Clustering

Vlad Gladkikh

IBS CMCM

Discover the underlying structure of the data

– unsupervised task, not predicting anything specific

What sub-populations exist in the data?

– how many are there?

– what are their sizes?

Do elements in a sub-population have any common properties?

Are sub-populations cohesive? Can they be further split up?

Are there outliers?

Clustering divides objects based on features.

A machine chooses the best way.

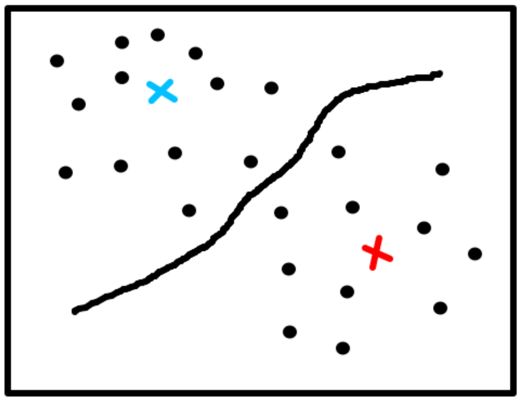
Clustering is a classification with no predefined classes.

A clustering algorithm is trying to find similar (by some features) objects and merge them in a cluster.

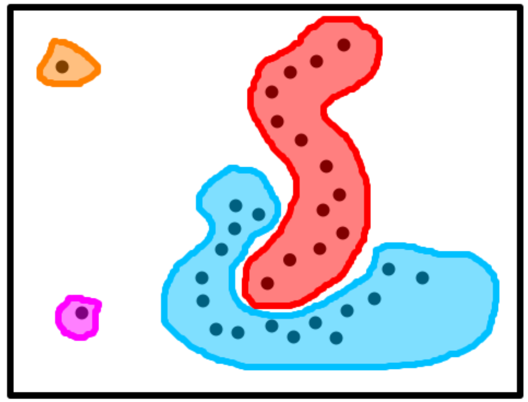
Those who have lots of similar features are joined in one class.

Basic types of cluster analysis

Centroid clustering



Density clustering



Hard clustering

Clusters do not overlap

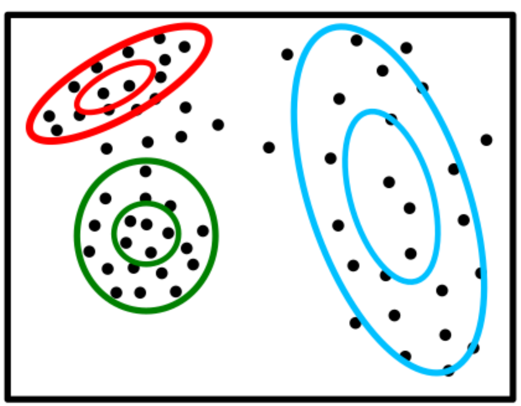
An element either belongs to a cluster or not

Soft clustering

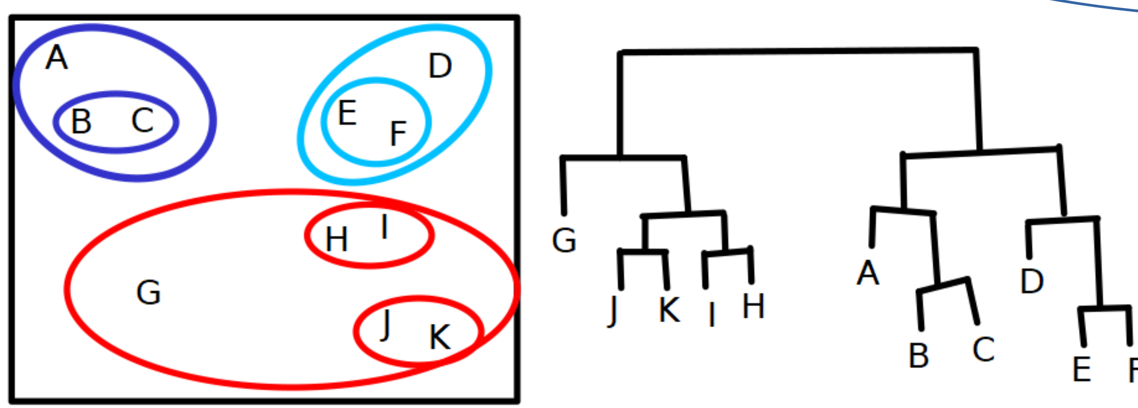
Clusters may overlap

Strength of association between element and cluster

Distribution clustering



Connectivity clustering



https://www.youtube.com/watch?v=Se28XHI2_xE

k-means

Input: k, set of points $x_1, \ldots, x_n$

Place centroids $c_1, \ldots, c_k$ at random locations

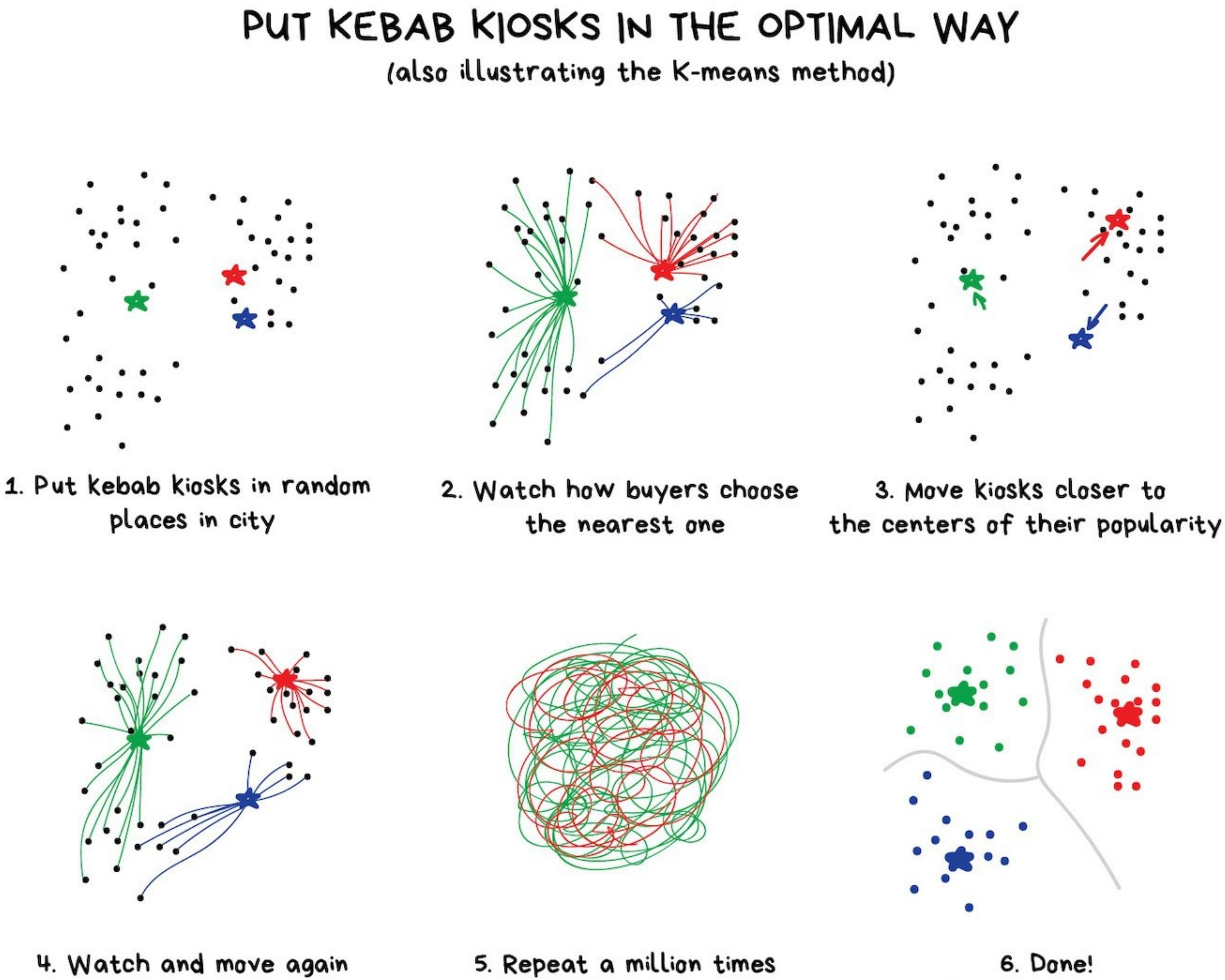
Repeat until convergence

$\forall x_i \; : \; \underset{j}{argmin}\, D(x_i, c_j)$ find nearest centroid

assign x_i to cluster j

$\forall j = 1, \ldots k: \; c_j = \frac{1}{n_j} \sum_{x_i \rightarrow c_j} x_i$ move the centroids to the centers of the clusters

https://www.youtube.com/watch?v=mHl5P-qlnCQ&list=PLBv09BD7ez_6cgkSUAqBXENXEhCkb_2wl



https://vas3k.com/blog/machine_learning/

k-means minimizes the aggregate intra-cluster distance

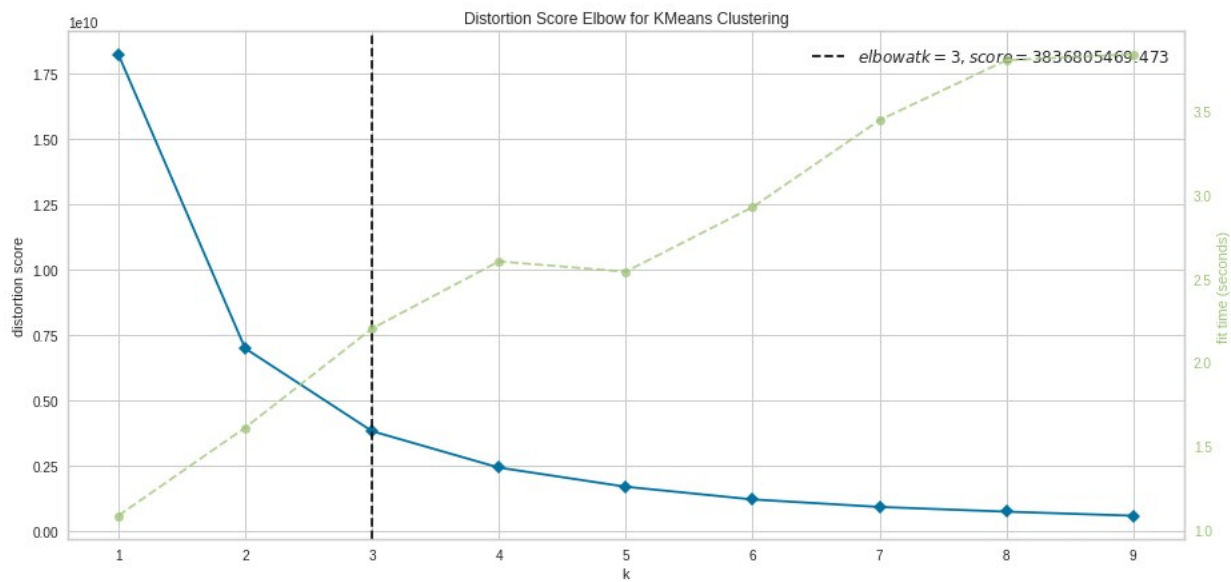
$$\sum_{j} \sum_{x_i \to c_j} D(c_j, x_i)^2$$

It converges to a local minimum

Nearby points may not end up in the same cluster (local min)

Different starting points lead to different results

Pick one that yields the smallest aggregate distance



https://www.coursera.org/learn/build-regression-classification-clustering-models

```
from yellowbrick.cluster import KElbowVisualizer

# Use the elbow method to find the optimal number of clusters.
plt.rcParams["figure.figsize"] = (15, 7)

visualizer = KElbowVisualizer(KMeans(init = 'k-means++'), k = (1, 10))
visualizer.fit(X)
visualizer.poof();
```

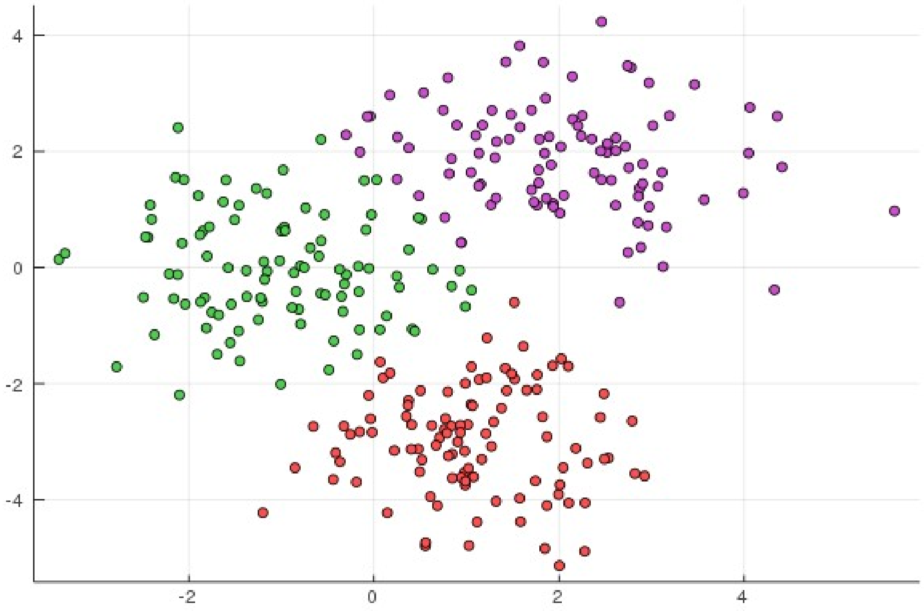
K-means in Julia

```
using Clustering, Plots
gr(fmt=:png);

r1 = (randn(2,100) .+ [-1,0]);
r2 = (randn(2,100) .+ [2,2]);
r3 = (randn(2,100) .+ [1,-3]);

X = [r1 r2 r3];
k = 3;       # number of clusters
r = kmeans(X, k);

@assert nclusters(r) == 3 # verify the number of clusters

counts(r) # get the cluster sizes
```

```
3-element Array{Int64,1}:
  98
  99
 103
```

```
r.centers # get the cluster centers
```

```
2×3 Array{Float64,2}:
 2.04066  -1.02762     1.05792
 1.88428  -0.0389211  -2.99537
```




```
scatter(X[1,:], X[2,:], marker_z=r.assignments, color=:lightrainbow, legend=false)
```

Silhouettes

– evaluates the quality of clustering

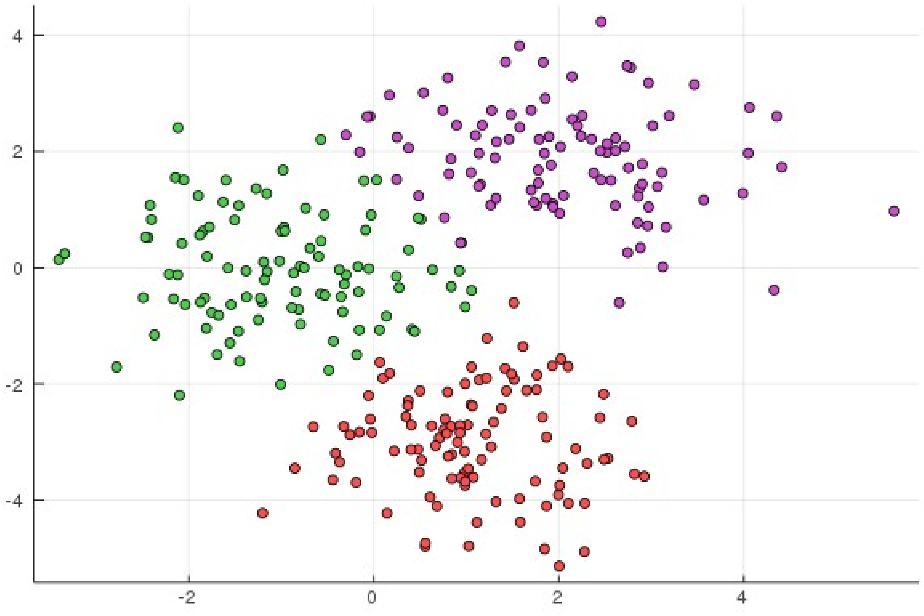
– measure how well each point lies within its cluster in comparison to the other clusters

```
using Distances, Statistics

dists = pairwise(Euclidean(), X, dims=2);

for k ∈ 2:10
    r = kmeans(X, k);
    slh = silhouettes(r, dists);
    println(k, " : ",  round(mean(slh),digits=3))
end
```

```
2 : 0.427
3 : 0.496
4 : 0.435
5 : 0.362
6 : 0.323
7 : 0.331
8 : 0.332
9 : 0.323
10 : 0.314
```

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)}$$

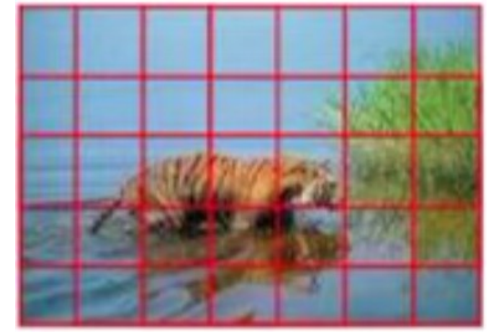
$$-1 \leq s_i \leq 1$$



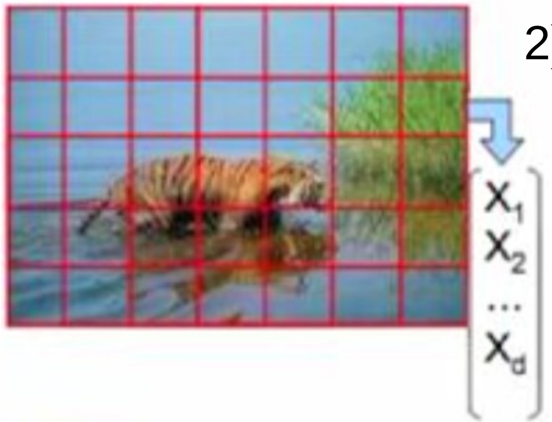
a_i – average distance from the *i*-th point to the other points in the same cluster

b_i – the smallest mean distance from the *i*-th point to the points in other clusters

Application: image representation https://www.youtube.com/watch?v=yDi2uX5tihc



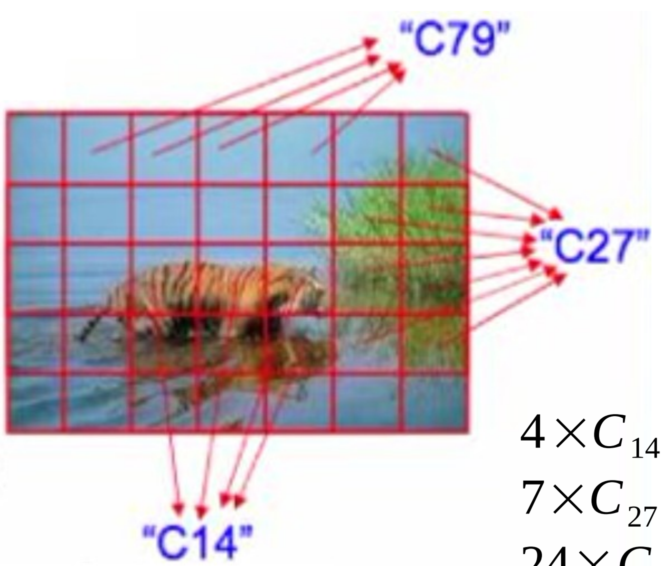
1) Partition an image using a rectangular grid



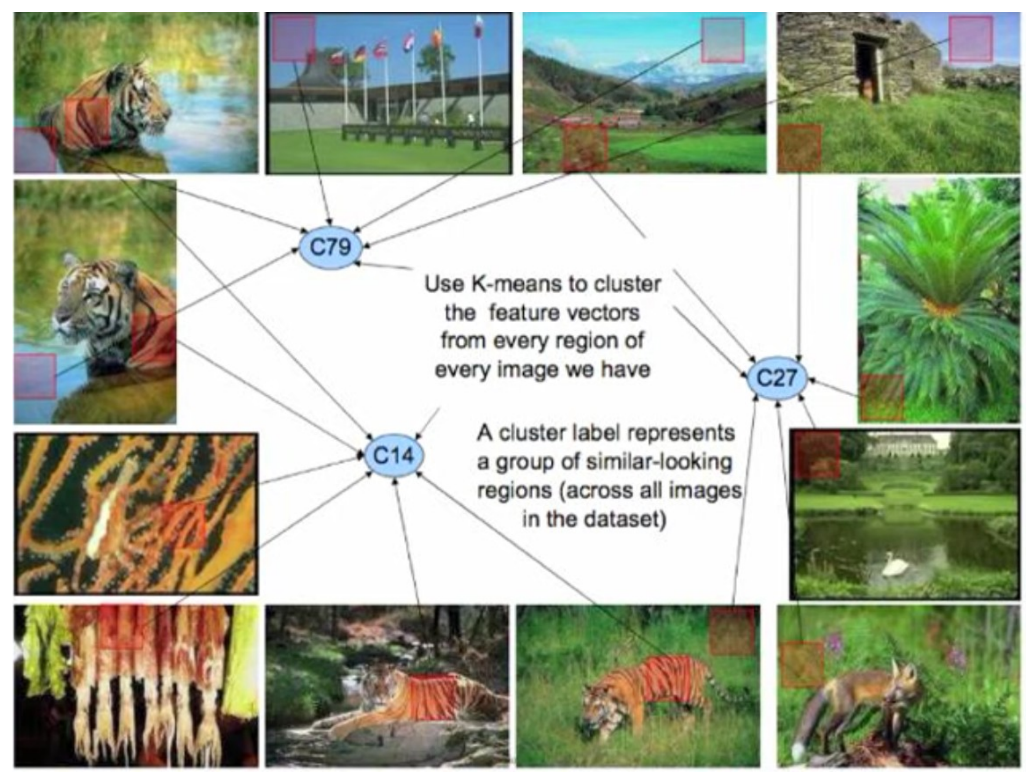
2) Compute a feature vector for each cell

distribution of colors, texture,
edge orientation, etc

3) Use *k*-means to assign each vector to a cluster

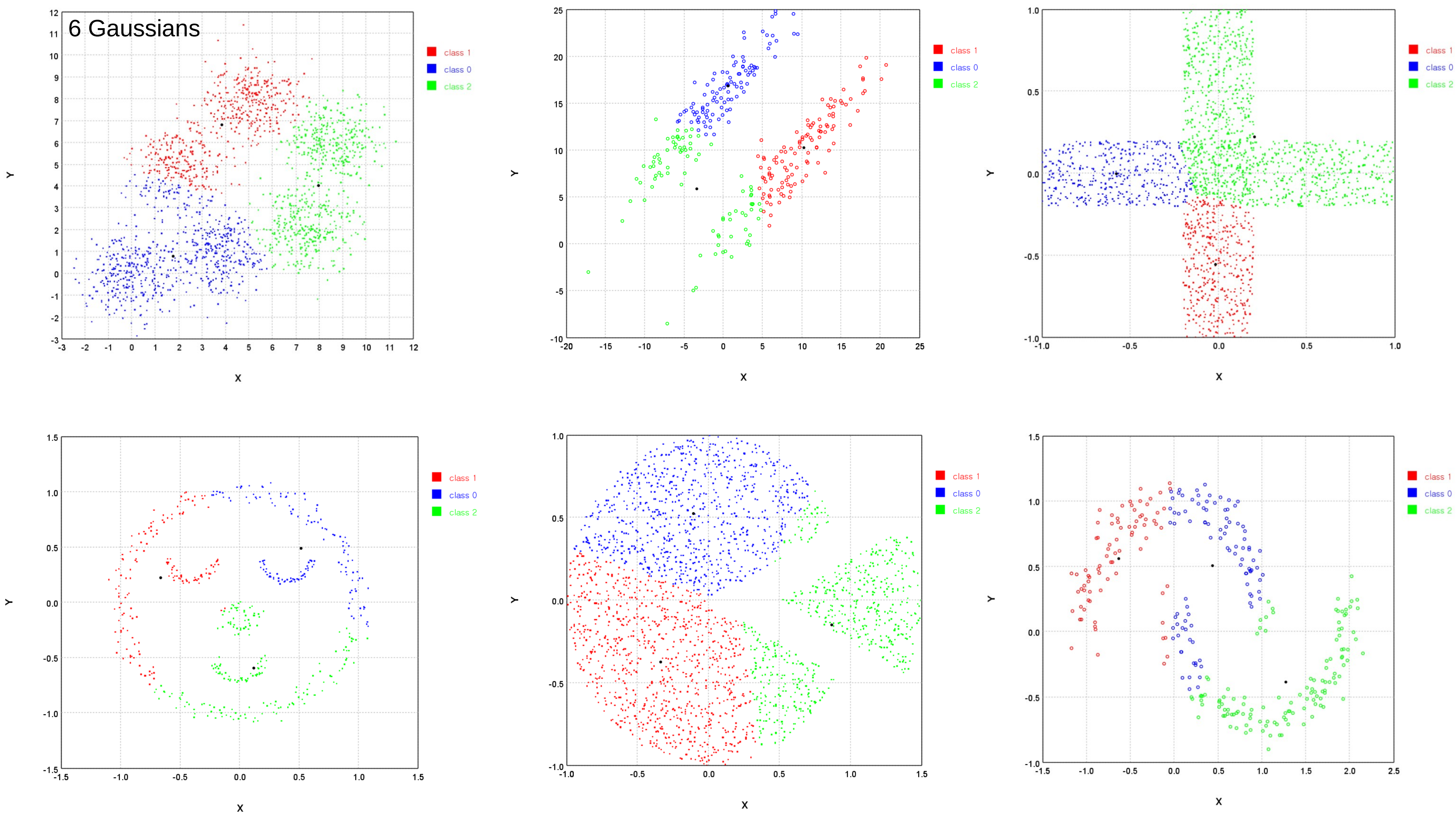


$4 \times C_{14}$
$7 \times C_{27}$
$24 \times C_{79}$
$0 \times other$



The clusters can be used as discrete attributes for representing the entire image.
This is known as a **bag-of-visual-terms** representation.

The clusters are not always circles. They can be weirdly shaped and even nested. Also, we don't even know how many of them to expect.



https://haifengl.github.io/index.html

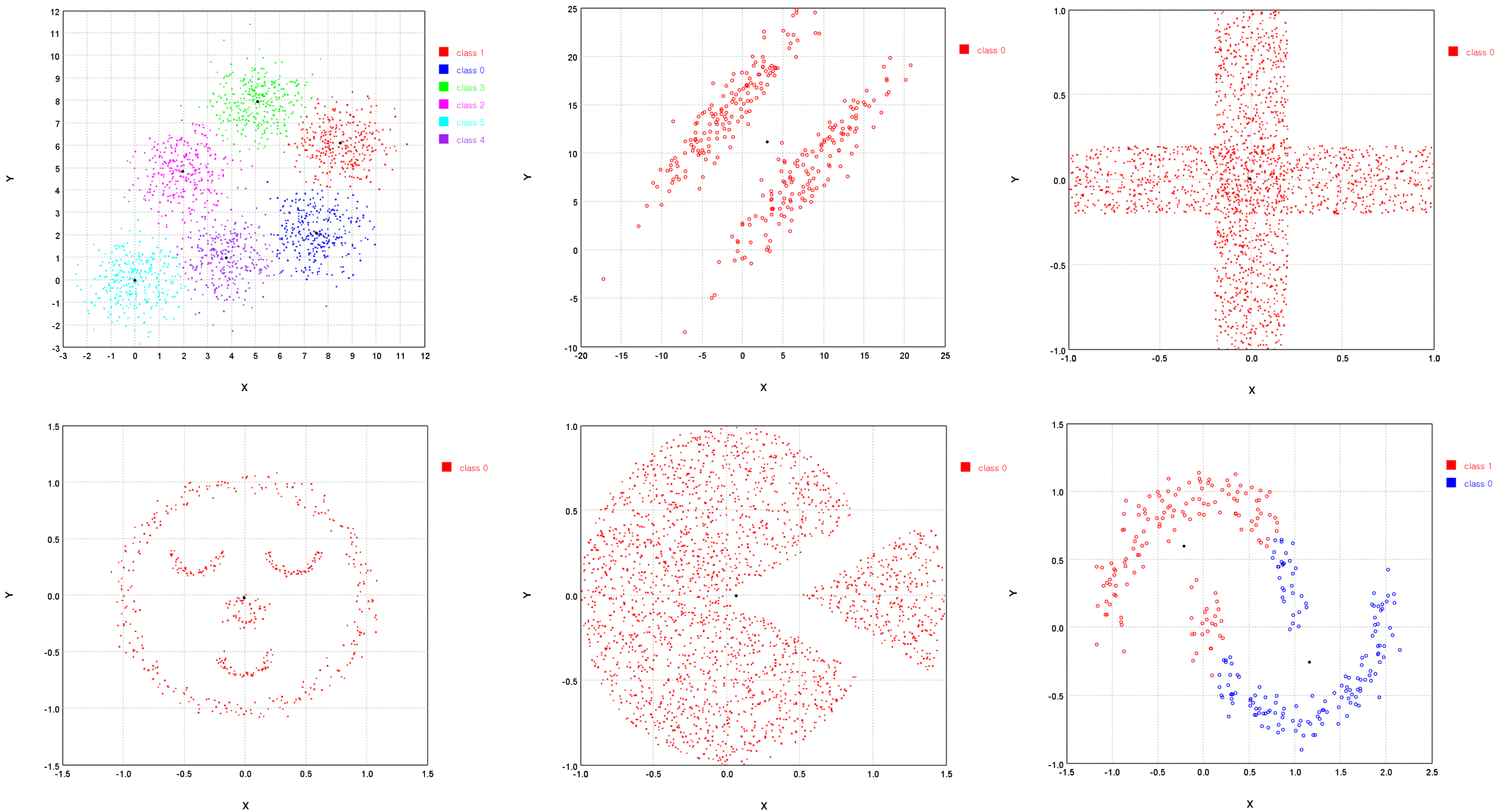
https://github.com/haifengl/smile

Smile - Statistical Machine Intelligence and Learning Engine

X-Means

– an extended *k*-means which tries to automatically determine the number of clusters based on Bayesian Information Criterion (BIC).

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.3377



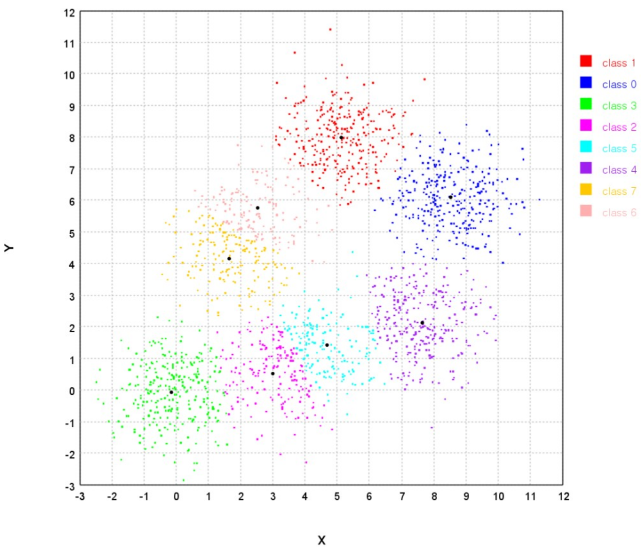
https://github.com/haifengl/smile https://haifengl.github.io/index.html

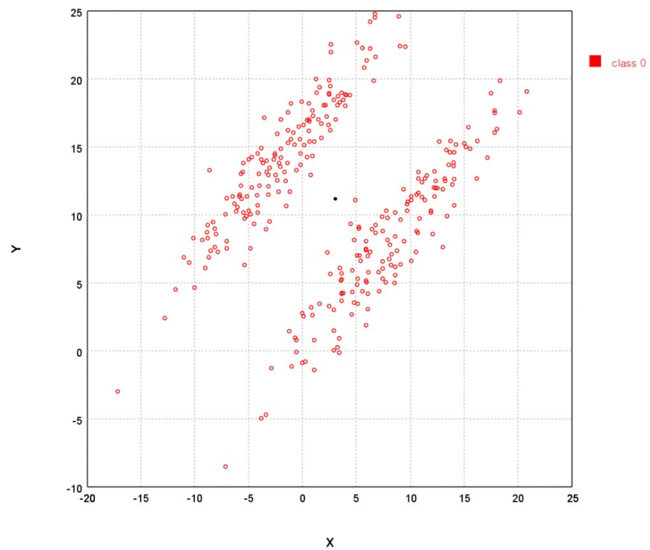
G-Means

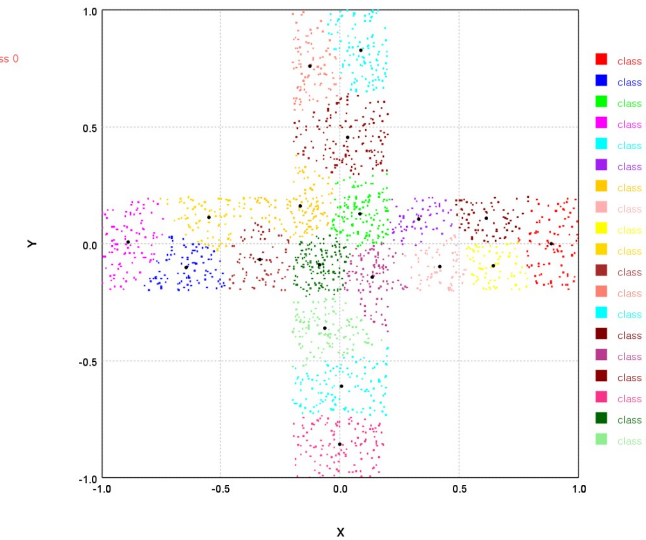
– runs *k*-means with increasing *k* in a hierarchical fashion until the test accepts the hypothesis that the data assigned to each *k*-means center are Gaussian.

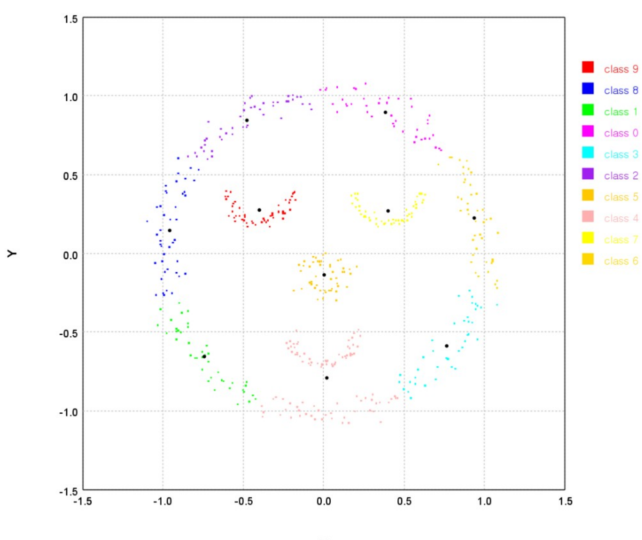
Greg Hamerly and Charles Elkan. Learning the k in k-means https://dl.acm.org/doi/10.5555/2981345.2981381

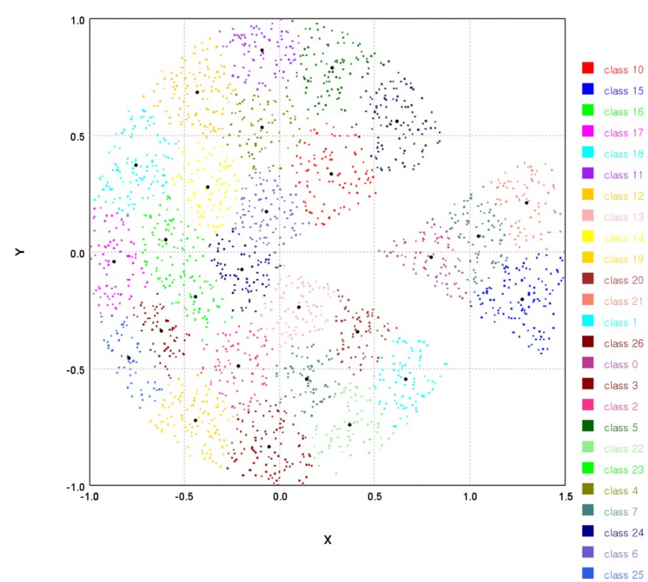
https://annoviko.github.io/G-Means/ https://github.com/haifengl/smile https://haifengl.github.io/index.html

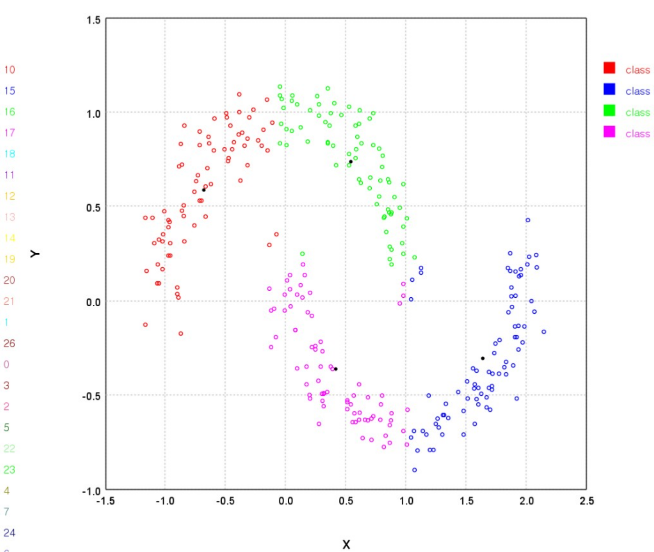












Hierarchical clustering

Hierarchical agglomerative clustering (HAC)

At the beginning, each point of an input data is considered as a separate cluster.

Then two the closest clusters are merged and the algorithm checks whether it is a time to stop by checking current amount of clusters.

Hierarchical divisive clustering (HDC)

Opposite to HAC

Assign all of the observations to a single cluster and then partition the cluster to two least similar clusters.

https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/

https://www.youtube.com/playlist?list=PLBv09BD7ez_7qIbBhyQDr-LAKWUeycZtx

https://www.coursera.org/learn/build-regression-classification-clustering-models

https://www.kdnuggets.com/2019/09/hierarchical-clustering.html

https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering/

Hierarchical agglomerative clustering

Clustering quality depends on a link that is used to find two the closest clusters.

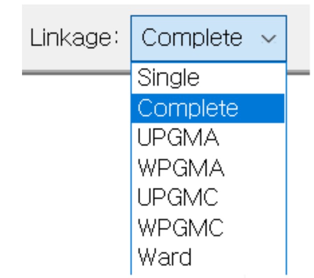
https://annoviko.github.io/Agglomerative/

https://pyclustering.github.io/

Single link
Complete link
Average distance between objects in clusters.
Centroid link

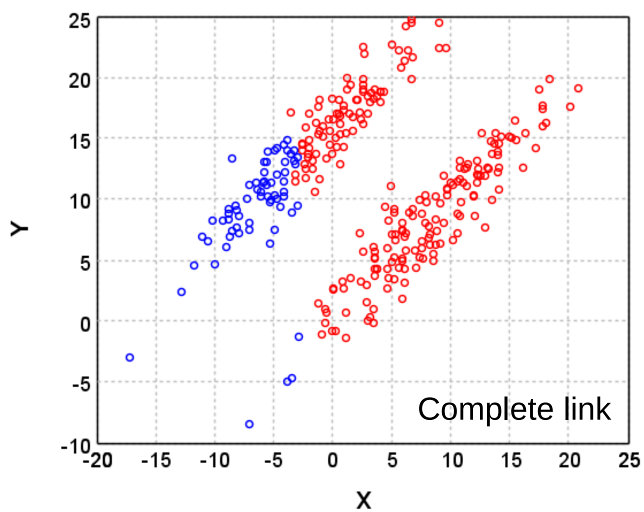
https://haifengl.github.io/index.html

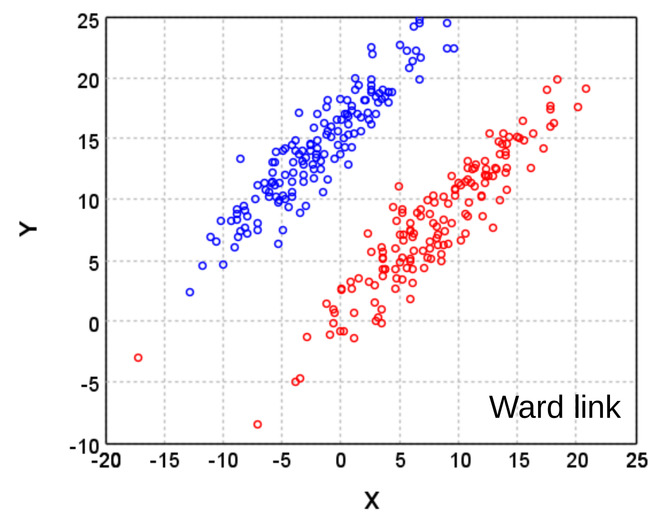
https://github.com/haifengl/smile

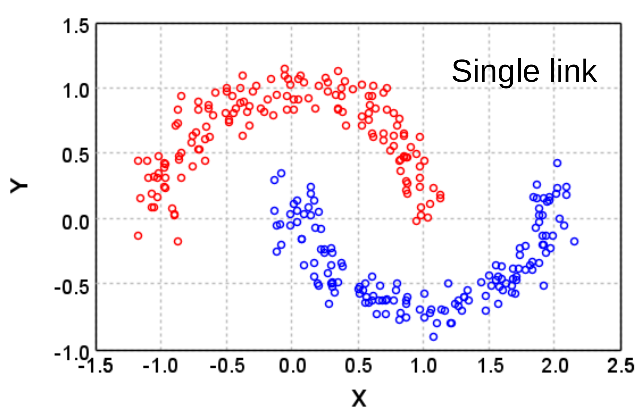


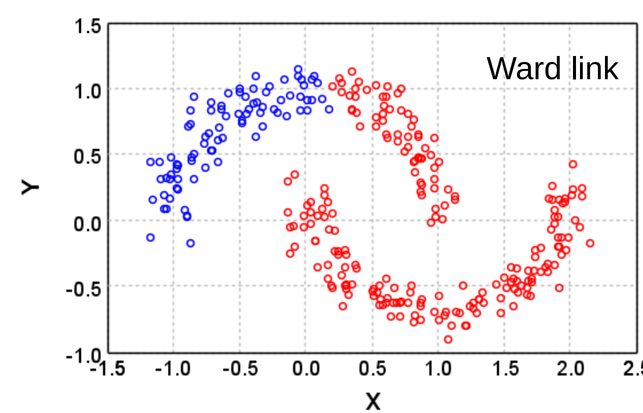
`scipy.cluster.hierarchy`

– many linkage methods







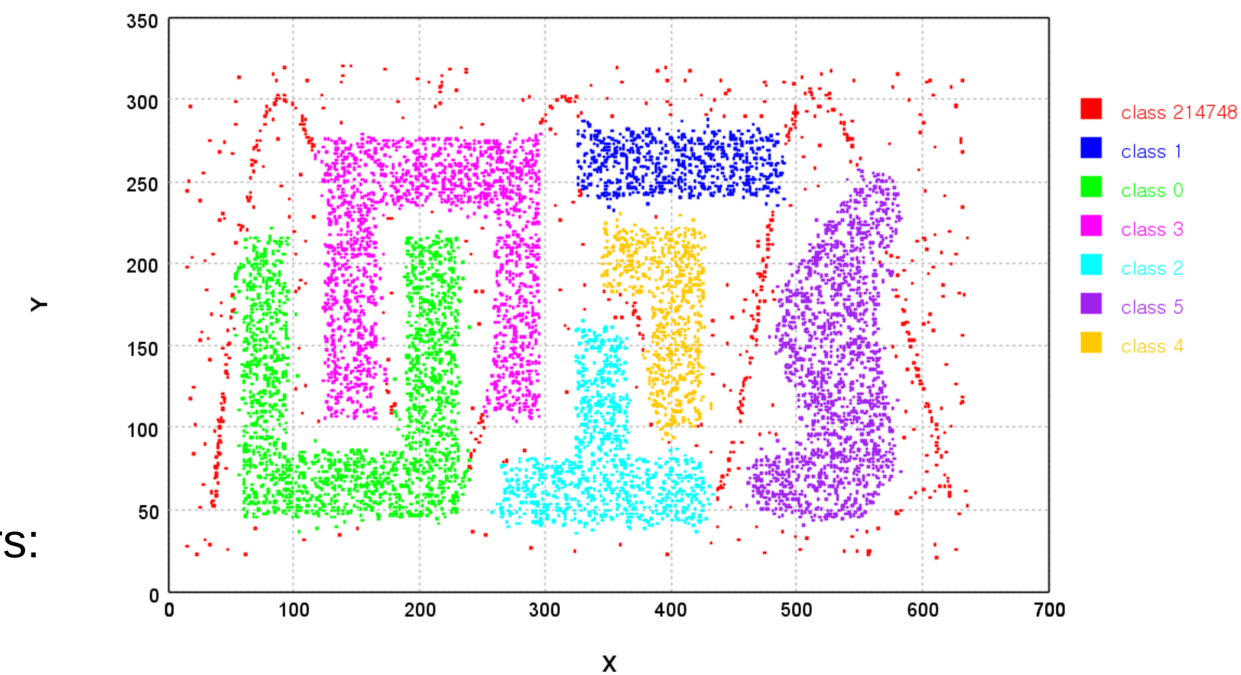


DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

– finds a number of clusters starting from the estimated density distribution of corresponding nodes.

```
val x = read.csv("data/clustering/chameleon/t4.8k.txt", header=false, delimiter=' ').toArray
val clusters = dbscan(x, 20, 10)
plot(x, clusters.y, '.', Palette.COLORS)
```

https://haifengl.github.io/clustering.html



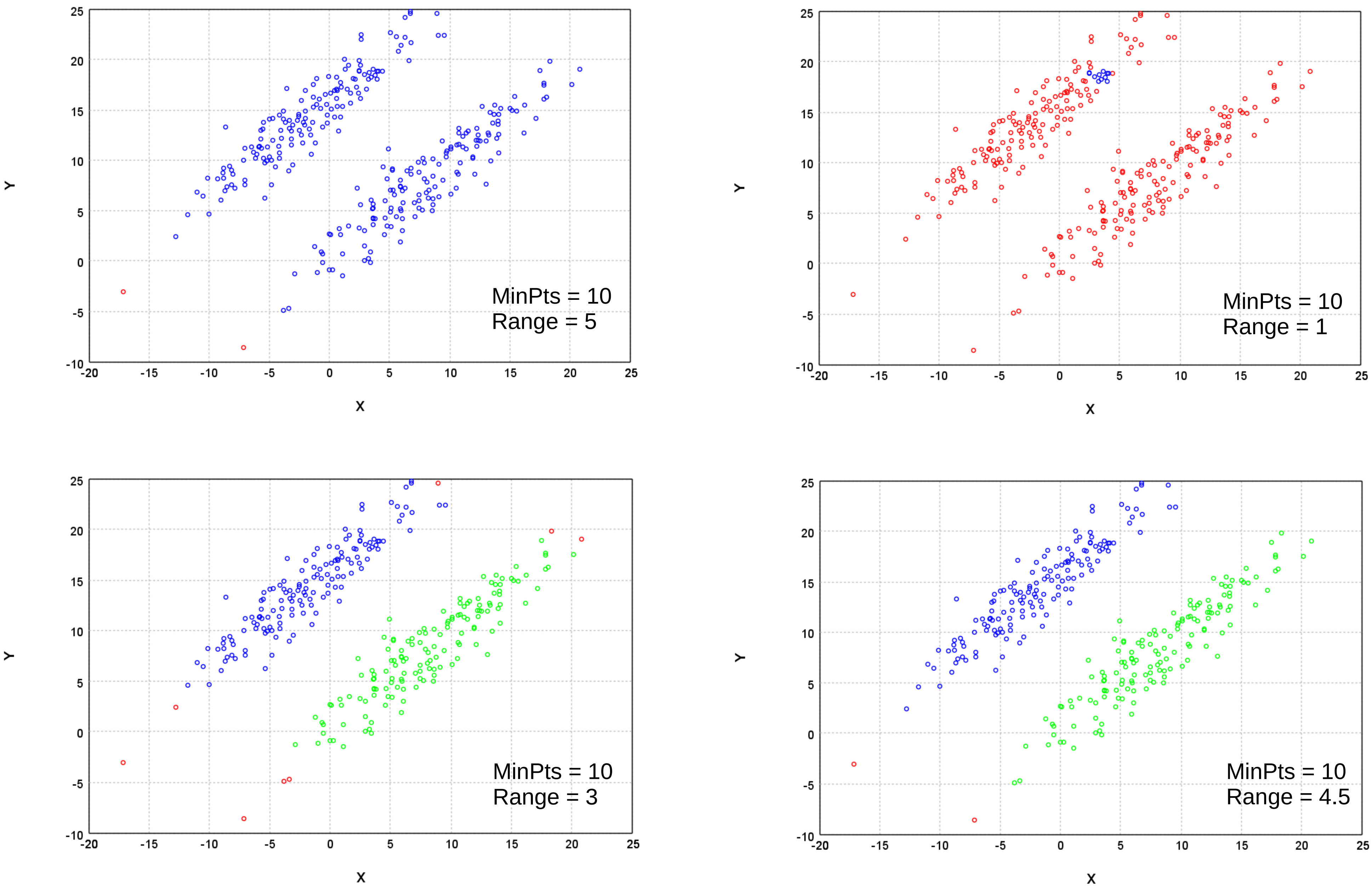
DBSCAN requires two parameters:

1) neighborhood radius

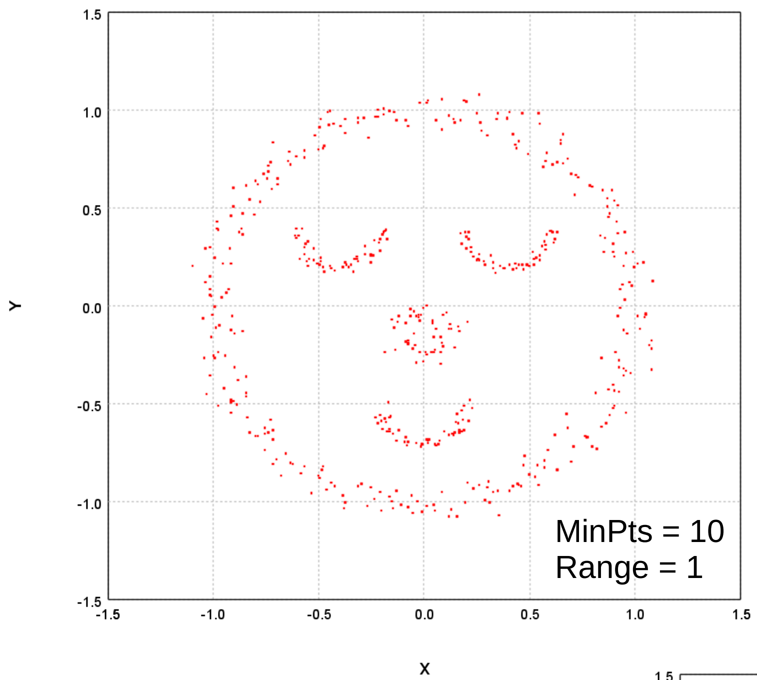
2) number of minimum points required to form a cluster

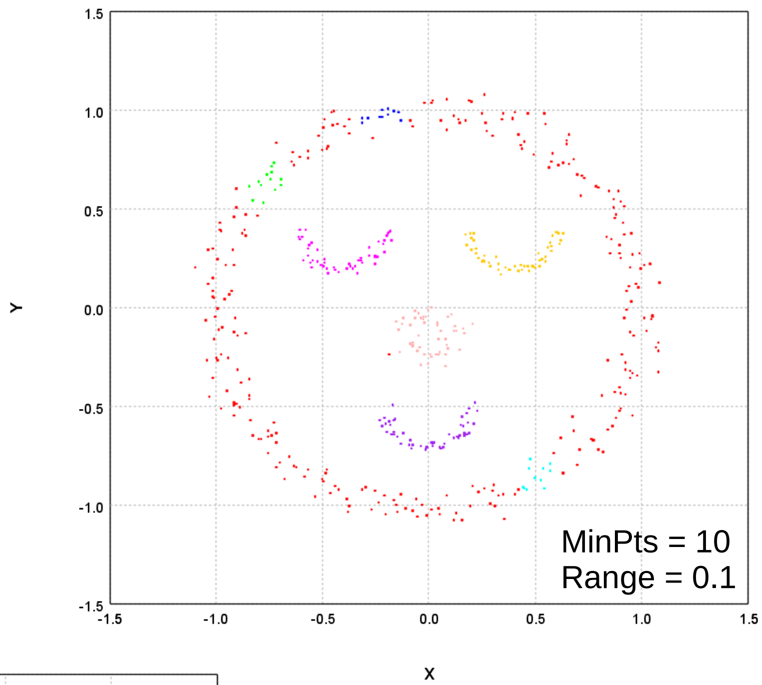
Problem: how to choose these parameters?

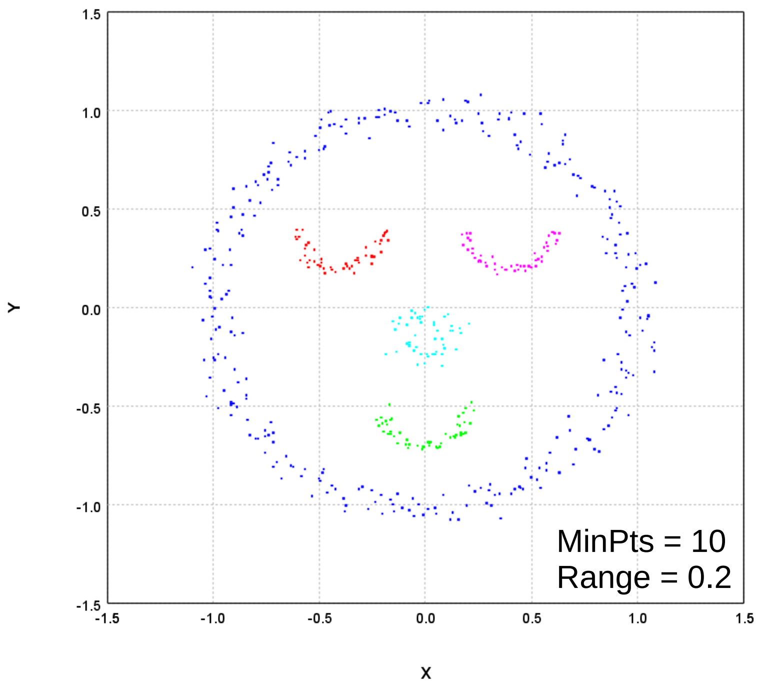
DBSCAN (Density-Based Spatial Clustering of Applications with Noise)



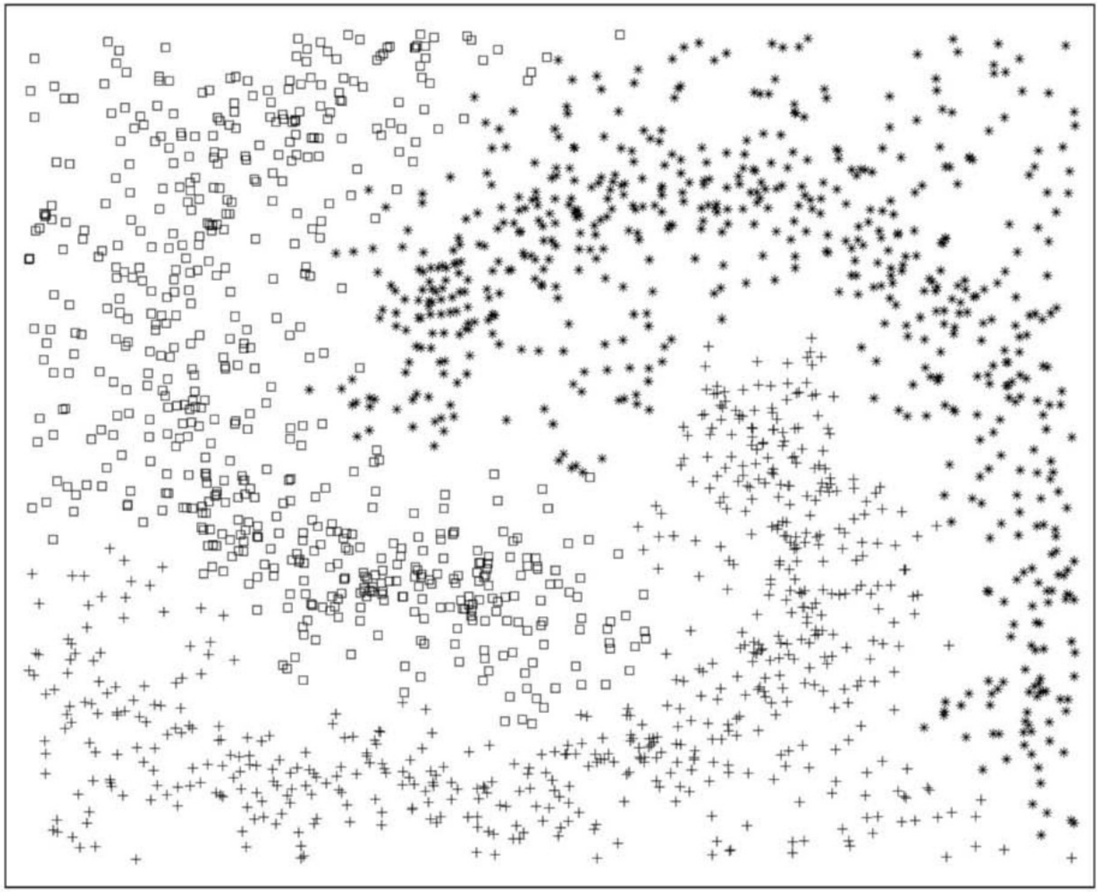
DBSCAN (Density-Based Spatial Clustering of Applications with Noise)







Bagging for clustering



Bagging for path-based clustering
https://ieeexplore.ieee.org/document/1240115

clue: Cluster Ensembles

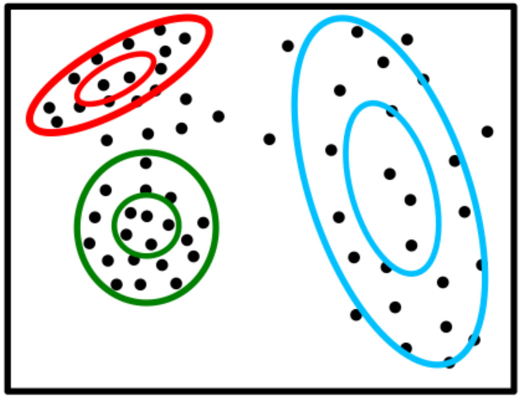
https://cran.r-project.org/web/packages/clue/index.html
https://rdrr.io/cran/clue/f/inst/doc/clue.pdf

K-Means Clustering with Bagging and MapReduce
https://ieeexplore.ieee.org/document/5718506

A Mixture Model for Clustering Ensembles
https://doi.org/10.1137/1.9781611972740.35

Random clustering forest for extended belief rule-based system https://doi.org/10.1007/s00500-020-05467-6

Mixture Models



– a probabilistic way to do soft clustering

– assigns samples to different clusters with different probabilities

Each cluster corresponds to a generative model (Gaussian or multinomial)

Parameters (e.g. mean/covariance) are unknown

The Expectation-Maximization algorithm allows to infer these parameters

Start with randomly placed Gaussians (μ_k, σ_k^2)

For each data point x_i: $P(b_k, x_i)$ = does it look like it came from b_k ?

Adjust (μ_k, σ_k^2) to fit points assigned to them

Iterate until convergence

https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html

https://www.youtube.com/playlist?list=PLBv09BD7ez_4e9LtmK626Evn1ion6ynrt