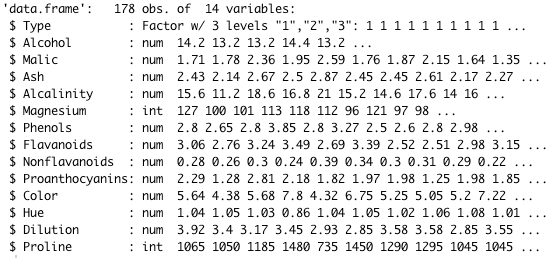
**Annex A – Applying k Means Clustering In R**

1. In this example we will use a dataset which has information on 178 wines. These 178 wines are from three different wine varieties. Here will be see how effective the k-means clustering algorithm is at clustering the 178 wines into these three varieties based on the wines characteristics (level of alcohol, malic acid, magnesium etc).
2. To access the data we need to load the rattle package

|  |
| --- |
| #Load the rattle package to access the wine data  install.packages("rattle")  require(rattle)  data(wine, package="rattle") |

1. Next we have a quick look at our wine dataset to see what we are working with. There are 14 variables. The wine variety (type) is the first value.

|  |
| --- |
| #View the structure of the wine database  str(wine) |



1. We’ll need to standardise the data before we can run the clustering. We keep the wine variety (type) unchanged in column 1 and standardise the data in columns 2 to 14.

|  |
| --- |
| #Standardise the wine data (columns 2-14), keep the wine labels as column 1  stand\_wine <- data.frame(Type=wine$Type,scale(wine[,2:14])) |

1. Next we split our data into two parts. 100 cases will be the ones we train our model on. The remaining 78 will be the ones we test the model on. Refer to manuals on the use of R for more on the commands used here. Since this is a random sample your results may differ from those in the example.

|  |
| --- |
| #Pick 100 cases to sample out of the 178 in our file  sample.cases <- sample(nrow(stand\_wine), 100, replace = FALSE)  #Split our data into the 100 training cases & the 78 remaining test cases  wine\_train <- stand\_wine[sample.cases,]  wine\_test <- stand\_wine[-sample.cases,] |

1. We can then apply our k nearest neighbour function for a single case in our training data. Here we’ll apply it to the 4 case, which should give us a result of the case being in cluster 1.

|  |
| --- |
| #Run the k nearest neighbour (knn) function  #The structure of the function is knn(train, test, cl, k)  #wine\_train[,2:14] are the data points training data (train)  #We are trying to classify case 4 of our (test) dataset - wine\_test[4,2:14]  #We use the same cluster labels (cl) as the Type field in our training dataset  #In this case we set k=1  #Our result should be that the point is in cluster 1  knn(wine\_train[,2:14], wine\_test[4,2:14], wine\_train$Type, 1) |



1. We can do the same for case 50 in the test dataset. This should give us a result of the answer being in cluster 2.

|  |
| --- |
| #Repeat but for case 50 in our training dataset  #Our result should be that the point is in cluster 2  knn(wine\_train[,2:14], wine\_test[50,2:14], wine\_train$Type, 1) |



1. In practice we want to apply run the algorithm for all cases in our data. We do this as follows – saving the result to a new variable which will have the cluster each of our 78 test data points are in.

|  |
| --- |
| #Now run for all our dataset and once and save this to a new variable  wine\_test.knn1 <- knn(wine\_train[,2:14], wine\_test[,2:14], wine\_train$Type, 1) |

1. We can get R to print out these results, but this isn’t very helpful.

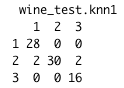
|  |
| --- |
| #See which cluster each of our test cases is in  wine\_test.knn1 |



1. It is better to compare them against the categories they are actually in. This shows us that of the cases where wine type = 2 our algorithm
   1. coded 30 cases accurately as type 2,
   2. coded 2 inaccurately as type 1,
   3. coded 2 inaccurately as type 3,

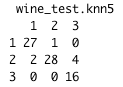
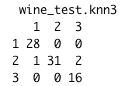
It got all of wine type 1 and wine type 3 right.

|  |
| --- |
| #See how our 3 actual wine types compare with our 3 knn clusters  table(wine\_test$Type, wine\_test.knn1) |



1. We can repeat this for k=3 and k=5 and see how they perform.

|  |
| --- |
| #Re-run for k=5 and k=7 to see how well they would have fitted  wine\_test.knn3 <- knn(wine\_train[,2:14], wine\_test[,2:14], wine\_train$Type, 3)  wine\_test.knn5 <- knn(wine\_train[,2:14], wine\_test[,2:14], wine\_train$Type, 5)  table(wine\_test$Type, wine\_test.knn3)  table(wine\_test$Type, wine\_test.knn5) |



1. We could manually add up the number of correct and incorrect cases for the k=1, k=3 and k=5 models to see which is most accurate. Here we’ll get R to do it for us using some IF functions and loops to say whether each case was categorised correctly or not.

|  |
| --- |
| #We can add up the number of correct cases using the previous tables  #Or we can get R to do this for us  #Work out whether each case has been correctly classified or not  #For k=1  knn1\_results <- vector(length=length(wine\_test.knn1))  for (i in 1:length(wine\_test.knn1)) {  knn1\_results[i] <- if (wine\_test$Type[i] == wine\_test.knn1[i]) {'Correct'}  else {'Incorrect'}  }  #For k=3  knn3\_results <- vector(length=length(wine\_test.knn3))  for (i in 1:length(wine\_test.knn3)) {  knn3\_results[i] <- if (wine\_test$Type[i] == wine\_test.knn3[i]) {'Correct'}  else {'Incorrect'}  }  #For k=5  knn5\_results <- vector(length=length(wine\_test.knn5))  for (i in 1:length(wine\_test.knn5)) {  knn5\_results[i] <- if (wine\_test$Type[i] == wine\_test.knn5[i]) {'Correct'}  else {'Incorrect'}  } |

1. We can then count the number of cases that were correct for each of the models. In this case k=3 was the best as it got 75 out of 78 cases correct. You may get different answers when you run this code. In practice we’d want to run this a number of times to see whether k=3 always comes out best or not.

|  |
| --- |
| #Compare accuracy of results  table(knn1\_results)  table(knn3\_results)  table(knn5\_results) |







1. Image and examples taken from <http://www.math.le.ac.uk/people/ag153/homepage/KNN/OliverKNN_Talk.pdf> [↑](#footnote-ref--1)
2. We would want to [standardise](https://en.wikipedia.org/wiki/Standard_score) the data first and then calculate the [Euclidian distance](http://www.cut-the-knot.org/pythagoras/DistanceFormula.shtml) across the five dimensions we are using [↑](#footnote-ref-0)