NAVAL
POSTGRADUATE
SCHOOL

OS4118

Statistical and Machine Learning

Clustering

Prof. Sam Buttrey

Fall AY 2020

Clustering

- Techniques for finding structure in a set of measurements
- Group X's without knowing their y's
- Usually we don't know number of clusters
- Method 1: **Visual**
- Difficult because of (usually) complicated correlation structure in the data
- Particularly hard in high dimensions

Clustering as Classification

- Clustering is a classification problem in which the Y values have to be estimated
- $Y_i \mid X_i$ is multinomial as before
- Most techniques give an assignment, but we can also get a probability vector
- Clustering remains under-developed
  - Model quality? Variable selection? Scaling? Transformations, interactions etc.? Model fit? Prediction?

Clustering by PCs

- Method 2: **Principal Components**
- If the PCs capture spread in a smart way, then “nearby” observations should have similar values on the PCs
- Plot 2 or 3 and look (e.g. `state.x77`)
- We know how to transform new (test set) observations into PC space
- We still need a rule for assigning observations to clusters

Our Dissimilarity Should Be…

1. Applicable to **all types of data** including data with mixed linear and categorical variables
2. **Invariant** to linear scaling, and robust to monotonic transformations, of numeric variables
3. Able to handle **missing values** and outliers
4. Capable of **variable selection** (ignoring noise variables, redundant variables), including weighting
5. Able to handle **large data sets** (in particular, we want to avoid computing all *n*-choose-2 pairwise distances when data sets are large)

Inter-point Distances

- Many clustering techniques rely on being able to measure distance
  - between two points;
  - between a point and a cluster; and
  - between two clusters.
- **How do we…**
  define a distance that meets the criteria of the preceding page, especially weighting and selection, categorical values, and missing values?
- Our choices are Euclidean, Gower, etc…

Distance Between Clusters

- In addition to measuring distance between two observations, …
- …We also need to measure distance between a point and a cluster, and between two clusters
- Example: Euclidean between the two cluster **averages**
- Example: Manhattan between the two points **farthest apart**
- These choices may make a difference, and we don't have much guidance

A. Partition Methods

- Given number of clusters (!), try to find observations that are means or medians
- Goal: each observation should be closer to its cluster's center than to the center of another cluster; this partitions space
  - As we have seen, measuring "closer" requires some choices to be made
- Classic approach: ***k*-means** algorithm
  - R implementation predates `daisy()`, requires all numeric columns

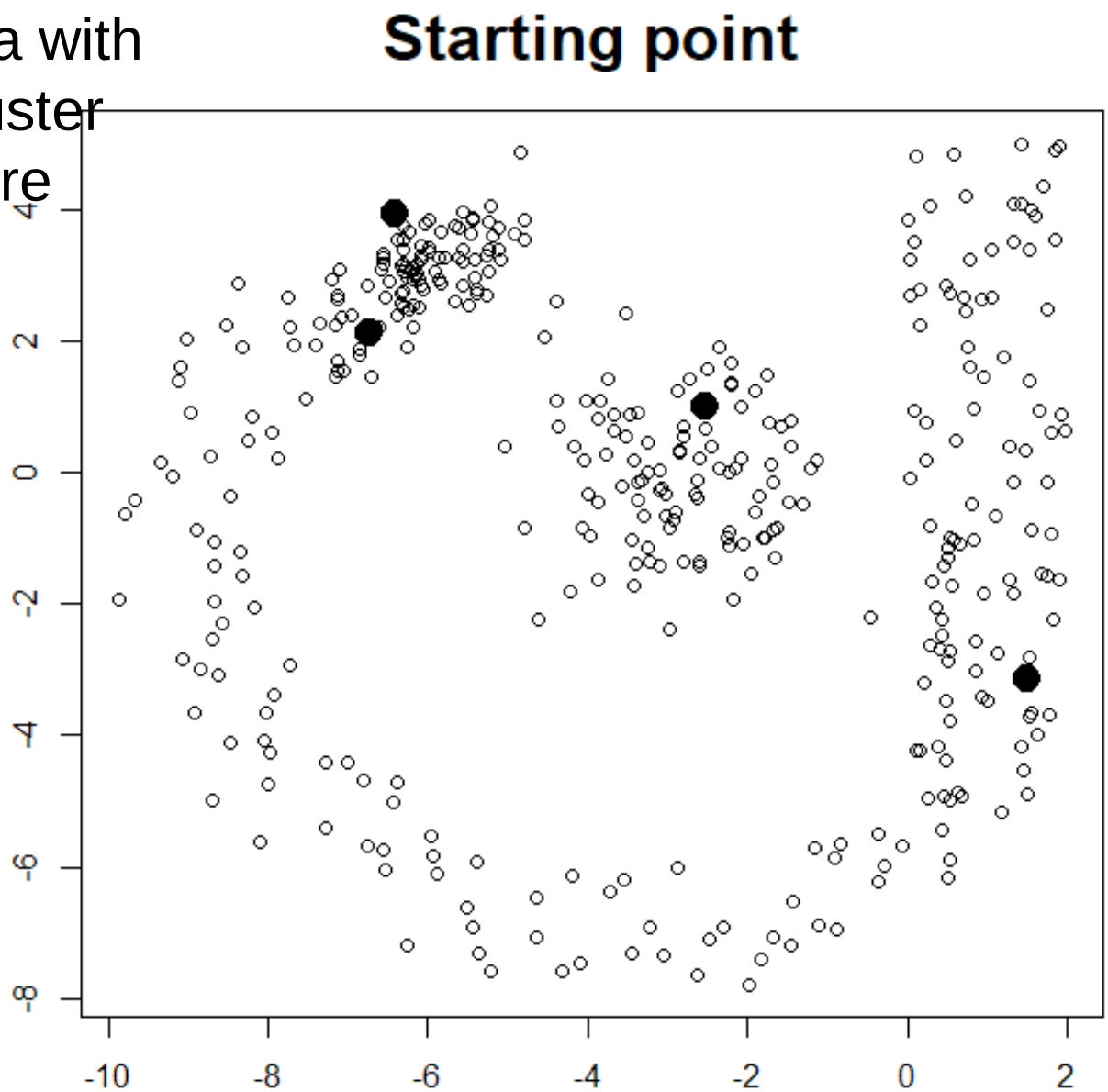
K-means Algorithm

1. Select *k* candidate cluster centers at random
2. Assign each observation to the nearest cluster center (w/Euclidean distance)
3. Recompute the cluster means
4. Repeat from 2. until convergence

- Guaranteed to converge, but not optimally; depends on step 1; *k* assumed known (try with many *k*'s)

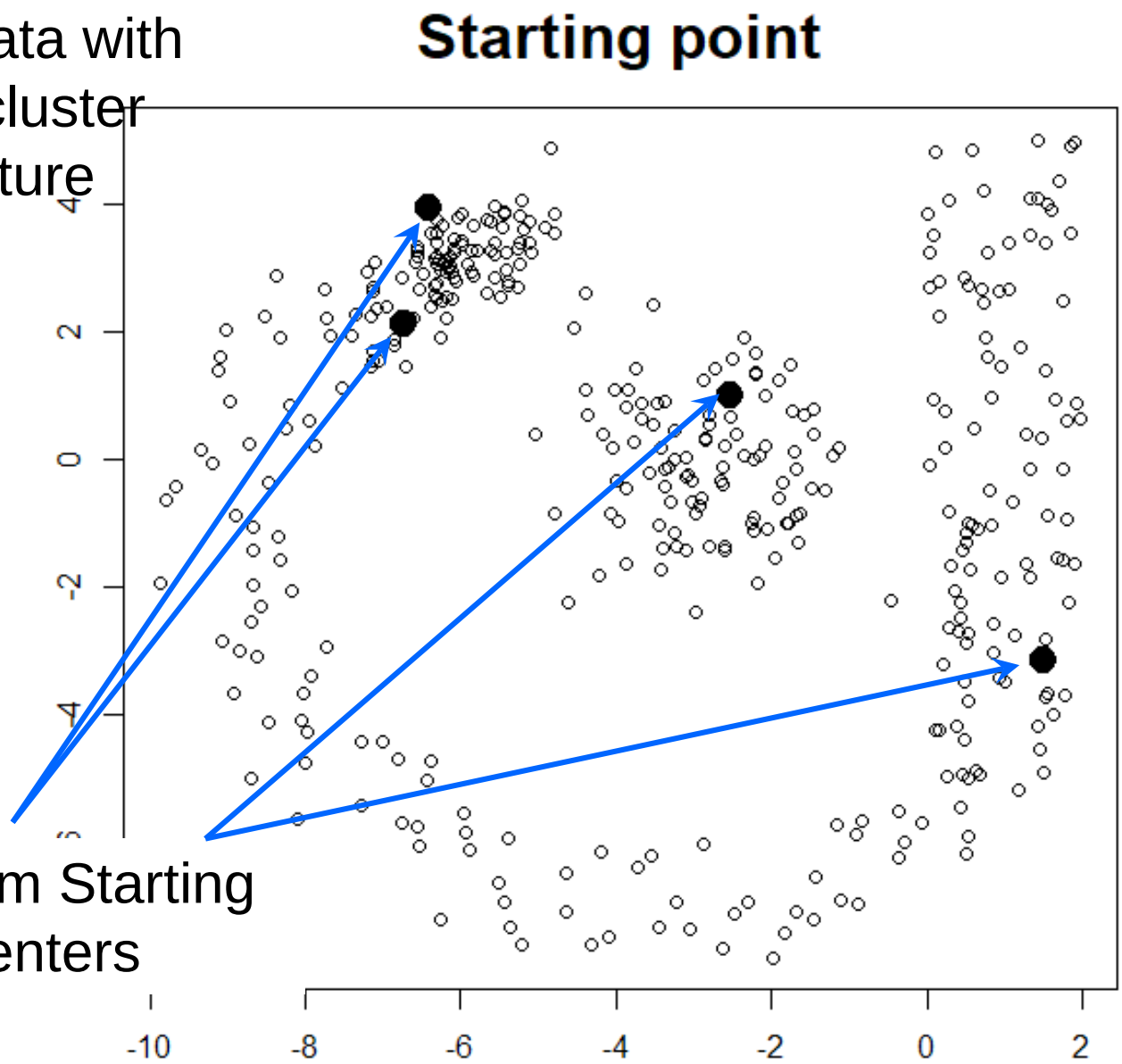
K-means Example (start)

Two-d data with some cluster structure



K-means Example (start)

Two-d data with some cluster structure

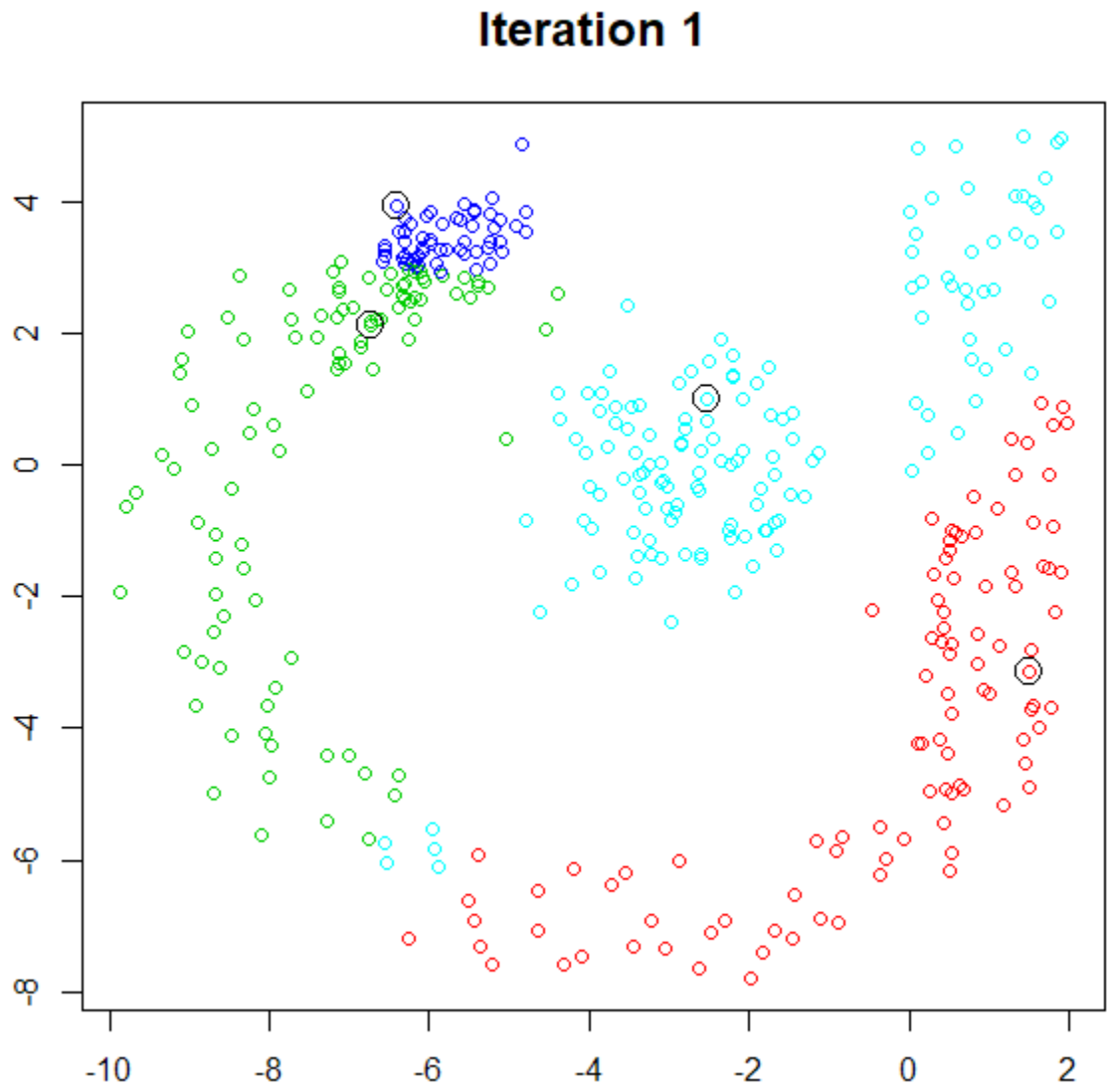


Random Starting Centers

K-means Example (iteration 1)

Assign each point to the nearest center, using unweighted Euclidean distance

(Assignments shown here by color)

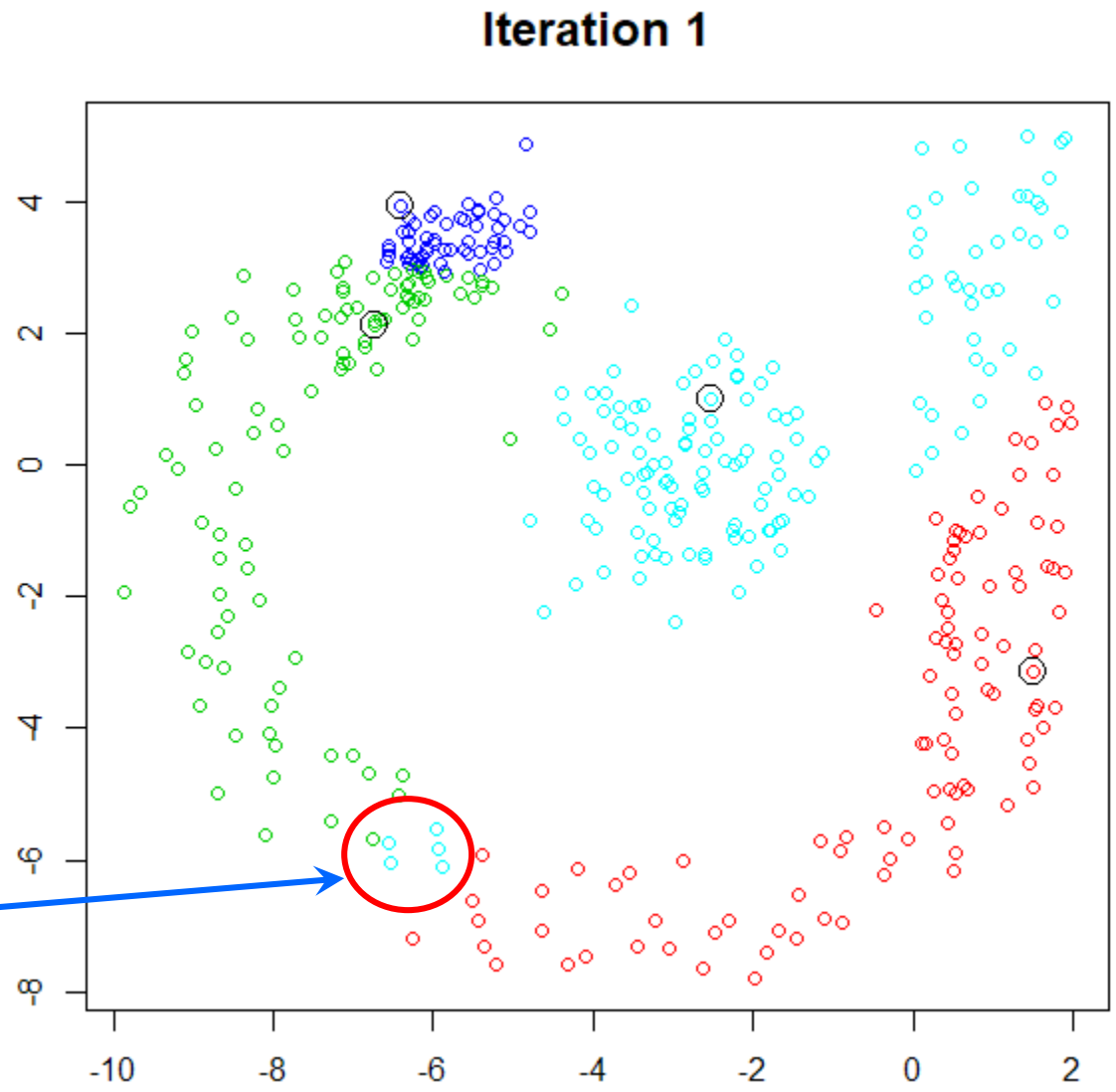


K-means Example (iteration 1)

Assign each point to the nearest center, using unweighted Euclidean distance

(Assignments shown here by color)

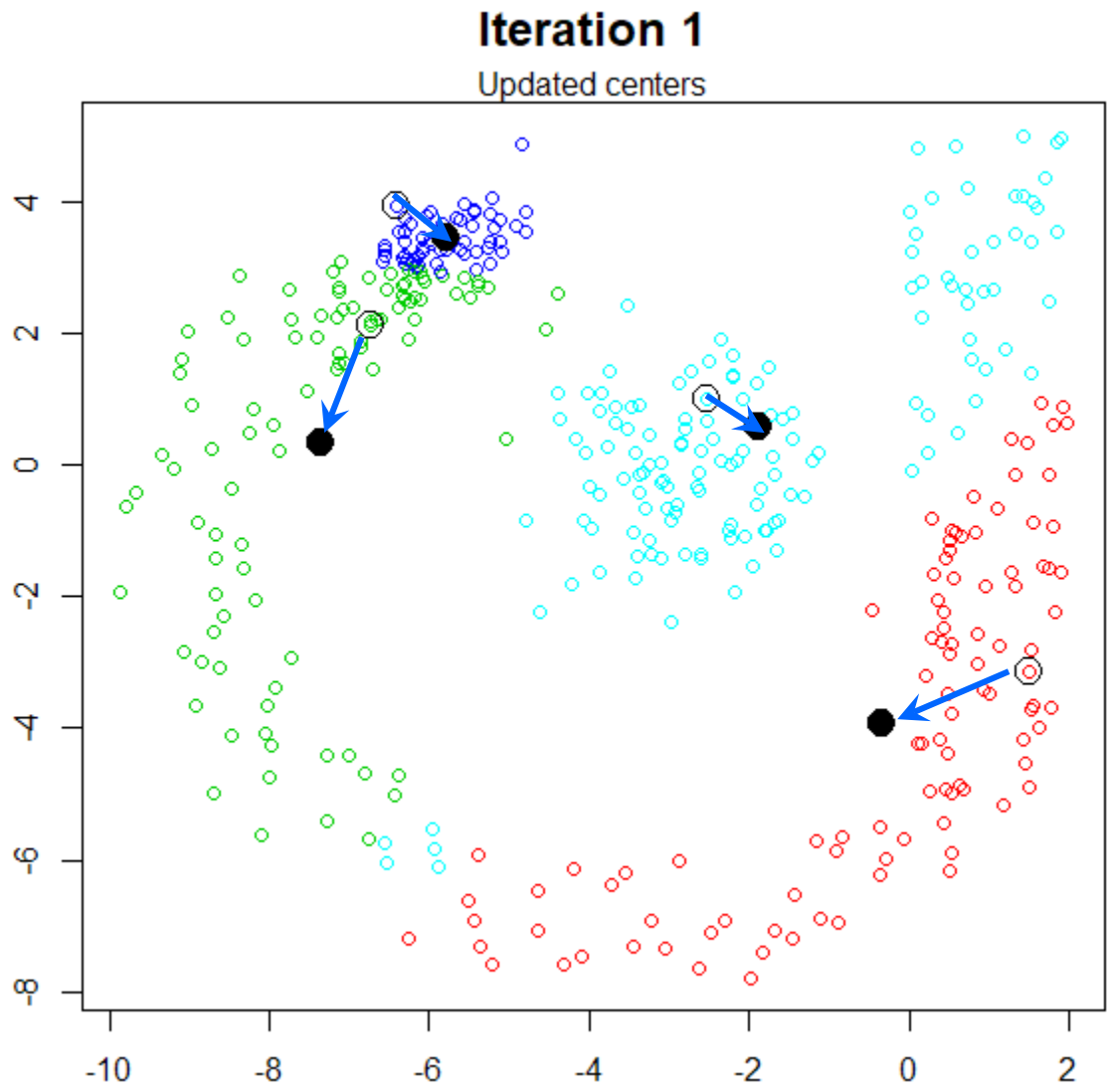
Check out these light blue points



K-means example (iteration 1)

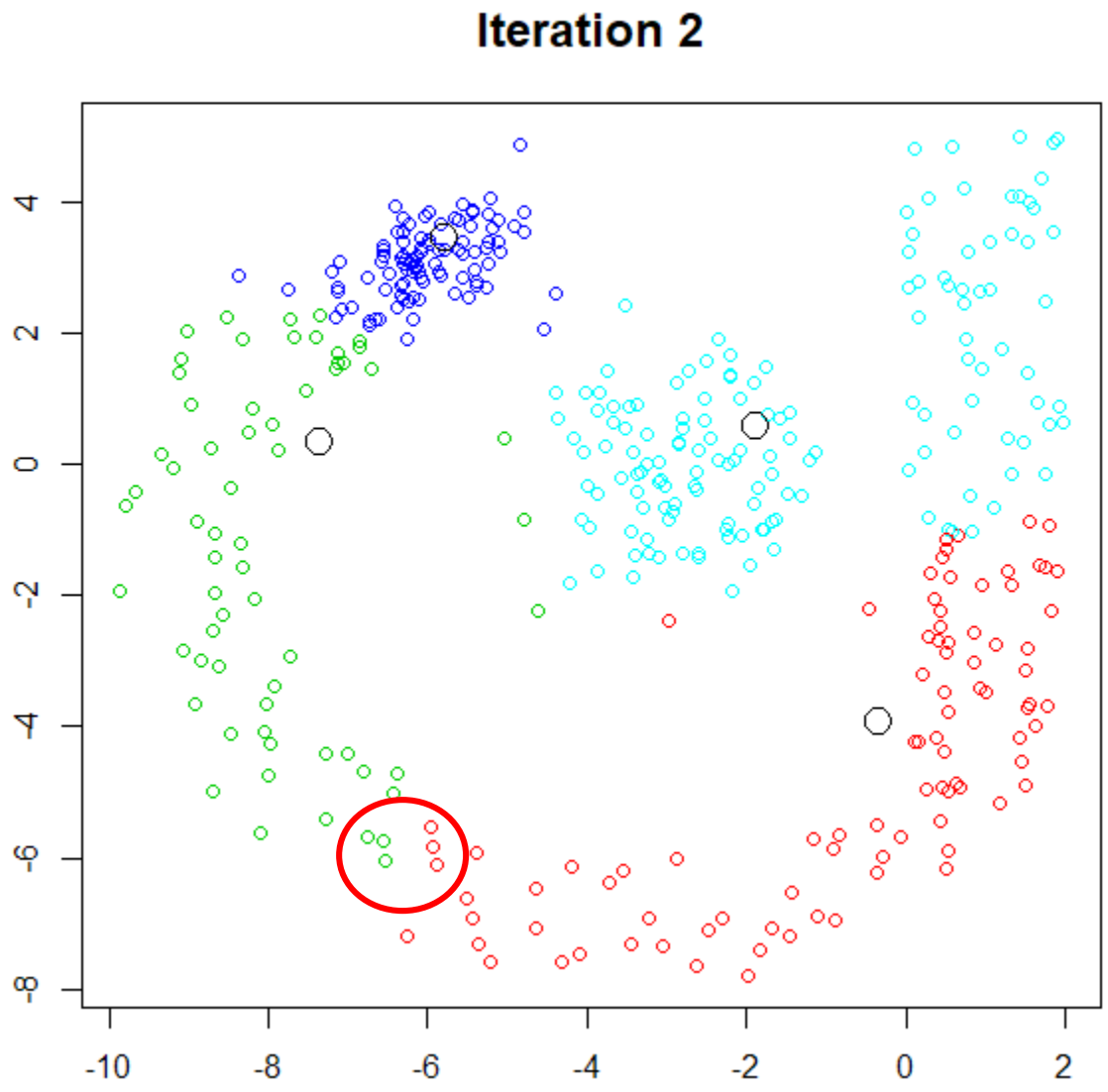
Use those assignments to re-compute the cluster means, which causes them to change position

New positions are shown by black dots



K-means Example (iteration 2)

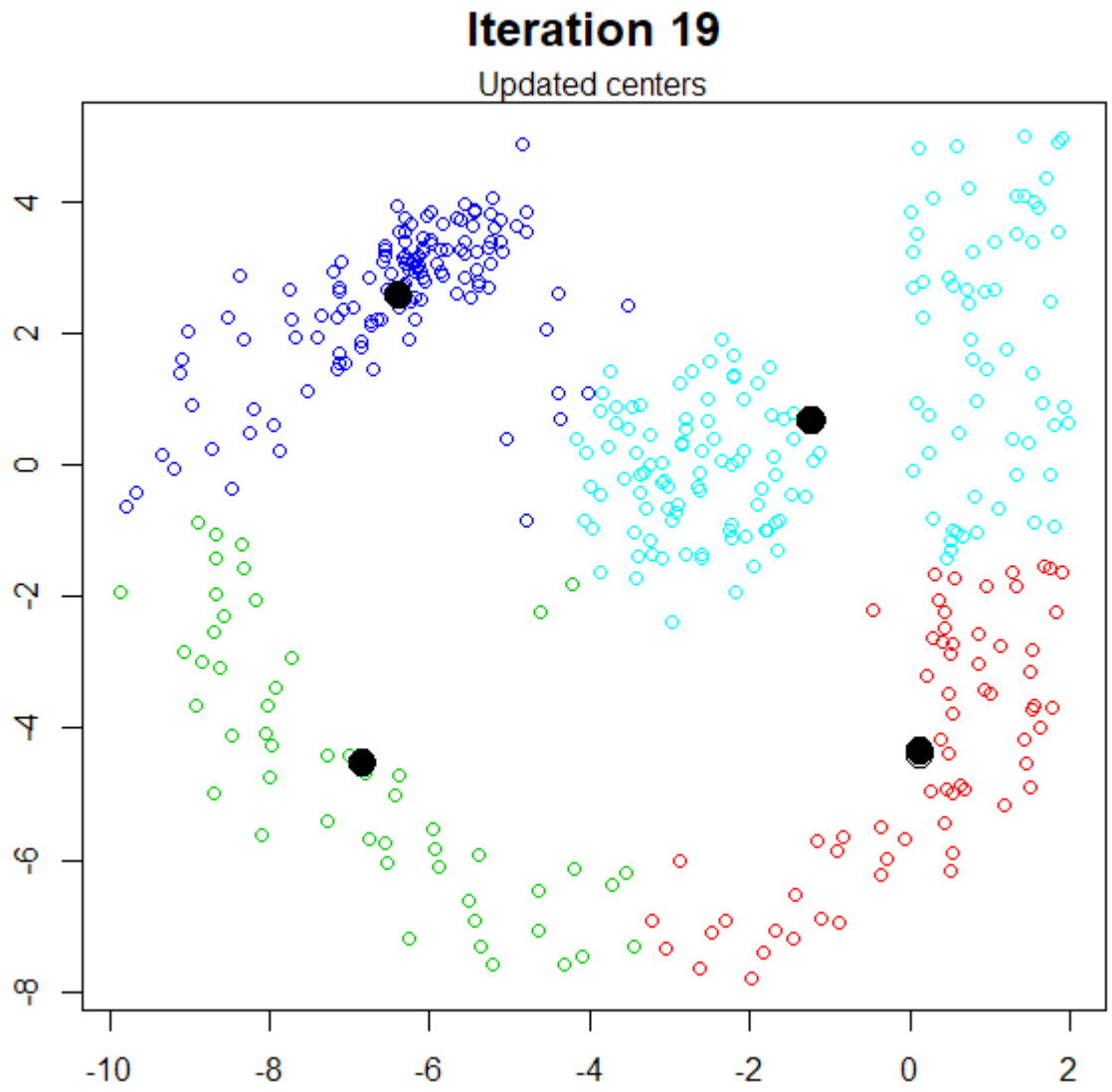
Now use those new positions to re-assign each observation to the closest center



K-means Example (finish)

This example took 19 iterations – but modern algorithms are more clever than the basic one I laid out here

Starting points matter too!



K-means Stopping

- Stop when no assignment changes, or a pre-set max. number of iterations is reached
- One measure of quality: an R^2-like thing
- Compute TotalSS = $\sum_i \sum_j \left(x_{ij} - \bar{x}_j\right)^2$
- Compute WithinSS for cluster 1 as $\sum_{\text{cluster 1}} \sum_j \left(x_{ij} - c_{1j}\right)^2$

K-means Stopping

- Stop when no assignment changes, or a pre-set max. number of iterations is reached
- One measure of quality: an R^2-like thing
- Compute TotalSS = $\sum_i \sum_j \left(x_{ij} - \bar{x}_j\right)^2$
- Compute WithinSS for cluster 1 as $\sum_{\text{cluster } 1} \sum_j \left(x_{ij} - c_{1j}\right)^2$

Cluster 1's center

- Now compute 1 – TotalWithin/Total SS

K-means (cont'd)

- Only *kn* (not $n(n-1)/2$) computations per iteration, helps with big data
- Well-suited to separated spherical clusters, not to narrow ellipses, snakes, linked chains, concentric spheres…
- Susceptible to influence from extreme outliers, which perhaps belong in their own clusters of size 1
- Example: `state.x77` data

Choosing *k*

- Choosing *k* is important but hard
- Classical scheme: plot 1 – TotWith/Tot, or just TotWith, against k for a bunch of different *k* values and identify…
- …the **knee** in the curve
- Just as with principal components, the "knee" is not always easy to find

Pam and Clara

- `pam` (Kaufman & Rousseeuw, 1990) is *k*-means-like, but on medoids
  - A cluster **medoid** is the observation for which the sum of distances to other cluster members is the smallest in the cluster
  - Can use `daisy()` output, handle factors
  - Resistant to outliers
  - Expensive ($O(n^2)$ for time and memory)
- `clara` is pam's big sister
  - Operates on small subsets of the data
  - `pamk()` in `{fpc}` helps pick "best" *k*, but criteria can disagree

Cluster Validation

- *K*-means vs. pam
- How to evaluate how well we're doing?
  - Cluster validation is an open problem
  - Goals: ensure we're not just picking up sets of random fluctuations
    - If our clustering is "better" on our data than what we see with the same technique on random noise, do we feel better?
  - Determine which of two clusterings is "better"
  - Determine how many "real" clusters there are

Cluster Validity

- **External** Validity: Compare cluster labels to "truth," maybe in a classification context
  - True class labels often not known
    - We cluster without knowing classes
  - Classes can span clusters: f vs. f vs. f, so in any case…
  - …True number of *clusters* rarely known, even if we knew how many *classes* there were

Cluster Validity

- **Internal** Validity: Measure something about "inherent" goodness
  - Perhaps R^2-style, 1 – SSW/SSB, using "sum of squares within" and "sum of squares between" as in k-means
  - Whatever metric the clustering algorithm optimizes will look good in our results
  - "Always" better than using our technique on noise
  - Not obvious how to use training/test set

The Silhouette Plot

- For each point, compute avg. distance to all points in its cluster (a), and avg. dist. to points in second-closest cluster (b)
- Silhouette coeff. is then (b – a)/max(a, b)
  - Can be computed for individual observations, or averaged within clusters and overall
- Usually in [0,1]; larger better
- Drawn by `plot.pam()`, `plot.clara()`
- (Different from bannerplot!)

Examples

- *K*-means vs. pam (cont'd)
- How to evaluate how well we're doing?
  - For the moment let's measure **agreement**
  - One choice: Cramér's *V*
  - $V = \sqrt{[\chi^2 / n\,(k\text{-}1)]}$, $k = \min(\#\text{row}, \#\text{col})$
  - $V \in [0, 1]$; more rows, cols $\Rightarrow$ higher *V*
- Rules of thumb: .15 weak, .35 strong, .50+ "essentially measuring same thing"
  - Sorta kinda

B. Hierarchical Clustering

- Techniques to preserve hierarchy (so to get from the "best" six clusters to the best five, we join two existing clusters)
  - Advantages: hierarchy is good; nice pictures make it easier to choose number of clusters
  - Disadvantages: small data sets only
- Typically "agglomerative" or "divisive"
- **Agglomerative**, implemented in `agnes()`: each object starts as one cluster; keep "merging" the two closest clusters till there's one huge cluster

Agnes (cont'd)

- Each step reduces the # of clusters by 1
- At each stage we need to know every entity's distance to every other
- We merge the two closest objects…
- …Then compute distances of new object to all other entities
- As before, we need to be able to measure the distances between cluster, or between a point and a cluster

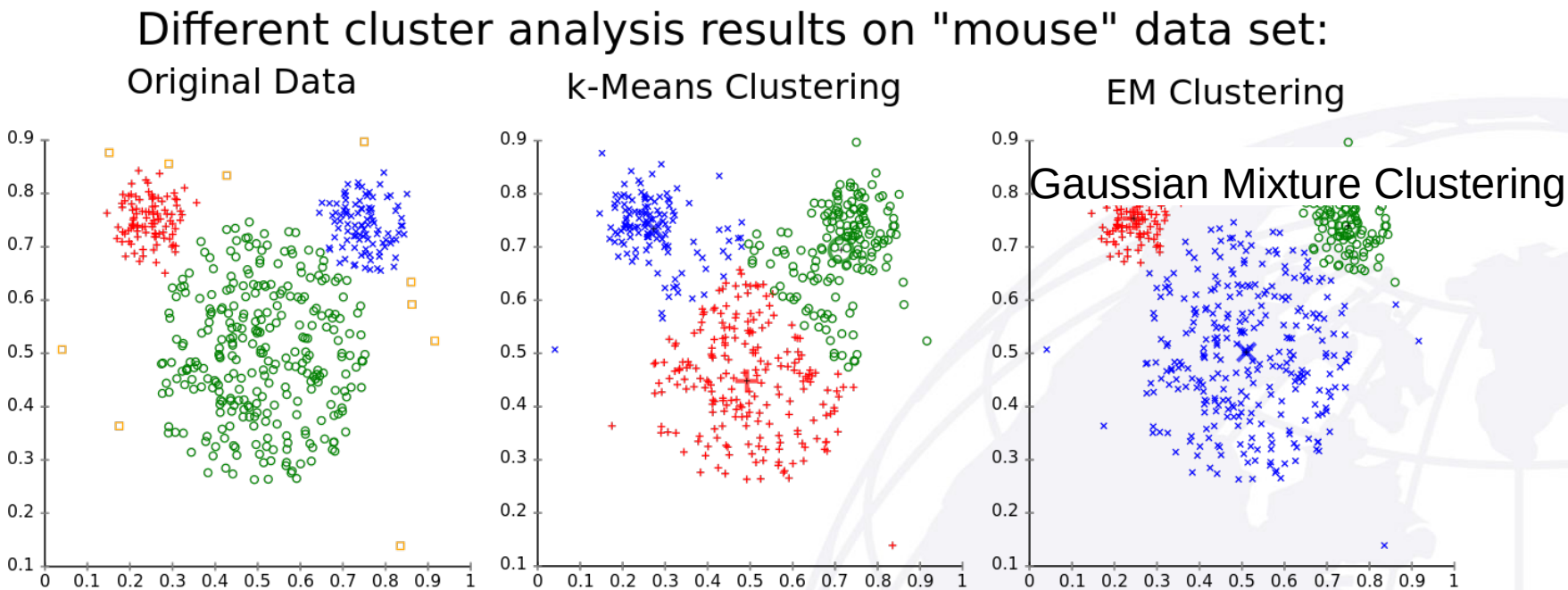
Hierarchical Clustering (cont'd)

- **Divisive**, implemented in `diana()`:
  - Start with all objects in one group
  - At each step, find the largest cluster
  - Remove its "weirdest" observation
  - See if others from that parent want to join the splinter group
  - Repeat until each obs. is its own cluster
- Clustering techniques often don't agree!

Dendrogram

- The tree picture ("dendrogram") shows the merging distance vertically and the observations horizontally
- Any horizontal line specifies a number of clusters (implemented in `cutree()`)
- Both Agnes + Diana require all n-choose-2 distances up front; ill-suited to large samples

Let’s talk about K-means first

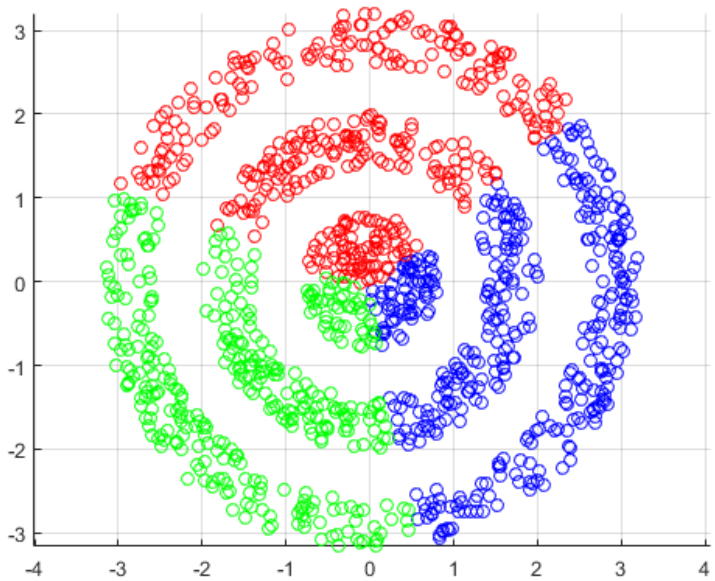


- It’s not too good at finding clusters of different shapes.
- It can’t help identify the stray yellow anomalies
- And like K-medoids (pam), you need to know the number of clusters.

https://en.wikipedia.org/wiki/K-means_clustering

Our methods can't cluster this

- Here the clusters are kind of silly, but the point is that they are not linearly separable

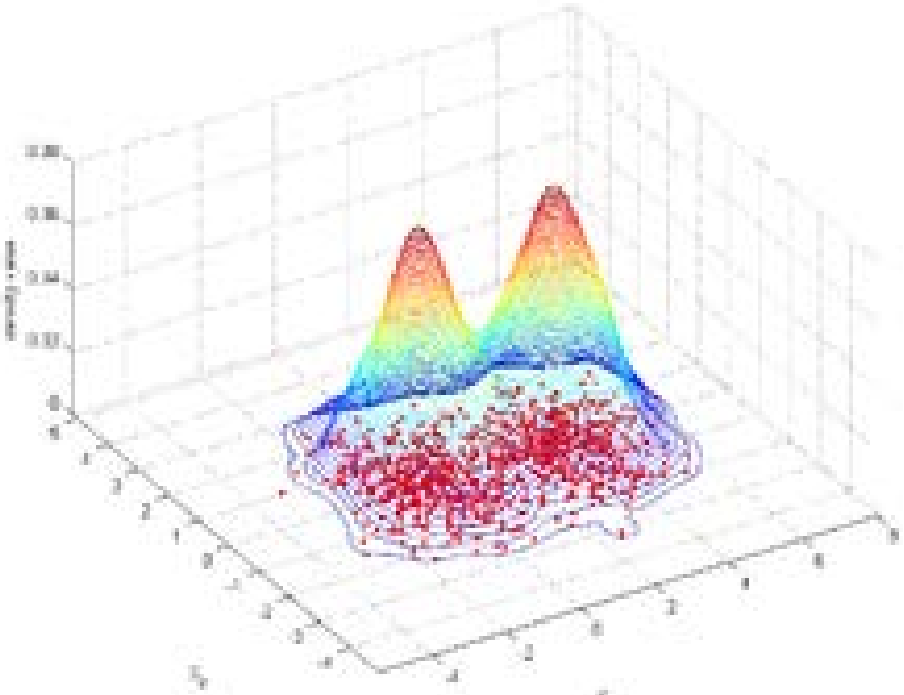


K-means clustering

https://stackoverflow.com/questions/40455334/matlab-kmeans-clustering-for-non-linearly-separable-data

Density Based Clustering

- Density based clustering, an old idea of Hartigan (1975) i.e. cluster observations in high density regions. Observations in low density regions are allowed to be noise.



- Does not require knowledge of the number of clusters or their shapes.

https://en.wikipedia.org/wiki/Multivariate_kernel_density_estimation

Density-Based Clustering

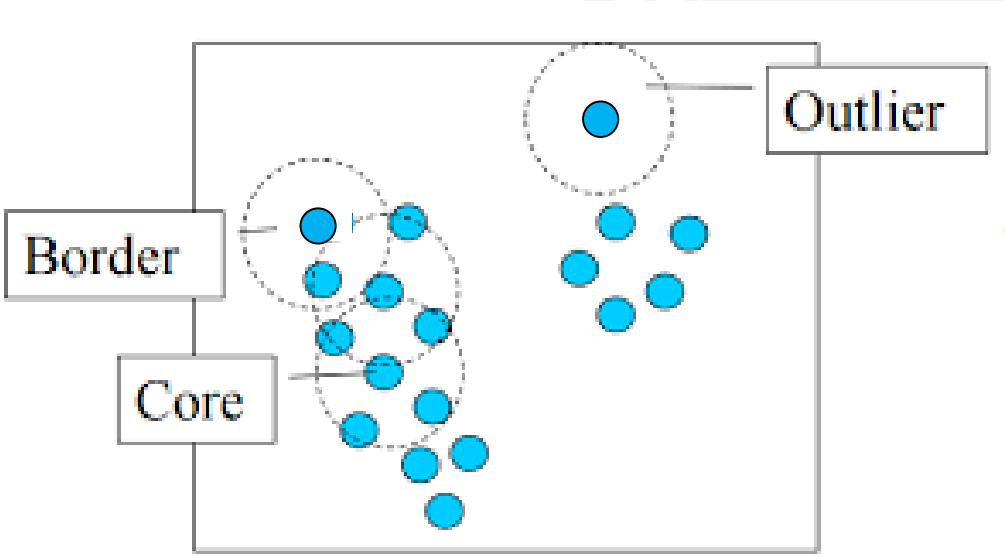
- The density-based algorithm **DBSCAN** of Ester, Kiegel, Sander, Xu (1996) is a very successful clustering algorithm of this type.
  - "Density-Based Spatial Clustering of Applications with Noise"
- The algorithm "scans" the data one time. Fast, scalable, intuitive, don't need to know the number of clusters, works with different shapes, and can assign observations to "noise" – that is, some points don't get put into any cluster

DBSCAN's Two Parameters

Pick a starting point (one of your observations) at random and specify two tuning parameters:

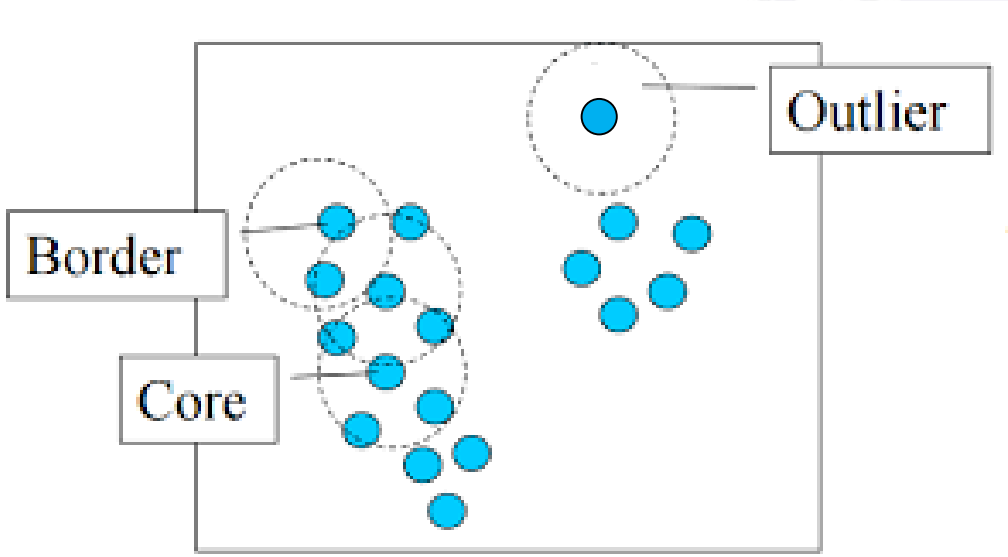
Epslion: The radius of a local neighborhood around a point.

Min Points: The minimum number of points that must be in that neighborhood for the point to be considered a "core point".



DBSCAN

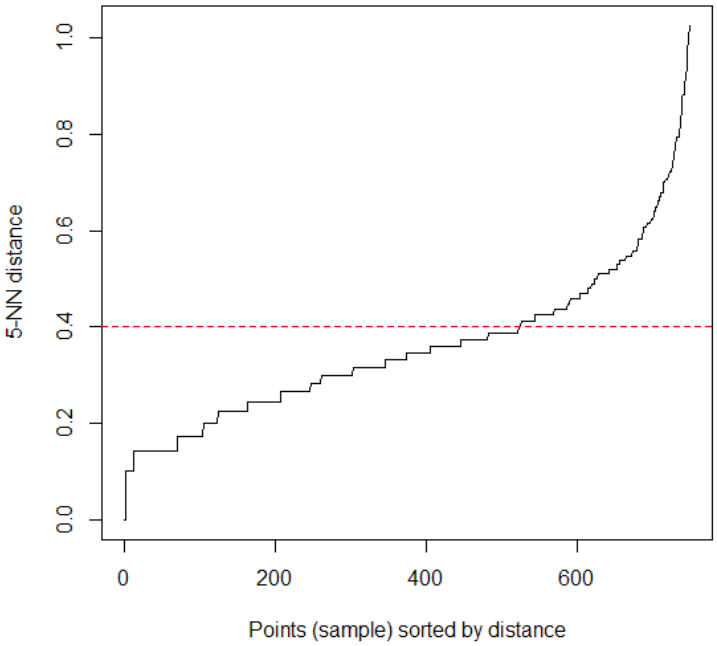
- When a cluster is formed, expand the cluster by checking all new points. **Border points** can belong to more than one cluster.



- https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Choosing Parameters

- One rule of thumb: choose minPoints = (# of columns + 1)…
  - But what if some columns are just noise?
- …Choose Epsilon by plotting the sorted distances to the minPoints nearest-neighbor and looking for the "knee"



Strengths and Weaknesses

- DBSCAN Strengths:
  - Can discover clusters of arbitrary shapes, whereas k-means likes spherical clusters
  - Number of clusters need not be specified
  - Scalable to very large data sets; only one scan through the data require
  - Allows for anomalies (noise) that don't belong to any cluster

Weaknesses

- Weaknesses:
  - Can be defeated by clusters with different densities, because minPts and eps are held constant over the whole space
    - Newer **OPTICS** algorithm tries to address this
  - Choosing epsilon can be difficult
  - Curse of dimensionality in high dimensions, where observations are always far apart – can produce lots of tiny clusters

Clustering Considerations

- Lots of other methods (e.g. mixture models) exist, including ensembles
- Scaling/weighting, transformation are not automatic (although methods are being proposed to do this)
- Hierarchical methods don't scale well
  - Must avoid computing all pairwise distances
- Validation and finding *k* are hard
  - Clustering is inherently much more harder than, say, linear regression