
# K-means Clustering

## Speed-ups

# K-means Clustering

## K-means Algorithm

- *Randomly initialize means* $[\mu_1, \ldots, \mu_K]$
- Repeat until L(μ, z) unchanged
  - *Assign all points to **nearest** cluster*

$$z_n = \underset{k}{\operatorname{argmin}} \, ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

  - *Update cluster means*

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n:\, z_n = k} \boldsymbol{x}_n$$

### Finding new cluster assignments

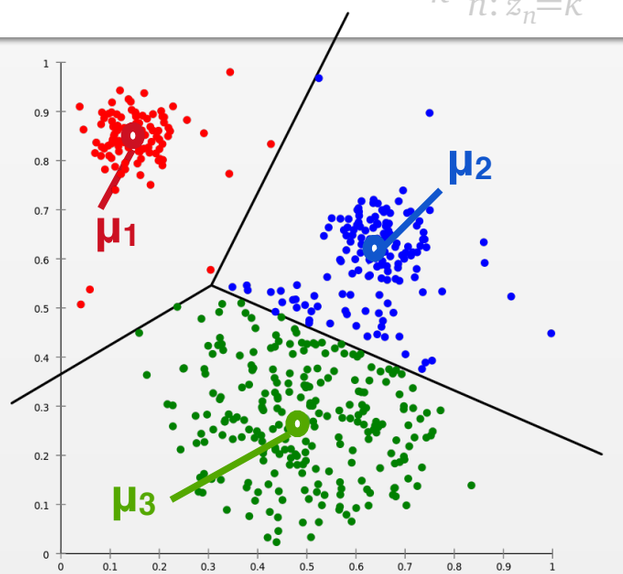
) compute all
oint-center
distances

*O(KND)* computational complexity (per iteration) for *K* clusters, *N* points, and *D* features.

### Updating the cluster centers

*O(ND)* computational complexity (per iteration)

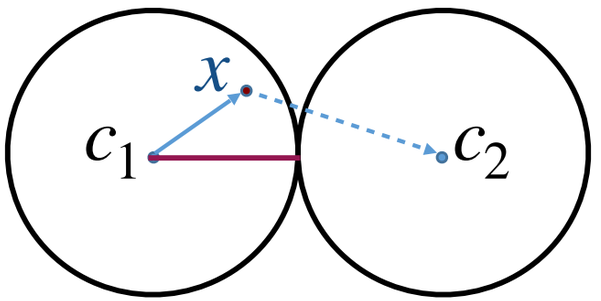
*Can it be reduced further if only a few cluster assignments change?*



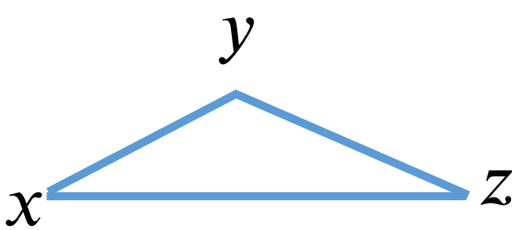
$O(NKD)$ per iteration is prohibitive in high dimensions and large K!

# The core idea for cutting on distance computation

When updating the cluster assignments not all point-center distances need be computed




Exploit triangle inequality




$d(x, z) \leq d(x, y) + d(y,$

**Also true when**

$$d(x, c_1) \leq u \leq \frac{d(c_1, c_2)}{2}$$

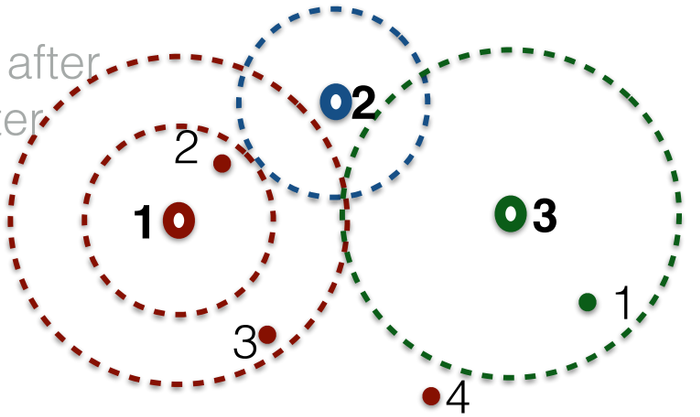
**Upper bound for $d(x, c_1)$**

$$d(x, c_1) \leq \frac{d(c_1, c_2)}{2} \Rightarrow d(x, c_1) \leq d(x, c_2)$$

If distance between $x$ and center $c_1$ is relatively small compared to that between $c_1$ and another center $c_2$, the distance between $x$ and $c_2$ need not be computed

# Elkan's accelerated K-means

Before cluster assignments. Right after centers have moved. Closest center might not be the assigned center.




## Conditions Checked:

1. $u(i) \leq s(a(i))$

   $i^{th}$ point cluster assignment need not be changed. No distance involving the $i^{th}$ point needs to be computed.

2. $u(i) \leq l(i,j)$ or $u(i) \leq \frac{d(c(a(i)), c(j))}{2}$

   $i^{th}$ point cluster assignment might change, but it won't be assigned to center $j$. Distance from the $j^{th}$ center need not be computed.

$a(i)$: contains the cluster index currently assigned to the $i^{th}$ point.

$u(i)$: contains an upper bound to the distance of the $i^{th}$ point to its current center

$l(i,j)$: contains a lower bound of the distance of the $i^{th}$ point to the $j^{th}$ center

$c(j)$: is the $j^{th}$ center.

$s(j)$: is equal to half the distance of $j^{th}$ center to its closest center

# Bounding the distance of $x$ from a center $c$ after it moves to $c^*$

Distance computation: vector operation
Upper and lower bound: scalar operation

## *Lower bound*

$$d(x, c^*) \geq \max(0,\ d(x, c) - d(c, c^*))$$
$$\geq \max(0,\ l - d(c, c^*))$$
$$= l^*$$

Old lower bound

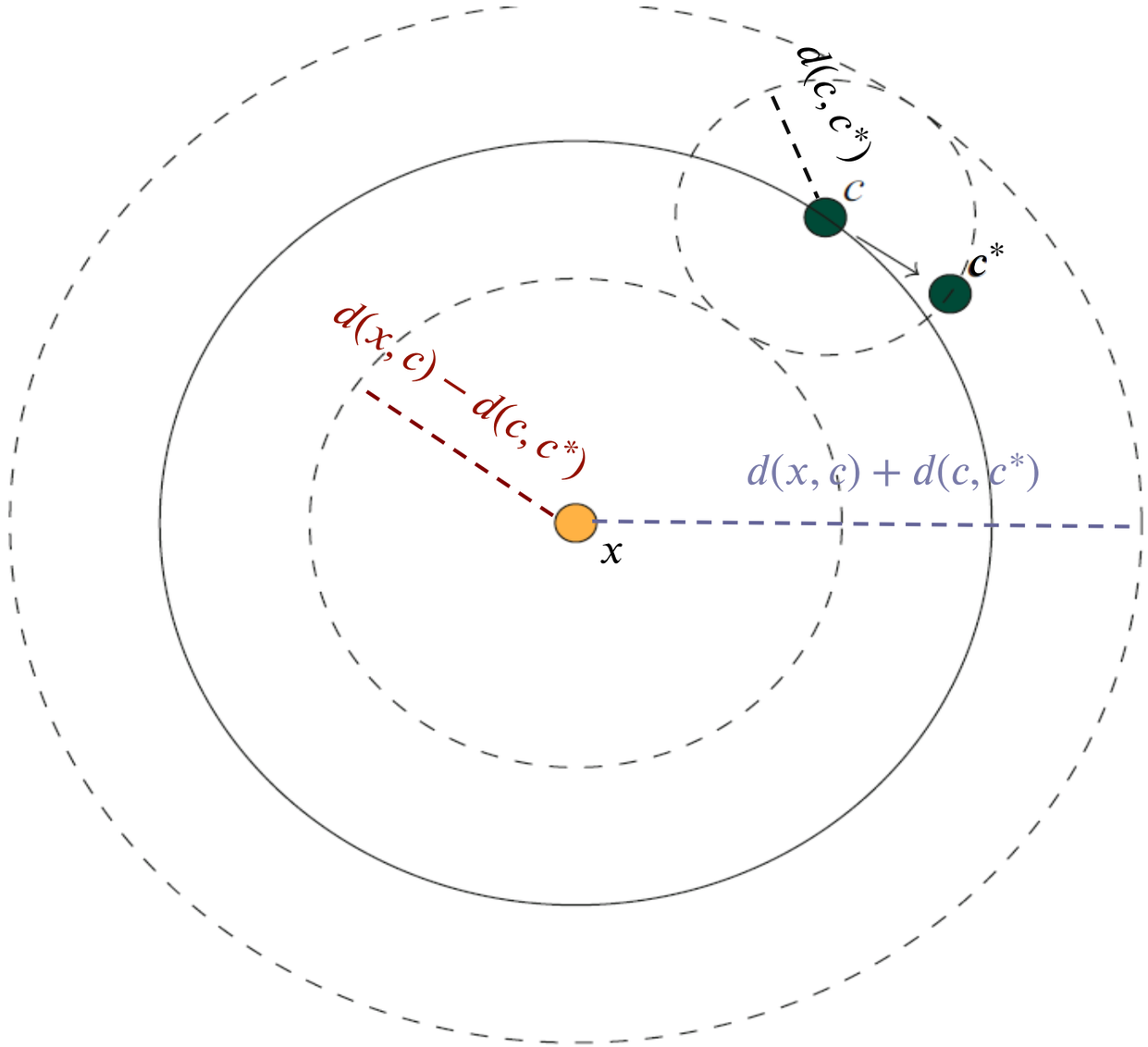
New lower bound

## *Upper bound*

$$d(x, c^*) \leq d(x, c) + d(c, c^*)$$
$$\leq u + d(c, c^*)$$
$$= u^*$$

Old upper bound

New upper bound



**Algorithm 3** Elkan's algorithm—using $k$ lower bounds per point and $k^2$ center-center distances

```
procedure ELKAN(X, C)
    a(i) ← 1, u(i) ← ∞, ∀i ∈ N {Initialize invalid bounds, all in one cluster.}
    ℓ(i, j) ← 0, ∀i ∈ N, j ∈ K
    while not converged do
5:      compute ‖c(j) − c(j′)‖, ∀ j, j′ ∈ K
        compute s(j) ← min_{j′≠j} ‖c(j) − c(j′)‖/2, ∀ j ∈ K
        for all i ∈ N do
            if u(i) ≤ s(a(i)) then continue with next i
            r ← True
10:         for all j ∈ K do
                z ← max(ℓ(i, j), ‖c(a(i)) − c(j)‖/2)
                if j = a(i) or u(i) ≤ z then continue with next j
                if r then
                    u(i) ← ‖x(i) − c(a(i))‖
15:                 r ← False
                    if u(i) ≤ z then continue with next j
                ℓ(i, j) ← ‖x(i) − c(j)‖
                if ℓ(i, j) < u(i) then a(i) ← j
        for all j ∈ K do {Move the centers and track their movement}
20:         move c(j) to its new location
            let δ(j) be the distance moved by c(j)
        for all i ∈ N do {Update the upper and lower distance bounds}
            u(i) ← u(i) + δ(a(i))
            for all j ∈ K do
25:             ℓ(i, j) ← ℓ(i, j) − δ(j)
```

$r$: tells if the upper bound needs to be tightened.

Both upper bound and the lower bound are tight on this step.

$u(i) \leftarrow l(i,j)$

The upper bound should be updated at this step

$$\max\left(0, l(i,j) - \delta(j)\right)$$

**Algorithm 3** Elkan's algorithm—using $k$ lower bounds per point and $k^2$ center-center distances

```
procedure ELKAN(X, C)
    a(i) ← 1, u(i) ← ∞, ∀i ∈ N {Initialize invalid bounds, all in one cluster.}
    ℓ(i, j) ← 0, ∀i ∈ N, j ∈ K
    while not converged do
5:      compute ‖c(j) − c(j′)‖, ∀ j, j′ ∈ K
        compute s(j) ← min_{j′≠j} ‖c(j) − c(j′)‖/2, ∀ j ∈ K
        for all i ∈ N do
            if u(i) ≤ s(a(i)) then continue with next i
            r ← True
10:         for all j ∈ K do
                z ← max(ℓ(i, j), ‖c(a(i)) − c(j)‖/2)
                if j = a(i) or u(i) ≤ z then continue with next j
                if r then
                    u(i) ← ‖x(i) − c(a(i))‖
                    r ← False
15:                 if u(i) ≤ z then continue with next j
                ℓ(i, j) ← ‖x(i) − c(j)‖
                if ℓ(i, j) < u(i) then a(i) ← j
        for all j ∈ K do {Move the centers and track their movement}
20:         move c(j) to its new location
            let δ(j) be the distance moved by c(j)
        for all i ∈ N do {Update the upper and lower distance bounds}
            u(i) ← u(i) + δ(a(i))
            for all j ∈ K do
25:             ℓ(i, j) ← ℓ(i, j) − δ(j)
```

$O(K^2D)$

$O(N)$

$O(\alpha_1 NK)$

$\alpha_1$ is the fraction of tim the first condition is n satisfied

Since the bounds are ɔose in the first iteration, all distances will be computed: $O(NDK)$

$O(\alpha_1\alpha_2 NKD)$

$\alpha_2$ is the fraction of times the second condition is not satisfied.

$O(KD)$

$\max\left(0, l(i,j) - \delta(j)\right)$ $O(NK)$

# Running time of Elkan's K-means

Major computations

- Computing point-center distances
  - $O(NKD)$ in the first/first-few iteration.
  - $O(ND)$ over all later iterations combined. For most datasets with significant cluster structure.
- Computing pairwise center distances
  - $O(K^2DE)$
- Updating the lower bound
  - $O(NKE)$

Most points (in the core of the cluster) won't change cluster assignments after the first few iterations and will satisfy the pruning conditions. The more the clusters looks like gaussians, the more this true. This might no longer be true if the data lacks a cluster structure.

$N$: dataset size

$K$: number of clusters

$D$: number of dimensions

$E$: number of iterations

# Results for Elkan

| | | $k = 3$ | $k = 20$ | $k = 100$ |
|---|---|---|---|---|
| birch | iterations | 17 | 38 | 56 |
| | standard | 5.100e+06 | 7.600e+07 | 5.600e+08 |
| | fast | 4.495e+05 | 1.085e+06 | 1.597e+06 |
| | speedup | 11.3 | 70.0 | 351 |
| covtype | iterations | 18 | 256 | 152 |
| | standard | 8.100e+06 | 7.680e+08 | 2.280e+09 |
| | fast | 9.416e+05 | 7.147e+06 | 7.353e+06 |
| | speedup | 8.60 | 107 | 310 |
| kddcup | iterations | 34 | 100 | 325 |
| | standard | 9.732e+06 | 1.908e+08 | 3.101e+09 |
| | fast | 6.179e+05 | 3.812e+06 | 1.005e+07 |
| | speedup | 15.4 | 50.1 | 309 |
| mnist50 | iterations | 38 | 178 | 217 |
| | standard | 6.840e+06 | 2.136e+08 | 1.302e+09 |
| | fast | 1.573e+06 | 9.353e+06 | 3.159e+07 |
| | speedup | 4.35 | 22.8 | 41.2 |
| mnist784 | iterations | 63 | 60 | 165 |
| | standard | 1.134e+07 | 7.200e+07 | 9.900e+08 |
| | fast | 1.625e+06 | 7.396e+06 | 3.055e+07 |
| | speedup | 6.98 | 9.73 | 32.4 |
| random | iterations | 52 | 33 | 18 |
| | standard | 1.560e+06 | 6.600e+06 | 1.800e+07 |
| | fast | 1.040e+06 | 3.020e+06 | 5.348e+06 |
| | speedup | 1.50 | 2.19 | 3.37 |

| name | cardinality | dimensionality | description |
|---|---|---|---|
| birch | 100000 | 2 | 10 by 10 grid of Gaussian clusters, DS1 in (Zhang et al., 199 |
| covtype | 150000 | 54 | remote soil cover measurements, after (Moore, 2000) |
| kddcup | 95413 | 56 | KDD Cup 1998 data, un-normalized |
| mnist50 | 60000 | 50 | random projection of NIST handwritten digit training data |
| mnist784 | 60000 | 784 | original NIST handwritten digit training data |
| random | 10000 | 1000 | uniform random data |

*Table 2.* Rows labeled "standard" and "fast" give the number of distance calculations performed by the unaccelerated $k$-means algorithr and by the new algorithm. Rows labeled "speedup" show how many times faster the new algorithm is, when the unit of measurement distance calculations.

# Limitations of Elkan

Storing and updating the lower bounds ($N \times K$ dimension) can be a bottleneck for large $K$

Can a smaller set of lower bounds be used?

# Hamerly's accelerated K-means

Main difference from Elkan:

$l(i)$ instead of $l(i,j)$

Maintains one lower bound per point instead of $K$

$l(i)$: lower bound of the distance of the $i^{th}$ point to the second closest centroid

Conditions Checked

$u(i) \leq s\left(a(i)\right)$ or $u(i) \leq l(i)$.

No distance involving the $i^{th}$ point needs to be computed.

$O(N)$ instead of $O(N \times K)$ space for storing the lower bounds

Tradeoff

- Less memory for storing lower bounds.
- Fewer computations for updating lower bounds.
- However, there is less pruning and consequently more distance computation.

**Algorithm 4** Hamerly's algorithm—using 1 lower bound per point

**procedure** HAMERLY($X, C$)

$a(i) \leftarrow 1, u(i) \leftarrow \infty, \ell(i) \leftarrow 0, \forall i \in N$ {Initialize invalid bounds, all in one cluster.}

**while** not converged **do**

compute $s(j) \leftarrow \min_{j' \neq j} \|c(j) - c(j')\|/2, \forall j \in K$

**for all** $i \in N$ **do**

$z \leftarrow \max(\ell(i), s(a(i)))$

**if** $u(i) \leq z$ **then** continue with next $i$

$u(i) \leftarrow \|x(i) - c(a(i))\|$ {Tighten the upper bound}

**if** $u(i) \leq z$ **then** continue with next $i$

Find $c(j)$ and $c(j')$, the two closest centers to $x(i)$, as well as the distances to each.

**if** $j \neq a(i)$ **then**

$a(i) \leftarrow j$

$u(i) \leftarrow \|x(i) - c(a(i))\|$

$\ell(i) \leftarrow \|x(i) - c(j')\|$

**for all** $j \in K$ **do** {Move the centers and track their movement}

move $c(j)$ to its new location

let $\delta(j)$ be the distance moved by $c(j)$

$\delta' \leftarrow \max_{j \in K} \delta(j)$

**for all** $i \in N$ **do** {Update the upper and lower distance bounds}

$u(i) \leftarrow u(i) + \delta(a(i))$

$\ell(i) \leftarrow \ell(i) - \delta'$ $\max(0, l(i) - \delta')$

$l(i)$ by definition is also a lower bound to the distances to other centers, except the closest one.

$\delta'$ ensures that if the second closest cluster changes the lower bound is still valid.

| Dataset | | Total user CPU Seconds (User CPU seconds per iteration) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $k = 3$ | | $k = 20$ | | $k = 100$ | | $k = 500$ | |
| uniform random | iterations | 44 | | 227 | | 298 | | 710 | |
| $n = 1250000$ | lloyd | 4.0 | (0.058) | 61.4 | (0.264) | 320.2 | (1.070) | 3486.9 | (4.909) |
| $d = 2$ | kd-tree | 3.5 | (**0.006**) | **11.8** | (**0.035**) | 34.6 | (0.102) | 338.8 | (0.471) |
| | elkan | 7.2 | (0.133) | 75.2 | (0.325) | 353.1 | (1.180) | 2771.8 | (3.902) |
| | hamerly | **2.7** | (0.031) | 14.6 | (0.058) | **28.2** | (**0.090**) | **204.2** | (**0.286**) |
| uniform random | iterations | 121 | | 353 | | 312 | | 1405 | |
| $n = 1250000$ | lloyd | 21.8 | (0.134) | 178.9 | (0.491) | 660.7 | (2.100) | 13854.4 | (9.857) |
| $d = 8$ | kd-tree | 117.5 | (0.886) | 622.6 | (1.740) | 2390.8 | (7.633) | 46731.5 | (33.254) |
| | elkan | 14.1 | (0.071) | 130.6 | (0.354) | 591.8 | (1.879) | 11827.9 | (8.414) |
| | hamerly | **10.9** | (**0.045**) | **40.4** | (**0.099**) | **169.8** | (**0.527**) | **1395.6** | (**0.989**) |
| uniform random | iterations | 137 | | 4120 | | 2096 | | 2408 | |
| $n = 1250000$ | lloyd | 66.4 | (0.323) | 5479.5 | (1.325) | 12543.8 | (5.974) | 68967.3 | (28.632) |
| $d = 32$ | kd-tree | 208.4 | (1.324) | 29719.6 | (7.207) | 74181.3 | (35.380) | 425513.0 | (176.697) |
| | elkan | 48.1 | (0.189) | 1370.1 | (0.327) | 2624.9 | (1.242) | 14245.9 | (5.907) |
| | hamerly | **46.9** | (**0.180**) | **446.4** | (**0.103**) | **1238.9** | (**0.581**) | **9886.9** | (**4.097**) |
| birch | iterations | 52 | | 179 | | 110 | | 99 | |
| $n = 100000$ | lloyd | 0.53 | (0.004) | 4.60 | (0.024) | 11.80 | (0.104) | 48.87 | (0.490) |
| $d = 2$ | kd-tree | **0.41** | (<**0.001**) | 0.96 | (**0.003**) | 2.67 | (0.021) | 17.68 | (0.173) |
| | elkan | 0.58 | (0.005) | 4.35 | (0.023) | 11.80 | (0.104) | 54.28 | (0.545) |
| | hamerly | 0.44 | (0.002) | **0.90** | (**0.003**) | **1.86** | (**0.014**) | **7.81** | (**0.075**) |
| covtype | iterations | 19 | | 204 | | 320 | | 111 | |
| $n = 150000$ | lloyd | 3.52 | (0.048) | 48.02 | (0.222) | 322.25 | (0.999) | 564.05 | (5.058) |
| $d = 54$ | kd-tree | 6.65 | (0.205) | 266.65 | (1.293) | 2014.03 | (6.285) | 3303.27 | (29.734) |
| | elkan | 3.07 | (0.022) | 11.58 | (0.044) | 70.45 | (0.212) | **152.15** | (**1.347**) |
| | hamerly | **2.95** | (**0.019**) | **7.40** | (**0.024**) | **42.83** | (**0.126**) | 169.53 | (1.505) |
| kddcup | iterations | 39 | | 55 | | 169 | | 142 | |
| $n = 95412$ | lloyd | 4.74 | (0.032) | 12.35 | (0.159) | 116.63 | (0.669) | 464.22 | (3.244) |
| $d = 56$ | kd-tree | 9.68 | (0.156) | 58.55 | (0.996) | 839.31 | (4.945) | 3349.47 | (23.562) |
| | elkan | 4.13 | (0.012) | 6.24 | (0.049) | 32.27 | (0.169) | **132.39** | (**0.907**) |
| | hamerly | **3.95** | (**0.011**) | **5.87** | (**0.042**) | **28.39** | (**0.147**) | 197.26 | (1.364) |
| mnist50 | iterations | 37 | | 249 | | 190 | | 81 | |
| $n = 60000$ | lloyd | 2.92 | (0.018) | 23.18 | (0.084) | 75.82 | (0.387) | 162.09 | (1.974) |
| $d = 50$ | kd-tree | 4.90 | (0.069) | 100.09 | (0.393) | 371.57 | (1.943) | 794.51 | (9.780) |
| | elkan | 2.42 | (0.005) | 7.02 | (0.019) | **21.58** | (**0.101**) | **55.61** | (**0.660**) |
| | hamerly | **2.41** | (**0.004**) | **4.54** | (**0.009**) | 21.95 | (0.104) | 77.34 | (0.928) |

Table 3: These results show the fraction of times that each algorithm was able to skip the innermost loop on data of different dimensions (values closer to 1 are better). These results are averaged over runs using $k$ = 3, 20, 100, and 500 (one run for each $k$). The randX datasets are uniform random hypercube data with X dimensions.

| dataset | rand2 | rand8 | rand32 | rand128 |
|---|---|---|---|---|
| elkan | 0.56 | 0.01 | 0.00 | 0.00 |
| hamerly | 0.97 | 0.88 | 0.91 | 0.83 |

| dataset | birch | covtype | kddcup | mnist50 |
|---|---|---|---|---|
| elkan | 0.52 | 0.34 | 0.18 | 0.22 |
| hamerly | 0.94 | 0.89 | 0.82 | 0.82 |

# Memory requirements

| | | Megabytes | | | |
|---|---|---|---|---|---|
| Dataset | Algorithm | $k$=3 | $k$=20 | $k$=100 | $k$=500 |
| uniform | lloyd | 7.5 | 7.5 | 7.5 | 7.5 |
| random | kd-tree | 32.1 | 32.1 | 32.1 | 32.1 |
| $n$=1.25M | elkan | 19.8 | 60.3 | 251.0 | 1205.2 |
| $d$=2 | hamerly | 14.7 | 14.7 | 14.7 | 14.7 |
| uniform | lloyd | 21.9 | 21.9 | 21.9 | 21.9 |
| random | kd-tree | 54.8 | 54.8 | 54.8 | 54.8 |
| $n$=1.25M | elkan | 34.1 | 74.6 | 265.3 | 1219.5 |
| $d$=8 | hamerly | 29.0 | 29.0 | 29.0 | 29.0 |
| uniform | lloyd | 79.1 | 79.1 | 79.1 | 79.1 |
| random | kd-tree | 145.2 | 145.2 | 145.2 | 145.3 |
| $n$=1.25M | elkan | 91.3 | 131.8 | 322.6 | 1276.8 |
| $d$=32 | hamerly | 86.2 | 86.2 | 86.2 | 86.3 |
| birch | lloyd | 1.4 | 1.1 | 1.1 | 1.3 |
| $n$=100K | kd-tree | 2.9 | 2.9 | 2.8 | 2.7 |
| $d$=2 | elkan | 2.1 | 5.2 | 20.6 | 97.3 |
| | hamerly | 1.5 | 1.7 | 1.6 | 1.5 |
| covtype | lloyd | 16.2 | 16.2 | 16.1 | 16.4 |
| $n$=150K | kd-tree | 27.2 | 27.2 | 27.2 | 27.3 |
| $d$=54 | elkan | 17.4 | 22.5 | 45.3 | 160.4 |
| | hamerly | 17.0 | 17.0 | 16.8 | 17.2 |
| kddcup | lloyd | 10.9 | 10.8 | 11.1 | 11.2 |
| $n$=95412 | kd-tree | 18.8 | 18.9 | 19.1 | 19.0 |
| $d$=56 | elkan | 11.9 | 15.1 | 29.6 | 103.1 |
| | hamerly | 11.6 | 11.6 | 11.3 | 11.7 |
| mnist50 | lloyd | 6.3 | 6.6 | 6.4 | 6.8 |
| $n$=60K | kd-tree | 10.5 | 10.4 | 10.6 | 10.7 |
| $d$=50 | elkan | 7.0 | 9.1 | 18.4 | 64.8 |
| | hamerly | 6.9 | 6.9 | 6.9 | 6.8 |

# Summary

- For moderate $D$ (< 50) and $K$(< 100), Hamerly is well-suited (has smaller time and memory footprint).
- Large $D$ (greater than 50), Elkan might be better (has smaller time footprint, in spite of large memory requirements).

# Speed up with an approximate algorithm

# Mini-batch K-means

Web-scale k-means **clustering**
**D Sculley** - Proceedings of the 19th international conference on …, 2010 - dl.acm.org
Abstract We present two modifications to the popular k-means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. First, we propose the use of mini-batch optimization for k-means ...
Cited by 152 Related articles All 11 versions Cite Save

# Mini-batch K-means



## Web-Scale K-Means Clustering

D. Sculley
Google, Inc. Pittsburgh. PA USA
dsculley@google.com


### ABSTRACT

We present two modifications to the popular $k$-means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. First, we propose the use of mini-batch optimization for $k$-means clustering. This reduces computation cost by orders of magnitude compared to the classic batch algorithm while yielding significantly better solutions than online stochastic gradient descent. Second, we achieve sparsity with projected gradient descent, and give a fast $\epsilon$-accurate projection onto the $L1$-ball. Source code is freely available: `http://code.google.com/p/sofia-ml`



### Categories and Subject Descriptors

I.5.3 [**Computing Methodologies**]: Pattern Recognition—*Clustering*

### General Terms

Algorithms, Performance, Experimentation

### Keywords

unsupervised clustering, scalability, sparse solutions


### 1. CLUSTERING AND THE WEB

Unsupervised clustering is an important task in a range of web-based applications, including grouping search results, near-duplicate detection, and news aggregation to name but a few. Lloyd's classic $k$-means algorithm remains a popular choice for real-world clustering tasks [6]. However, the standard batch algorithm is slow for large data sets. Even optimized batch $k$-means variants exploiting triangle inequality [3] cannot cheaply meet the latency needs of user-facing applications when clustering results on large data sets are required in a fraction of a second.

This paper proposes a mini-batch $k$-means variant that yields excellent clustering results with low computation cost on large data sets. We also give methods for learning sparse cluster centers that reduce storage and network cost.

### 2. MINI-BATCH K-MEANS

The $k$-means optimization problem is to find the set $C$ of cluster centers $\mathbf{c} \in \mathbb{R}^m$, with $|C| = k$, to minimize over a set $X$ of examples $\mathbf{x} \in \mathbb{R}^m$ the following objective function:

$$\min \sum_{\mathbf{x} \in X} ||f(C, \mathbf{x}) - \mathbf{x}||^2$$

Here, $f(C, \mathbf{x})$ returns the nearest cluster center $\mathbf{c} \in C$ to $\mathbf{x}$ using Euclidean distance. It is well known that although this problem is NP-hard in general, gradient descent methods converge to a local optimum when seeded with an initial set of $k$ examples drawn uniformly at random from $X$ [1].

The classic batch $k$-means algorithm is expensive for large data sets, requiring $O(kns)$ computation time where $n$ is the number of examples and $s$ is the maximum number of non-zero elements in any example vector. Bottou and Bengio proposed an online, stochastic gradient descent (SGD) variant that computed a gradient descent step on one example at a time [1]. While SGD converges quickly on large data sets, it finds lower quality solutions than the batch algorithm due to stochastic noise [1].

**Algorithm 1** Mini-batch $k$-Means.

```
Given: k, mini-batch size b, iterations t, data set X
Initialize each c ∈ C with an x picked randomly from X
v ← 0
for i = 1 to t do
  M ← b examples picked randomly from X
  for x ∈ M do
    d[x] ← f(C, x)    // Cache the center nearest to x
  end for
  for x ∈ M do
    c ← d[x]          // Get cached center for this x
    v[c] ← v[c] + 1   // Update per-center counts
    η ← 1/v[c]        // Get per-center learning rate
    c ← (1 − η)c + ηx // Take gradient step
  end for
end for
```

We propose the use of mini-batch optimization for $k$-means clustering, given in Algorithm 1. The motivation behind this method is that mini-batches tend to have lower stochastic noise than individual examples in SGD (allowing convergence to better solutions) but do not suffer increased computational cost when data sets grow large with redundant examples. We use per-center learning rates for fast convergence, in the manner of [1]; convergence properties follow closely from this prior result [1].

**Experiments.** We tested the mini-batch $k$-means against both Lloyd's batch $k$-means [6] and the SGD variant of [1]. We used the standard `RCV1` collection of documents [4] for our experiments. To assess performance at scale, the set of 781,265 examples were used for training and the remaining 23,149 examples for testing. On each trial, the same random initial cluster centers were used for each method. We evaluated the learned cluster centers using the $k$-means objective function on the held-out test set; we report fractional error from the best value found by the batch algorithm run to convergence. We set the mini-batch $b$ to 1000 based on separate initial tests; results were robust for a range of $b$.

The results (Fig. 1) show a clear win for mini-batch $k$-means. The mini-batch method converged to a near optimal value several orders of magnitude faster than the full batch method, and also achieved significantly better solutions than SGD. Additional experiments (omitted for space) showed that mini-batch $k$-means is several times faster on large data sets than batch $k$-means exploiting triangle inequality [3].

For small values of $k$, the mini-batch methods were able to produce near-best cluster centers for nearly a million documents in a fraction of a CPU second on a single ordinary 2.4 GHz machine. This makes real-time clustering practical for user-facing applications.


*WWW 2010*, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

WWW 2010 • Poster April 26-30 • Raleigh • NC • USA

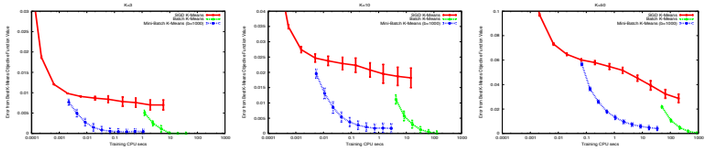


**Figure 1: Convergence Speed. The mini-batch method (blue) is orders of magnitude faster than the full batch method (green), while converging to significantly better solutions than the online SGD method (red).**

### 3. SPARSE CLUSTER CENTERS

We modify mini-batch $k$-means to find sparse cluster centers, allowing for compact storage and low network cost. The intuition for seeking sparse cluster centers for document clusters is that term frequencies follow a power-law distribution. Many terms in a given cluster will only occur in one or two documents, giving them very low weight in the cluster center. It is likely that for a locally optimal center $\mathbf{c}$, there is a nerby point $\mathbf{c}'$ with many fewer non-zero values.

Sparsity may be induced in gradient descent using the projected-gradient method, projecting a given $\mathbf{v}$ to the nearest point in an $L1$-ball of radius $\lambda$ after each update [2]. Thus, for mini-batch $k$-means we achieve sparsity by performing an $L1$-ball projection on each cluster center $\mathbf{c}$ after each mini-batch iteration.

**Algorithm 2** $\epsilon$**-L1**: an $\epsilon$-Accurate Projection to L1 Ball.

```
Given: ε tolerance, L1-ball radius λ, vector c ∈ R^m
if ||c||_1 ≤ λ + ε then exit
upper ← ||c||_∞ ; lower ← 0 ; current ← ||c||_1
while current > λ(1 + ε) or current < λ do
  θ ← (upper+lower)/2.0          // Get L1 value for this θ
  current ← Σ_{c_i≠0} max(0, |c_i| − θ)
  if current ≤ λ then upper ← θ else lower ← θ
end while
for i = 1 to m do
  c_i ← sign(c_i) * max(0, |c_i| − θ) // Do the projection
end for
```

**Fast $L1$ Projections.** Applying $L1$ constraints to $k$-means clustering has been studied in forthcoming work by Witten and Tibshirani [5]. There, a hard $L1$ constraint was applied in the full batch setting of maximizing between-cluster distance for $k$-means (rather than minimizing the $k$-means objective function directly); the work did not discuss how to perform this projection efficiently.

The projection to the $L1$ ball can be performed effectively using, for example, the linear time $L1$-ball projection algorithm of Duchi *et al.* [2], referred to here as LTL1P. We give an alternate method in Algorithm 2, observing that the exact $L1$ radius is not critical for sparsity. This simple approximation algorithm uses bisection to find a value $\theta$ that projects $\mathbf{c}$ to an $L1$ ball with radius between $\lambda$ and $(1+\epsilon)\lambda$. Our method is easy to implement and is also significantly faster in practice than LTL1P due to memory concurrency.

| METHOD | $\lambda$ | #NON-ZERO'S | TEST OBJECTIVE | CPUS |
|---|---|---|---|---|
| full batch | - | 200,319 | 0 (baseline) | 133.96 |
| LTL1P | 5.0 | 46,446 | .004 (.002-.006) | 0.51 |
| $\epsilon$-L1 | 5.0 | 44,060 | .007 (.005-.008) | 0.27 |
| LTL1P | 1.0 | 3,181 | .018 (.016-.019) | 0.48 |
| $\epsilon$-L1 | 1.0 | 2,547 | .028 (.027-.029) | 0.19 |

**Results.** Using the same set-up as above, we tested Duchi *et al.*'s linear time algorithm and our $\epsilon$-accurate projection for mini-batch $k$-means, with a range of $\lambda$ values. The value of $\epsilon$ was arbitrarily set to 0.01. We report values for $k = 10$, $b = 1000$, and $t = 16$ (results for other values qualitatively similar). Compared with the full batch method, we achieve much sparser solutions. The approximate projection is roughly twice as fast as LTL1P and finds sparser solutions, but gives slightly worse performance on the test set. These results show that sparse clustering may cheaply be achieved with low latency for user-facing applications.

### 4. REFERENCES


[1] L. Bottou and Y. Bengio. Convergence properties of the kmeans algorithm. In *Advances in Neural Information Processing Systems*. 1995.

[2] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008.

[3] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML '03: Proceedings of the 20th international conference on Machine learning*, 2003.

[4] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5, 2004.

[5] D. Witten and R. Tibshirani. A framework for feature selection in clustering. To Appear: *Journal of the American Statistical Association*, 2010.

[6] X. Wu and V. Kumar. *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC, 2009.


(2-page abstract)

# Mini-batch K-means



## Web-Scale K-Means Clustering

D. Sculley
Google, Inc. Pittsburgh. PA USA
dsculley@google.com

### ABSTRACT

We present two modifications to the popular $k$-means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. First, we propose the use of mini-batch optimization for $k$-means clustering. This reduces computation cost by orders of magnitude compared to the classic batch algorithm while yielding significantly better solutions than online stochastic gradient descent. Second, we achieve sparsity with projected gradient descent, and give a fast $\epsilon$-accurate projection onto the $L1$-ball. Source code is freely available: `http://code.google.com/p/sofia-ml`

### Categories and Subject Descriptors

I.5.3 [**Computing Methodologies**]: Pattern Recognition—*Clustering*

### General Terms

Algorithms, Performance, Experimentation

### Keywords

unsupervised clustering, scalability, sparse solutions

### 1. CLUSTERING AND THE WEB

Unsupervised clustering is an important task in a range of web-based applications, including grouping search results, near-duplicate detection, and news aggregation to name but a few. Lloyd's classic $k$-means algorithm remains a popular choice for real-world clustering tasks [6]. However, the standard batch algorithm is slow for large data sets. Even optimized batch $k$-means variants exploiting triangle inequality [3] cannot cheaply meet the latency needs of user-facing applications when clustering results on large data sets are required in a fraction of a second.

This paper proposes a mini-batch $k$-means variant that yields excellent clustering results with low computation cost on large data sets. We also give methods for learning sparse cluster centers that reduce storage and network cost.

### 2. MINI-BATCH K-MEANS

The $k$-means optimization problem is to find the set $C$ of cluster centers $\mathbf{c} \in \mathbb{R}^m$, with $|C| = k$, to minimize over a set $X$ of examples $\mathbf{x} \in \mathbb{R}^m$ the following objective function:

$$\min \sum_{\mathbf{x} \in X} ||f(C, \mathbf{x}) - \mathbf{x}||^2$$

Here, $f(C, \mathbf{x})$ returns the nearest cluster center $\mathbf{c} \in C$ to $\mathbf{x}$ using Euclidean distance. It is well known that although this problem is NP-hard in general, gradient descent methods converge to a local optimum when seeded with an initial set of $k$ examples drawn uniformly at random from $X$ [1].

The classic batch $k$-means algorithm is expensive for large data sets, requiring $O(kns)$ computation time where $n$ is the number of examples and $s$ is the maximum number of non-zero elements in any example vector. Bottou and Bengio proposed an online, stochastic gradient descent (SGD) variant that computed a gradient descent step on one example at a time [1]. While SGD converges quickly on large data sets, it finds [illegible] due to stochastic noise [1].

**Algorithm 1** Mini-batch $k$-Means.

```
Given: k, mini-batch size b, iterations t, data set X
Initialize each c ∈ C with an x picked randomly from X
v ← 0
for i = 1 to t do
    M ← b examples picked randomly from X
    for x ∈ M do
        d[x] ← f(C, x)    // Cache the center nearest to x
    end for
    for x ∈ M do
        c ← d[x]          // Get cached center for this x
        v[c] ← v[c] + 1   // Update per-center counts
        η ← 1/v[c]        // Get per-center learning rate
        c ← (1 − η)c + ηx // Take gradient step
    end for
end for
```

We propose the use of mini-batch optimization for $k$-means clustering, given in Algorithm 1. The motivation behind this method is that mini-batches tend to have lower stochastic noise than individual examples in SGD (allowing convergence to better solutions) but do not suffer increased computational cost when data sets grow large with redundant examples. We use per-center learning rates for fast convergence, in the manner of [1]; convergence properties follow closely from this prior result [1].

**Experiments.** We tested the mini-batch $k$-means against both Lloyd's batch $k$-means [6] and the SGD variant of [1]. We used the standard `RCV1` collection of documents [4] for our experiments. To assess performance at scale, the set of 781,265 examples were used for training and the remaining 23,149 examples for testing. On each trial, the same random initial cluster centers were used for each method. We evaluated the learned cluster centers using the $k$-means objective function on the held-out test set; we report fractional error from the best value found by the batch algorithm run to convergence. We set the mini-batch $b$ to 1000 based on separate initial tests; results were robust for a range of $b$.

The results (Fig. 1) show a clear win for mini-batch $k$-means. The mini-batch method converged to a near optimal value several orders of magnitude faster than the full batch method, and also achieved significantly better solutions than SGD. Additional experiments (omitted for space) showed that mini-batch $k$-means is several times faster on large data sets than batch $k$-means exploiting triangle inequality [3].

For small values of $k$, the mini-batch methods were able to produce near-best cluster centers for nearly a million documents in a fraction of a CPU second on a single ordinary 2.4 GHz machine. This makes real-time clustering practical for user-facing applications.


*WWW 2010*, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

WWW 2010 • Poster April 26-30 • Raleigh • NC • USA

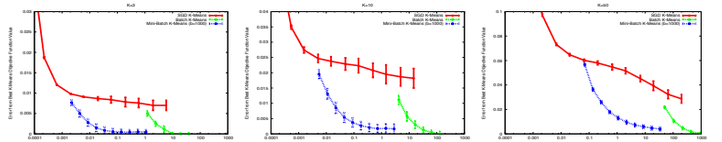


**Figure 1: Convergence Speed. The mini-batch method (blue) is orders of magnitude faster than the full batch method (green), while converging to significantly better solutions than the online SGD method (red).**

### 3. SPARSE CLUSTER CENTERS

We modify mini-batch $k$-means to find sparse cluster centers, allowing for compact storage and low network cost. The intuition for seeking sparse cluster centers for document clusters is that term frequencies follow a power-law distribution. Many terms in a given cluster will only occur in one or two documents, giving them very low weight in the cluster center. It is likely that for a locally optimal center $\mathbf{c}$, there is a nerby point $\mathbf{c}'$ with many fewer non-zero values.

Sparsity may be induced in gradient descent using the projected-gradient method, projecting a given $\mathbf{v}$ to the nearest point in an $L1$-ball of radius $\lambda$ after each update [2]. Thus, for mini-batch $k$-means we achieve sparsity by performing an $L1$-ball projection on each cluster center $\mathbf{c}$ after each mini-batch iteration.

**Algorithm 2** $\epsilon$-**L1**: an $\epsilon$-Accurate Projection to L1 Ball.

```
Given: ε tolerance, L1-ball radius λ, vector c ∈ R^m
if ||c||_1 ≤ λ + ε then exit
upper ← ||c||_∞ ; lower ← 0 ; current ← ||c||_1
while current > λ(1 + ε) or current < λ do
    θ ← (upper+lower)/2.0        // Get L1 value for this θ
    current ← Σ_{c_i ≠ 0} max(0, |c_i| − θ)
    if current ≤ λ then upper ← θ else lower ← θ
end while
for i = 1 to m do
    c_i ← sign(c_i) * max(0, |c_i| − θ) // Do the projection
end for
```

**Fast $L1$ Projections.** Applying $L1$ constraints to $k$-means clustering has been studied in forthcoming work by Witten and Tibshirani [5]. There, a hard $L1$ constraint was applied in the full batch setting of maximizing between-cluster distance for $k$-means (rather than minimizing the $k$-means objective function directly); the work did not discuss how to perform this projection efficiently.

The projection to the $L1$ ball can be performed effectively using, for example, the linear time $L1$-ball projection algorithm of Duchi *et al.* [2], referred to here as LTL1P. We give an alternate method in Algorithm 2, observing that the exact $L1$ radius is not critical for sparsity. This simple approximation algorithm uses bisection to find a value $\theta$ that projects $\mathbf{c}$ to an $L1$ ball with radius between $\lambda$ and $(1+\epsilon)\lambda$. Our method is easy to implement and is also significantly faster in practice than LTL1P due to memory concurrency.

| METHOD | $\lambda$ | #NON-ZERO'S | TEST OBJECTIVE | CPUs |
|---|---|---|---|---|
| full batch | - | 200,319 | 0 (baseline) | 133.96 |
| LTL1P | 5.0 | 46,446 | .004 (.002-.006) | 0.51 |
| $\epsilon$-L1 | 5.0 | 44,060 | .007 (.005-.008) | 0.27 |
| LTL1P | 1.0 | 3,181 | .018 (.016-.019) | 0.48 |
| $\epsilon$-L1 | 1.0 | 2,547 | .028 (.027-.029) | 0.19 |

**Results.** Using the same set-up as above, we tested Duchi *et al.*'s linear time algorithm and our $\epsilon$-accurate projection for mini-batch $k$-means, with a range of $\lambda$ values. The value of $\epsilon$ was arbitrarily set to 0.01. We report values for $k = 10$, $b = 1000$, and $t = 16$ (results for other values qualitatively similar). Compared with the full batch method, we achieve much sparser solutions. The approximate projection is roughly twice as fast as LTL1P and finds sparser solutions, but gives slightly worse performance on the test set. These results show that sparse clustering may cheaply be achieved with low latency for user-facing applications.

### 4. REFERENCES

[1] L. Bottou and Y. Bengio. Convergence properties of the kmeans algorithm. In *Advances in Neural Information Processing Systems*. 1995.

[2] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008.

[3] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML '03: Proceedings of the 20th international conference on Machine learning*, 2003.

[4] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5, 2004.

[5] D. Witten and R. Tibshirani. A framework for feature selection in clustering. To Appear: *Journal of the American Statistical Association*, 2010.

[6] X. Wu and V. Kumar. *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC, 2009.

(2-page abstract)

# Mini-batch K-means

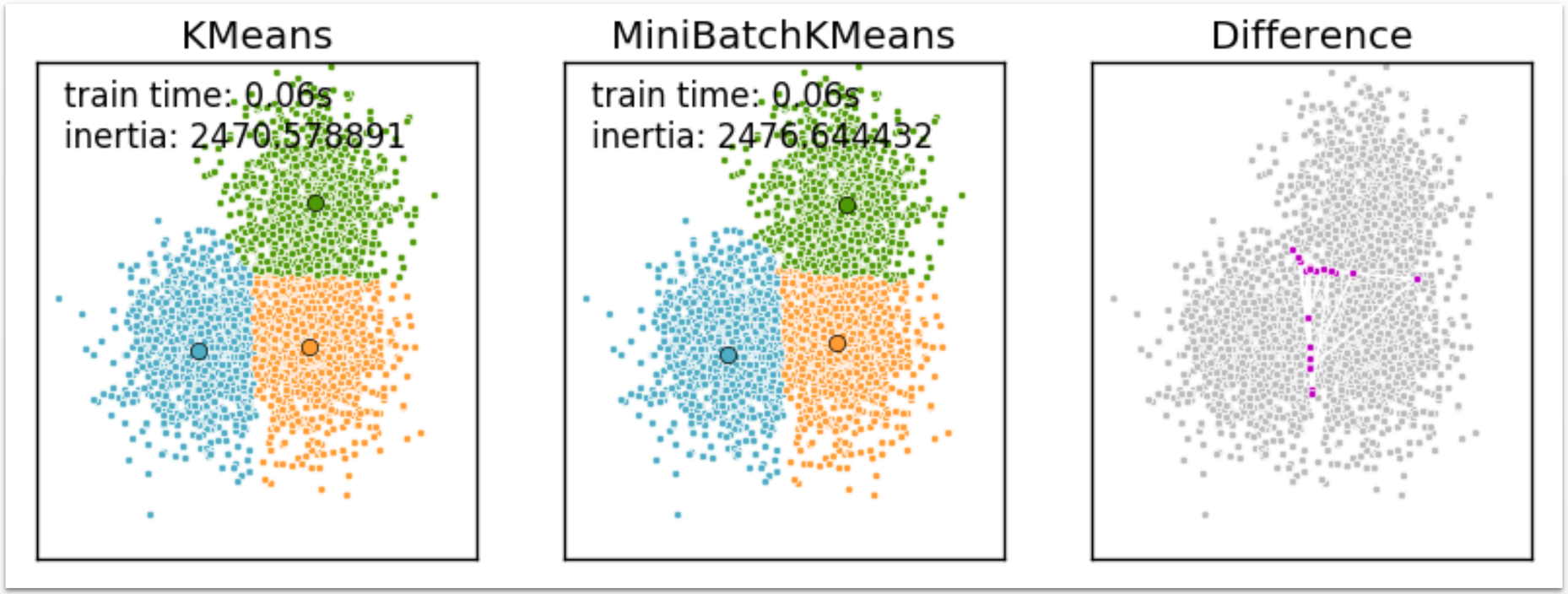
**Algorithm 1 Mini-batch $k$-Means.**

Given: $k$, mini-batch size $b$, iterations $t$, data set $X$
Initialize each $\mathbf{c} \in C$ with an $\mathbf{x}$ picked randomly from $X$
$\mathbf{v} \leftarrow 0$
**for** $i = 1$ to $t$ **do**
  $M \leftarrow b$ examples picked randomly from $X$
  **for** $\mathbf{x} \in M$ **do**
    $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$ // Cache the center nearest to $\mathbf{x}$
  **end for**
  **for** $\mathbf{x} \in M$ **do**
    $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$ // Get cached center for this $\mathbf{x}$
    $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$ // Update per-center counts
    $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$ // Get per-center learning rate
    $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$ // Take gradient step
  **end for**
**end for**

Complexity: O(~~N~~ M K D t)

Here t is the number of iterations

# Mini-batch K-means



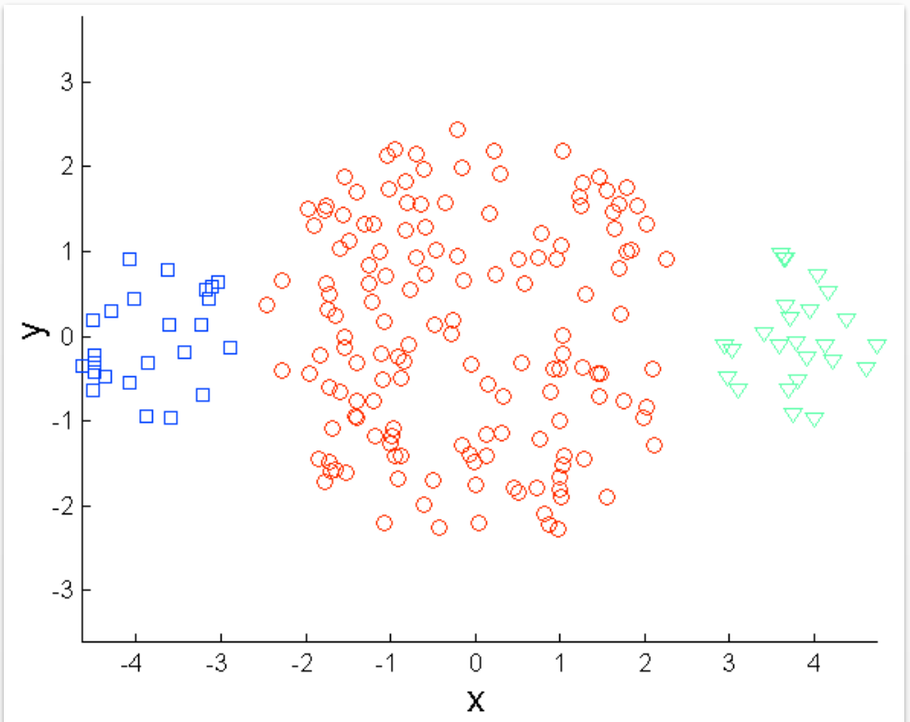

# Clustering

Shantanu Jain

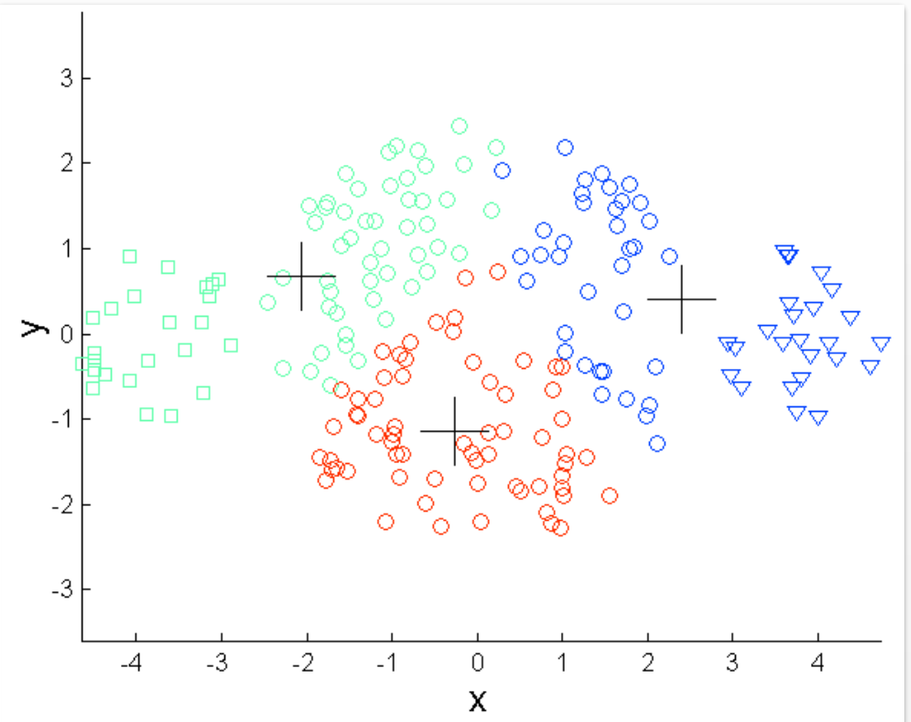

# K-means Clustering

## Limitations

# K-means Limitations: Differing Sizes

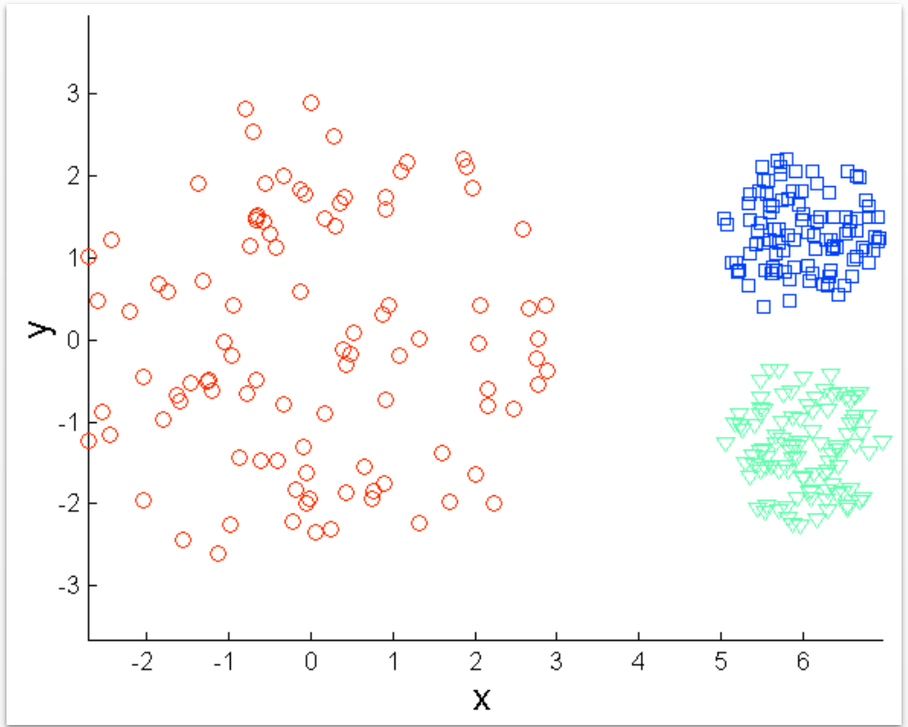



Original Points

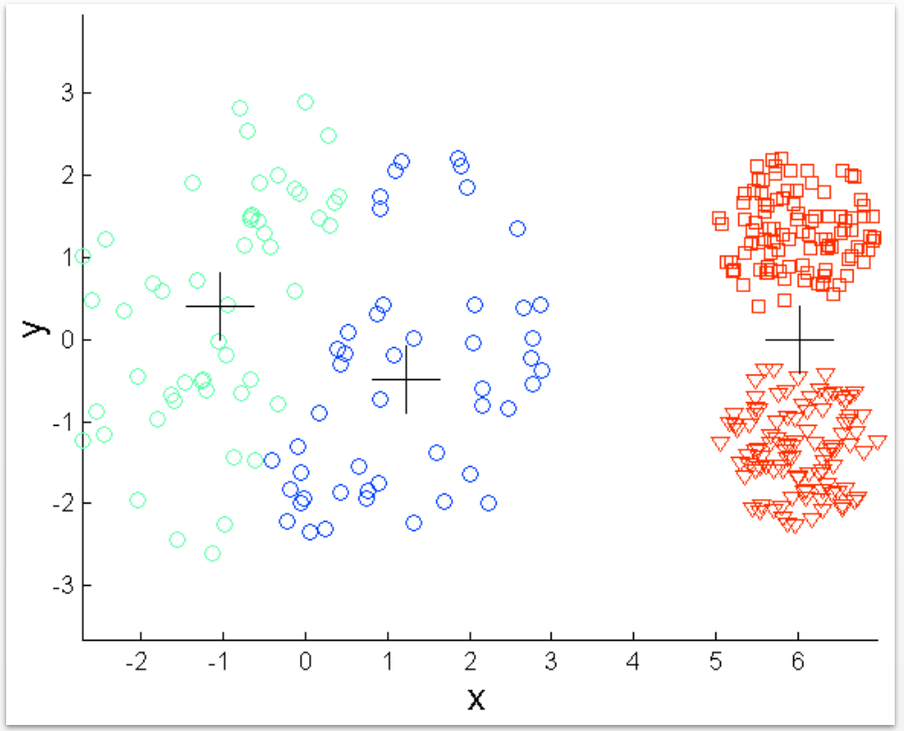



K-means (3 clusters)

# K-means Limitations: Different Densities

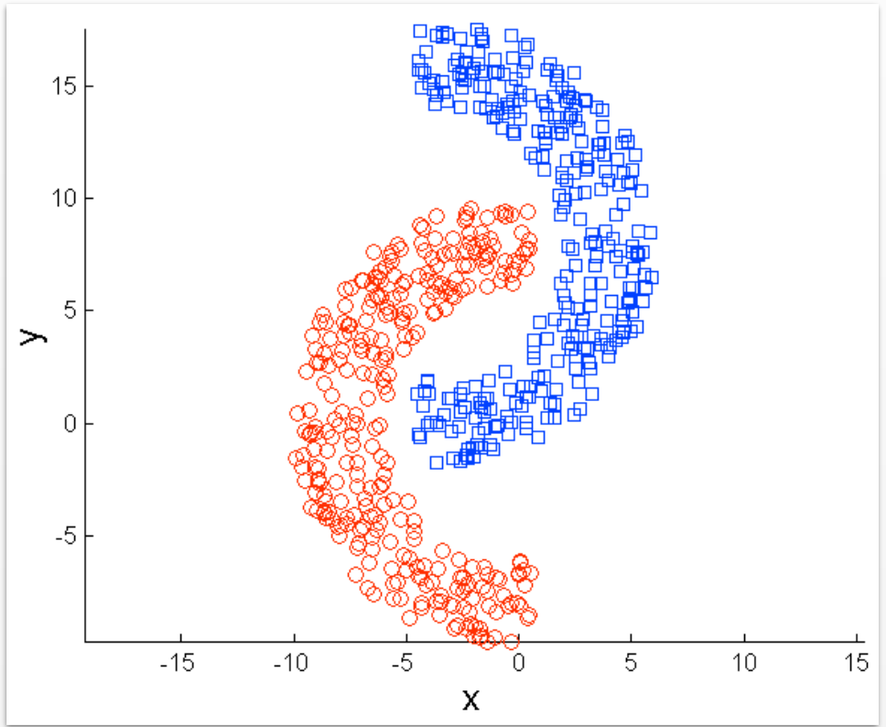



Original Points

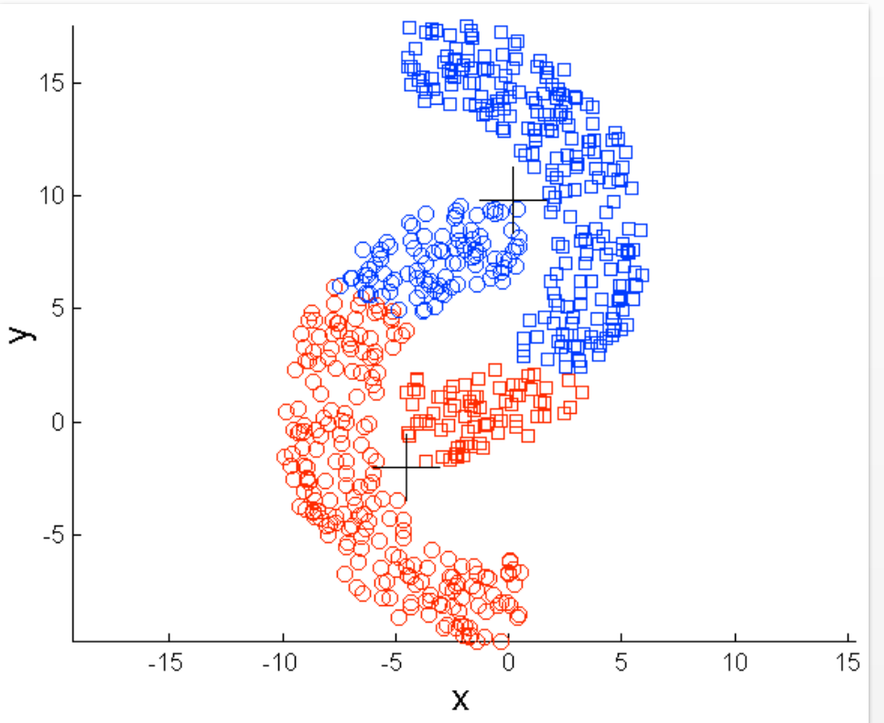



K-means (3 clusters)

# K-means Limitations: Non-globular Shapes

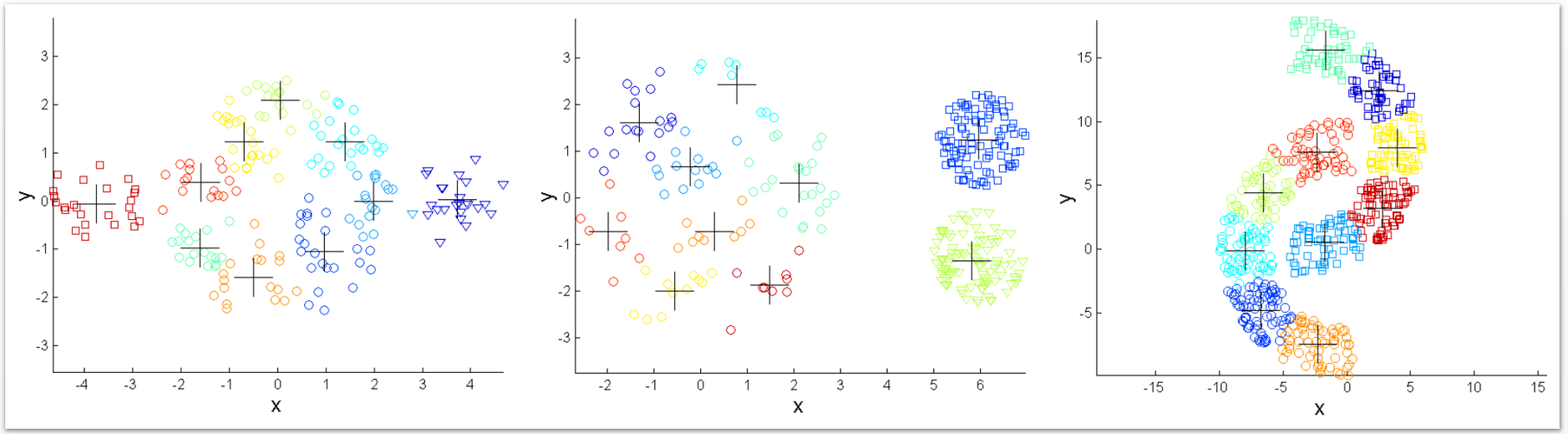



Original Points




K-means (2 clusters)

# Overcoming K-means Limitations




*Intuition:* "Combine" smaller clusters into larger clusters

- *One Solution:* Hierarchical Clustering
- *Another Solution:* Density-based Clustering

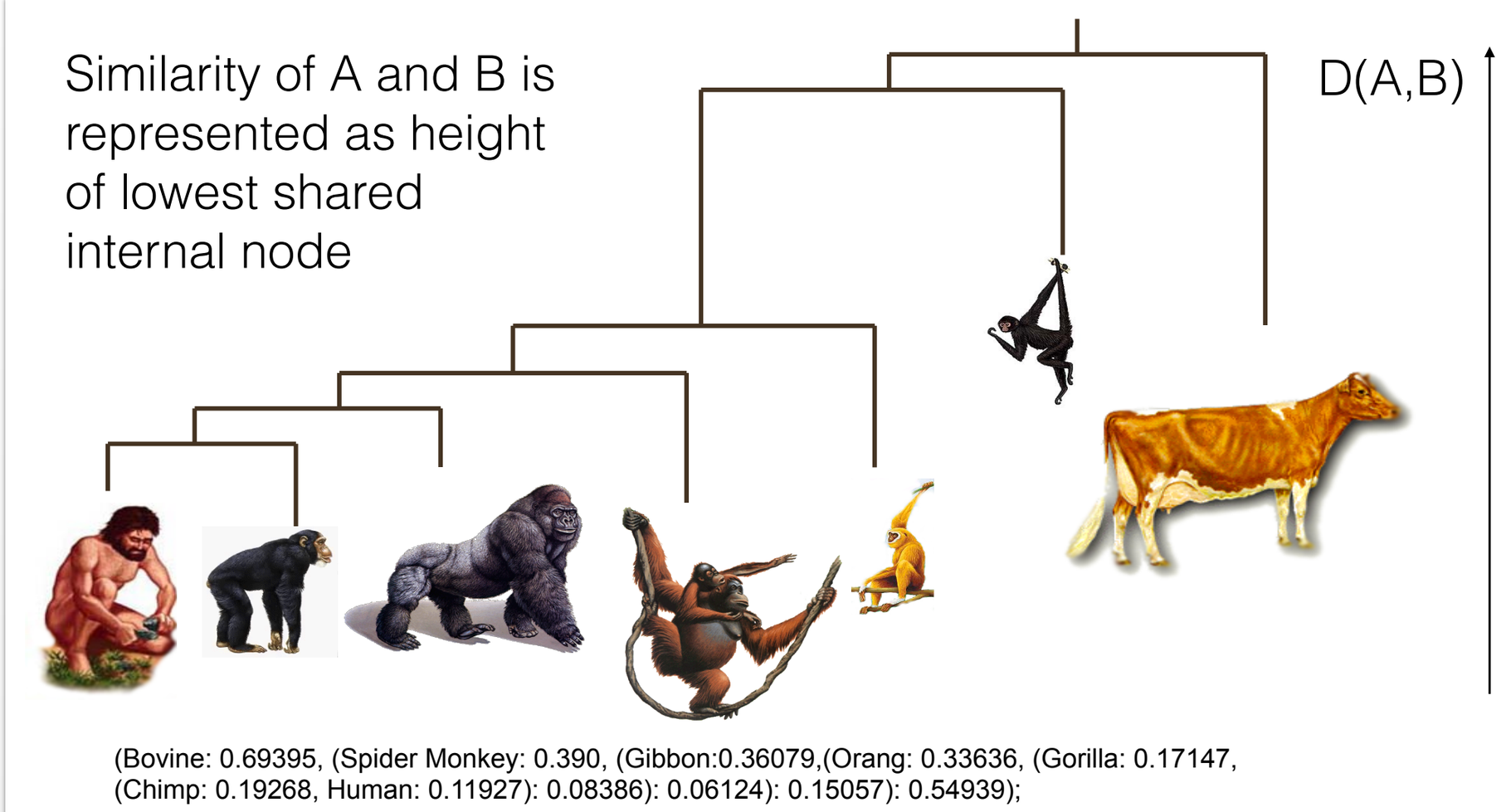

# Clustering

Shantanu Jain


# Hierarchical Clustering

# Dendrogram

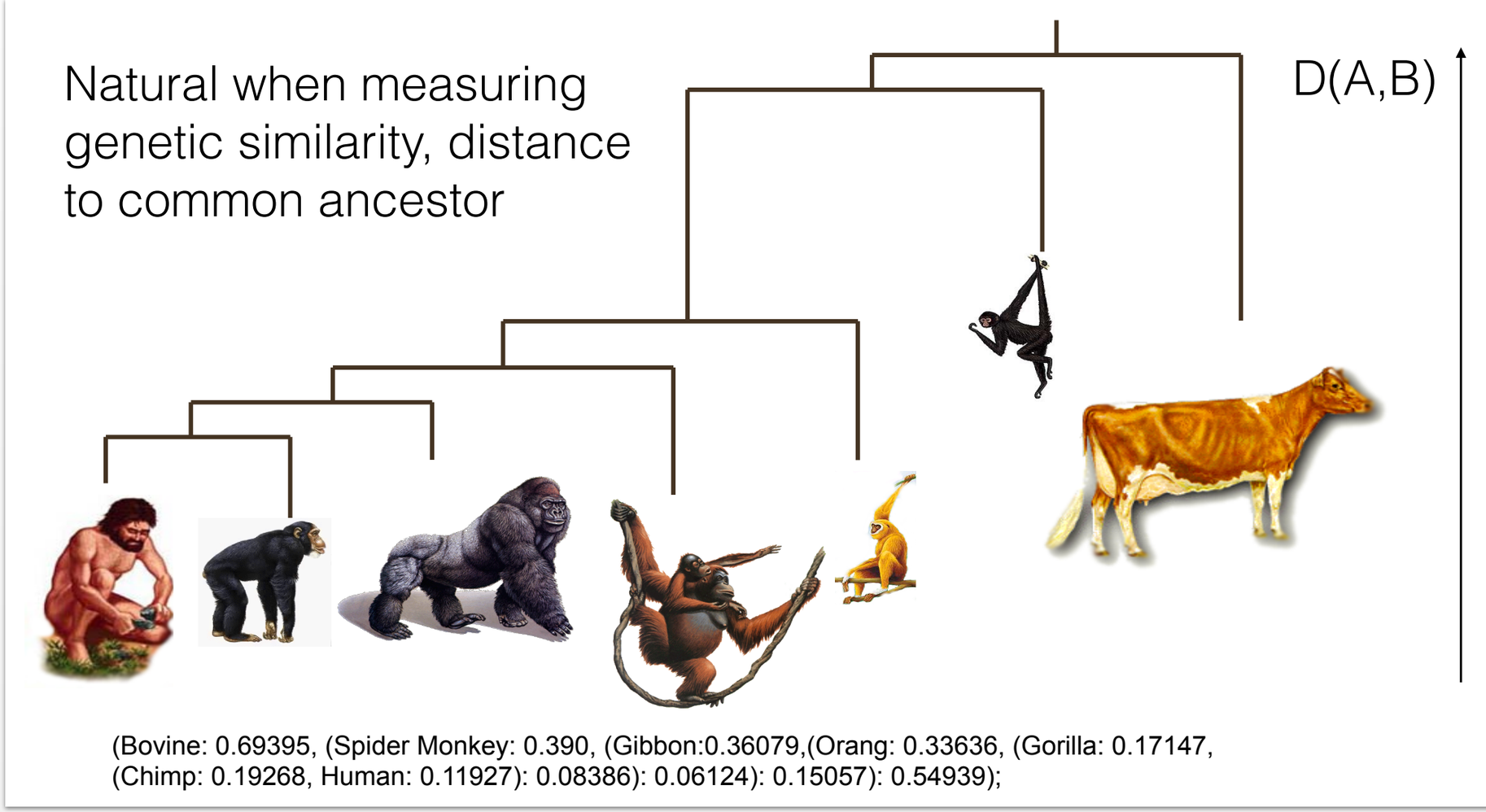
(*a.k.a. a similarity tree*)




(Bovine: 0.69395, (Spider Monkey: 0.390, (Gibbon:0.36079,(Orang: 0.33636, (Gorilla: 0.17147, (Chimp: 0.19268, Human: 0.11927): 0.08386): 0.06124): 0.15057): 0.54939);

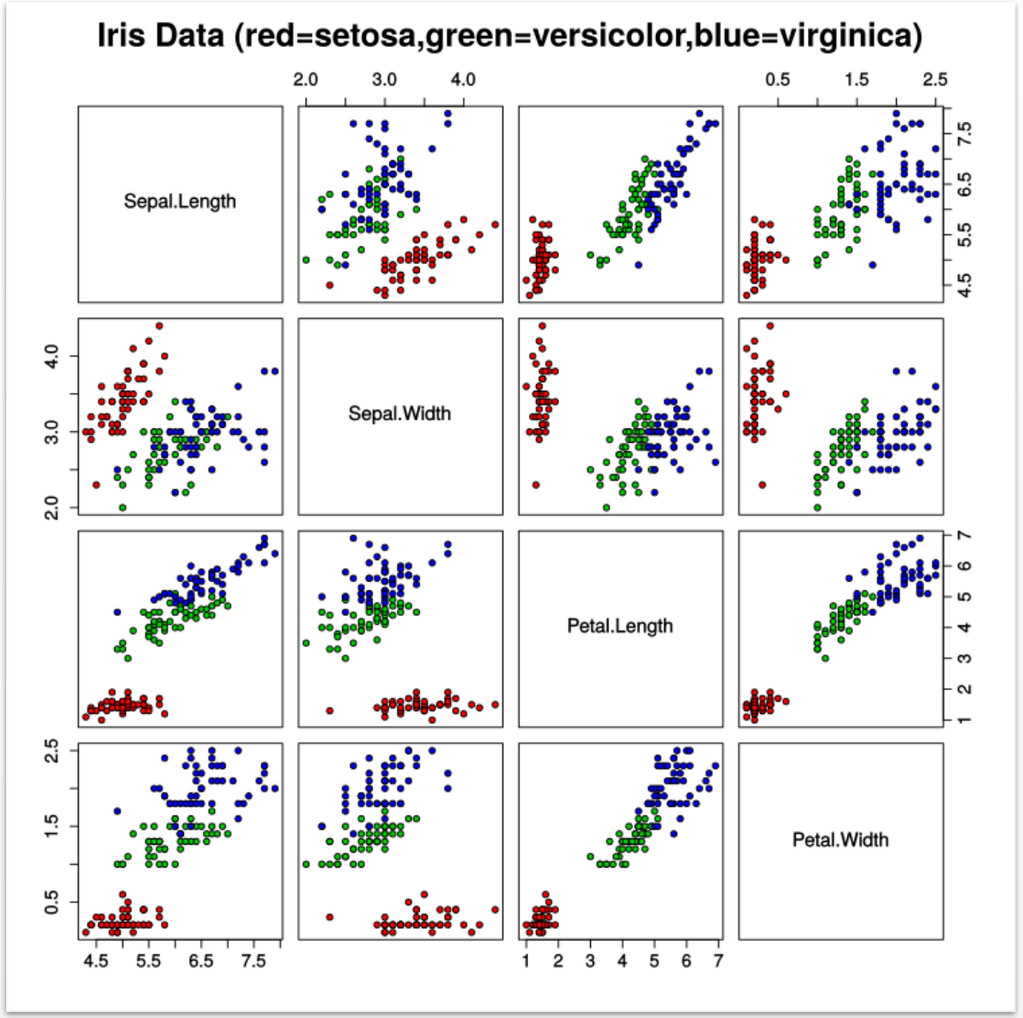
# Dendrogram

(*a.k.a. a similarity tree*)




(Bovine: 0.69395, (Spider Monkey: 0.390, (Gibbon:0.36079,(Orang: 0.33636, (Gorilla: 0.17147, (Chimp: 0.19268, Human: 0.11927): 0.08386): 0.06124): 0.15057): 0.54939);

# Example: Iris data




Iris Setosa

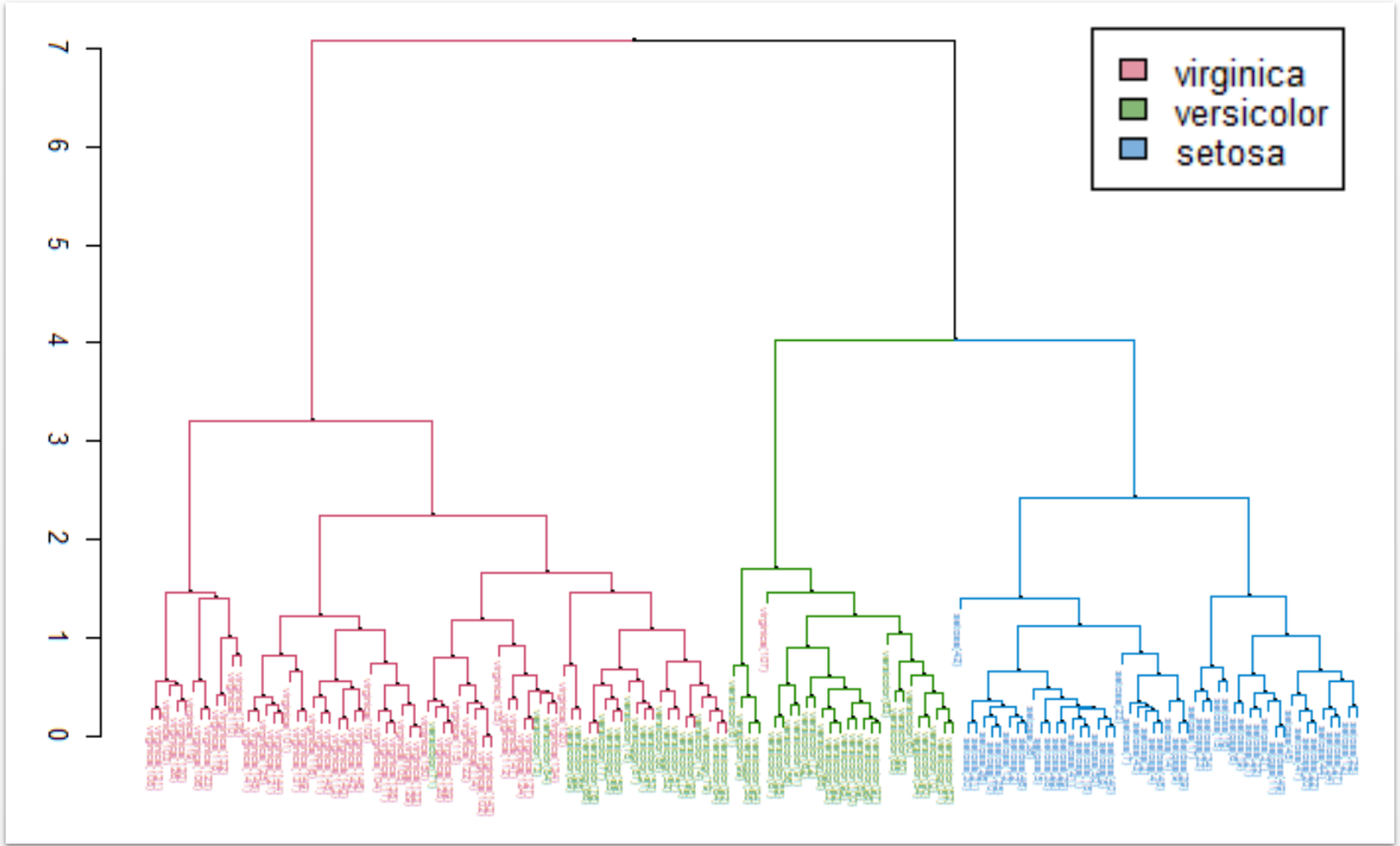
Iris versicolor

Iris virginica

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

# Hierarchical Clustering

(*Euclidian Distance*)




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

# Hamming Distance

## Distance Patty and Selma

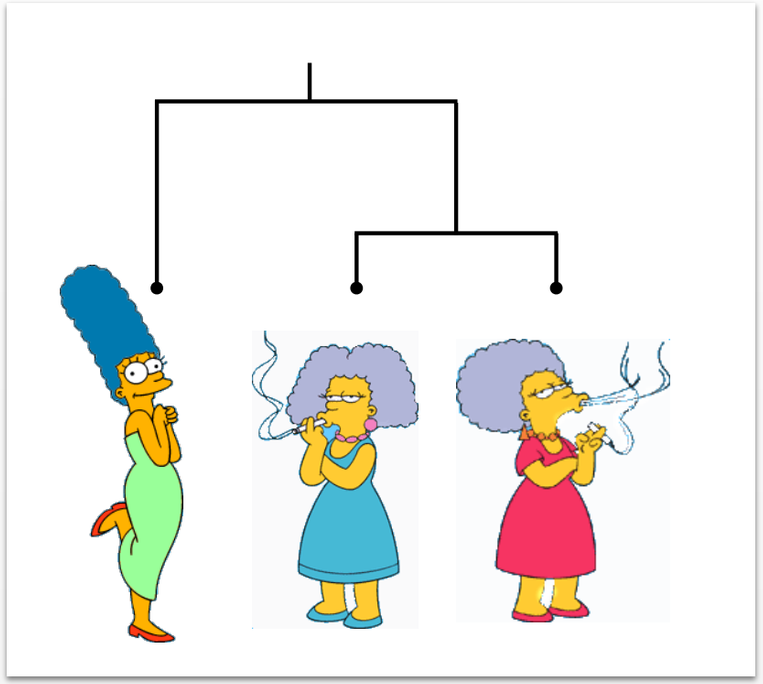
```
Change dress color,    1 point
Change earring shape,  1 point
Change hair part,      1 point
```

D(Patty, Selma) = **3**

## Distance Marge and Selma

```
Change dress color,    1 point
Add earrings,          1 point
Decrease height,       1 point
Take up smoking,       1 point
Lose weight,           1 point
```

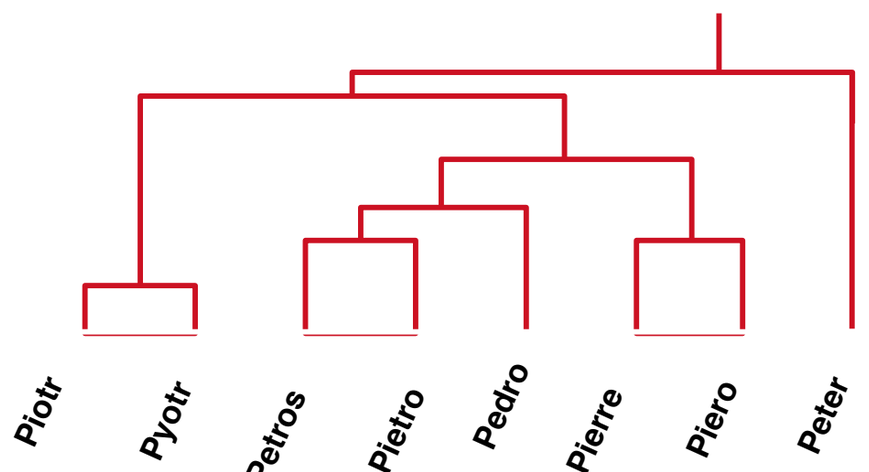
D(Marge,Selma) = **5**



Can be defined for any set of discrete features

# Edit Distance for Strings

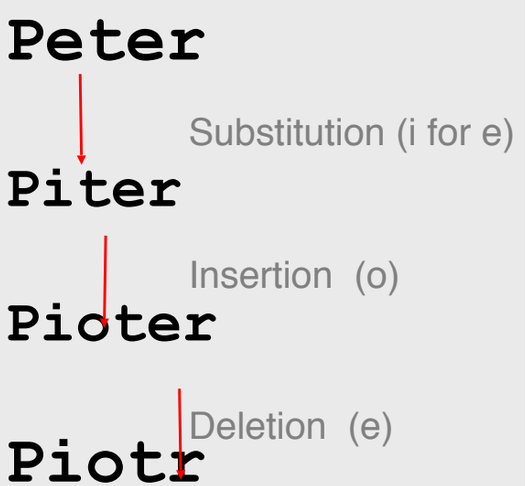
- Transform string *Q* into string *C*, using only ***Substitution***, ***Insertion*** and ***Deletion***.
- Assume that each of these operators has a **cost** associated with it.
- The similarity between two strings can be defined as the cost of the ***cheapest*** transformation from *Q* to *C.*




Similarity “Peter” and “Piotr”?

| | |
|---|---|
| *Substitution* | 1 Unit |
| *Insertion* | 1 Unit |
| *Deletion* | 1 Unit |

*D*(**Peter**,**Piotr**) is 3



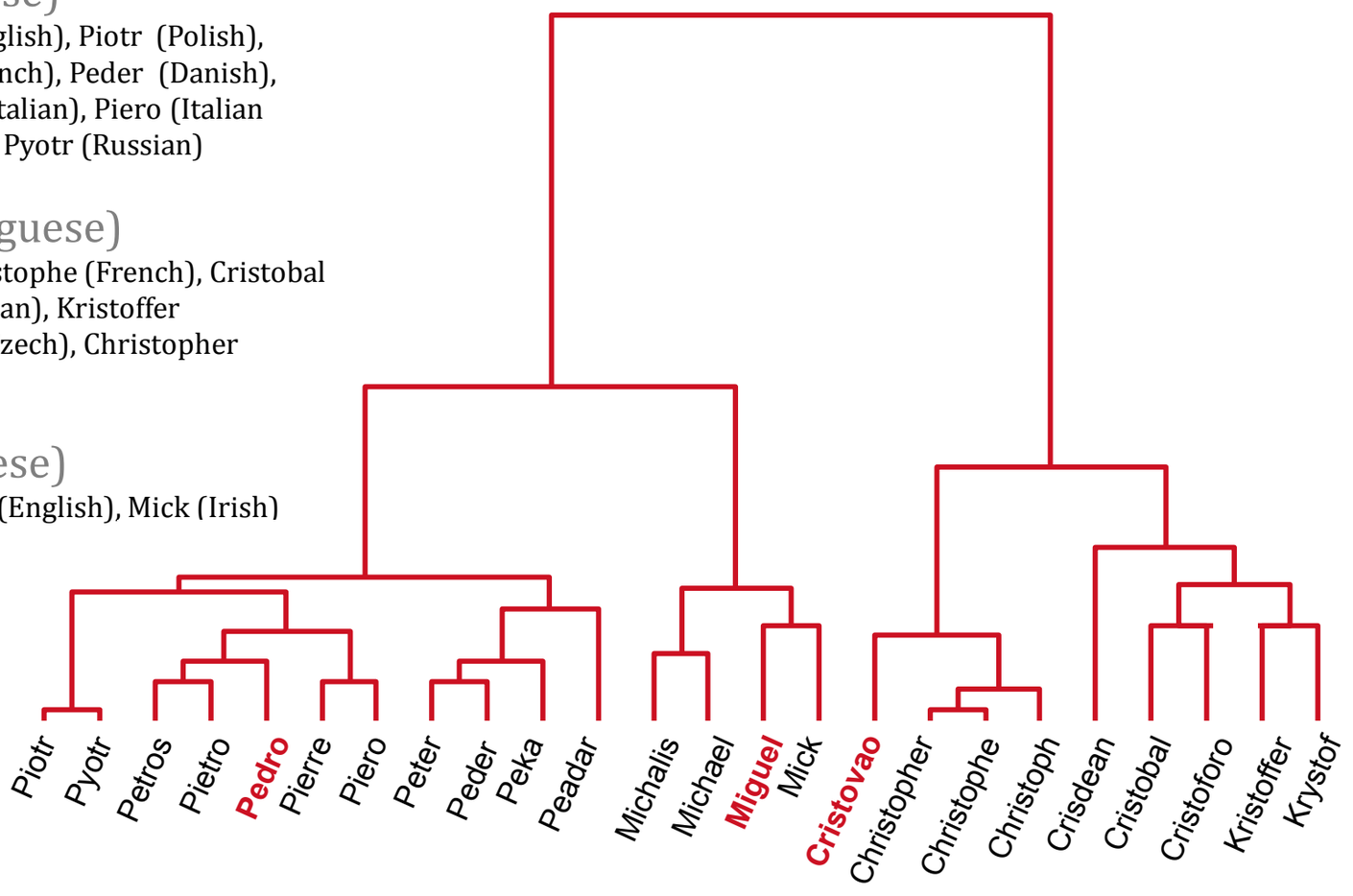
# Hierarchical Clustering

(*Edit Distance*)

**Pedro** (Portuguese)
Petros (Greek), Peter (English), Piotr (Polish), Peadar (Irish), Pierre (French), Peder (Danish), Peka (Hawaiian), Pietro (Italian), Piero (Italian Alternative), Petr (Czech), Pyotr (Russian)

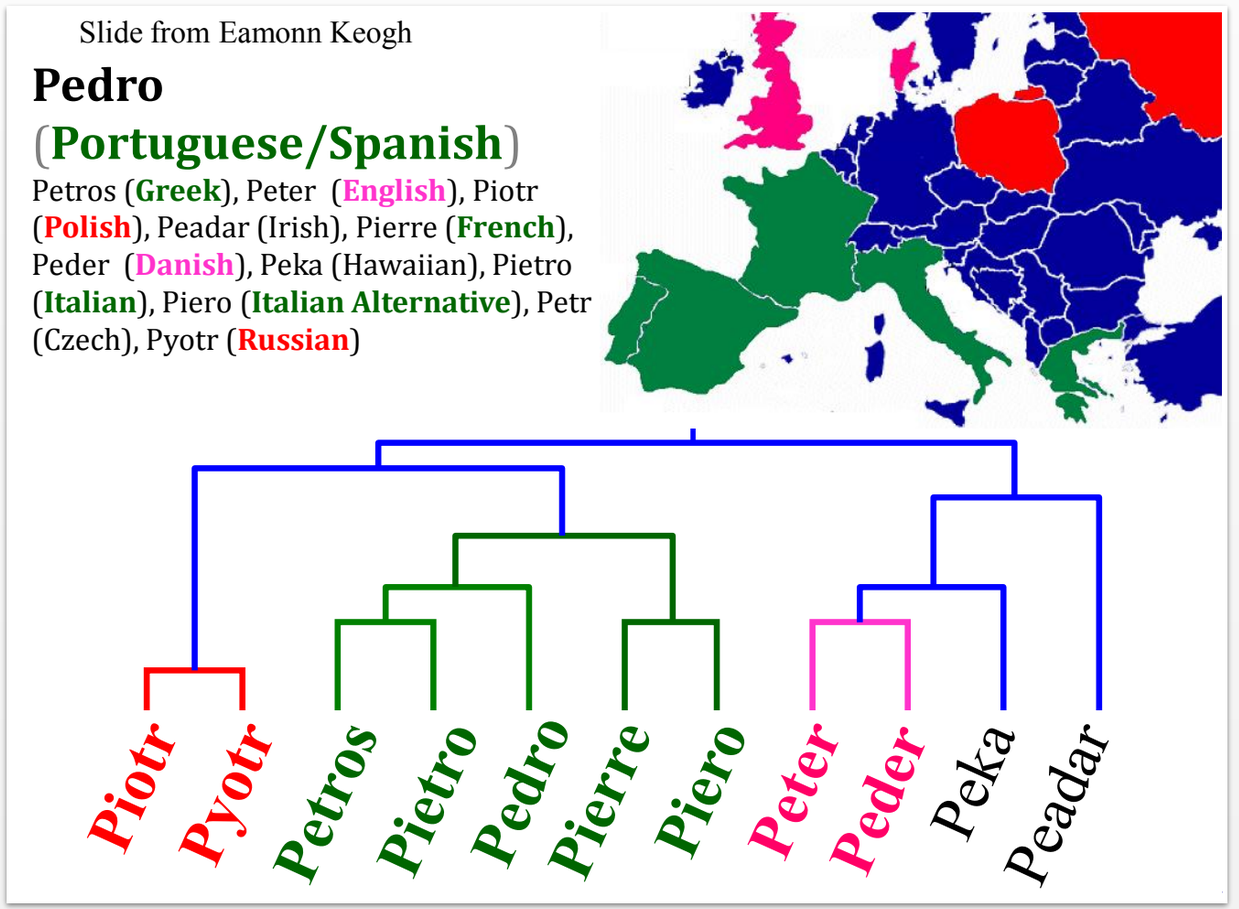
**Cristovao** (Portuguese)
Christoph (German), Christophe (French), Cristobal (Spanish), Cristoforo (Italian), Kristoffer (Scandinavian), Krystof (Czech), Christopher (English)

**Miguel** (Portuguese)
Michalis (Greek), Michael (English), Mick (Irish)



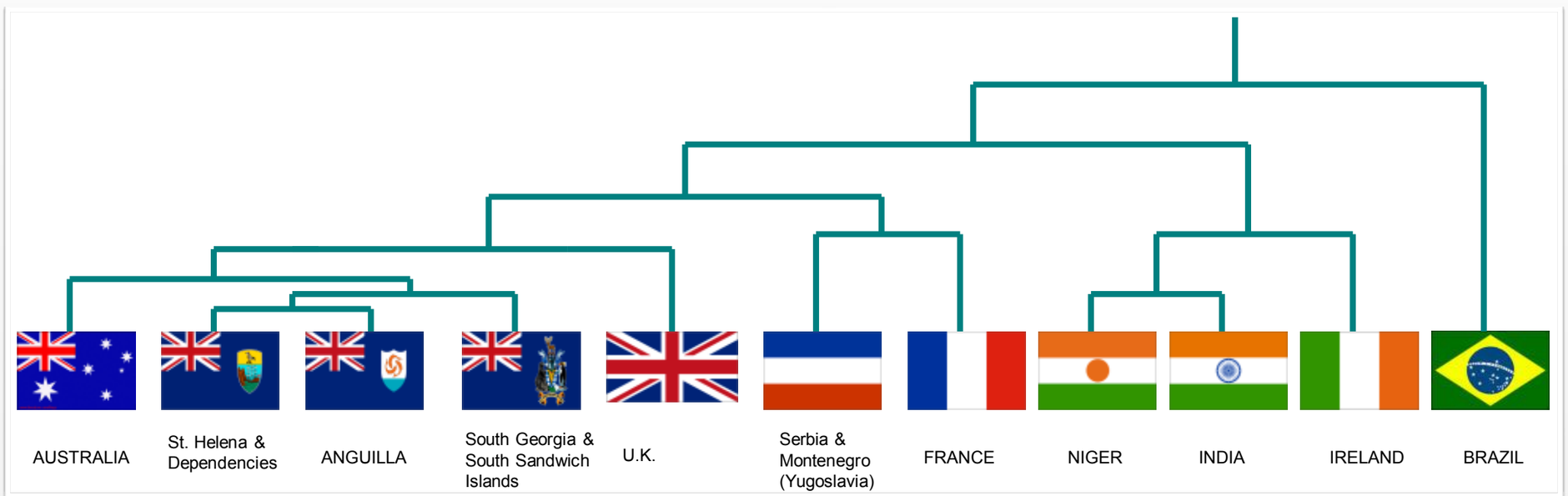
# Meaningful Patterns

Edit distance yields clustering according to geography



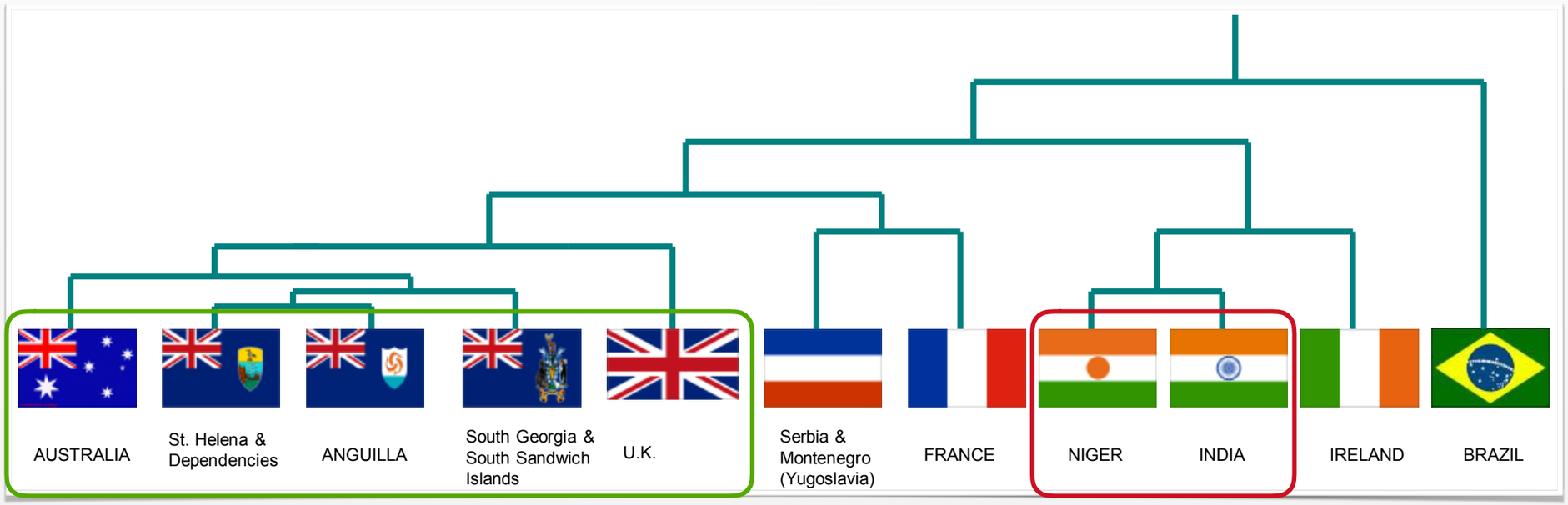
# Spurious Patterns

In general clusterings will only be
as meaningful as your distance metric



# Spurious Patterns

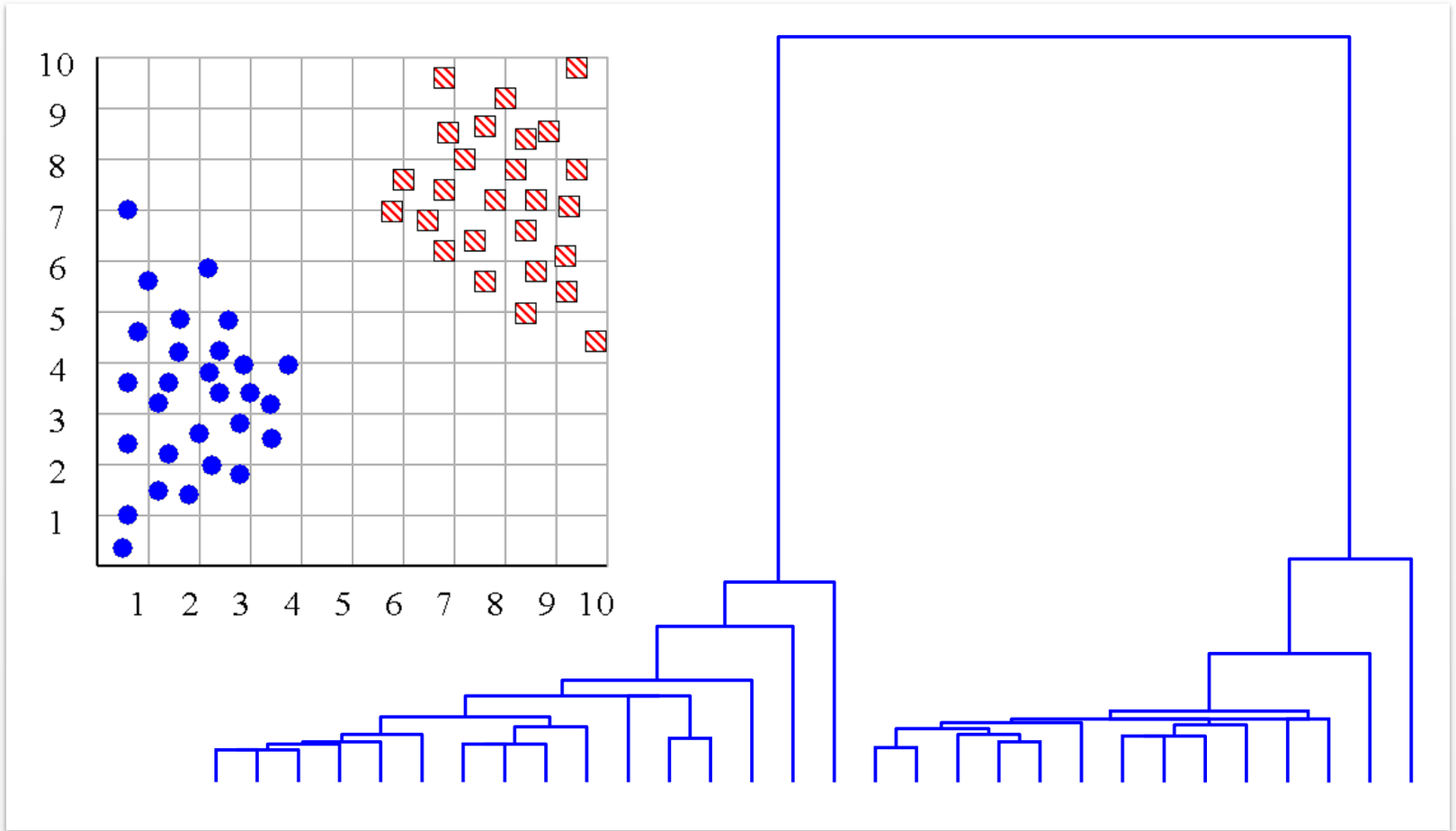
In general clusterings will only be
as meaningful as your distance metric



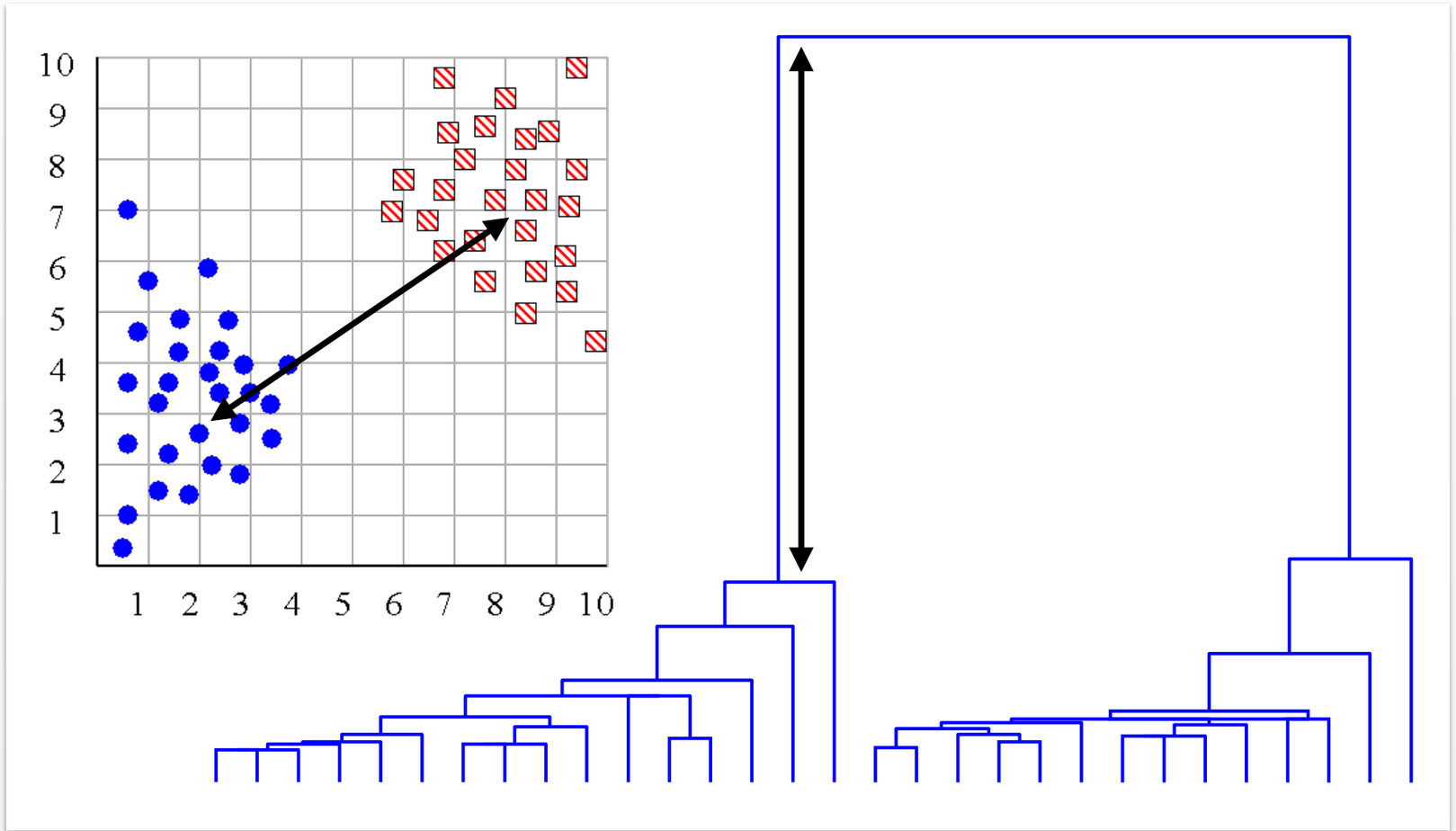

*Former UK colonies*

*No relation*

# “Correct” Number of Clusters

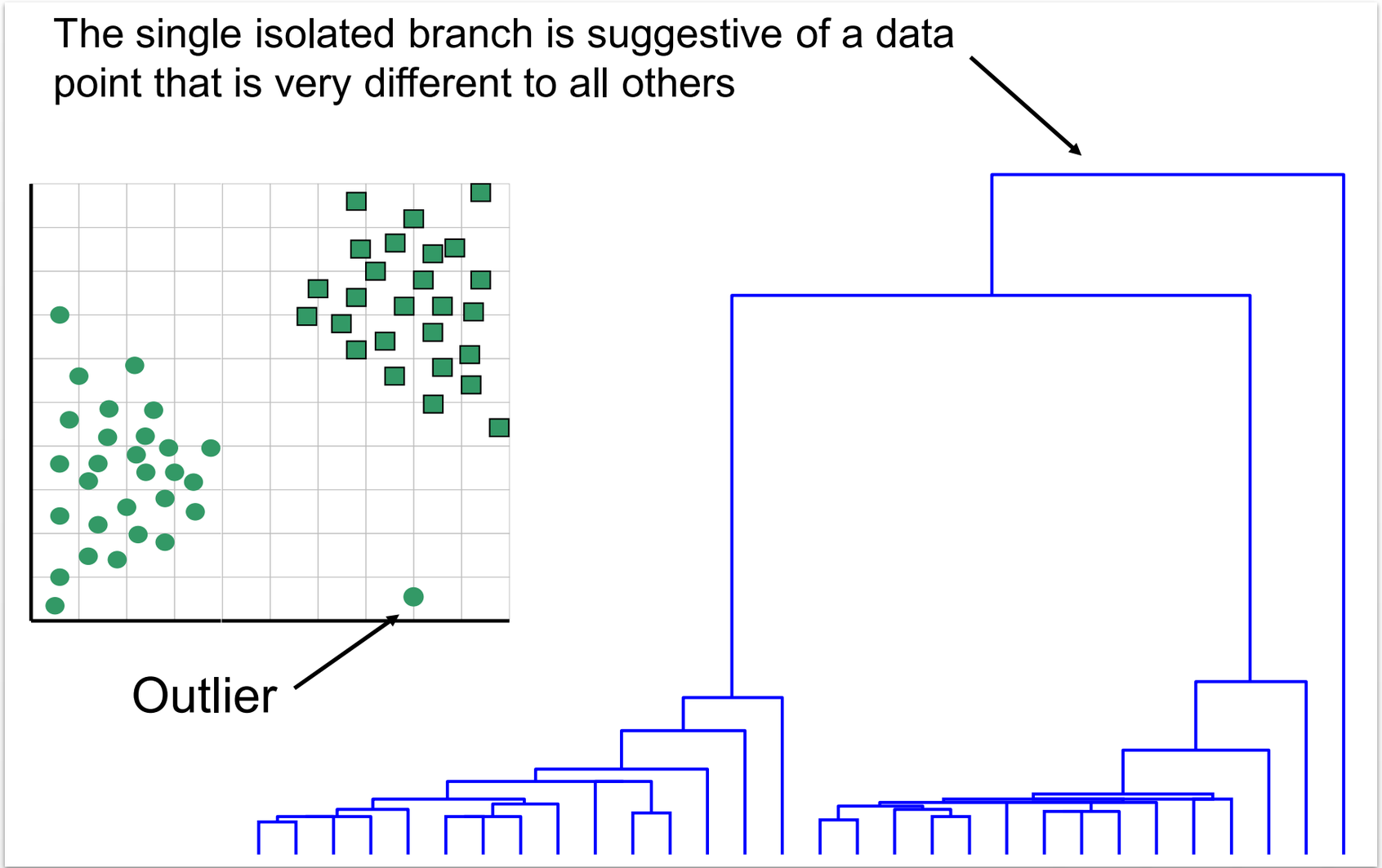


# “Correct” Number of Clusters

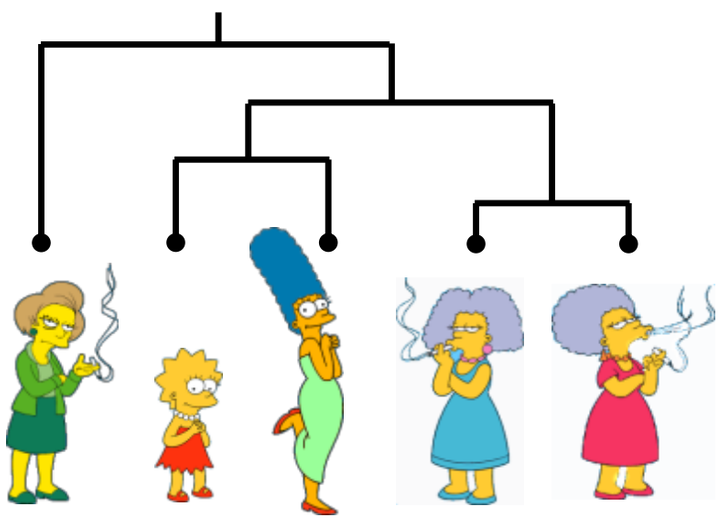



Determine number of clusters by looking at distance

# Detecting Outliers



# Bottom-up vs Top-down

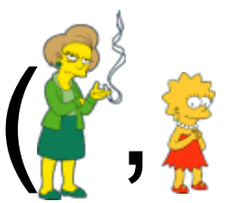


**Bottom-Up (agglomerative):** Starting with each item in its own cluster, find the best pair to merge into a new cluster. Repeat until all clusters are fused together.

**Top-Down (divisive):** Starting with all the data in a single cluster, consider every possible way to divide the cluster into two. Choose the best division and recursively operate on both sides.

# Bottom-up: Distance Matrix

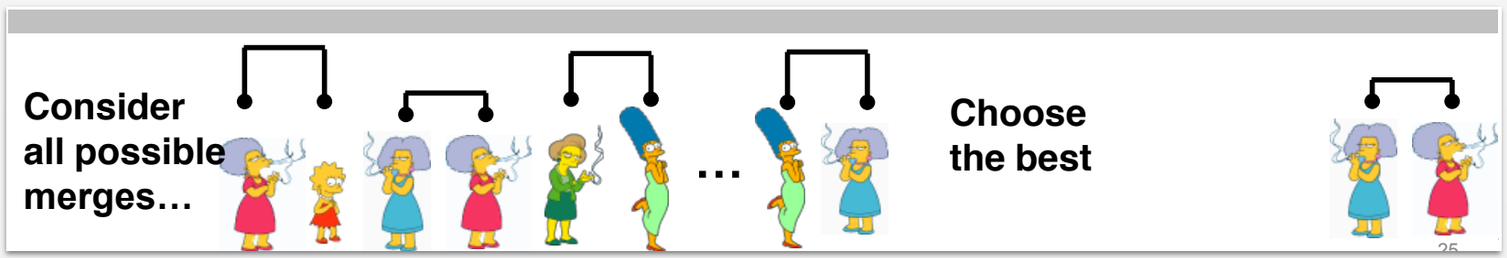
**We begin with a distance matrix which contains the distances between every pair of objects in our database.**

D(  ) = 8

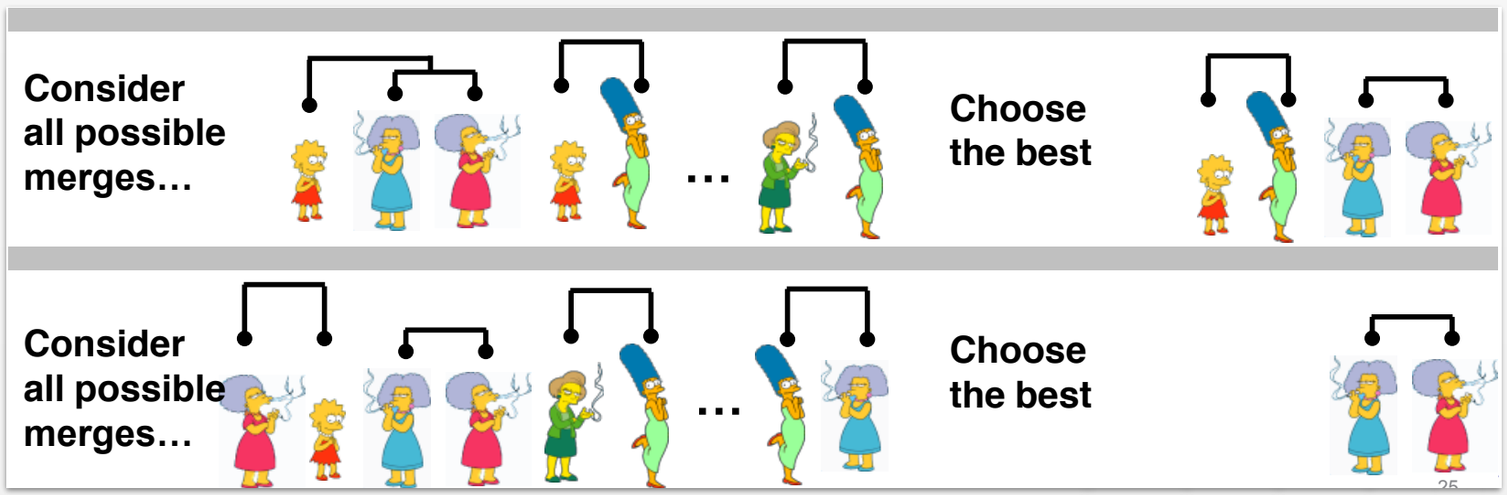
D(  ) = 1

| | | | | |
|---|---|---|---|---|
| 0 | 8 | 8 | 7 | 7 |
| | 0 | 2 | 4 | 4 |
| | | 0 | 3 | 3 |
| | | | 0 | 1 |
| | | | | 0 |

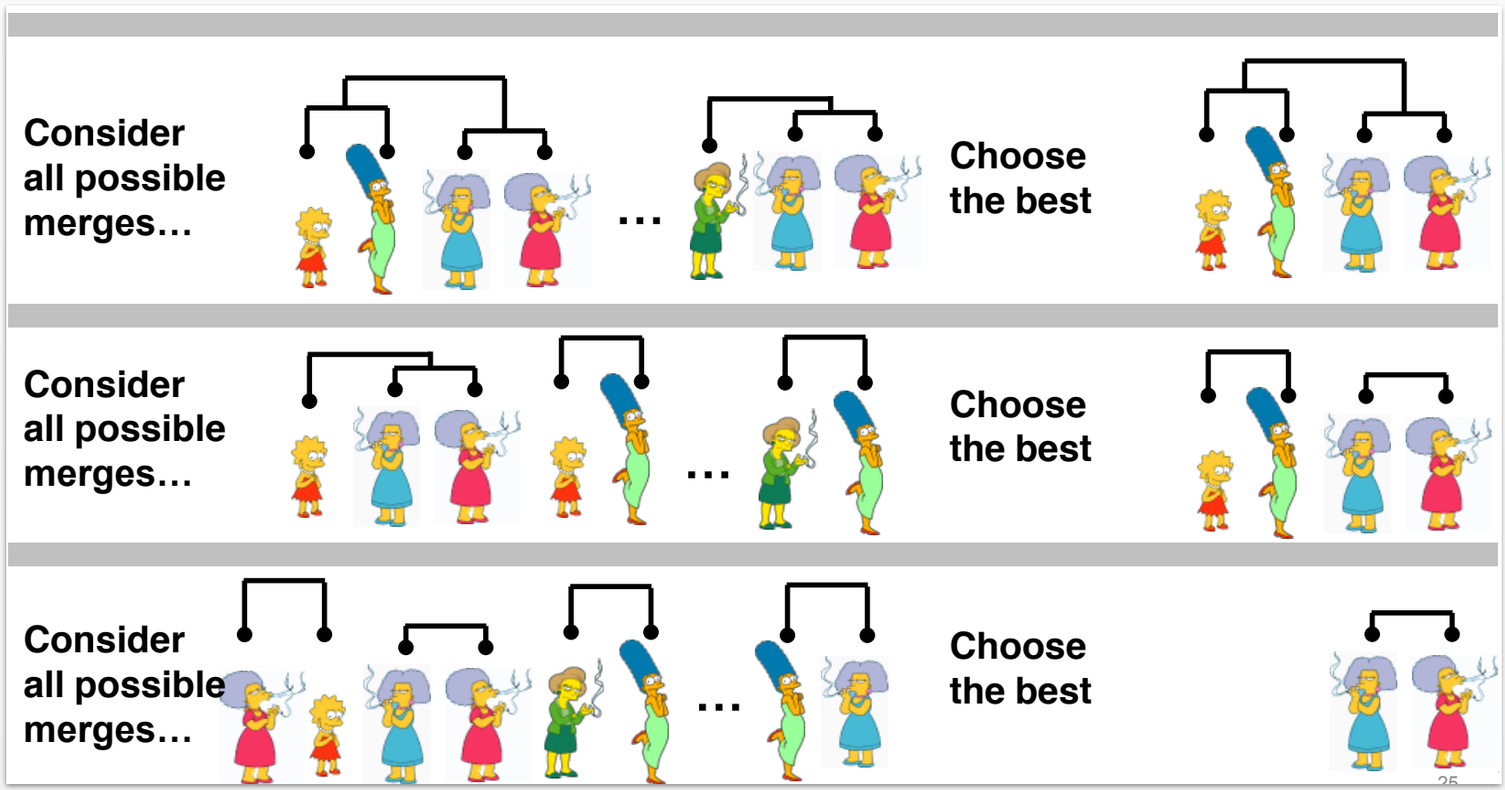
# Bottom-up (Agglomerative Clustering)



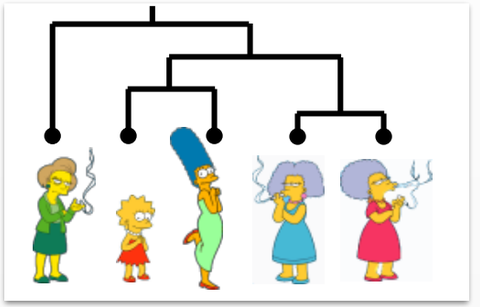
# Bottom-up (Agglomerative Clustering)

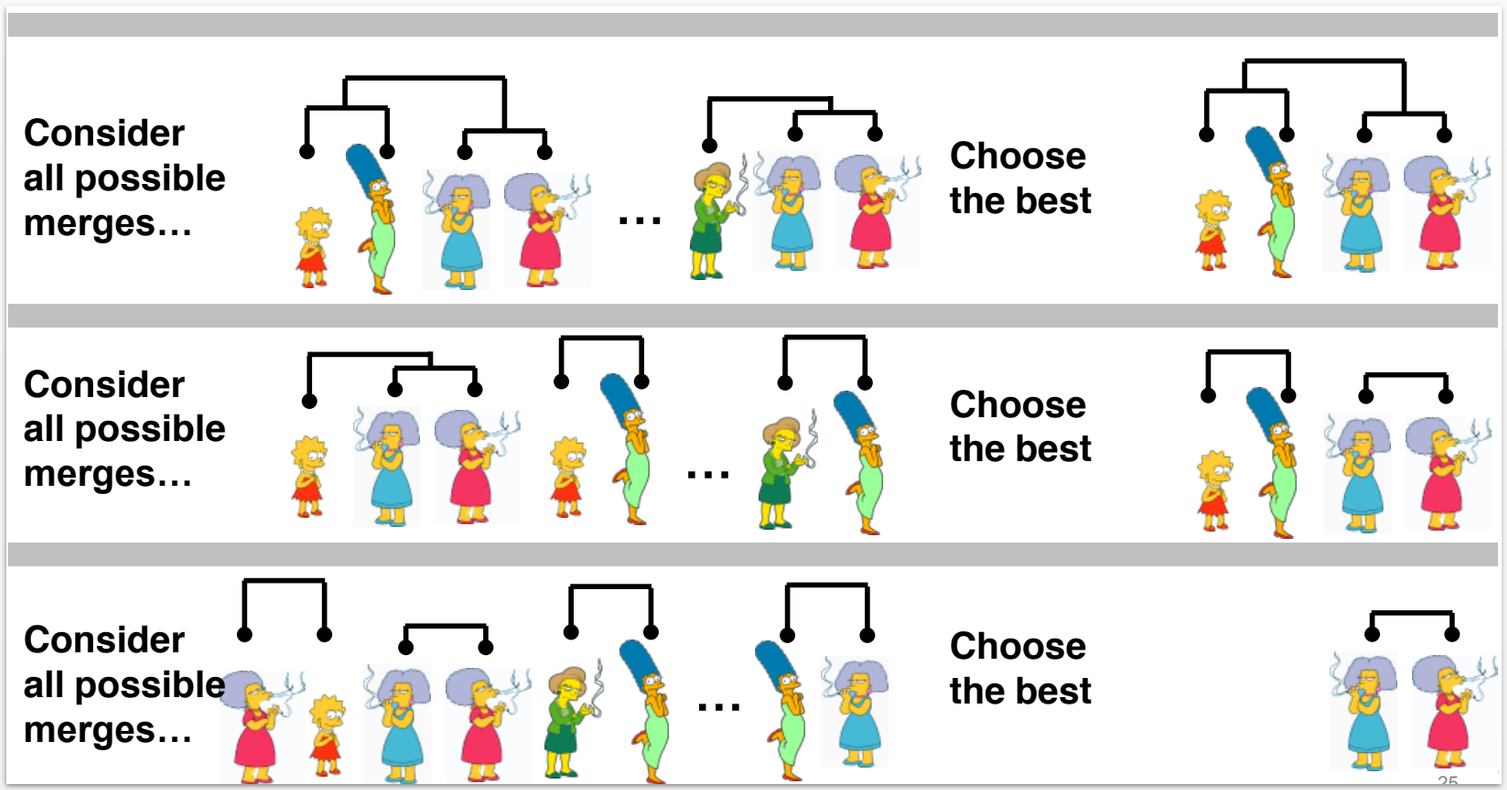


# Bottom-up (Agglomerative Clustering)



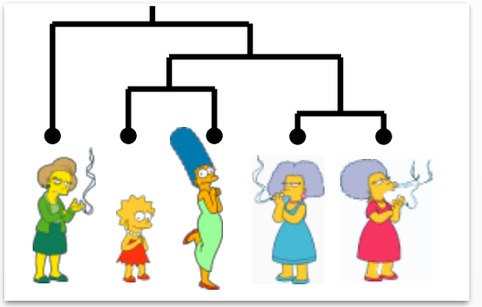
# Bottom-up (Agglomerative Clustering)

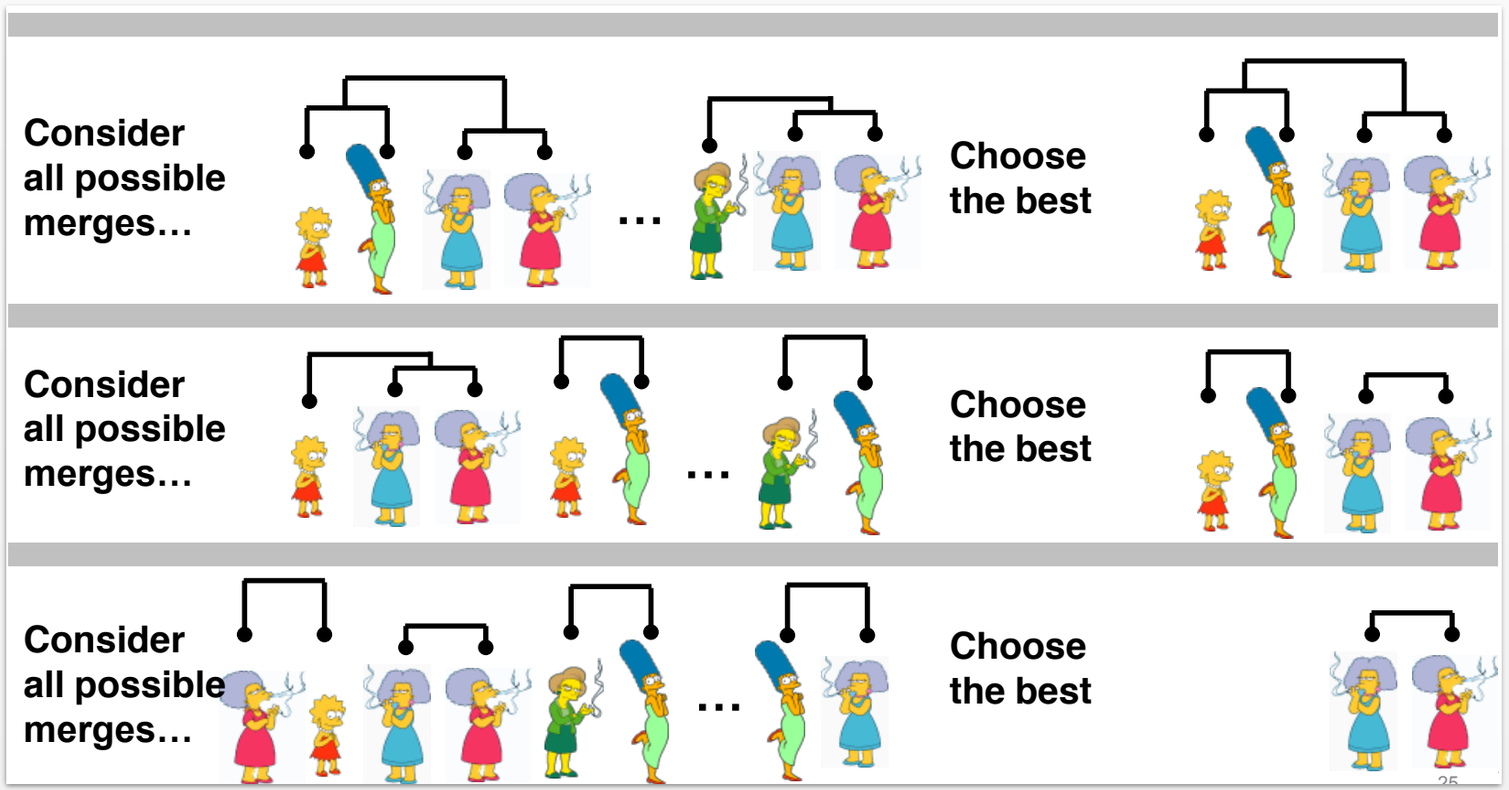




# Bottom-up (Agglomerative Clustering)

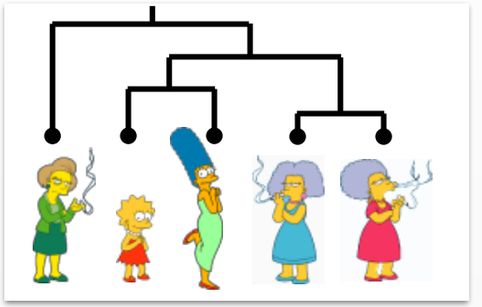
Can you now implement this?

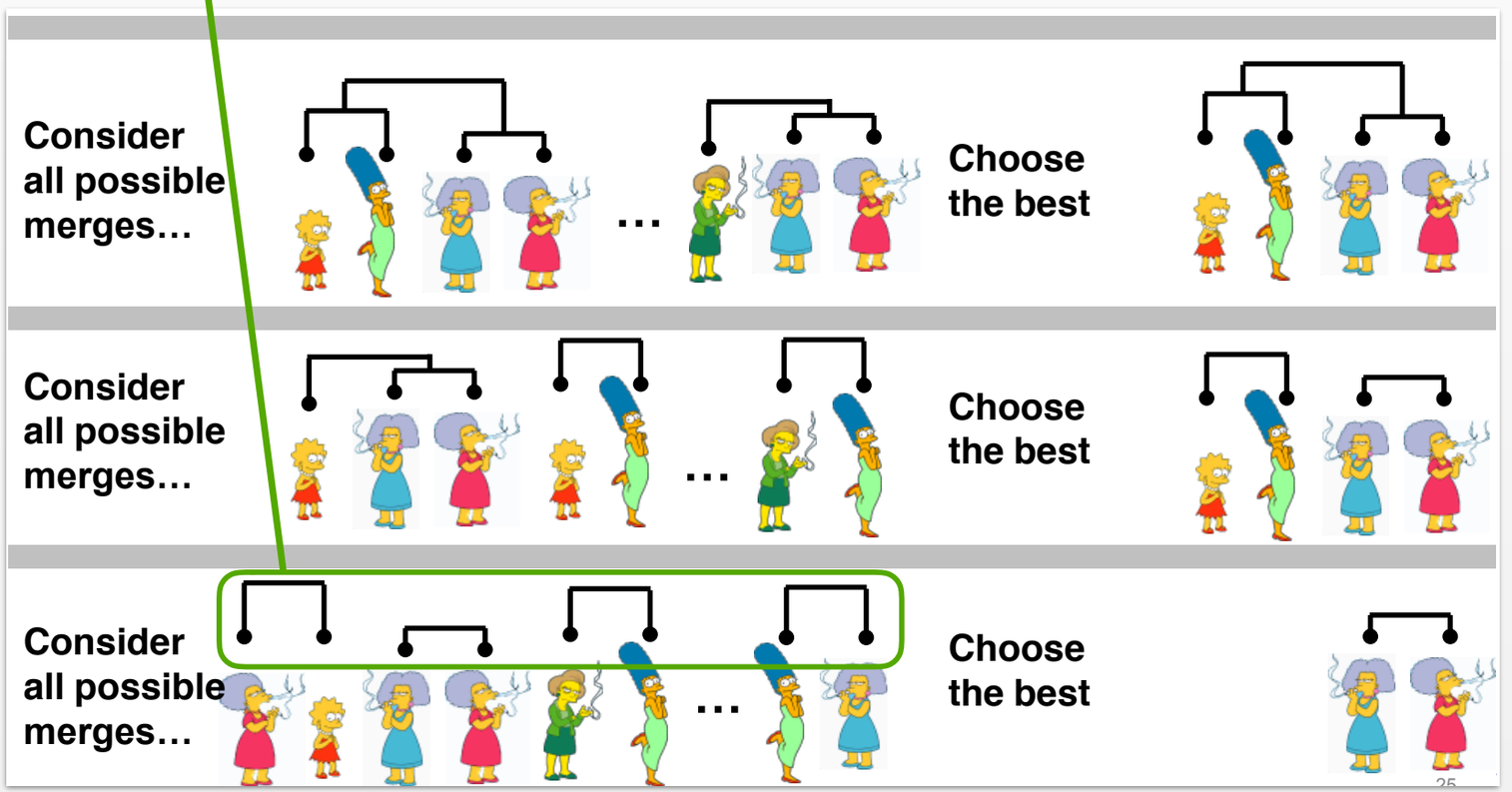




# Bottom-up (Agglomerative Clustering)

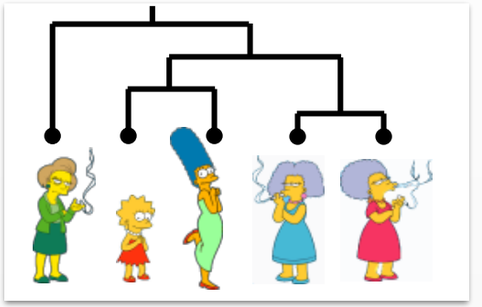
Distances between examples
(can calculate using metric)

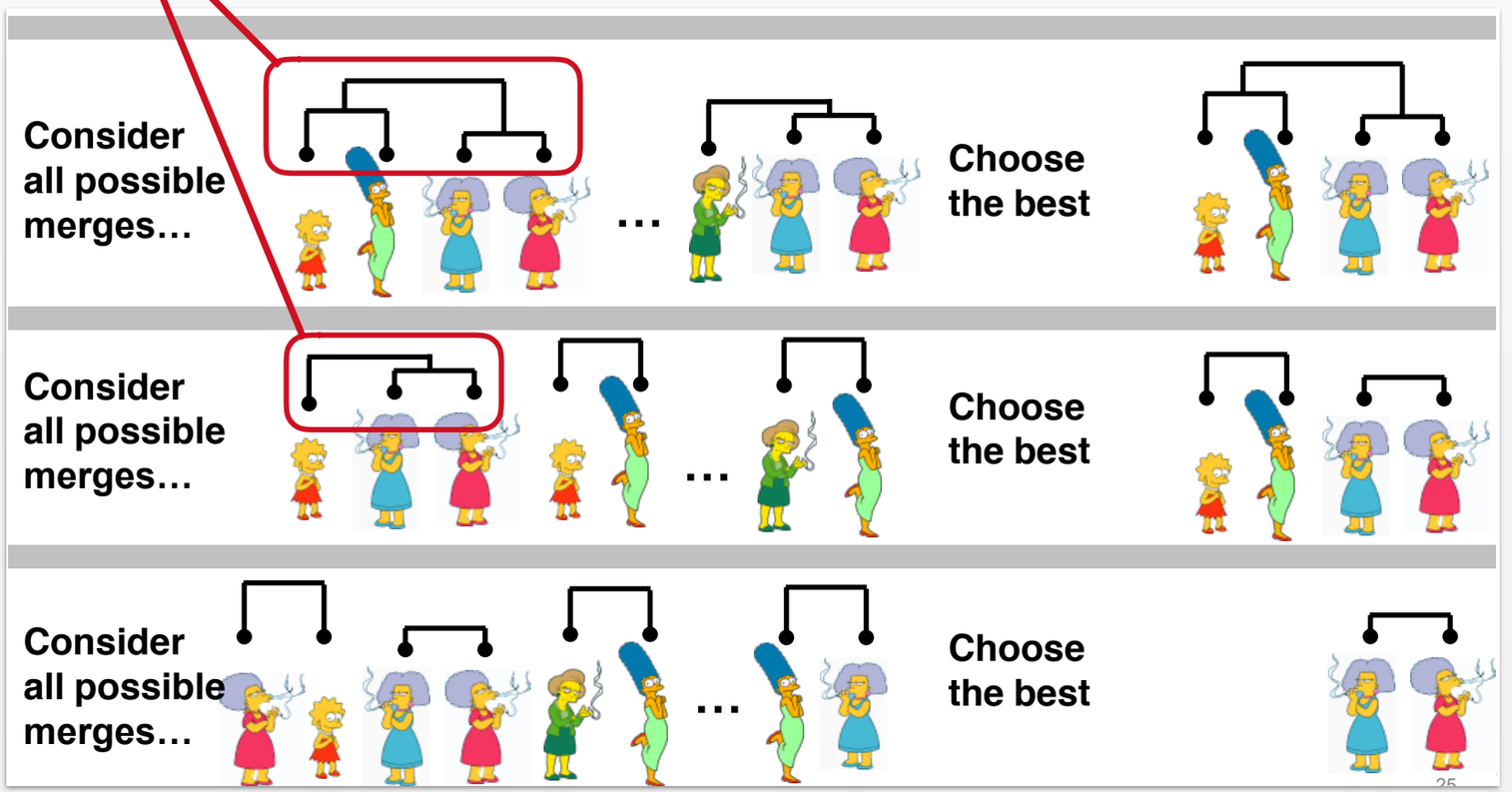




# Bottom-up (Agglomerative Clustering)

How do we calculate the distance to a cluster?

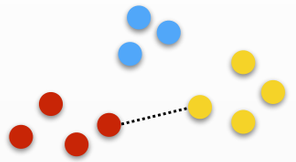




# Clustering Criteria

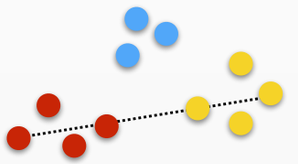
*Single link:*
(Closest point)

$$d(A,B) = \min_{a \in A, b \in B} d(a,b)$$



*Complete link:*
(Furthest point)

$$d(A,B) = \max_{a \in A, b \in B} d(a,b)$$

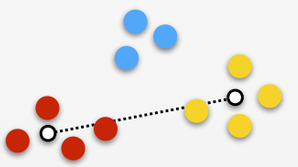


*Group average:*
(Average distance)

$$d(A,B) = \frac{1}{|A||B|} \sum_{a \in A, b \in B} d(a,b)$$

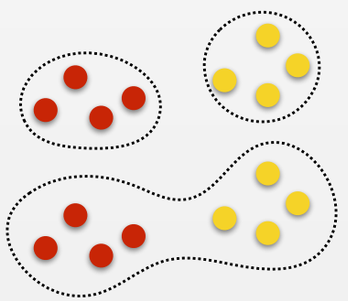
*Centroid:*
(Distance of average)

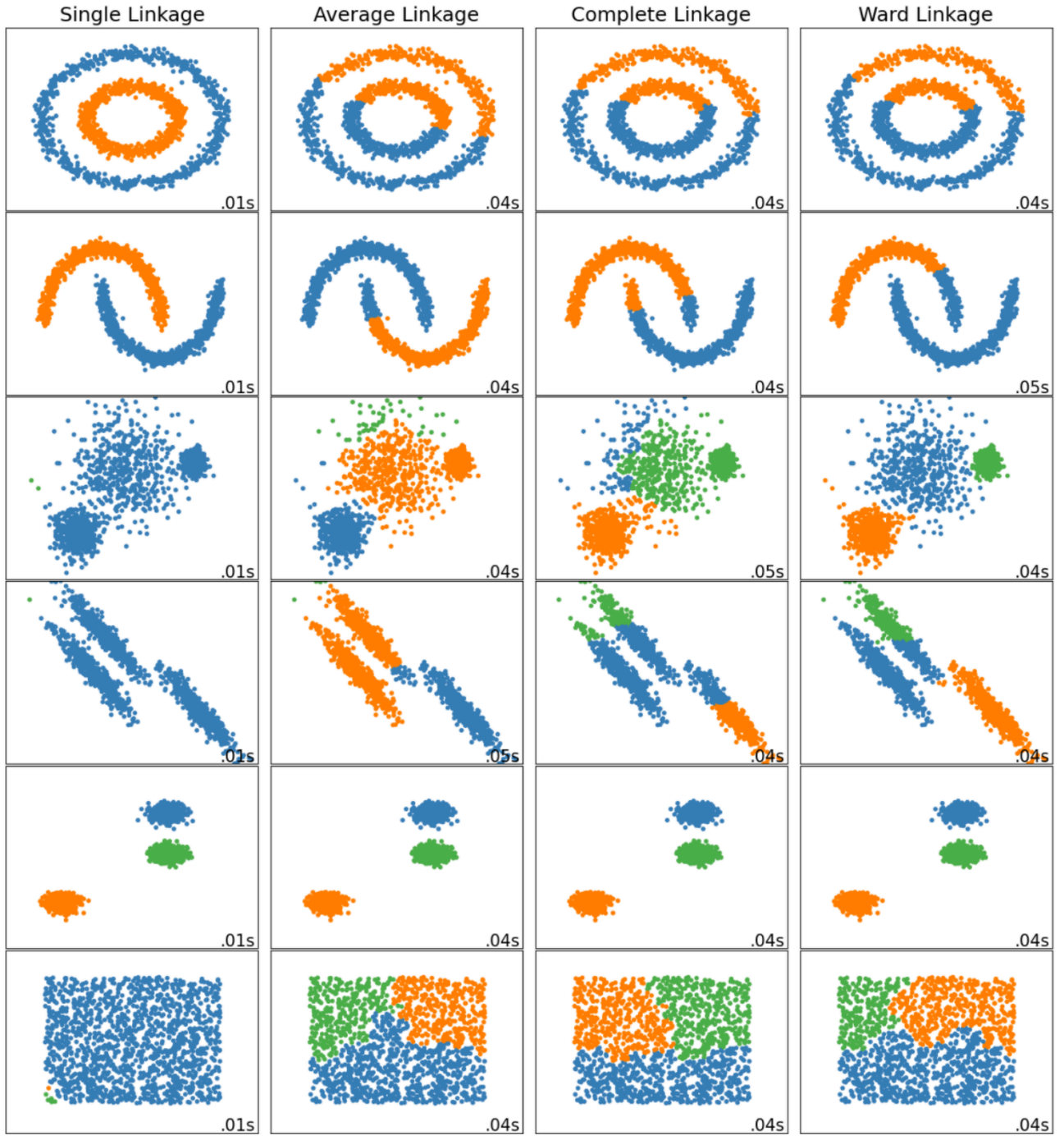
$$d(A,B) = d(\mu_A, \mu_B) \quad \mu_X = \frac{1}{|X|} \sum_{x \in X} x$$



*Ward*:
(Intra-cluster variance)

$$S_{A \cup B} = \sum_{x \in A \cup B} d(x, \mu_{A \cup B})^2$$




Single Linkage
Average Linkage
Complete Linkage
Ward Linkage
.01s
.04s
.04s
.04s
.01s
.04s
.04s
.05s
.01s
.04s
.05s
.04s
.01s
.05s
.04s
.04s
.01s
.04s
.04s
.04s
.01s
.04s
.04s
.04s

# Naive time complexity

**Algorithm 8.3** Basic agglomerative hierarchical clustering algorithm.

1: Compute the proximity matrix, if necessary. $O(N^2D)$
2: **repeat**
3: Merge the closest two clusters. $O((N-i+1)^2)$
4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters. $O((N-i-1)D)$
5: **until** Only one cluster remains.

Performed O(N) iterations

Naive approach
$O(N^2D + N^3)$

N: number of points
D: dimensionality

# True Time complexity

**Algorithm 8.3** Basic agglomerative hierarchical clustering algorithm.

1: Compute the proximity matrix, if necessary. $O(N^2D)$
2: **repeat**
3: Merge the closest two clusters. $O(1)$
4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters. $O((N-i)\log(N-i+1))$ and $O(N-i-1)$
5: **until** Only one cluster remains.

Performed O(N) iterations

Overall complexity

$O(N^2D + N^2 \log N)$

Trick 1

Use Min heap

- Allows accessing the minimum distance in $O(1)$
- Insertion of new distance and deletion of old ones into the heap in step 4 takes $O(\log(N-i+1))$ per distance

Trick 2

Recompute distances from merged cluster distances

- $d(AB, C) = \min(d(A, C), d(B, C))$ for single linkage, where $AB$ denotes the merging of clusters $A$ and $B$

# Lance-Williams Methods

| **Clustering Method** | $\alpha_{\mathbf{A}}$ | $\alpha_{\mathbf{B}}$ | $\beta$ | $\gamma$ |
|---|---|---|---|---|
| Single Link | $1/2$ | $1/2$ | $0$ | $-1/2$ |
| Complete Link | $1/2$ | $1/2$ | $0$ | $1/2$ |
| Group Average | $\frac{m_A}{m_A+m_B}$ | $\frac{m_B}{m_A+m_B}$ | $0$ | $0$ |
| Centroid | $\frac{m_A}{m_A+m_B}$ | $\frac{m_B}{m_A+m_B}$ | $\frac{-m_A m_B}{(m_A+m_B)^2}$ | $0$ |
| Ward's | $\frac{m_A+m_Q}{m_A+m_B+m_Q}$ | $\frac{m_B+m_Q}{m_A+m_B+m_Q}$ | $\frac{-m_Q}{m_A+m_B+m_Q}$ | $0$ |

d(x,y) = |x-y|

d(x,y) = $|x-y|^2$

Recursively minimize/maximize proximity for a merger **R:=A∪B** relative to all existing **Q**

$$
\begin{aligned}
p(R,Q) = & \alpha_A p(A,Q) \\
& + \alpha_B p(B,Q) \\
& + \beta\, p(A,B) \\
& + \gamma |p(A,Q) - p(B,Q)|
\end{aligned}
$$

# Hierarchical Clustering Summary

+ Hierarchical structure maps nicely onto human intuition in some domains
+ No difficulty in choosing initial points
- *Heuristic method: No global objective criteria to optimize. Optimizes local objective at each merge.*
- Merging decisions are final: Prevents local optimization to from becoming global optimization. For e.g., Ward methods optimized local SSE doesn't translate to the optimized global SSE.
- *Scaling:* Time complexity at least $O(N^2D + N^2 \log N)$, Space complexity: $O(N^2)$
- Susceptibility to noise
- *Interpretation* of results is (very) subjective

Can be improve by initializing wit several small k- means clusters

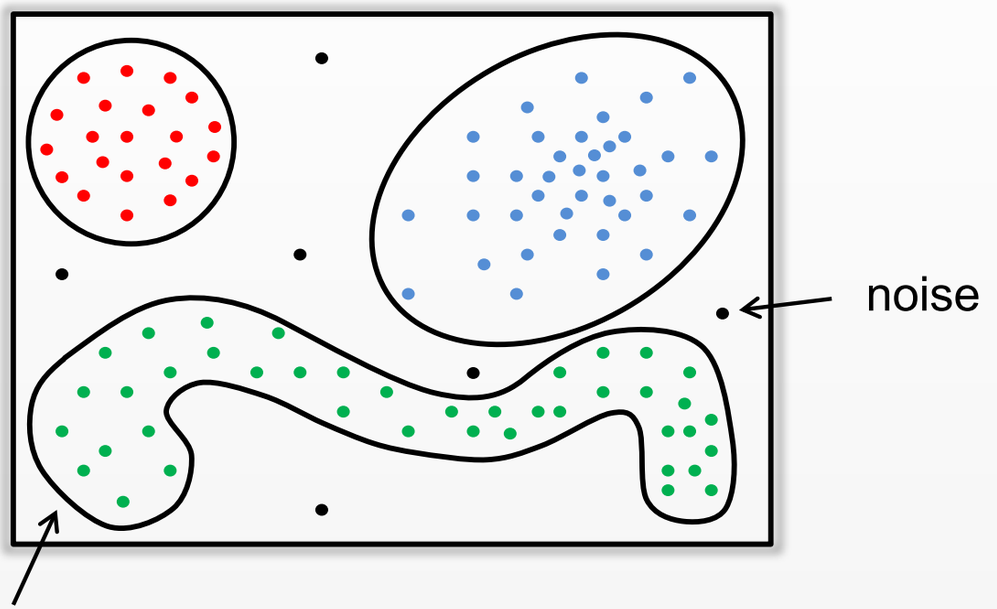

# Clustering

Shantanu Jain


# DBScan

## Density-based Clustering

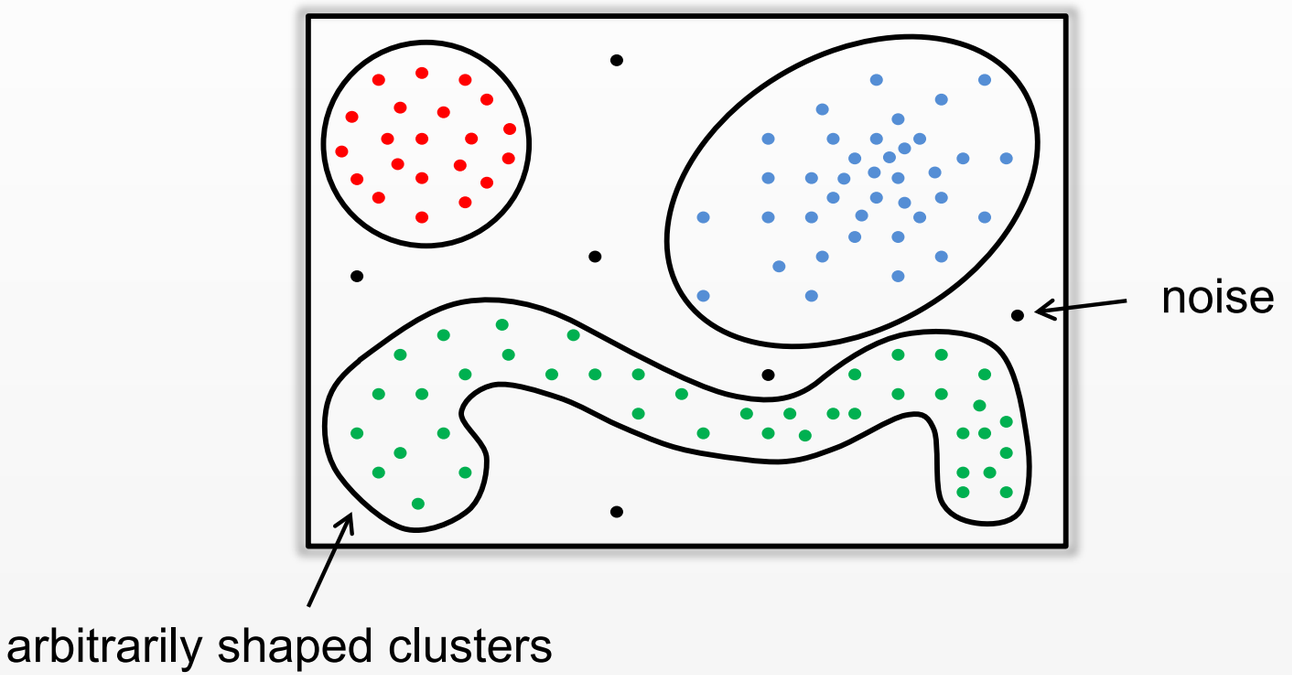
# DBSCAN




[PDF] A density-based algorithm for discovering clusters in large spatial databases with noise.
M Ester, HP Kriegel, J Sander, X Xu - Kdd, 1996 - aaai.org
Abstract Clustering algorithms are attractive for the task of class identification in spatial databases. However, the application to large spatial databases rises the following requirements for clustering algorithms: minimal requirements of domain knowledge to ...
Cited by 8901 Related articles All 70 versions Cite Save More

(one of the most-cited clustering methods)

# DBSCAN



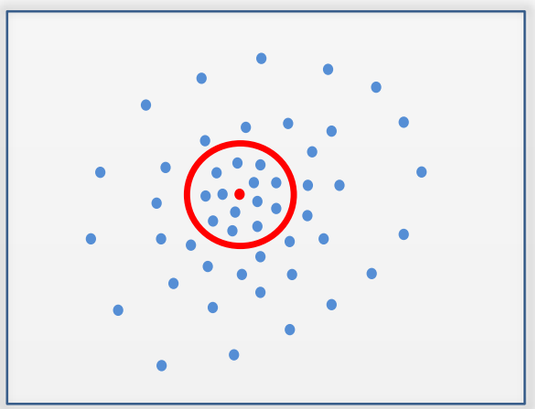

*Intuition*

- A *cluster* is a islands of *high* density
- *Noise* points lie in a sea of *low* density

# Defining “High Density”

**Naïve approach**

For each point in a cluster there are at least a minimum number (MinPts) of points in an Eps-neighborhood of that point.

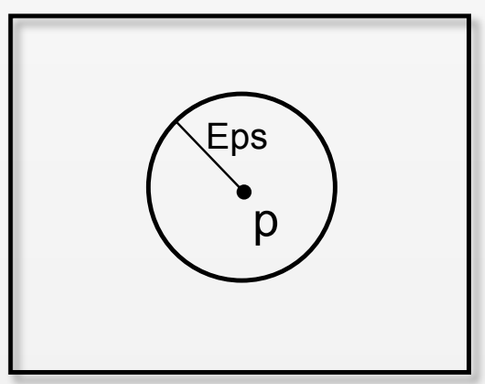


**cluster**

# Defining “High Density”

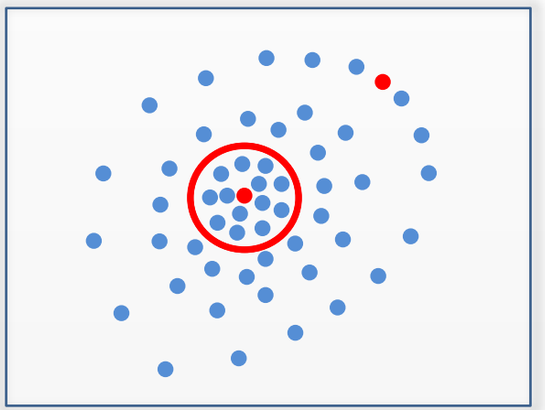
**Eps-neighborhood of a point p**

$$N_{Eps}(p) = \{ q \in D \mid dist(p, q) \leq Eps \}$$



# Defining “High Density”

- In each cluster there are two kinds of points:
  - points inside the cluster (core points)
  - points on the border (border points)



**cluster**

An Eps-neighborhood of a border point contains significantly less points than an Eps-neighborhood of a core point.

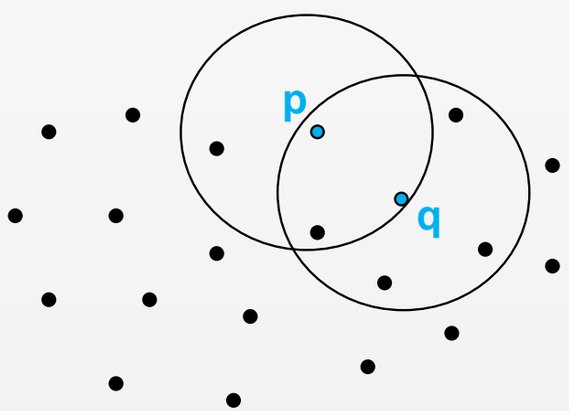
# Density Reachability

**Definition**

A point p is directly density-reachable from a point q with regard to the parameters Eps and MinPts, if

1) $p \in N_{Eps}(q)$ (reachability)

2) $| N_{Eps}(q) | \geq MinPts$ (core point condition)




**Parameter: MinPts = 5**

p directly density reachable from q

$p \in N_{Eps}(q)$

$| N_{Eps}(q) | = 6 \geq 5 = MinPts$ (core point condition)

q **not** directly density reachable from p

$| N_{Eps}(p) | = 4 < 5 = MinPts$ (core point condition)

*Note:* This is an asymmetric relationship

# Density Reachability

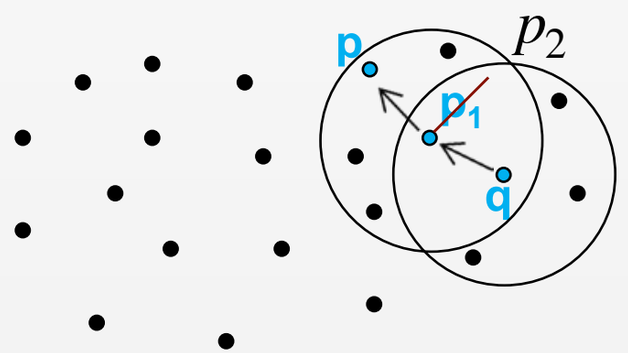
**Definition**

A point p is density-reachable from a point q

with regard to the parameters Eps and MinPts

if there is a chain of points $p_1, p_2, \ldots, p_s$ with $p_1 = q$ and $p_s = p$

such that $p_{i+1}$ is directly density-reachable from $p_i$ for all $1 < i < s-1$.




MinPts = 5

$| N_{Eps}(q) | = 5 = MinPts$ (core point condition)

$| N_{Eps}(p_1) | = 6 \geq 5 = MinPts$ (core point condition)

$p_2$

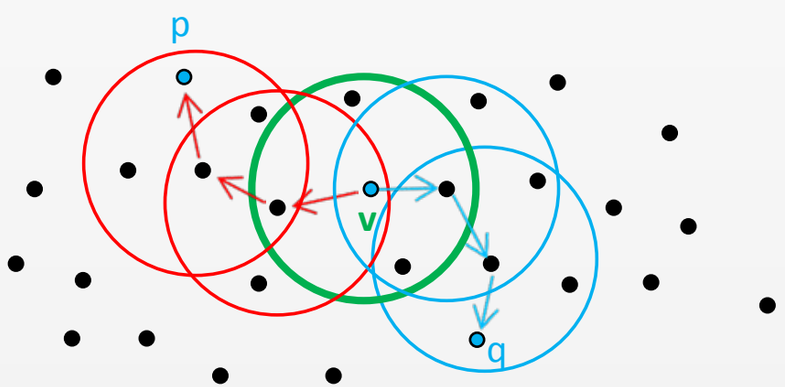
# Density Connectivity

**Definition (density-connected)**

A point p is density-connected to a point q

with regard to the parameters Eps and MinPts

if there is a point v such that both p and q are density-reachable from v.




MinPts = 5

*Note:* This is a symmetric relationship

# Definition of a Cluster

A cluster with regard to the parameters Eps and MinPts is a non-empty subset C of the database D with

1) For all p, q ∈ D: (Maximality)
   If p ∈ C and q is density-reachable from p with regard to the parameters Eps and MinPts, then q ∈ C.

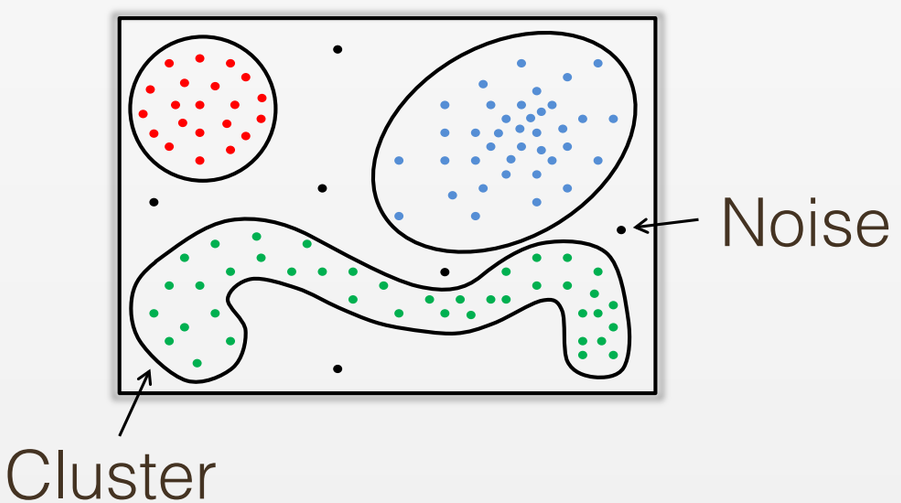
2) For all p, q ∈ C: (Connectivity)
   The point p is density-connected to q with regard to the parameters Eps and MinPts.

# Definition of Noise

Let $C_1,...,C_k$ be the clusters of the database D
with regard to the parameters $Eps_i$ and $MinPts_i$ (i=1,...,k).

The set of points in the database D not belonging to any cluster $C_1,...,C_k$
is called noise:

$$\text{Noise} = \{ p \in D \mid p \notin C_i \text{ for all } i = 1,...,k\}$$



# DBSCAN Algorithm

(1) Start with an arbitrary point p from the database and
retrieve all points density-reachable from p
with regard to Eps and MinPts.

The set of points reached from p may include points previously labeled as noise, but are in reality border points

(2) If p is a core point, the procedure yields a cluster
with regard to Eps and MinPts
and all points in the cluster are classified.

Low density point, label it as noise

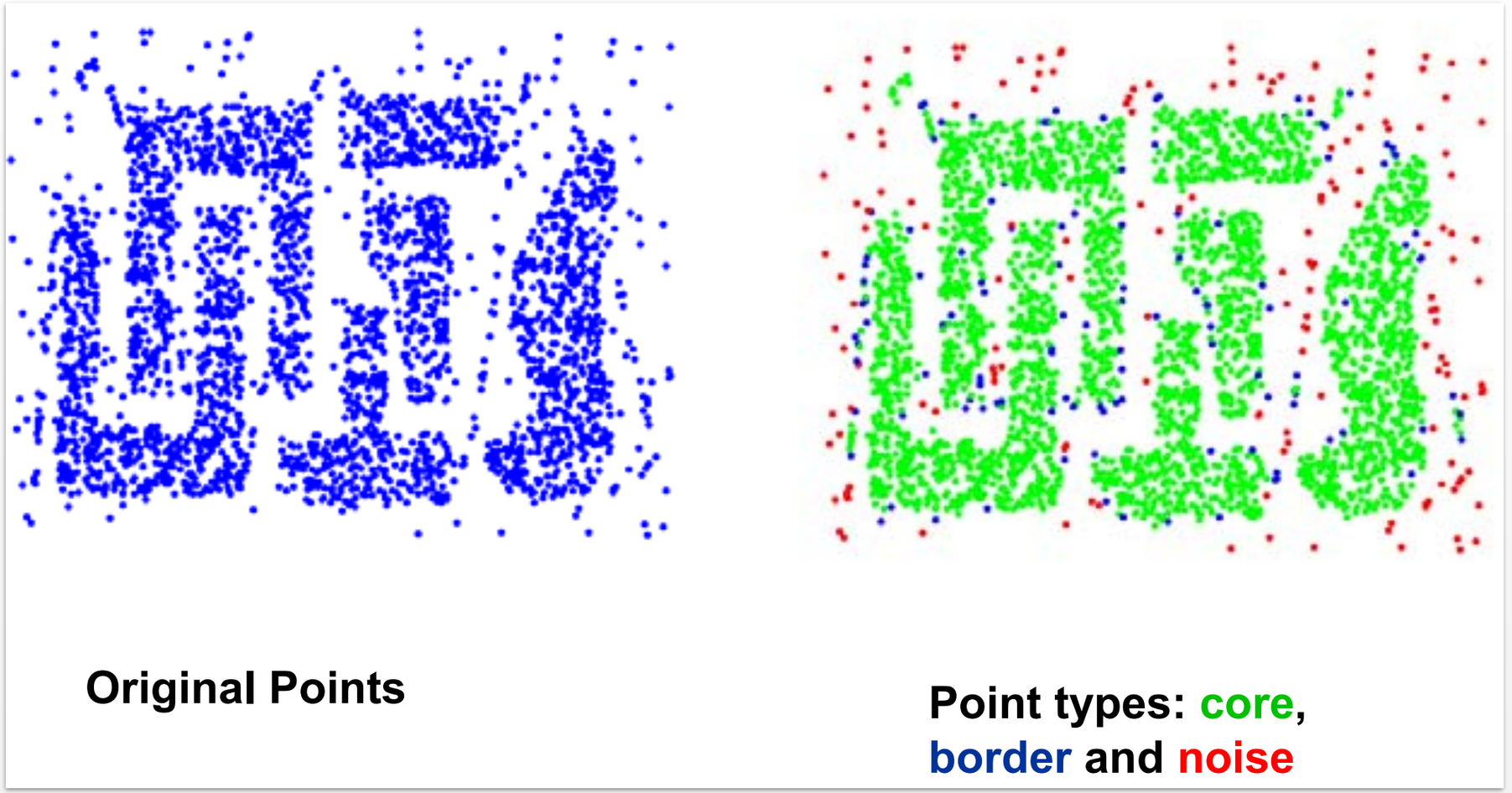
(3) If p is a ~~border point~~, no points are density-reachable from p
and DBSCAN visits the next unclassified point in the database.

and go to step 1

# DBSCAN Complexity

- *Time complexity:* $O(N^2D)$ if done naively, $O(DN \log N)$ when using a spatial index such as K-D tree.
  (*works in relatively low dimensions*)
- *Space complexity:* $O(ND)$

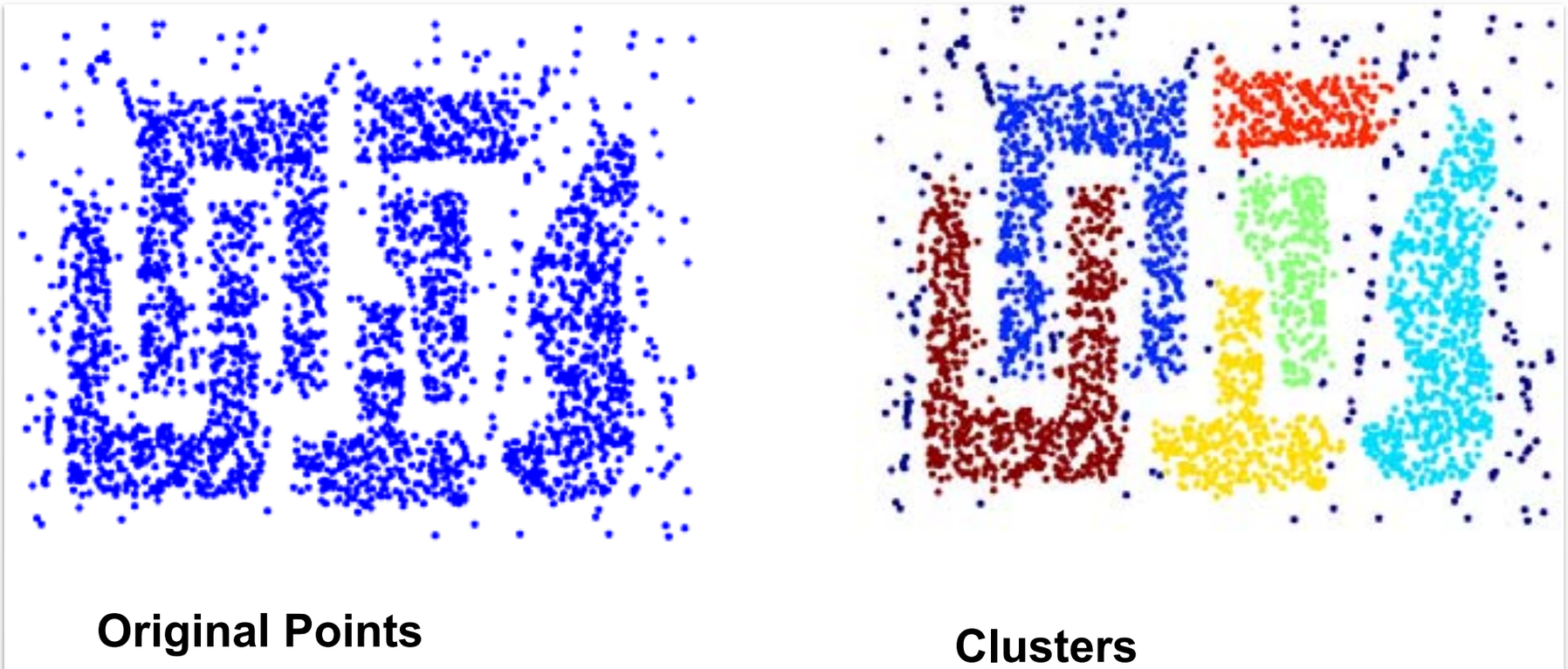
# DBSCAN Algorithm



**Original Points**

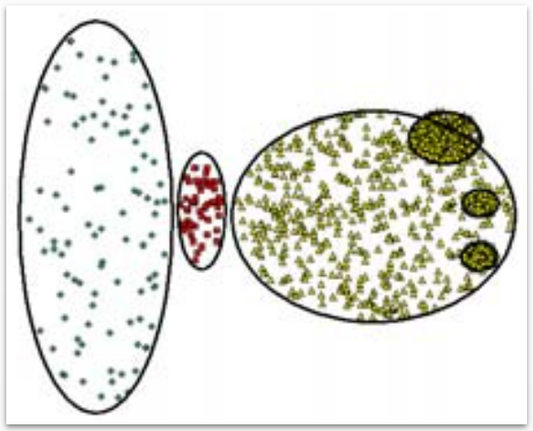
**Point types: core, border and noise**

# DBSCAN strengths

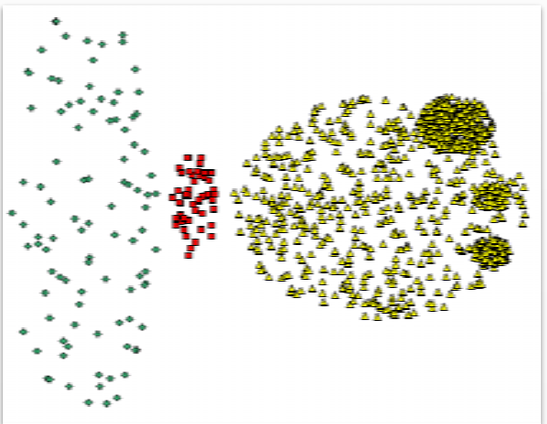



+ Resistant to noise
+ Can handle arbitrary shapes

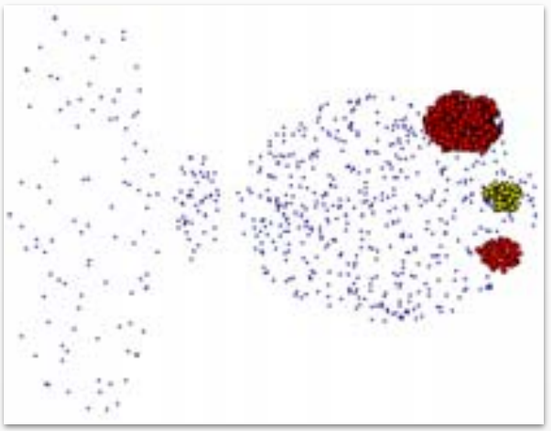
# DBSCAN Weaknesses



Ground Truth



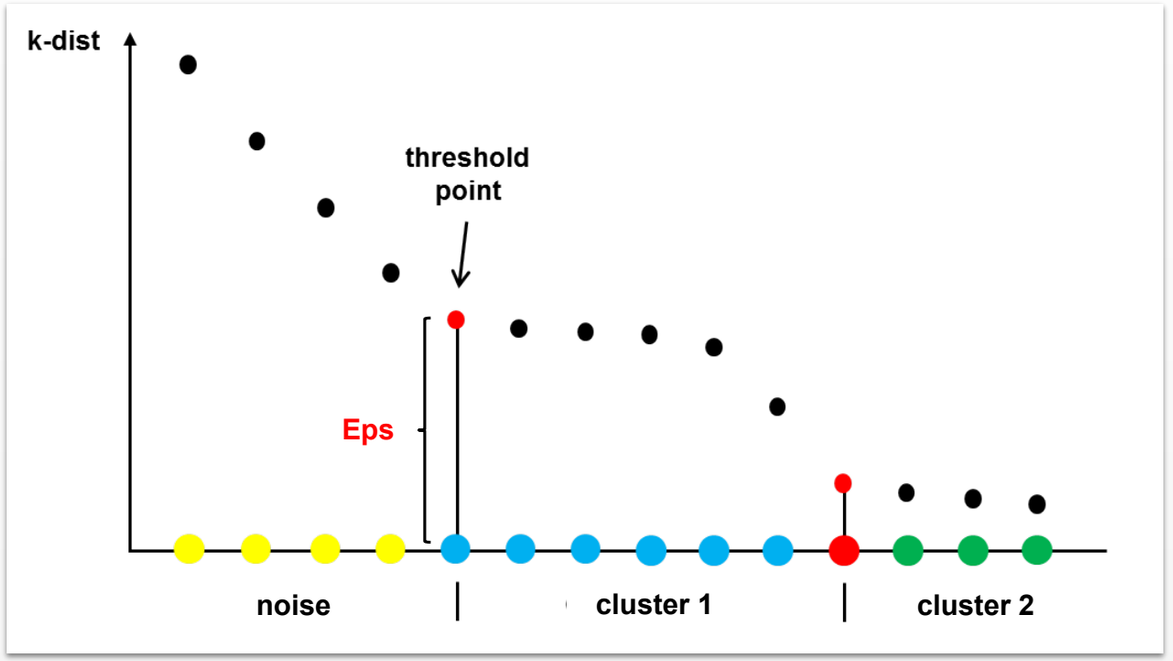
MinPts = 4, Eps=9.92



MinPts = 4, Eps=9.75

- Varying densities
- High dimensional data
- Cannot give overlapping clusters

# Determining EPS and MINPTS




- Calculate distance of ***k*-th** nearest neighbor for each point
- Plot in ascending / descending order
- Set **EPS** to max distance before “jump”
- Set Minpts to k.

# K-means vs DBSCAN