Statistics for Business Analytics 2: R Laboratory Homework 1

Instructions

In this assignment we are going to focus on clustering. In particular, we will do some examples with model based clustering and an algorithm known as DBSCAN first, and then we will let you try out the different approaches you have learned on some datasets so that you can investigate how various approaches work in different contexts.

Question 1 *(12 points)*

In this question we are going to learn a famous density-based algorithm for clustering known as DBSCAN. This algorithm is implemented in the R package `dbscan` so you should begin by installing this package.

The name DBSCAN stands for <u>D</u>ensity-<u>B</u>ased <u>S</u>patial <u>C</u>lustering of <u>A</u>pplications with <u>N</u>oise. The key idea behind this algorithm is that in our data, a cluster is basically a dense group of points. If a point belongs to a cluster, then it should be close to several other points in that cluster. This idea essentially describes density.

The algorithm requires us to define two important parameters. The first of these is a positive number called *epsilon (ε)*. This number defines a maximum distance from a point in our data set. For any given point in our data set *p*, we say that the *ε-neighborhood* of that point consists of all the points in the data set that lie at most a distance of *ε* from p. The idea here is that if point *p* is part of a cluster, then there will be several points that are close to it i.e. the number of points inside its *ε-neighborhood* will be large. To put a threshold on this number, we define a positive integer-valued second parameter, *minPoints*, which is the minimum number of points that should lie inside a point *p*'s *ε-neighborhood* in order for us to consider that point *p* is part of a cluster. Ok, make sure you understand these two simple definitions before moving on. Now that we have these under our belt, here is a simple description of how the algorithm works:

1) First define values for *ε* and *minPoints*
2) Now pick an arbitrary point p and check if there are at least *minPoints* with its *ε-neighborhood*. If this is the case, then define a new cluster consisting of all of these neighboring points as well as the original point we chose. Then, continue to grow this cluster recursively by checking that for each new point added to this cluster, whether there are at least *minPoints* neighbors within that point's *ε-neighborhood* and including these to the cluster. Repeat this process until you cannot add any new points. This cluster is then finalized.
3) Keep repeating step 2 above by considering points that have not been assigned to a cluster until you have considered all the points in the dataset or until all the points have been assigned to a cluster. Notice that at the end of this algorithm, some points do not belong to any cluster.

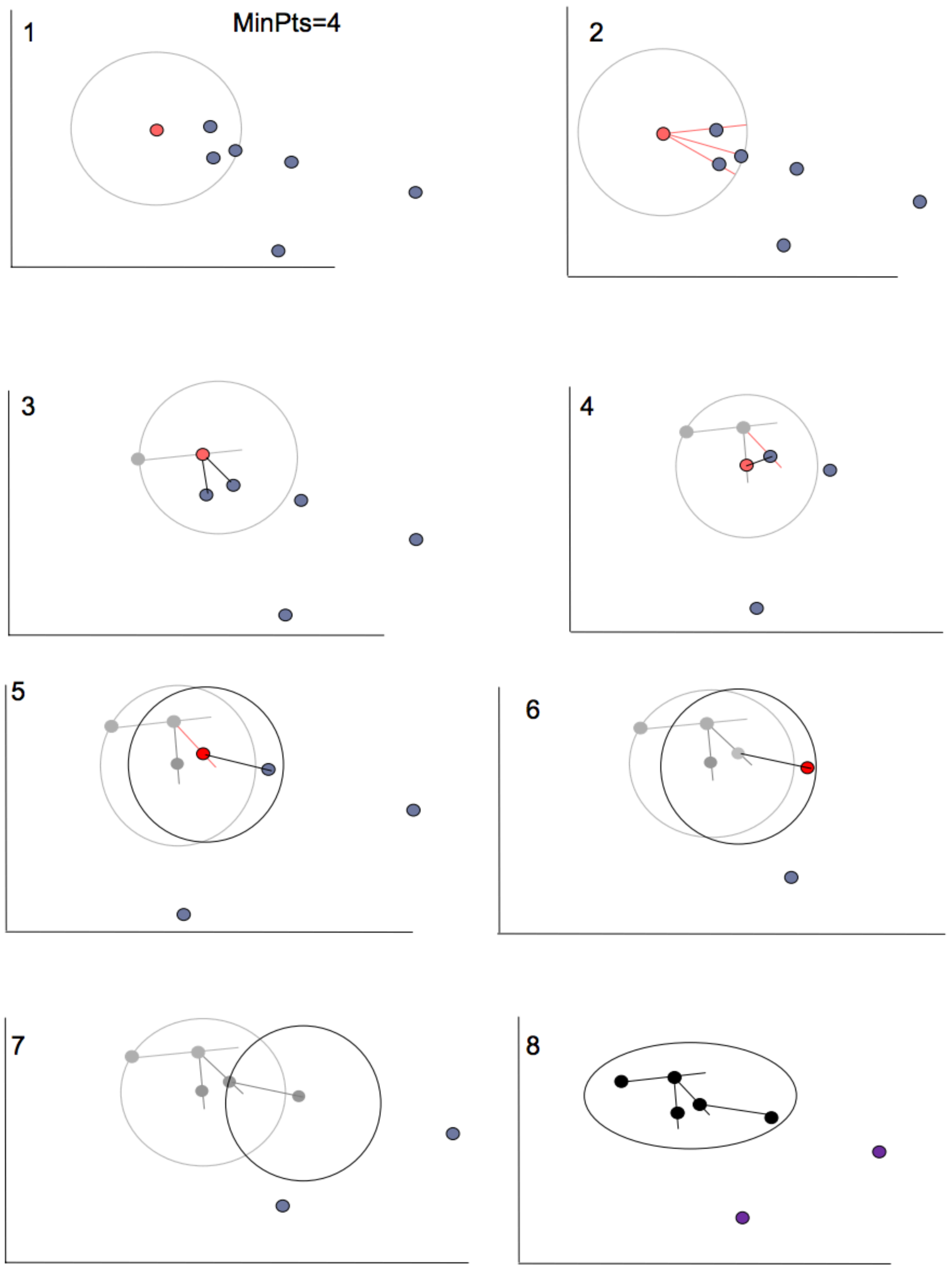
We can define three types of points at the end of this approach:

- **Core Points**: These are points in the data set that have at least *minPoints* neighbors with their *ε-neighborhood*. These are found in the interior of a cluster.
- **Border Points**: These points have fewer than *minPoints* neighbors with their *ε-neighborhood* but happened to lie within the *ε-neighborhood* of a core point. Thus, they were added to a cluster but they themselves did not add any new points to that cluster. Notice that it is possible for a point to be on the border of two different clusters! As a result, DBSCAN has an

element of randomness associated with the order in which we process the points in our database which will determine which clusters are formed first and hence the assignment of border points. In practice, because this only applies to border points, DBSCAN has much less variability in its results compared to something like K-Means.

- **Noise Points**: These points are neither core nor border points and were not added to any cluster at the end of the algorithm.

The following diagram should help you visualize the progress of DBSCAN on a tiny data set of points that yields 1 cluster and 2 noise points.



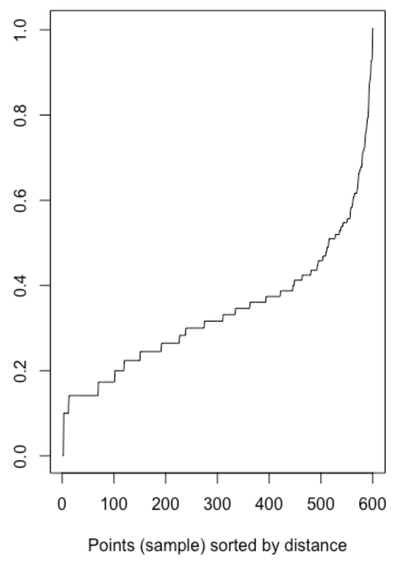
The key characteristics of this algorithm include:

- The algorithm can discover clusters of arbitrary shape.
- The algorithm is equipped to handle noise in the data set and it does not assign clusters to all of the points.
- The algorithm performs only one single scan of the data.
- The algorithm has two important parameters to be tuned and its effectiveness is essentially tied to a good choice of these. It can sometimes be hard to pick these correctly.
- Because these parameters essentially describe the requirements of what constitutes denseness in a cluster, the algorithm is not particularly great when we have clusters of varying density in our data.
- Depending on the number of border points, there could be an element of randomness in the results

Check out this blog post for a good visualization of this algorithm: https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Trying to understand how to pick good values for our DBSCAN parameters is an area of research. For *minPoints* the tradeoff is typically between how much noise we want to allow in our clusters (small values lead to more noise). If we pick a value of *minPoints* we can then use a simple rule of thumb approach to pick a value for *ε*. For each point in our data set, we can compute the distance to its nearest k neighbors, where *k* takes the value of *minPoints*. Then we will plot a histogram of the distance to the farthest k-Nearest Neighbour from every point. In this histogram we can look for a "knee" that is to say a point where this distance suddenly starts to rise abruptly and then pick this value as a value for *ε*. This is done for the iris dataset below using the R function `kNNdistplot()` for 4 neighbors (k=4):

```
> kNNdistplot(iris[1:4], k=4)
```



Looking at the y-axis in the previous plot, we can see that around 0.5 the gradient of the slope starts to sharpen abruptly and so we could pick an ε around that value.

Now, let's put this knowledge to the test. We're going to the use the `planets` dataset inside the package `HSAUR2` and we will also use the package `scatterplot3d` to visualize the so install those two packages as well as `dbscan`. The following code snippet shows us how to visualize a 3D scatterplot with the package `scatterplot3d` using three dimensions chosen from the `mtcars` dataset.

```
library(scatterplot3d)
with(mtcars, {
  scatterplot3d(disp, wt, mpg,        # x y and z axis
                color="blue", pch=19, # filled blue circles
                type="h",             # lines to the horizontal plane
                main="3-D Scatterplot Example 2",
                xlab="Displacement (cu. in.)",
                ylab="Weight (lb/1000)",
                zlab="Miles/(US) Gallon")
})
```

Now we are ready to look at our `planets` data set. This is a dataset with 101 observations from 101 exoplanets on the following 3 variables:

- mass (mass of the planet relative to Jupiter)
- period (period in earth days)
- eccen (the radial eccentricity of the planet, meaning how much it deviates from a perfect circular orbit)

i. Use the above code example to plot a 3D scatterplot of this data set ***(1 point)***
ii. Run `summary()` on this data set. ***(1 point)***
iii. You will see that the three variables are not only measured on different scales but their maximum values are quite large because they are quite skewed. Create a new data frame, planets2, where we transform each of the three variables to a logarithmic scale. Note that some values in the `eccen` column are practically zero so we should apply a function `log(1+x)` to each scale. ***(2 points)***
iv. We will investigate this data set with *minPoints* = 7. Use the `kNNdistplot()` function to plot the histogram for the farthest 7-Nearest Neighbors in this data set and after studying it, suggest a suitable value of ε. ***(2 points)***
v. The function `dbscan()` can now be used to compute a clustering on our data. The first parameter we provide is our data frame. We must also set the `eps` and `minPts` parameters. Run this function using the values from the previous part and store this result in a variable `res`. Then print this variable to the screen to see how many points are assigned to the various vlusters and how many noise points we obtain. ***(2 points)***
vi. If you modify the different parameters we can get different clustering results and you could check this. For now, visualize the clustering you

obtained using the `scatterplot3d` code above but change the color of the points to match the cluster assignments. You can find the cluster assignment in the cluster property of your `res` object. **HINT**: setting the color value to 0 will not render the points properly with `scatterplot3d` ***(2 points)***

vii. Investigate the qualities of the clusters you found by studying the statistics of the points assigned to each cluster and try to describe the distinguishing characteristics of the clusters that you found. HINT: You may find the `clPairs` function in the `mclust` R library useful. ***(2 points)***

Question 2 (10 points)

In this question we are going to investigate how model based clustering with Gaussian mixture models works on our planets data set. You should have had a short introduction to this approach from your notes. In a nutshell, we assume that the data clusters are the result of a mixture of different multi-dimensional Gaussian components. The package `mclust` is very useful in that it has methods to try to discover an appropriate parameter setting for these Gaussian components (their number, location and standard deviations) using information theoretic criteria such as the BIC.

i. Load the `mclust` library if you have not done so already. Use the command `Mclust()` and supply the `planets` data frame as an input, storing the results in a variable `res2`. Use `summary()` on the result. **(2 points)**

ii. As you saw from your notes, model based clustering with Gaussian mixtures can result in a range of different model types by varying the characteristics of the covariance matrices. In particular, you can change the shape of the cluster, the overall volume, the symmetry and the orientation. How many clusters did you find, and what are the characteristics of the model? ***(1 point)***

iii. To understand why this particular model was chosen you can plot the variable `res2` with `plot()` and supply the parameter `what="BIC"`. The y-axis will then show the negative of the BIC value. Now draw a second plot same as the first but zoomed in on the interval (-2300, -1800) on the y-axis using the `ylim` parameter so you can distinguish some of the lines more clearly. Explain how these plots justify the model choices you reported on in the previous step. ***(3 points)***

iv. Using the `clPairs()` function on the model's cluster output, describe the clusters that this model has found. The classification output of the model is in the attribute `classification` in the `res2` object. Note that because this is not simply a vector of cluster labels you must pass this to the `classification` parameter of the `clPairs()` function. ***(2 points)***

v. Visualize the clusters in 3D exactly as you did for part vi of question 1 and comment on how these compare to those you found with DBSCAN ***(2 points)***

Question 3 (20 points + 5 Bonus points)

In this final question we will give you some artificial data sets to play with. The goal is to understand and investigate the behavior of different clustering algorithms and to see which ones work well in what cases.

First lets create the data sets. You will need to install some packages (`factoextra` and `mlbench`) for these. DO NOT INCLUDE CALLS TO `install.packages()` INSIDE YOUR MARKUP FILE. Include the following commands at the start of your answer to question 3.

```
library(factoextra)
data1 <- multishapes
library(mlbench)
set.seed(42) #
data2 <-  mlbench.smiley()
data3 <-  mlbench.cassini(1000)
data4 <-  mlbench.circle(1000)
data5 <-  mlbench.spirals(1000, cycles = 3,  sd = .05)
data6 <-  mlbench.threenorm(1000, d = 2)
data7 <-  mlbench.shapes(1000)
```

We now have 7 2-D dimensional data sets stored in the variables `data1` through `data7`.

i) Plot all these data sets and state for each data set, how many clusters you observe. **(2 points)**

ii) You know have a number of different approaches to clustering that you've seen (K-Means, Hierarchical clustering, model-based clustering and DBSCAN). In the laboratory R file, you also have a code snippet at the end where you can try out a variant of K-Means known as Partitioning Around Medioids (PAM), which uses medioids instead of centroids. For each of your 7 data sets, we want you to investigate the performance of the clustering techniques you've learned and write a 2-5 line short summary MAX (in English) that compares at contrasts the performance of the approaches you tried out. Note that in addition to the algorithms you've learned, you also know techniques for transforming your inputs before you cluster with techniques such as PCA and normalization. These might turn out to be useful on some data sets for some algorithms **(18 points)**.
HINT: This is less work than it seems. Try to use the same code you write for each algorithm across the different data sets.
HINT 2: Be organized! Try to organize your code, your results and your write-ups neatly and cleanly. Simple, short and to the point will get you all the points.

iii) As a bonus, we want to give you the chance to explore these data sets further. Investigate a new clustering algorithm on your own, find and run the code for this on these data sets and include that algorithm in your analysis. If you explain BRIEFLY how the algorithm works, you use it correctly in the code and you can draw conclusions from it, we'll give you another 5 bonus points (increasing the maximum score for this assignment above 100%). Two ideas for you to are CURE and OPTICS **(5 bonus points)**

If you want to find R packages related to clustering:
https://cran.r-project.org/web/views/Cluster.html