

# Clustering

Daniel Lawson University of Bristol

Lecture 03.2.1 (v2.0.0)

# Signposting

- ▶ We have made latent structures using SVD and PCA.
- ▶ This **dimensionality reduction** is essential for many types of analysis including clustering.
- ▶ Clustering is one of the most fundamental data analysis tools and the ideas form the cornerstone of more complex approaches.
- ▶ In Part 1 we cover:
  - ▶ How Clustering methods are organised,
  - ▶ Hierarchical clustering
- ▶ In Part 2 we cover:
  - ▶ K-means
  - ▶ Gaussian Mixture Modelling
  - ▶ Density-based model-free clustering (dbscan)

# Intended Learning Outcomes

- ▶ ILO1 Be able to **access and process cyber security data** into a format suitable for mathematical reasoning
- ▶ ILO2 Be able to **use and apply basic machine learning** tools
- ▶ ILO3 Be able to make and report appropriate inferences from the results of applying basic tools to data

# Clustering

- ▶ Clustering contains enough complexity to cover several courses by itself.
- ▶ You are likely to use clustering in several projects, sometimes as the goal and sometimes as a data processing step.
- ▶ We will talk about **computational complexity**. This is covered in full detail later in the course. Today,  $O(f(N))$  means that “the algorithm run-time increases as  $f(N)$ , ignoring complexity” (for the worst case data).

# Clustering paradigms

- ▶ Most clustering procedures fit one or more of these paradigms:
- ▶ **Algorithmic clustering**
  - ▶ An algorithm is run which outputs a clustering of the data
  - ▶ Usually fast
  - ▶ Usually data-type specific
  - ▶ Often hard to interpret
- ▶ **Distance-based clustering**
  - ▶ Distances between all items are considered and then clustered somehow
  - ▶ Widely applicable
  - ▶ Often can be linked to a model
- ▶ **Model-based clustering**
  - ▶ Explicit objective function used
  - ▶ Can be slower - unless a convenient model is chosen
  - ▶ Can be made to solve a specific task, handle uncertainty
  - ▶ Most appropriate when you want the clusters to “mean something”

# Most important clustering methods

- ▶ **Algorithmic:**
  - ▶ graph-cutting methods, e.g. modularity
  - ▶ space partitioning, e.g. KD-trees, etc
- ▶ **Hierarchical**, distance-based:
  - ▶ single linkage
  - ▶ complete linkage
  - ▶ average linkage
- ▶ **Model-based:**
  - ▶ k-means (though was introduced as an algorithm)
  - ▶ Gaussian mixture modelling (GMM)
  - ▶ Bayesian clustering

# Algorithmic clustering

Algorithmic approaches are best when used with a goal that exploits the structure provided. We'll visit them as needed. For example:

- ▶ There are really fast **graph clustering algorithms**. The clusters are not always “best” but they are useful.
  - ▶ See for example modularity maximisation, min-cut
  - ▶ General problem: community detection
- ▶ Some really useful **data structures** in computer science resemble clustering.
  - ▶ KD-trees are a binary splitting method for  $\mathbb{R}^d$
  - ▶ They partition the space using the specified points
  - ▶ See also Quadtree, R-tree, etc.
  - ▶ They solve lookup problems; for example, fast recall of approximate nearest-neighbours.

# Hierarchical clustering

This comes in two flavours:

- ▶ **Divisive clustering**: start with all objects in a single cluster and split them;
- ▶ **Agglomerative clustering**: start with all objects in a different cluster and merge them.
- ▶ In general divisive clustering is harder to “get right” so we focus on agglomerative methods. Broadly, these:
  1. start with  $N$  clusters  $c_i$ ; defined by the original points
  2. choose the closest two clusters  $a$  and  $b$  to **merge** based on a distance measure  $d_{ab}$
  3. **update** the locations and hence the **distances** of the clusters according to some rule.



# Distances

- ▶ The choice of distance is very important for clustering. Here are some common ones:

Model	Norm	Equation
Euclidean	$\ x - y\ _2$	$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
Squared Euclidean	$\ x - y\ _2^2$	$\sum_{i=1}^n (x_i - y_i)^2$
Manhattan	$\ x - y\ _1$	$\sum_{i=1}^n  x_i - y_i $
Maximum	$\ x - y\ _\infty$	$\max_i  x_i - y_i $
Mahalanobis	$\ x - y\ _M$	$[(\vec{x} - \vec{y})C^{-1}(\vec{x} - \vec{y})^T]^{1/2}$

- ▶ Note the connection of the Mahalanobis norm to PCA<sup>1</sup>!
- ▶ See also: Hamming Distance (for binary variables), edit distance, etc.

---

<sup>1</sup>“The squared Mahalanobis distance is equal to the sum of squares of the scores of all non-zero standardised principal components.”

# Metrics and related objects

- ▶ Distances  $d : X \times X \rightarrow [0, \infty)$  are a **Metric** and satisfy:
  - ▶  $d(x, y) = d(y, x)$ : symmetry
  - ▶  $d(x, y) \geq 0$ : non-negativity
  - ▶  $d(x, y) = 0 \Leftrightarrow x = y$ : (the distance is only zero if the elements are the same)
  - ▶  $d(x, z) \leq d(x, y) + d(y, z)$ : Triangle inequality
- ▶ Some methods can work with **divergences**, which need not satisfy symmetry or the Triangle inequality.
- ▶ If instead  $d(x, z) \leq \max(d(x, y), d(y, z))$  the  $d$  is called **ultrametric**. This is important for certain types of tree.

# Hierarchical clustering

- ▶ Hierarchical clustering methods report **trees** as their output.
- ▶ We select the threshold  $k$  (a “tree cut”) to select the number of clusters
- ▶ Many criteria exist to do this selection in an automated way:
  - ▶ Within-vs Between cluster variation<sup>2</sup>
  - ▶ Gap statistic<sup>3</sup>
  - ▶ etc . . .
  - ▶ Why not use **Cross validation**!

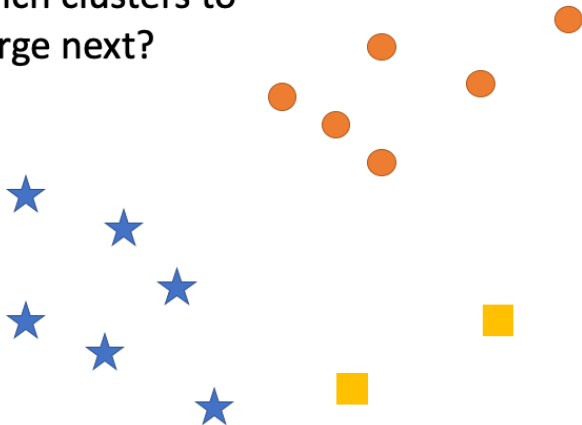
---

<sup>2</sup>Calinski and Harabasz (1974), “A dendrite method for cluster analysis”

<sup>3</sup>Tibshirani et al. (2001), “Estimating the number of clusters in a data set via the gap statistic”

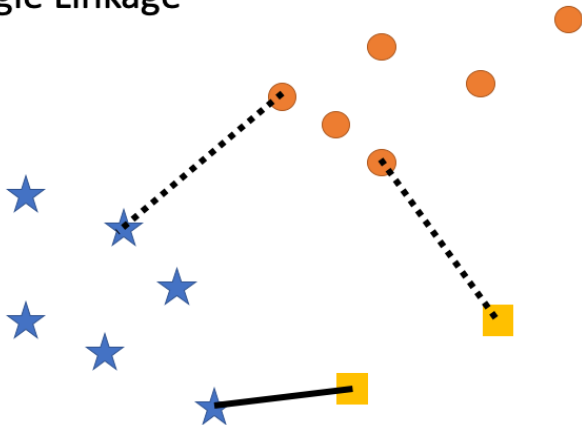
## Linkage clustering

Which clusters to merge next?



# Single linkage clustering

## Single Linkage



# Single linkage clustering

- ▶ Hierarchical clustering where we set

$$d_{a,b} = \min_{i \in a, j \in b} d_{i,j}$$

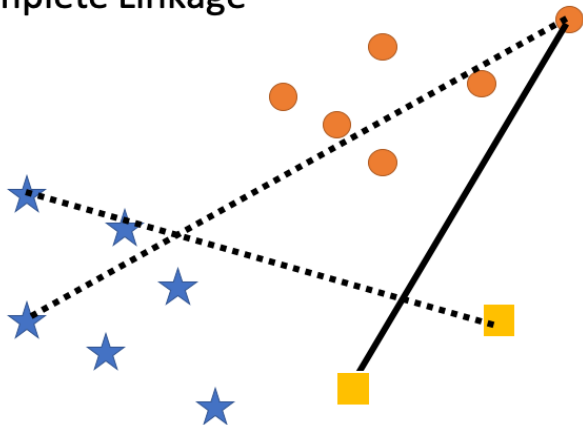
- ▶ i.e. the distance is the **closest point** in each cluster.
- ▶ The naive implementation would take  $O(N^3)$ .
- ▶ Good implementations are  $O(N^2)$  (e.g. SLINK, 1973)<sup>4</sup>, Kruskal's algorithm for minimum spanning trees.

---

<sup>4</sup>Sibson 1973, "SLINK: An optimally efficient algorithm for the single-link cluster method".

# Complete linkage clustering

## Complete Linkage



# Complete linkage clustering

- Hierarchical clustering where we set

$$d_{a,b} = \max_{i \in a, j \in b} d_{i,j}$$

- i.e. the distance is the **furthest point** in each cluster.
- The naive implementation would take  $O(N^3)$ .
- Good implementations are  $O(N^2)$  (CLINK, 1977)<sup>5</sup>.

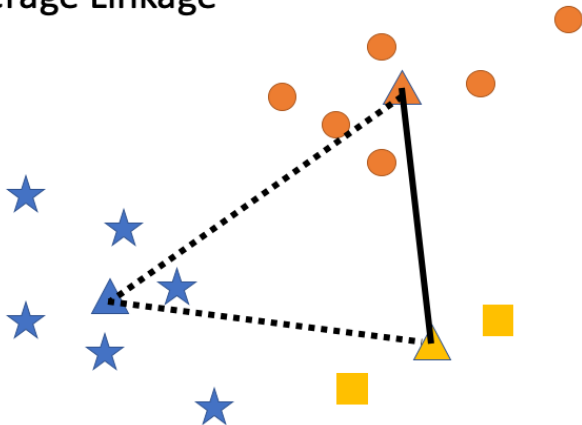
---

<sup>5</sup>Defays 1977, "An efficient algorithm for a complete link method".



## Average linkage clustering

### Average Linkage



# Average linkage clustering

- ▶ Also known as “Unweighted Pair Group Method with Arithmetic mean” (UPGMA).
- ▶ Hierarchical clustering where we set

$$d_{a,b} = \mathbb{E}_{i \in a, j \in b}(d_{i,j})$$

- ▶ i.e. the distance is the **average distance** between each cluster.
- ▶ The naive implementation would take  $O(N^3)$ .
- ▶ Good implementations are  $O(N^2 \log(N))$ .
- ▶ It can be “meaningful”:
  - ▶ the recovered tree is the “true tree” if the clusters diverged at constant rate.
  - ▶ This is plausible in evolution, for example.

## Hierarchical Clustering: See also

- ▶ **Centroid** Linkage: Define centres of each cluster, compute distance to cluster centres
- ▶ **Minimax** Clustering<sup>6</sup> : Minimise the maximum radius to the centre of each group
- ▶ NB: Minimax is an important concept in Machine Learning!

---

<sup>6</sup>Bien et al. (2011), "Hierarchical Clustering with Prototypes via Minimax Linkage"

# Implementations in R

```
library("hclust") # default hierarchical clustering  
library("fastcluster") # faster implementations
```

- Implementations are important for computational complexity and speed<sup>7</sup>

---

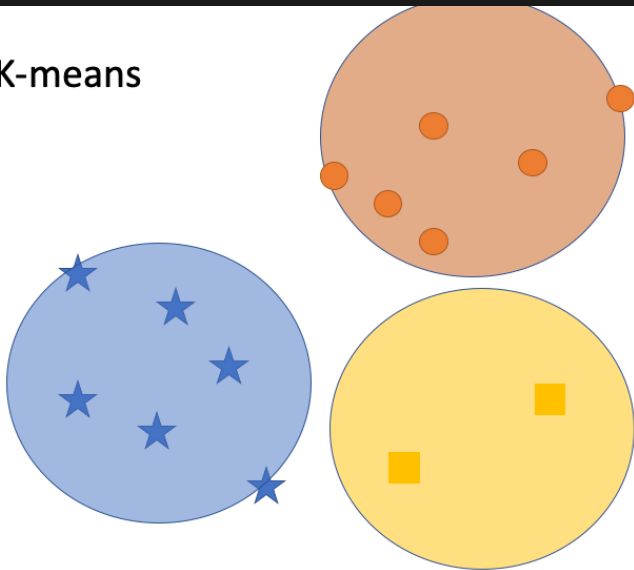
<sup>7</sup><http://danifold.net/fastcluster.html?section=1>

# K-means clustering

- ▶ Probably the **most widely used** clustering algorithm.
- ▶ Randomly (or otherwise) **initialise**  $K$  locations as initial cluster means  $\mu_k$
- ▶ Iteratively, until convergence:
  1. **Assign each sample**  $x_i$  to its closest cluster
$$c(x_i) = \min_k d(x_i, \mu_k)$$
  2. **Set each cluster mean** to the mean of its members
$$\mu_k = \frac{1}{n_k} \sum_{i:c(x_i)=k} x_i$$
- ▶ In practice, we:
  - ▶ Use a large number of starting values
  - ▶ Use “intelligent” initial guesses
- ▶ Computational complexity (per clustering) is  $O(N^2)$  but getting convergence is harder.
  - ▶ Approximate  $O(N)$  algorithms exist.

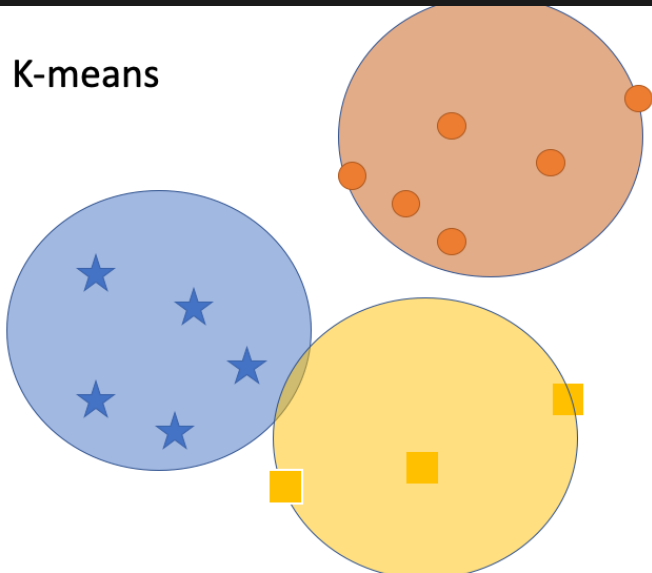
# K-means clustering

K-means



# K-means clustering

K-means



# Beyond K-means

- ▶ Soft K-means: replace assignment with cluster probabilities.
  - ▶ Typically better convergence than hard K-means.
- ▶ **K-means assumes that clusters are spherical.**
  - ▶ This might work when clusters are well-separated or the data scaled in the right way.
  - ▶ Sometimes high dimensionality makes this more plausible.
- ▶ Gaussian Mixture Modelling (GMM) allows ellipsoid clusters to be fit instead.
- ▶ GMMs are a more general class of model than K-means and therefore perform **uniformly better** when used correctly
  - ▶ There are model selection issues, resolved by CV or information criteria (BIC)



# Expectation Maximization

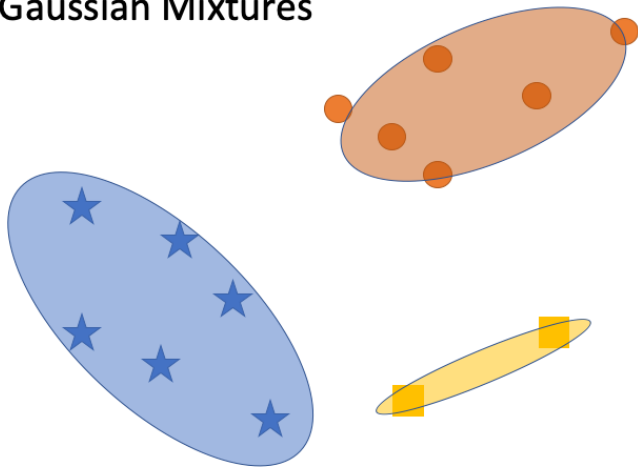
- ▶ The Expectation-Maximization (EM) is an optimization tool for problems with a latent parameter  $\mathbf{Z}$  of the form:

$$L(\theta, \mathbf{X}) = p(\mathbf{X}|\theta) = \int p(\mathbf{X}, \mathbf{Z}|\theta) d\mathbf{Z}$$

- ▶ Where we wish to maximise the Likelihood  $L(\theta, \mathbf{X})$  with respect to  $\theta$ , marginalising out  $\mathbf{Z}$ .
- ▶ In soft K-means,  $\mathbf{Z}$  is the probability of belonging to each cluster;  $\theta$  is the location of the clusters.
- ▶ EM solves this by iteratively:
  - ▶ Computing the **Expected value** of the latent  $\mathbb{E}(\mathbf{Z}|\theta)$ ,
  - ▶ Computing the **Maximum likelihood** estimate  $p(\mathbf{X}, \mathbf{Z}|\theta)$ .
- ▶ EM provably always improves  $L(\theta, \mathbf{X})$ .

# Gaussian Mixture Modelling

## Gaussian Mixtures



# Gaussian Mixture Modelling

- ▶ Randomly **initialise**  $K$  locations as initial cluster means  $\mu_k$ , each with an initial covariance  $\Sigma_k$  (can just be spherical)
- ▶ Iteratively, until convergence:
  1. Compute the **density of each cluster** at each point
$$d_{ik} = K_k(x_i | \mu_k, \Sigma_k)$$
  2. Compute the **probability** of each cluster for each point:
$$p_{ik} = d_{ik} / \sum_{k'} d_{ik'}$$
  3. **Update the cluster** parameters accounting for the probabilistic memberships
- ▶ In practice, we still want to:
  - ▶ Use several starting values
  - ▶ Use “intelligent” initial guesses
- ▶ probabilistic assignment speeds convergence over K-means
- ▶ **Computational complexity** is  $O(N^2)$ , though the constant is larger than for K-means. What is the dependency on  $K$ ?

# Gaussian Mixture Modelling

- ▶ GMMs work very well on a range of problems.
- ▶ However, choosing  $\Sigma$  and  $K$  can be awkward
- ▶ One solution is to use a (semi)Bayesian paradigm:
  - ▶ Fit the clusters using EM as in regular GMMs
  - ▶ Use Bayesian Model selection (BIC) to choose a model for  $\Sigma$  and select  $K$
  - ▶  $\Sigma$  choices: ellipsoid vs circular, volume, shape, orientation
  - ▶ Changes the dimension of  $\Sigma$ , hence affects BIC
- ▶ This isn't reliable **model selection** for whether GMM is appropriate, but it is good selection for what shape  $\Sigma$  to use
- ▶ In R: `library(mclust)`

## Example: K-means clustering

- ▶ Run K-means clustering on the whole example dataset:

```
km.all.raw=lapply(1:10,function(i){  
  km=kmeans(testdata_all_scaled,centers=i,nstart=10)  
})
```

## Example: K-means clustering

- ▶ Run K-means clustering on the whole example dataset:

```
km.all.raw=lapply(1:10,function(i){  
  km=kmeans(testdata_all_scaled,centers=i,nstart=10)  
})
```

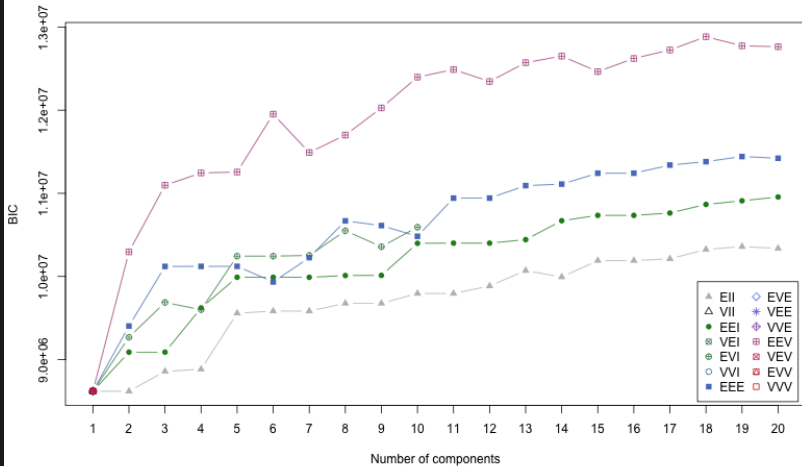
- ▶ **Spectral clustering** just means running the same clustering algorithm on the top PCs in a PCA/SVD

```
km.all.svd=lapply(1:10,function(i){  
  km=kmeans(testdata_all.svd$u,centers=i,nstart=10)  
})
```

## Example: GMM using mclust

```
library("mclust")
mc.all=mclustBIC(testdata_all.svd$u,G=1:20)
# mclustBIC Compares lots of models
mc.assignments=lapply(1:20,function(i){
  tmp=mclustModel(testdata_all.svd$u,mc.all,G=i)
  apply(tmp$z,1,which.max)
}) # extract the results for the best models
```

# Example: GMM using mclust: diagnostics





# DBSCAN

- ▶ “Density-Based Spatial Clustering of Applications with Noise”<sup>8</sup>.
- ▶ Clusters arbitrary shapes that are above some threshold density.
- ▶ Uses **K-Nearest-Neighbours** (next session) to approximate density.
  - ▶ “dense” points have many close neighbours, “outliers” have few
- ▶ Uses **KD-trees** to efficiently approximate k-NN calculation.
  - ▶ changes complexity from  $O(N^2)$  to  $O(N \log(N))$ ; nb relatively slow still as have to do this multiple times...
- ▶ **Overview**: Initialise: Assign a cluster to each “dense” point.  
Then iterate:
  1. All neighbours of a cluster are also in that cluster
  2. Merge joined clusters
  3. Update neighbours of each cluster

---

<sup>8</sup>Kriegel, Hans-Peter, Sander & Xu (1996). “A density-based algorithm for discovering clusters in large spatial databases with noise”

# HDBSCAN

- ▶ DBSCAN is limited because all clusters have to have the same minimum density threshold
- ▶ This sometimes leads to clusters being ignored as noise
- ▶ Many variants exist to address this
- ▶ One of the most important is **HDBSCAN**<sup>9</sup> : An extension of DBSCAN allowing variation in density across clusters

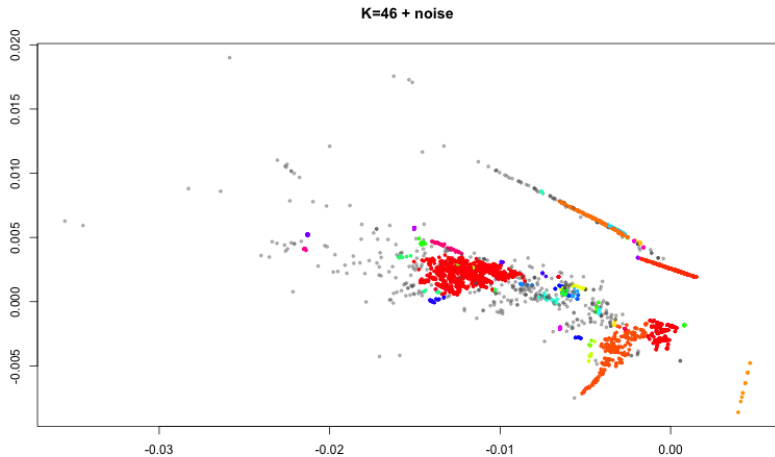
---

<sup>9</sup>McInnes & Healy (2017), “Accelerated Hierarchical Density Based Clustering”

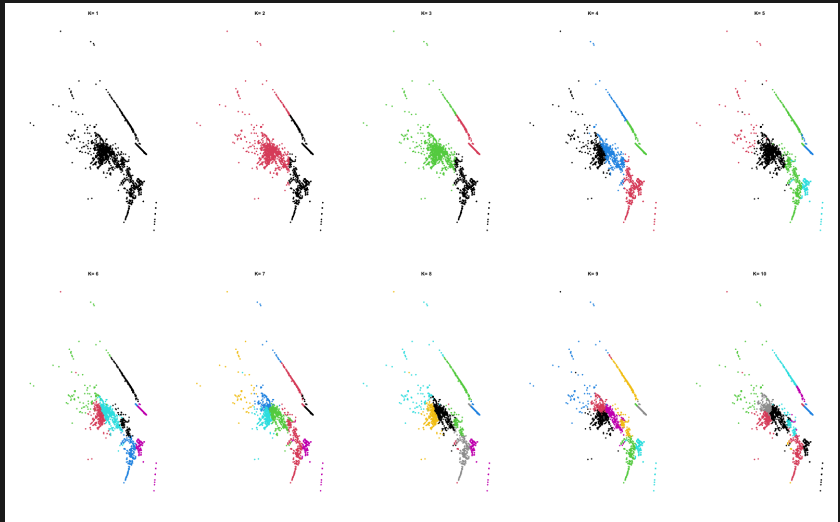
## Example: DBSCAN in R

```
library("dbscan")  
# Hardest part is choosing the threshold  
test=kNNdist(testdata_all.svd$u, k = 5)  
testmin=apply(test,1,min)  
plot(sort(testmin[testmin>1e-8]),log="xy")  
abline(h=0.001) # we chose  
abline(h=0.01) # would give bigger clusters  
abline(h=0.0001) # would give smaller clusters  
kNNdistplot(testdata_all.svd$u, k = 5)  
## This is actually running it (quite slow)  
dbscanres=dbscan(testdata_all.svd$u,0.001)
```

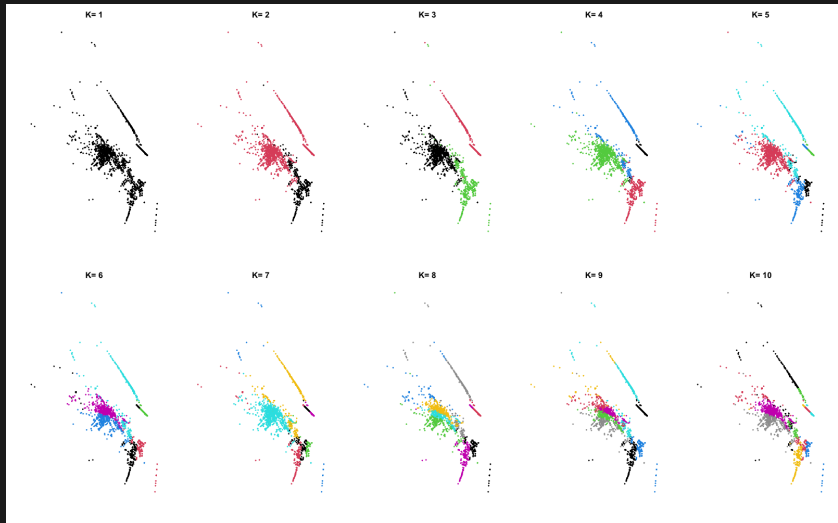
## Example: DBSCAN clustering



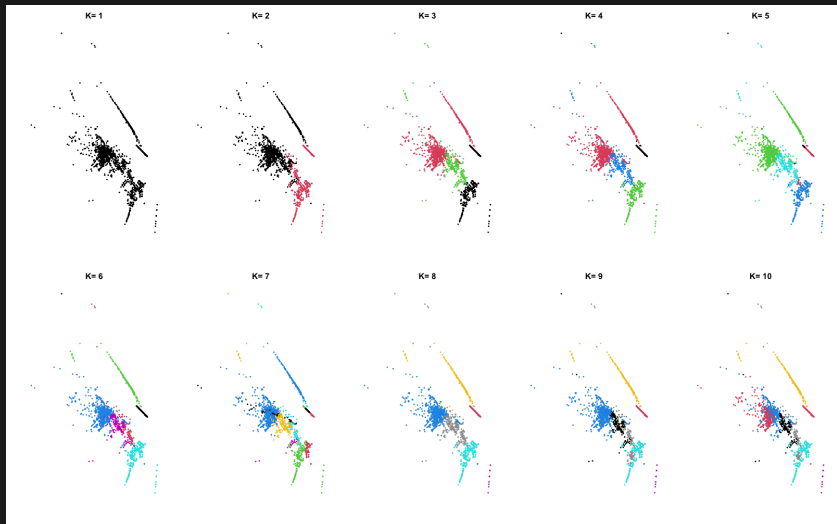
# Example: K-means clustering



# Example: K-means spectral clustering



# Example: GMM spectral clustering



## Example: generating the plots

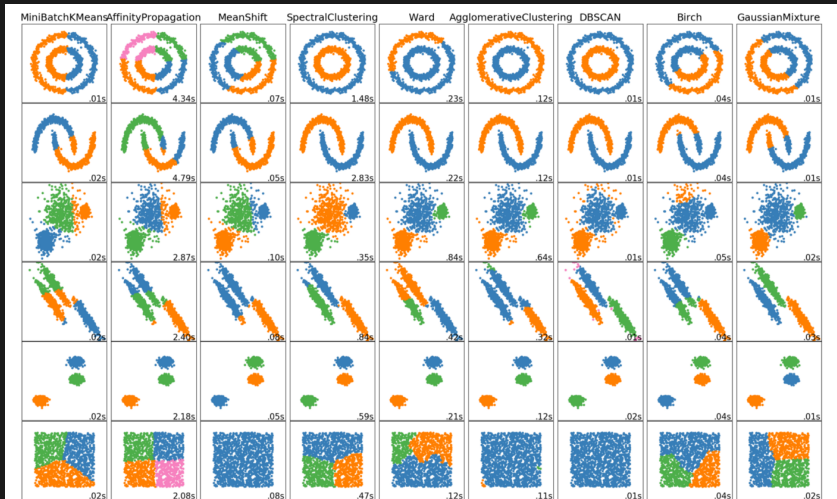
```
png(paste0("../media/03.2.5-Clustering_kmeans_svd.png"),  
    height=1000,width=1600)  
par(mfrow=c(2,5))  
for(i in 1:10){  
    plot(testdata_all.svd$u[,1],  
         testdata_all.svd$u[,2],xlab="",axes=F,  
         ylab="",  
         col=km.all.svd[[i]]$cluster,pch=19,cex=0.5)  
    title(main=paste("K=",i),cex.main=2)  
}  
dev.off()
```



## Important extensions: How many clusters, really?

- ▶ Any **model selection** approach can allow selection of the number of clusters.
- ▶ When the **model is supposed to be true** then careful model selection is important. The usual model selection rules apply.
- ▶ When the **model is for convenience** then the clustering is just a tool for understanding.
  - ▶ The number of clusters is a **tuning parameter** that can be chosen by convenience
  - ▶ **Sensitivity analysis** should be used to investigate whether it matters.

# Scikit Learn Diagram



# Reflection

- ▶ What is a cluster?
- ▶ When does it make sense to do clustering? When does it not?
- ▶ How does the scale of data interact with the choice of clustering algorithm?
- ▶ When might spectral clustering work, when direct clustering does not? And vice-versa?
- ▶ By the end of the course, you should:
  - ▶ Be able to describe the key approaches to clustering
  - ▶ Be able to interpret common hierarchical clustering algorithms
  - ▶ Be able to reason about the appropriate clustering algorithm for a particular problem

# Further Reading

- ▶ References:

- ▶ Tibshirani's Data Mining lecture notes (Lecture 2 and Lecture 5)
- ▶ 5 clustering algorithms you need to know
- ▶ The fastcluster packages for R and python implements “fastest”  $O(N^2)$  versions of hierarchical clustering.
- ▶ Python resources comparing hdbscan