Week 3: Distributed Computing

Tien-Nam Le

`tien-nam.le@ens-lyon.fr`
`perso.ens-lyon.fr/tien-nam.le/su`

Sciences U Lyon

Outline

- Memory Latency
- Clustering Data
- Distributed Computing with Spark

Part 1: Memory Latency

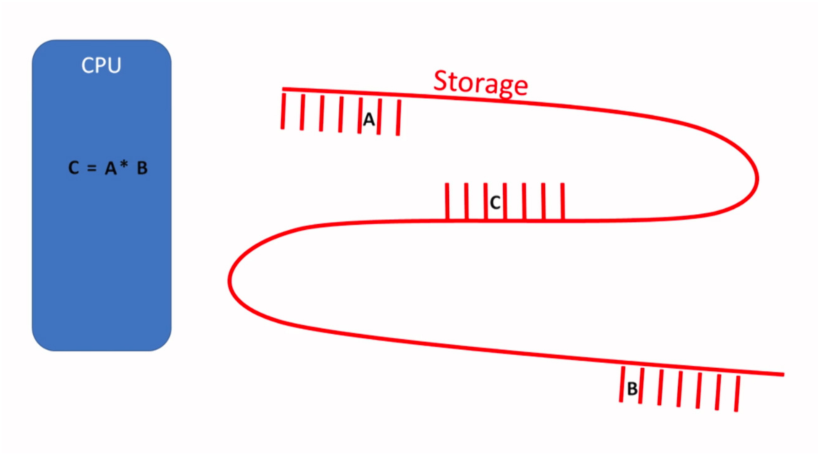
Big Data problem

- The amount of collected data is increasing.
- Before, this can be handled by just increasing computer memory.
- Now we use clusters with many computers for parallel data analysis.
- **Question:** What makes computation on big data slow? Why do we need multiple computers?

CPU computation

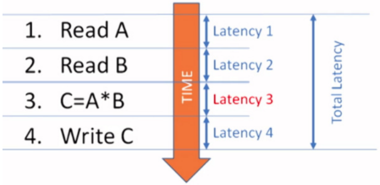
A computer has two main parts:

- CPU
- Storage



Memory Latency

Latencies

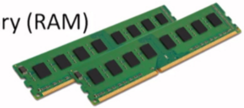


1. Read A
2. Read B
3. C=A*B
4. Write C

With big data, most of the latency is memory latency (1,2,4), not computation (3)

Storage Types

- Main Memory (RAM)

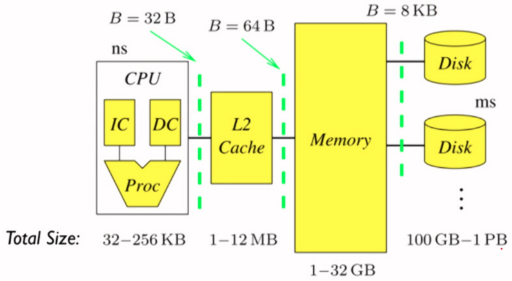


- Spinning disk

- Remote computer

The Memory Hierarchy

B=Block size



Sizes and latencies in a typical memory hierarchy.

	CPU (Registers)	L1 Cache	L2 Cache	L3 Cache	Main Memory	Disk Storage	Local Area Network
Size (bytes)	1KB	64KB	256KB	4MB	4-16GB	4-16TB	16TB – 10PB
Latency	300ps	1ns	5ns	20ns	100ns	2-10ms	2-10ms
Block size	64B	64B	64B	64B	32KB	64KB	1.5-64KB

Questions

Which of the following steps has the lowest step latency?

- Reading from memory
- Performing a CPU operation
- Writing to memory

True or false: Main memory has a higher latency than a mechanical hard drive.

- True
- False

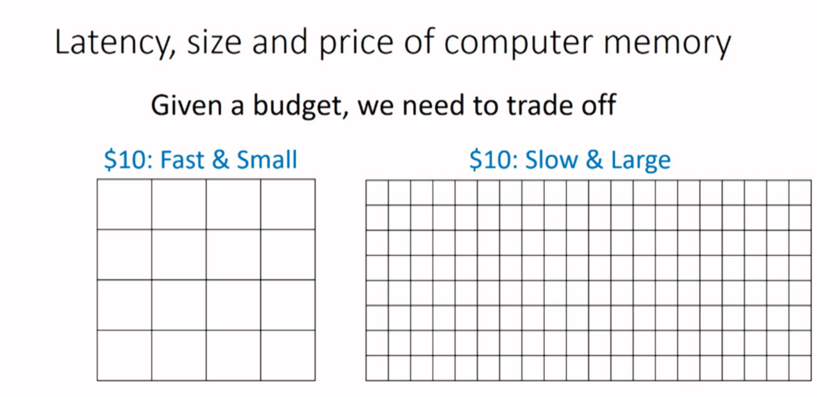
Where is the result stored immediately following a multiplication operation?

- L2 cache
- Main memory
- Registers

Summary

- The major source of latency in data analysis is reading and writing to storage
- Different types of storage offer different latency, capacity and price
- Objective: organizing storage + computation to
    - maximizing speed
    - minimizing cost.

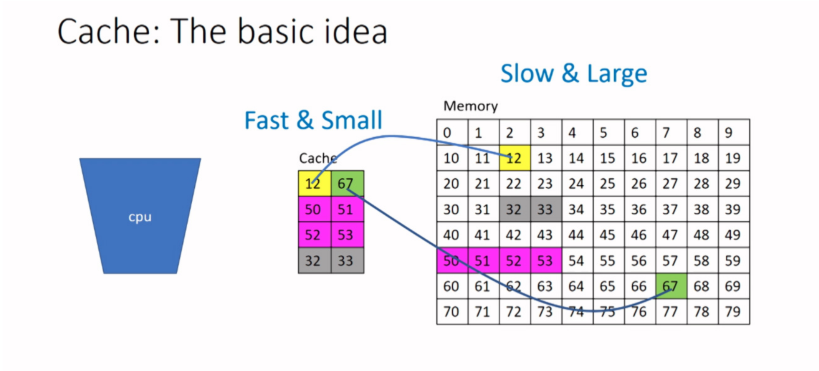
Trade off


Latency, size and price of computer memory

Given a budget, we need to trade off

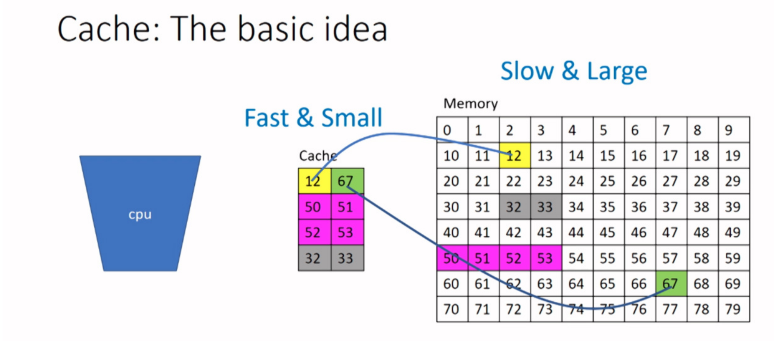
- For a given amount of money: you can buy either, memory that is fast and small, or memory that is slow and large.
- **Caching:** combining fast and slow memory, to create storage both fast and large.

Cache



- Cache hit: the data for computation is in cache
- Cache miss: the data for computation is not in cache
- How to have more cache hit?

Types of locality



- Frequently accessed data should stay in the cache. $\rightarrow$ **Temporal locality**
- Memory is partitioned into blocks (contains several bytes). Data are moved from memory to cache block by block.
- Data elements should be stored next to each other in the memory. $\rightarrow$ **Spacial locality**

Simple examples

- **Temporal locality:**
  - Want to compute vector multiplication $w \cdot x_i$ for $x_1, ..., x_N$
  - If size of w fits into cache $\rightarrow$ fast computation.
  - If size of w is larger than size of cache $\rightarrow$ much slower.
- **Spacial locality:**
  - Want to compute $\sum (x_i - x_{i+1})^2$ for $x_1, ..., x_N$
  - x_i stored in memory closed to each other $\rightarrow$ fast computation.
  - x_i stored randomly $\rightarrow$ much slower.

Question

Given the following code.

```
A = [0]*10000
sum = 0
for i in range(10000):
  sum += A[i] + i
```

Determine which type of locality is present:

- Temporal locality
- Spacial locality
- Both
- None

Question

Suppose you have a 16 KiB cache, with a 64 byte block size. You try to access a 4 byte integer and receive a cache miss. How much data will be copied to the cache from main memory?

- Only one byte of data can be moved from main memory to the cache at any one time.
- The amount of data copied to the cache will be the size of the variable being accessed. In this case, 4 bytes.
- The amount of data copied will be equal to the block size of the cache. In this case, 64 bytes.
- The amount of data copied to the cache will be equal to the size of the cache itself. In this case, 16 KiB.
- Data isn't necessarily copied to the cache upon each cache miss.

Question

How does sorting data improve access latency?

- Temporal locality is improved because data will have been accessed and copied to the cache during sorting.
- Temporal locality is improved because fewer elements will be replaced in the cache, allowing the existing elements to be accessed a greater number of times.
- Spacial locality is improved because all of the data is placed in adjacent locations in the cache, meaning we don't have to access main memory for any of the values after sorting.
- Spacial locality is improved because values are placed in consecutive memory locations during sorting and multiple values which will be accessed consecutively will be copied to the cache.

Memory locality: row vs column

- See Jupyter notebook

Conclusions

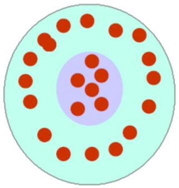
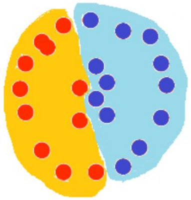
Impact of memory latency on Big Data:

- Clock rate is stuck at around 3GHz, and is likely to be stuck there for the foreseeable future.
- Faster computers / disks / networks are expensive
- Focus on data access: The main bottleneck on big data computation is moving data around, NOT calculation.
- **Solution:** a cluster of many cheap computers, each with many cores and divide the data so that each computer has a small part of the data.
- **Question**: How to divide data into parts?

Part 2: Clustering Data

What is clustering data?

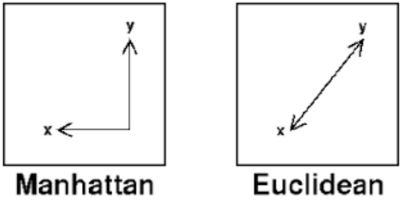
- **Clustering Data:** Dividing data into similarity groups called *clusters*.
- A **cluster** is a collection of data elements such that
  - elements in the same cluster are “similar”
  - elements in different clusters are “dissimilar”.

What we need for clustering

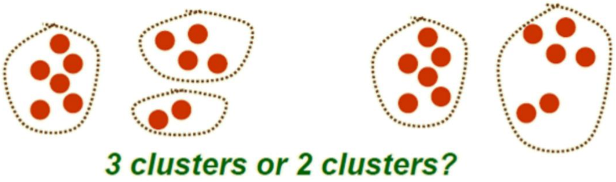
Q1: Need to define ”similarity”

- Numerical:
  - Euclidean distance
  - Manhattan distance (easier to compute)
- Categorical: Hamming distance



What we need for clustering

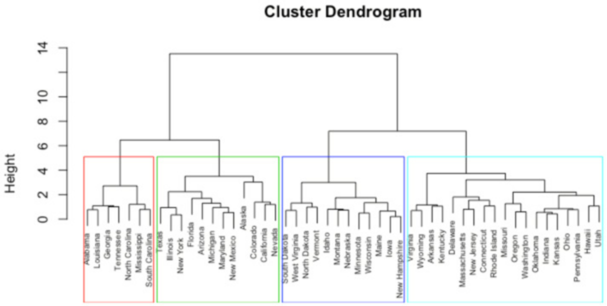
Q2: How many clusters?



- Fix the number of clusters beforehand.
- Try and find the best number of clusters

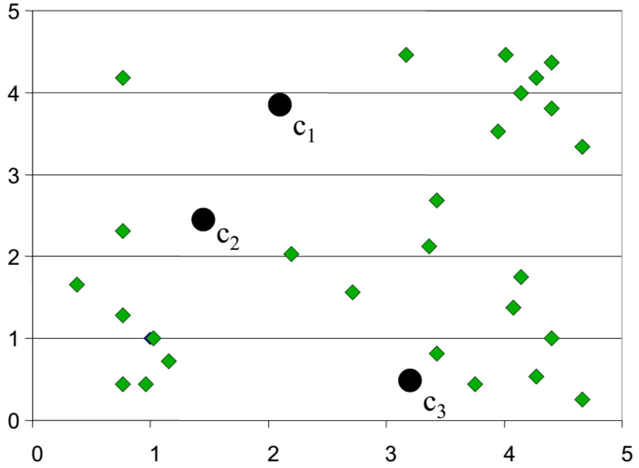
Types of clustering techniques

- **"Flat" techniques:** Determine all clusters at once.
  - Most popular technique: k-mean
- **Hierarchical techniques**: repeatedly find clusters based on previous clusters



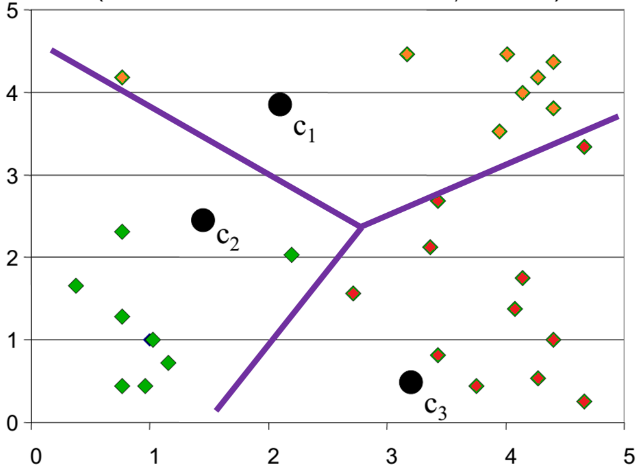
K-means clustering example: step 1

Randomly initialize the cluster centers (synaptic weights)



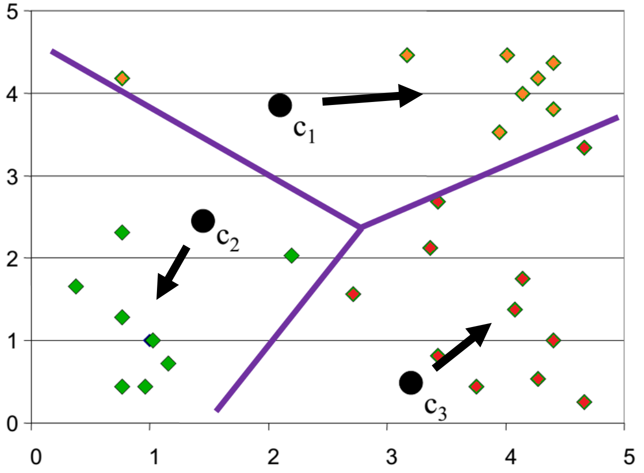
K-means clustering example – step 2

Determine cluster membership for each input
("winner-takes-all" inhibitory circuit)

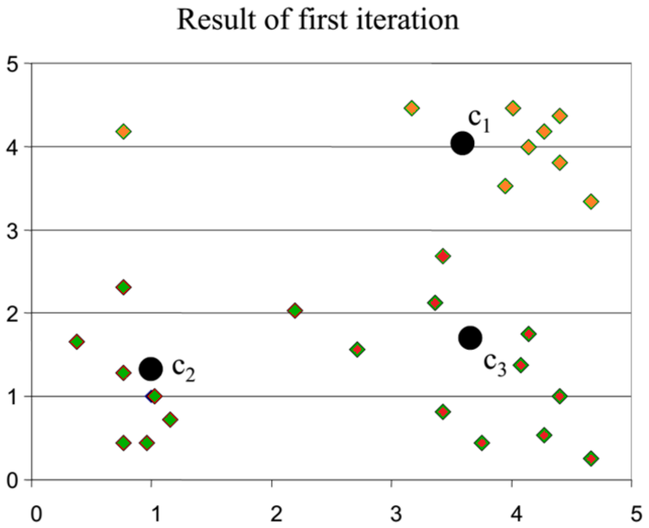


K-means clustering example – step 3

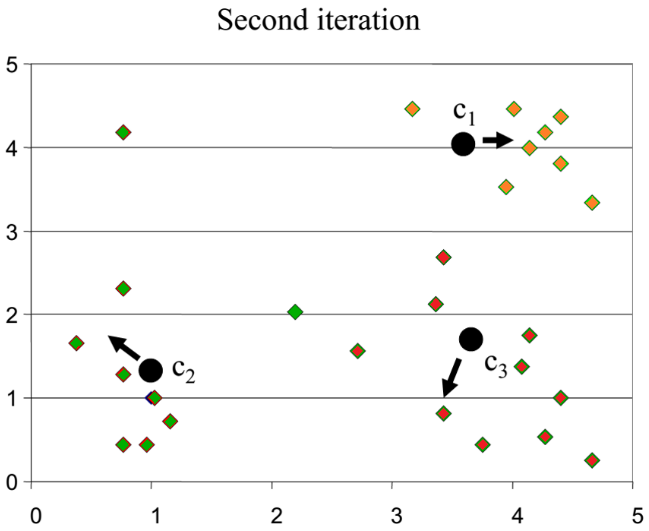
Re-estimate cluster centers (adapt synaptic weights)



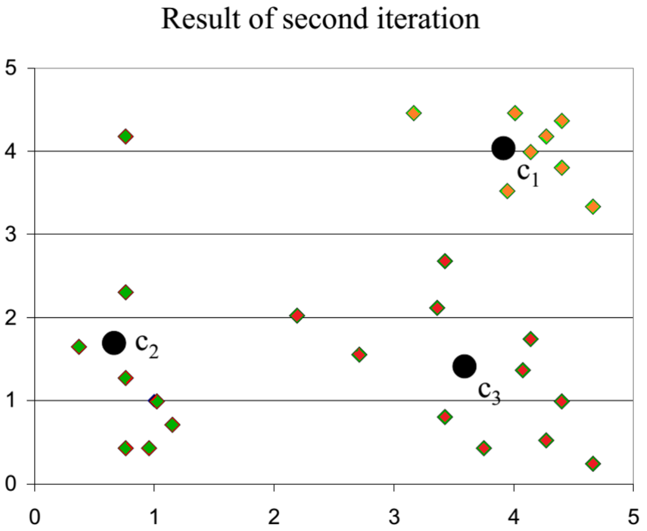
K-means clustering example



K-means clustering example



K-means clustering example

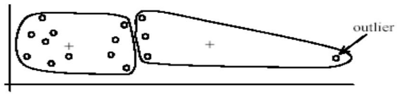


Strength and weakness of k-means

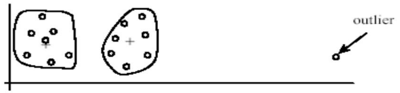
Strength: Simple and efficient

Weakness:

- The *mean* need to be defined. (For categorical data, use *mode* instead)
- Need to specify the number *k* of clusters
- Sensitive to outliers.



(A): Undesirable clusters



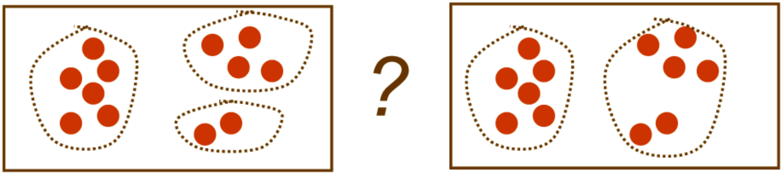
(B): Ideal clusters

Exercises with k-means

- Jupyter notebooks

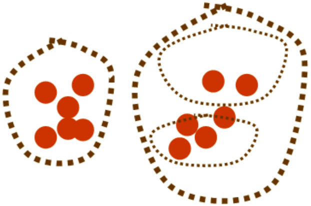
Hierarchical clustering

- Up to now, considered "flat" clustering

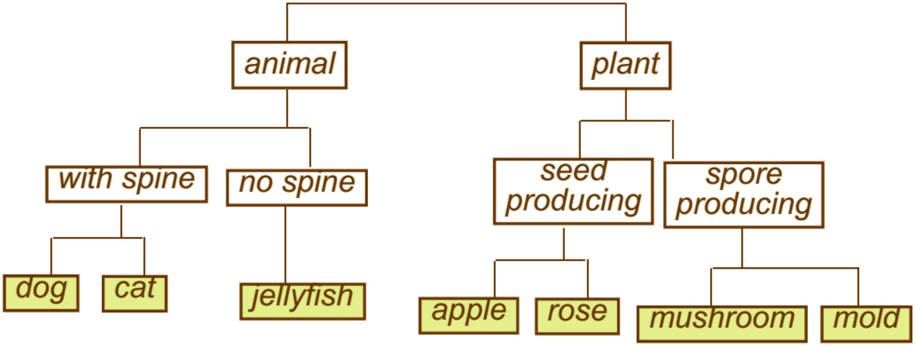


- For some data, hierarchical clustering is more appropriate than "flat" clustering

- Hierarchical clustering

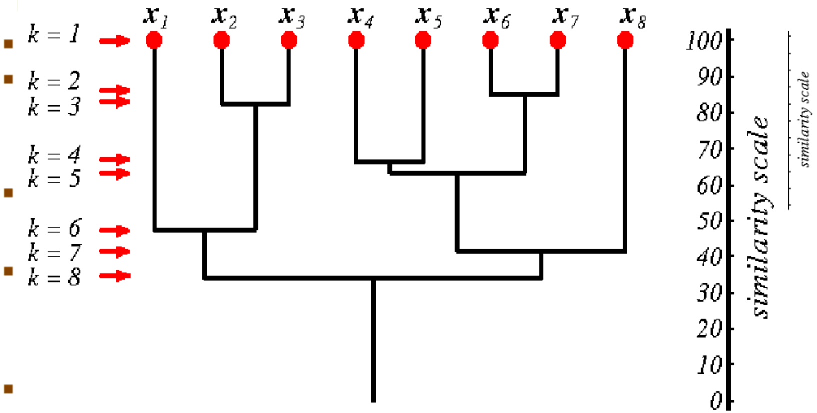


Example: biological taxonomy



A Dendrogram

- preferred way to represent a hierarchical clustering



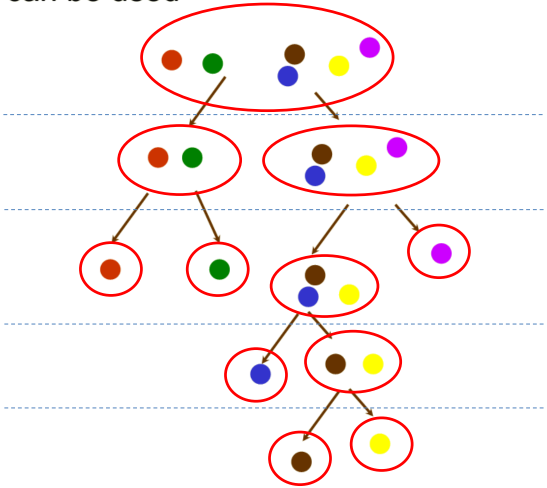
grouped clusters

Types of hierarchical clustering

- Divisive (top down) clustering
  Starts with all data points in one cluster, the root, then
  - Splits the root into a set of child clusters. Each child cluster is recursively divided further
  - stops when only singleton clusters of individual data points remain, i.e., each cluster with only a single point
- Agglomerative (bottom up) clustering
  The dendrogram is built from the bottom level by
  - merging the most similar (or nearest) pair of clusters
  - stopping when all the data points are merged into a single cluster (i.e., the root cluster).

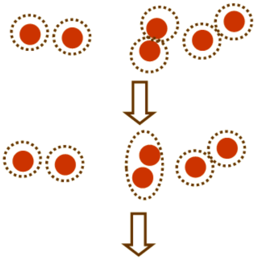
Divisive hierarchical clustering

- Any “flat” algorithm which produces a fixed number of clusters can be used
  - set ***c*** = ***2***



Agglomerative hierarchical clustering

```
initialize with each example in singleton cluster
while there is more than 1 cluster
    1. find 2 nearest clusters
    2. merge them
```

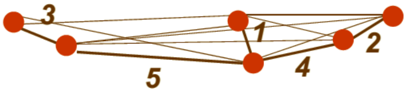


- Four common ways to measure cluster distance
  1. minimum distance $d_{\min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \| x - y \|$
  2. maximum distance $d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \| x - y \|$
  3. average distance $d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \| x - y \|$
  4. mean distance $d_{mean}(D_i, D_j) = \| \mu_i - \mu_j \|$

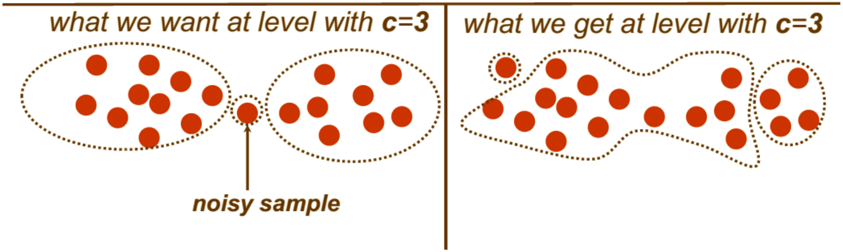
Single linkage or Nearest neighbor

- Agglomerative clustering with minimum distance

$$d_{\min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \| x - y \|$$



- generates minimum spanning tree
- encourages growth of elongated clusters
- disadvantage: very sensitive to noise



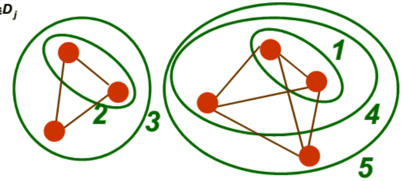
what we want at level with **c=3**
noisy sample
what we get at level with **c=3**

Complete linkage or Farthest neighbor

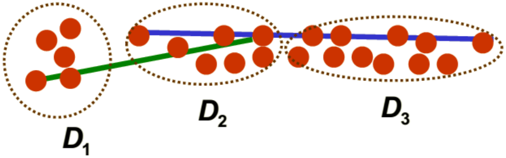
- Agglomerative clustering with maximum distance

$$d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \| x - y \|$$

- encourages compact clusters



- Does not work well if elongated clusters present



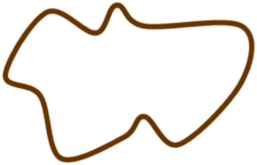
  - $d_{\max}(D_1, D_2) < d_{\max}(D_2, D_3)$
  - thus D_1 and D_2 are merged instead of D_2 and D_3

Divisive vs. Agglomerative

- Agglomerative is faster to compute, in general
- Divisive may be less "blind" to the global structure of the data

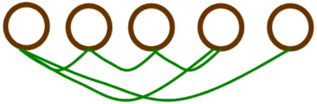
Divisive

when taking the first step (split), have access to all the data; can find the best possible split in 2 parts



Agglomerative

when taking the first step merging, do not consider the global structure of the data, only look at pairwise structure



Exercises with Hierarchical Clustering

- Jupyter notebooks

Part 3: Distributed Computing with Spark

Installations

- **Install Java idk8**
  https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.
- **Install pyspark in Anaconda**
  https://anaconda.org/conda-forge/pyspark
- Open a new notebook and run:

```
from pyspark import SparkContext
sc = SparkContext(master="local[4]")
print(sc)
```

- The output should be

```
<SparkContext master=local[4] appName=pyspark-shell>
```

References

- [1] UCSanDiegoX course: Big Data Analytics using Spark
- [2] MIT course: Computational Aspects of Machine Learning