Clustering Part 2

Daniel Lawson University of Bristol

Lecture 03.2.2 (v1.0.2)

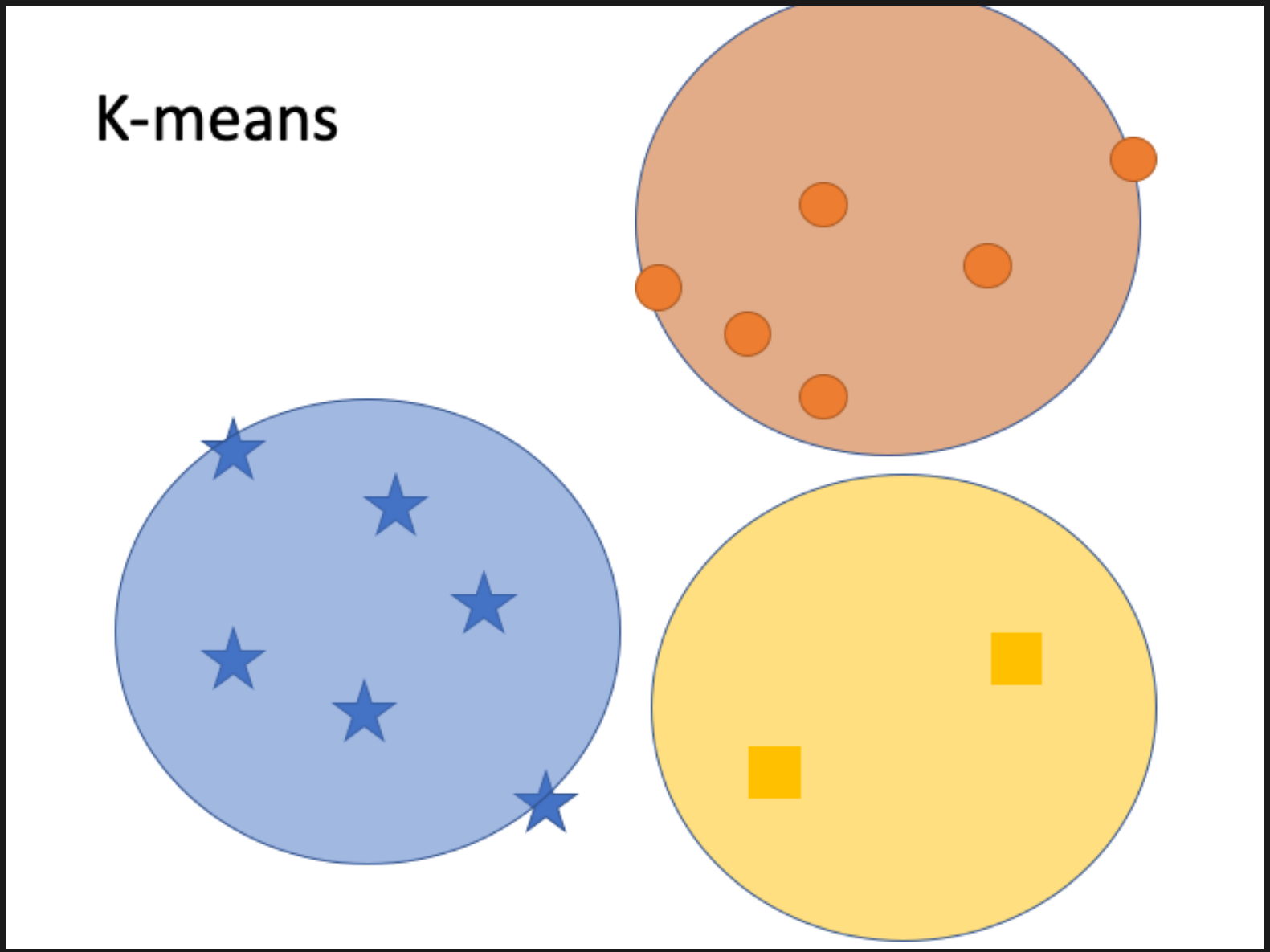
Signposting

- In Part 1 we covered:
  - How Clustering methods are organised,
  - Hierarchical clustering
- In Part 2 we cover:
  - K-means
  - Gaussian Mixture Modelling
  - Density-based model-free clustering (dbscan)

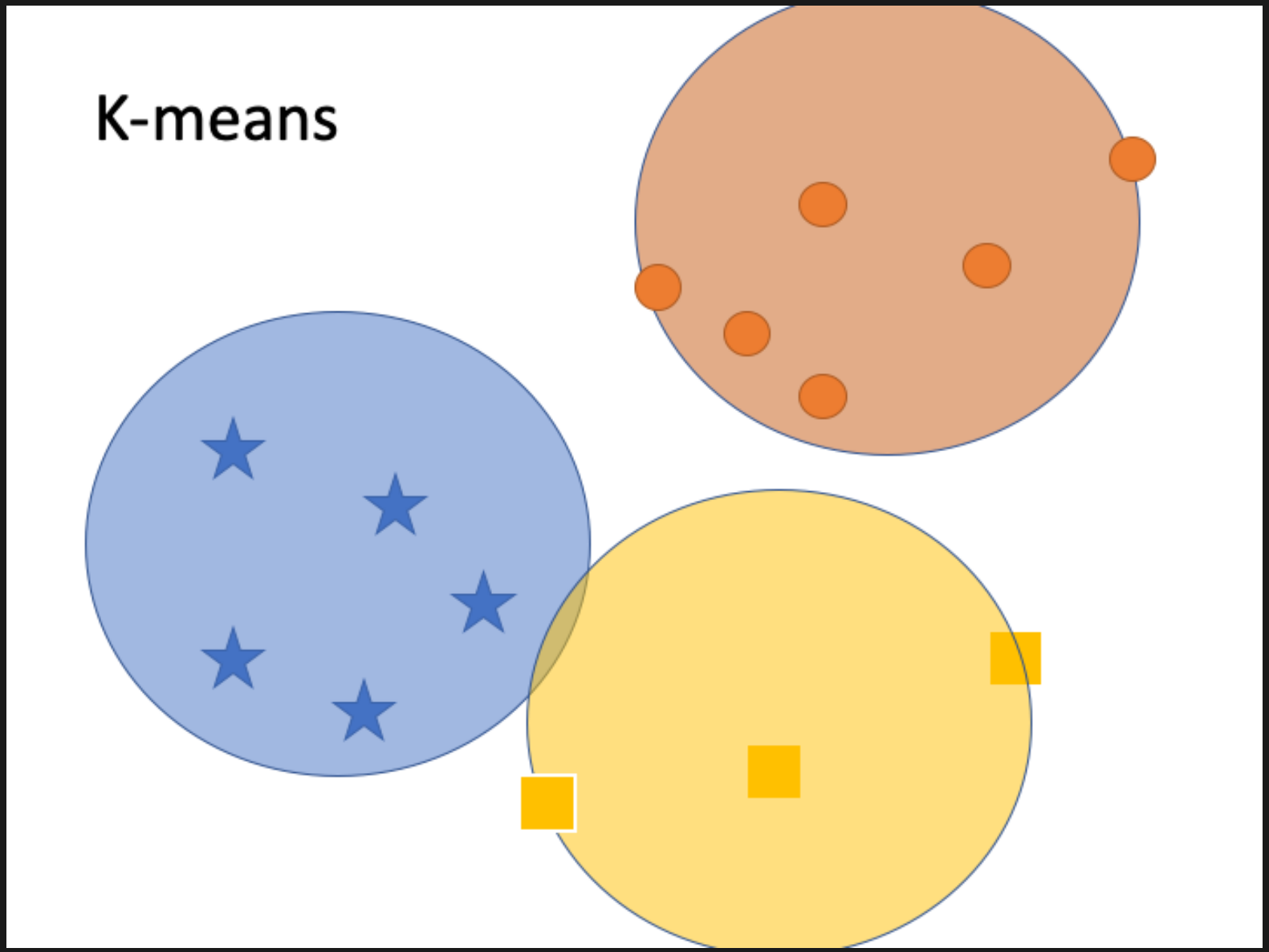
K-means clustering

- Probably the **most widely used** clustering algorithm.
- Randomly (or otherwise) **initialise** K locations as initial cluster means μ_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 clustering



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)

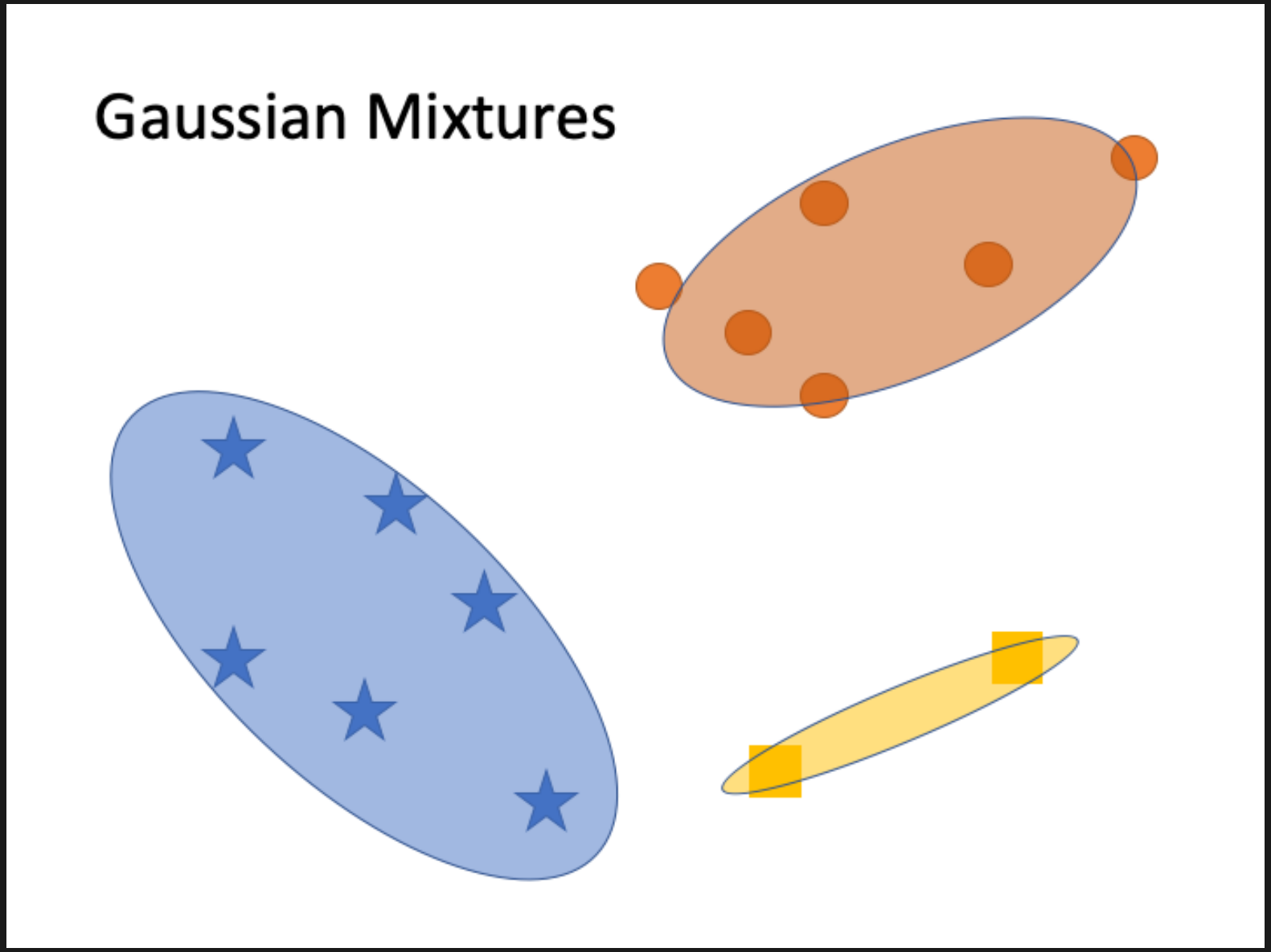
Expectation Maximization

- 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 θ, marginalising out $\mathbf{Z}$.
- In soft K-means, $\mathbf{Z}$ is the probability of belonging to each cluster; θ 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 Mixture Modelling

- Randomly **initialise** K locations as initial cluster means μ_k, each with an initial covariance Σ_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 Σ 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 Σ and select K
  - Σ choices: ellipsoid vs circular, volume, shape, orientation
  - Changes the dimension of Σ, hence affects BIC
- This isn't reliable **model selection** for whether GMM is appropriate, but it is good selection for what shape Σ 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)
})
```

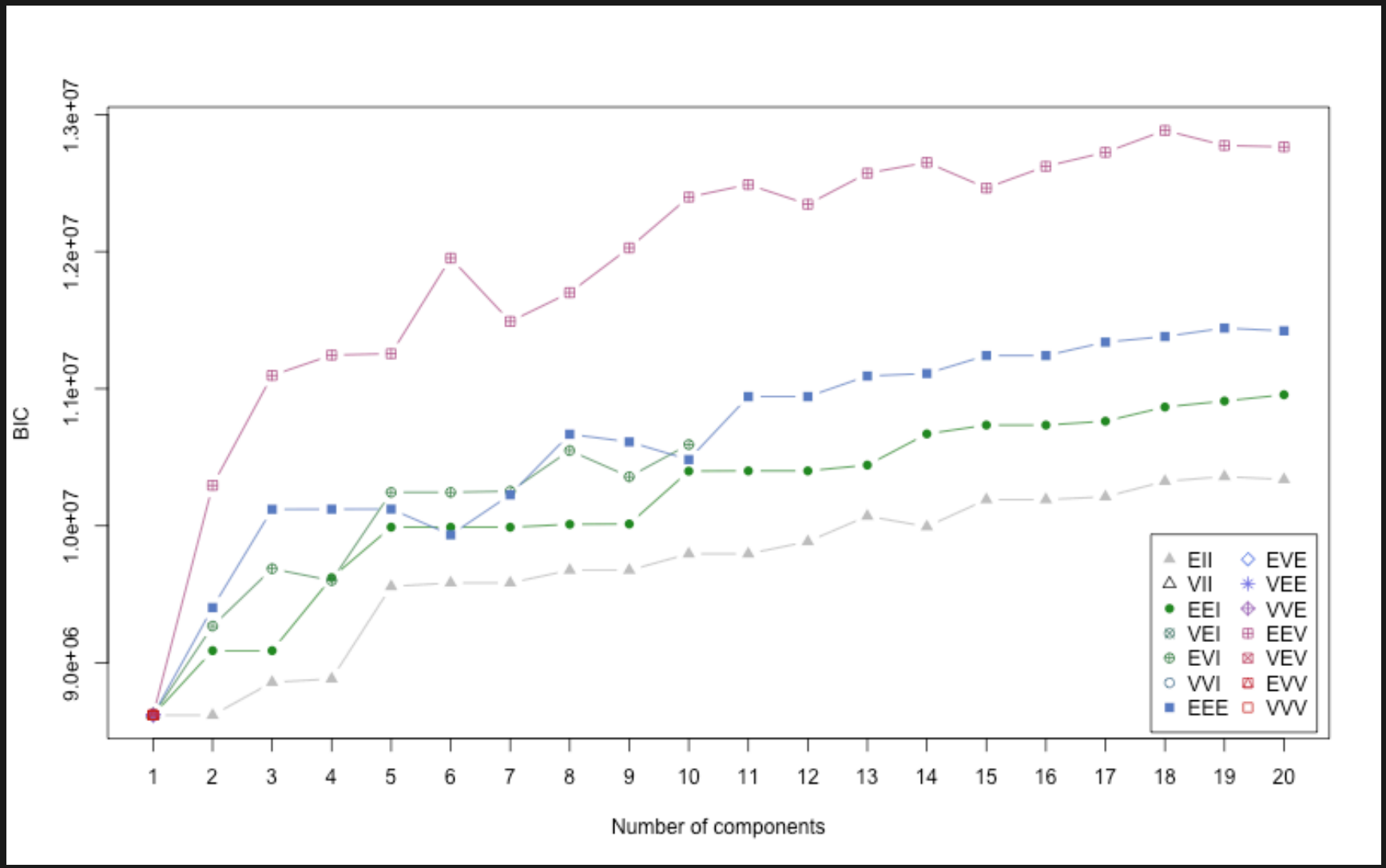
- **Spectral clustering** just means running the same clustering algorithm on the top PCs in a PCA/SVD

```
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"[1].
- 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

[1] Kriegel, Hans-Peter, Sander & Xu (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise"

HDBSCAN

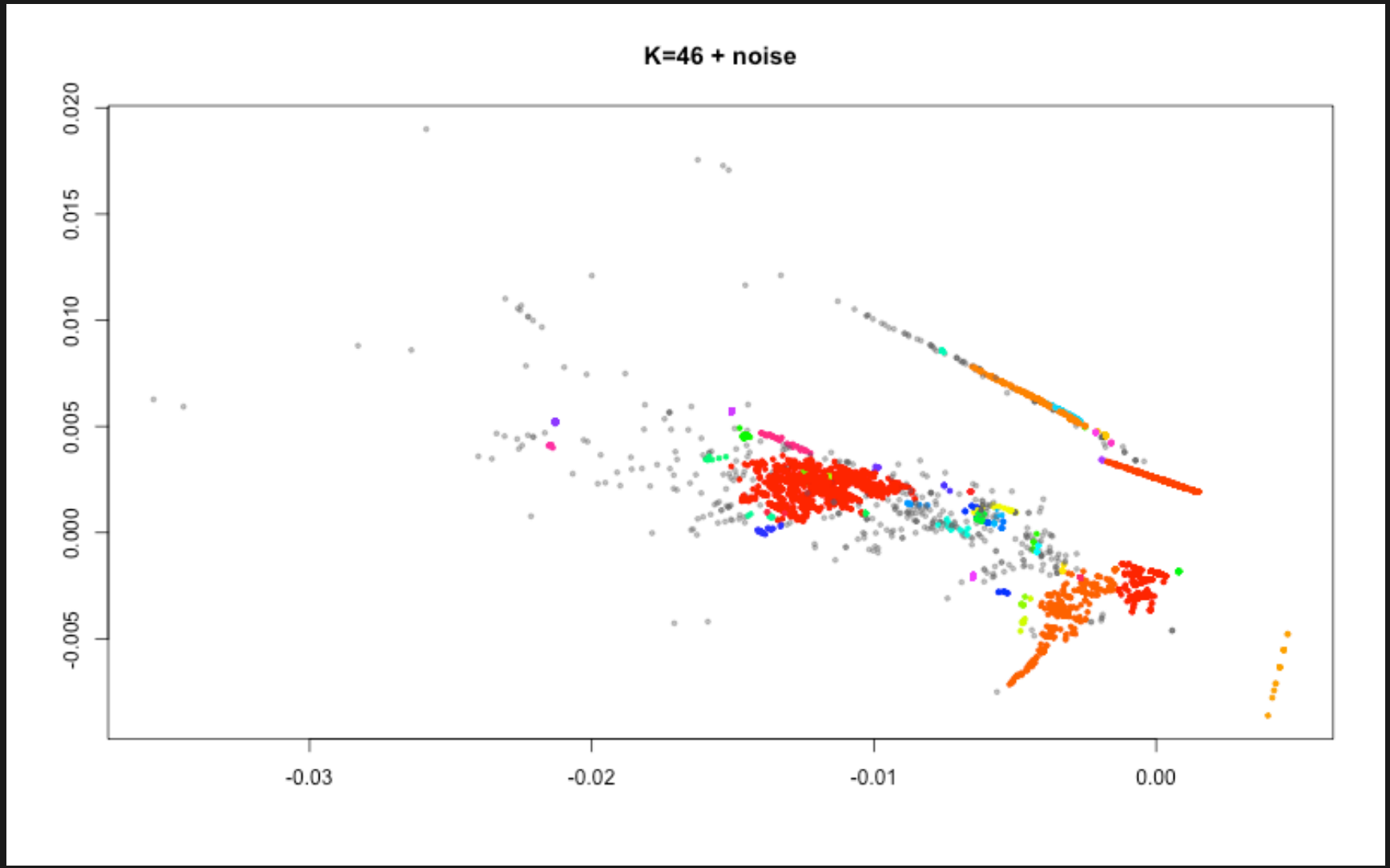
- DBSCAN is limited because all clusters have to have the same minimum density threshold
- This sometimes leads to clusters being ignored as noise
- Many variants exist to address this
- One of the most important is **HDBSCAN**[2] : An extension of DBSCAN allowing variation in density across clusters

[2] McInnes & Healy (2017), "Accelerated Hierarchical Density Based Clustering"

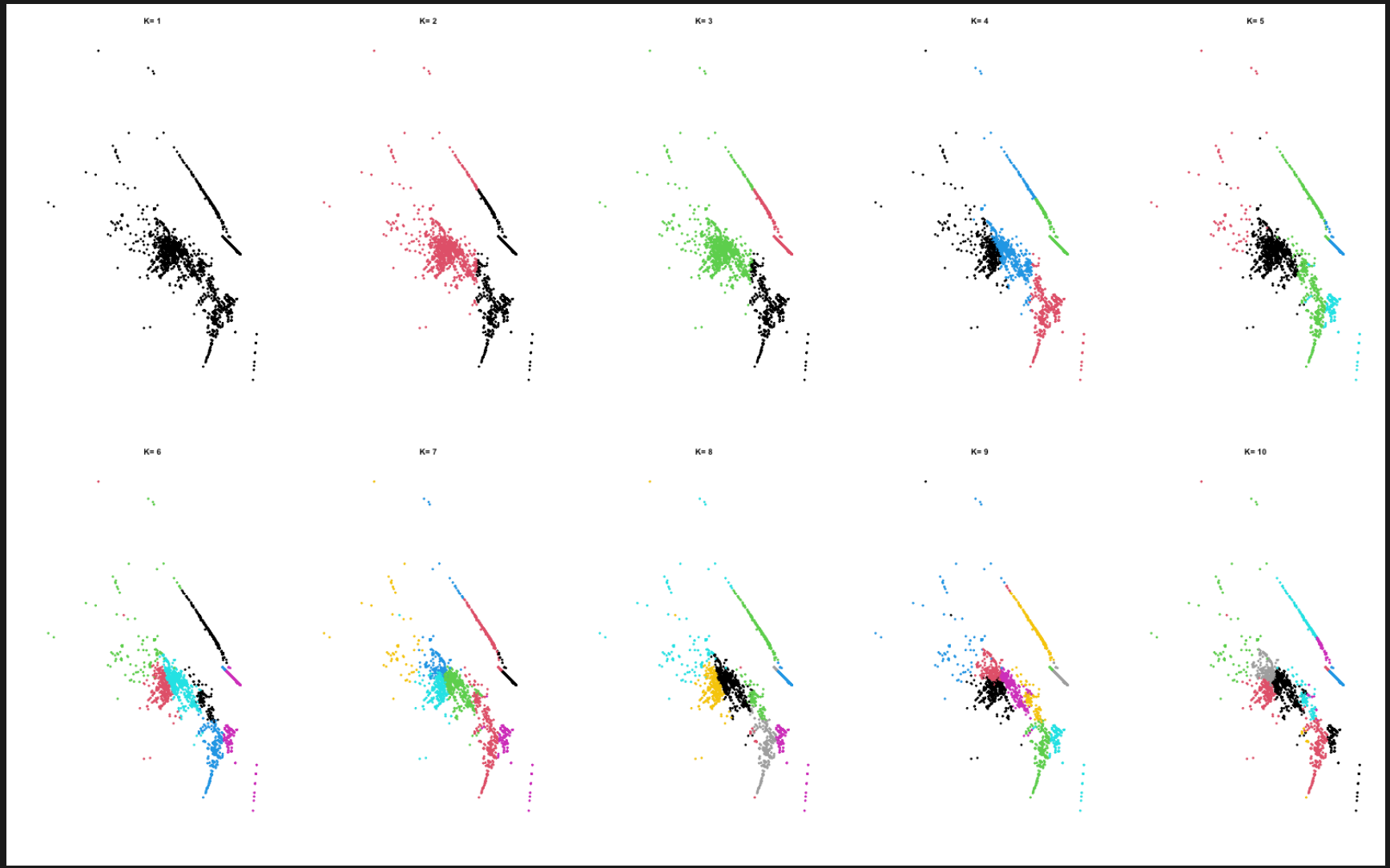
Example: DBSCAN in R

```
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)
```

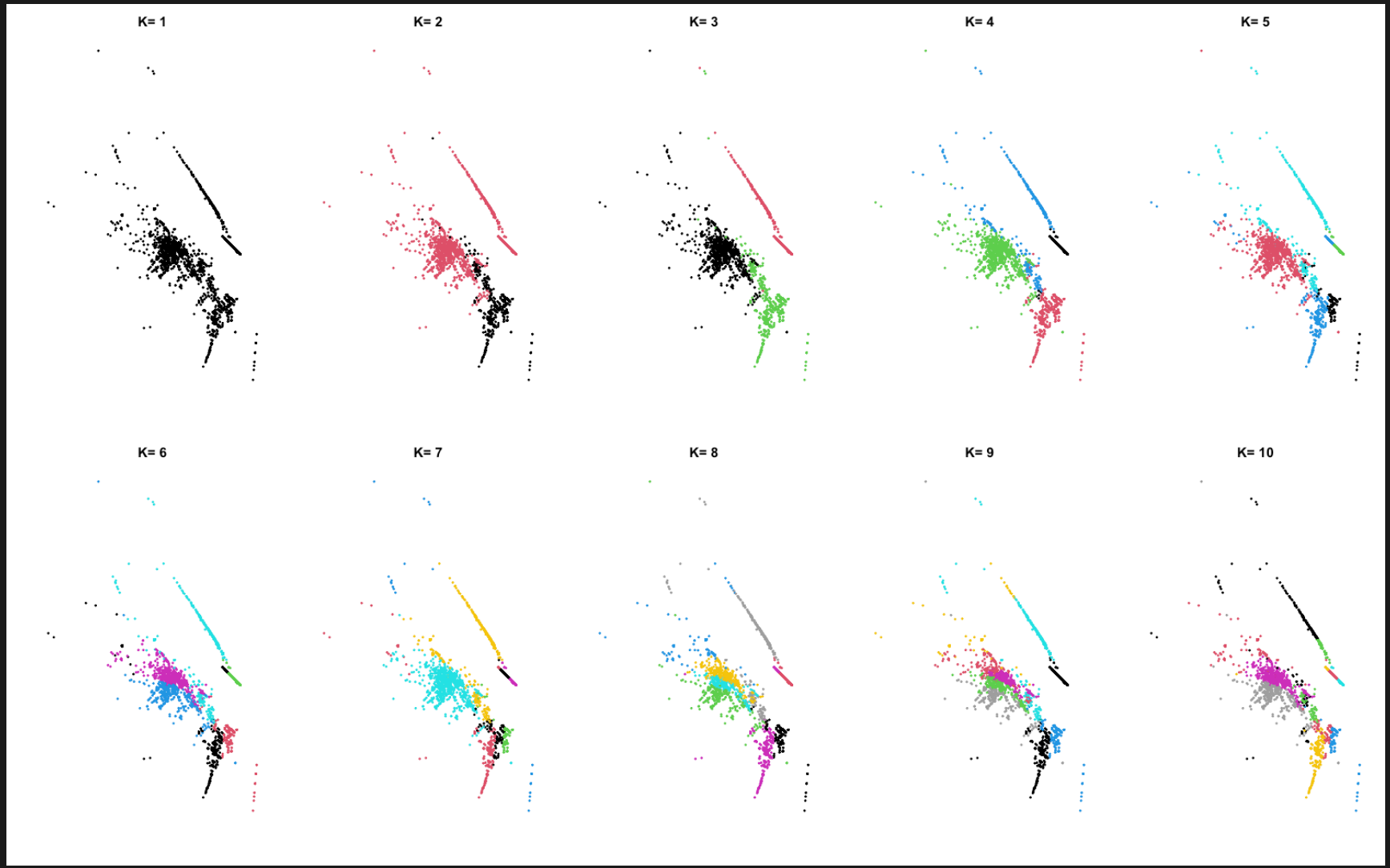
Example: DBSCAN clustering



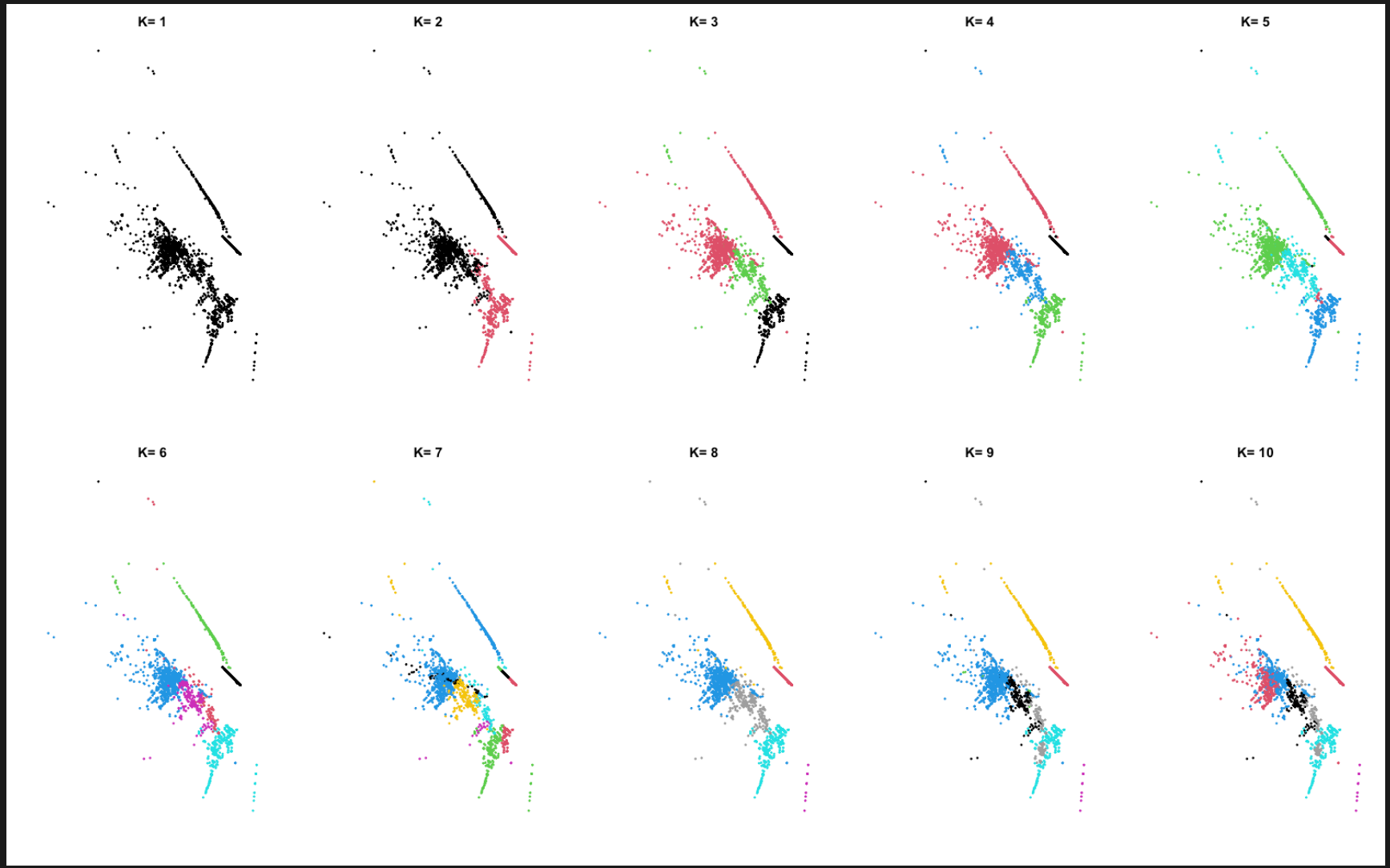
Example: K-means clustering



Example: K-means spectral clustering



Example: GMM spectral clustering



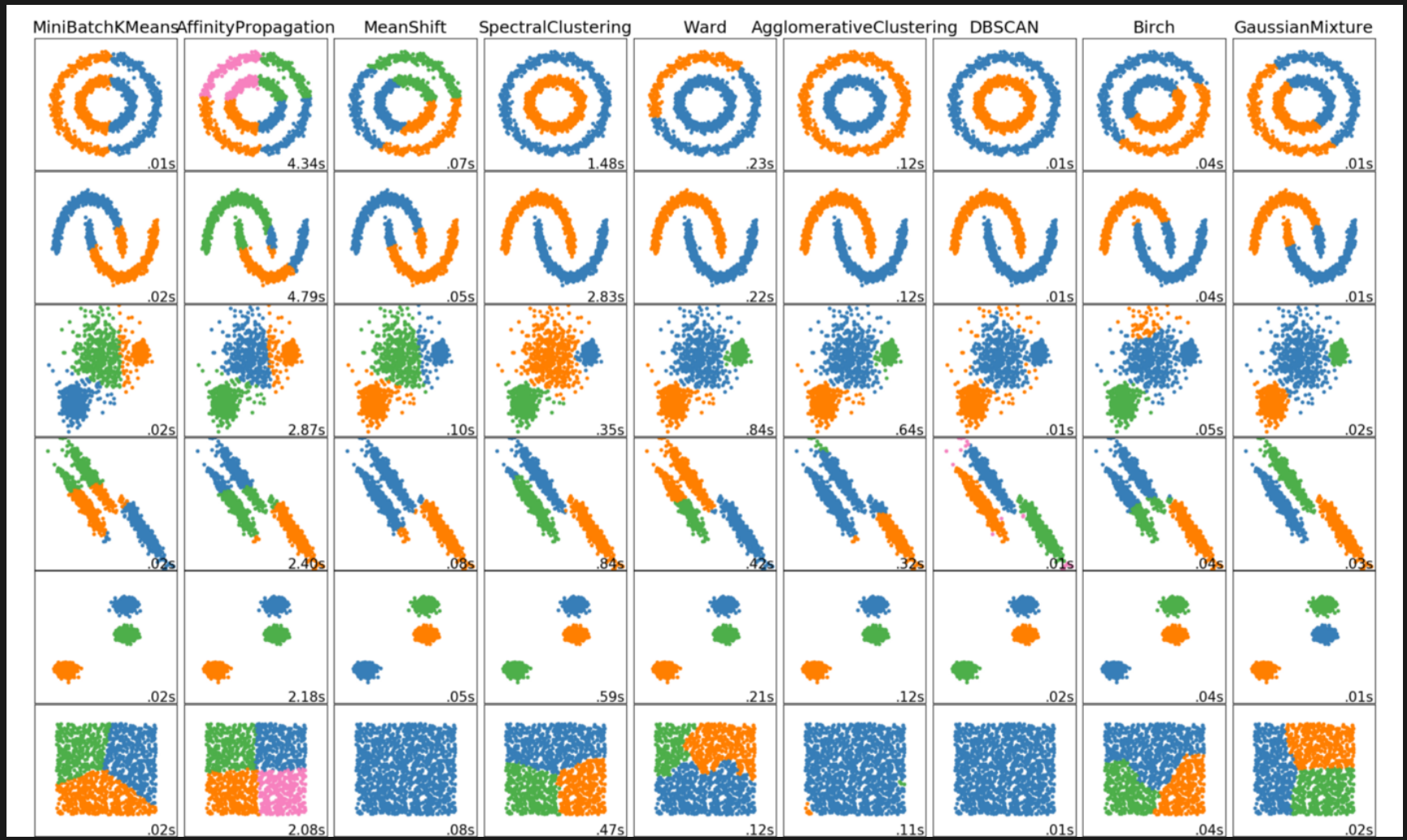
Example: generating the plots

```
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

Signposting

- There is a workshop associated with this lecture and PCA.
- Next week we cover nonparametric methods: transforms, kernel methods, and The Kernel Trick.
- References:
  - Tibsherani'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