DBSCAN.
Clustering by unmasking.
Hierarchical Clustering.

Radu Ionescu, Prof. PhD.

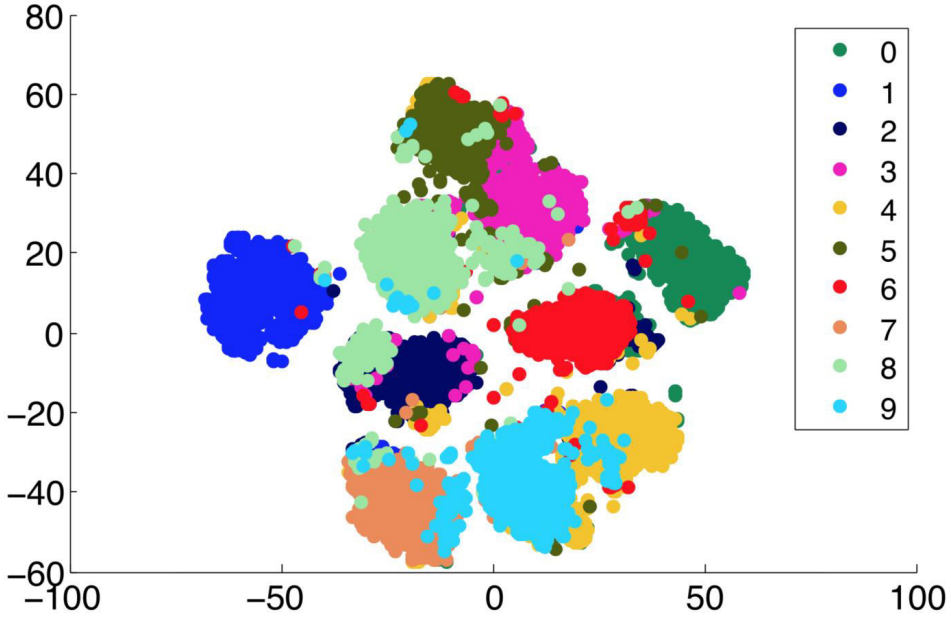
raducu.ionescu@gmail.com

Faculty of Mathematics and Computer Science

University of Bucharest

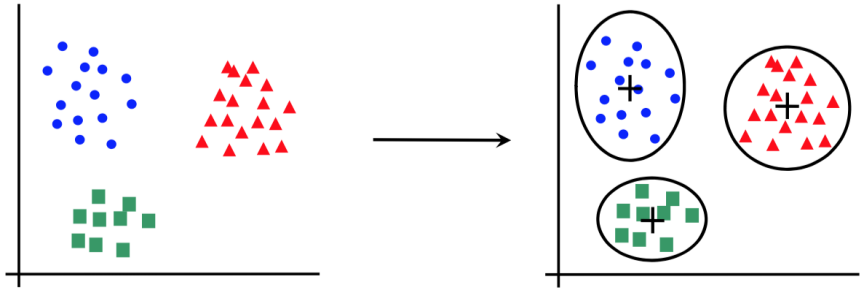
Reminder: Unsupervised Learning

- There are no labels for the training phase
- Our goal is to discover structure in data



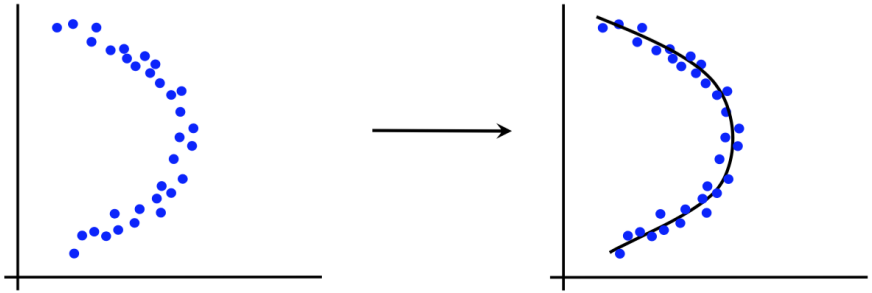
Canonical forms of unsupervised learning problems

- Clustering



- K-means
- DBSCAN
- Hierarchical Clustering
- …

- Dimensionality Reduction



- Principal Component Analysis
- t-SNE
- …

DBSCAN

DBSCAN

- DBSCAN stands for Density-based Spatial Clustering of Applications with Noise
- DBSCAN is data clustering algorithm that groups points which are closely packed together in feature space
- It was introduced in 1996:

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of KDD, pp. 226–231, 1996

- Unlike k-means, it does not require the number of clusters to be known in advance
- It has other hyperparameters that define the density of the formed clusters

Terminology

- Let X be a data set of points
- The ε-neighborhood of a point p is the set of points within an area of radius ε around the point p:

$$N_\varepsilon(p) = \{q \in X | \Delta(p, q) \leq \varepsilon\},$$

where is a distance function, e.g. the Euclidean distance

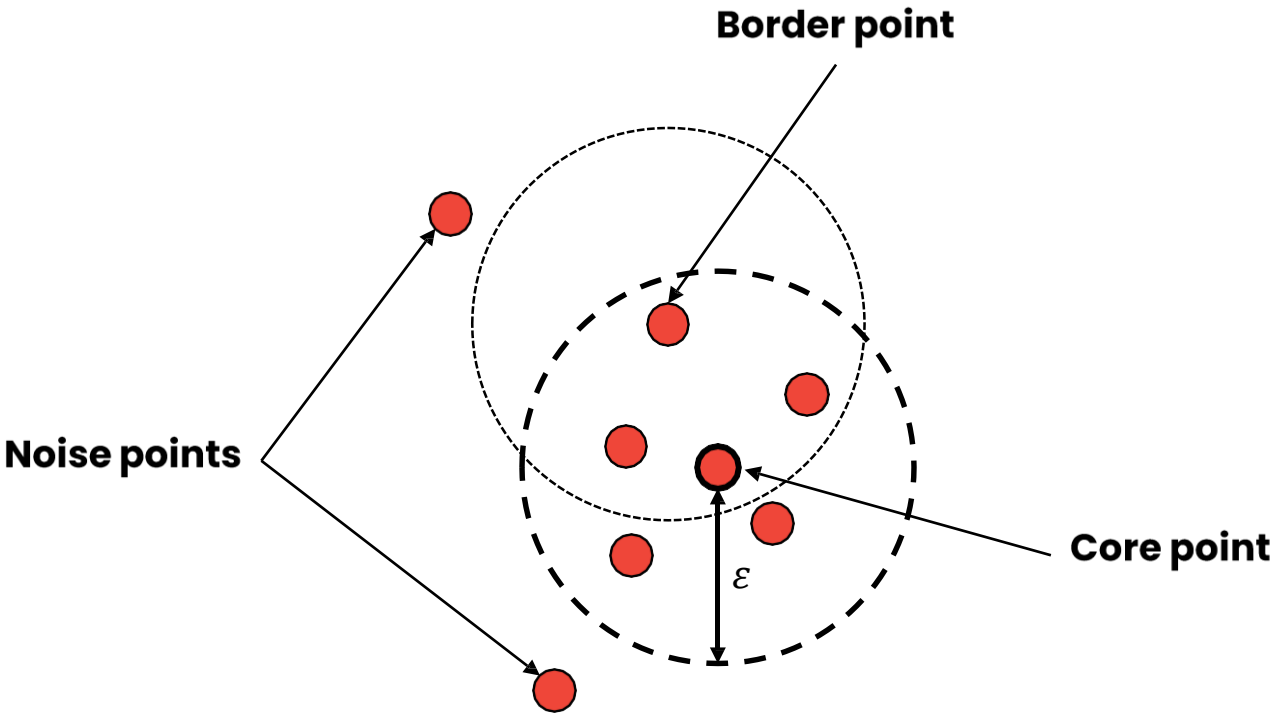
- A point p is a **core point** if its ε-neighborhood has at least m points:

$$|N_\varepsilon(p)| \geq m$$

- A point q is a **border point** if it is in the ε-neighborhood of a core point p, such that $q \neq p$
- All points which are neither core points nor border points are called noise points

Terminology

$m = 6$



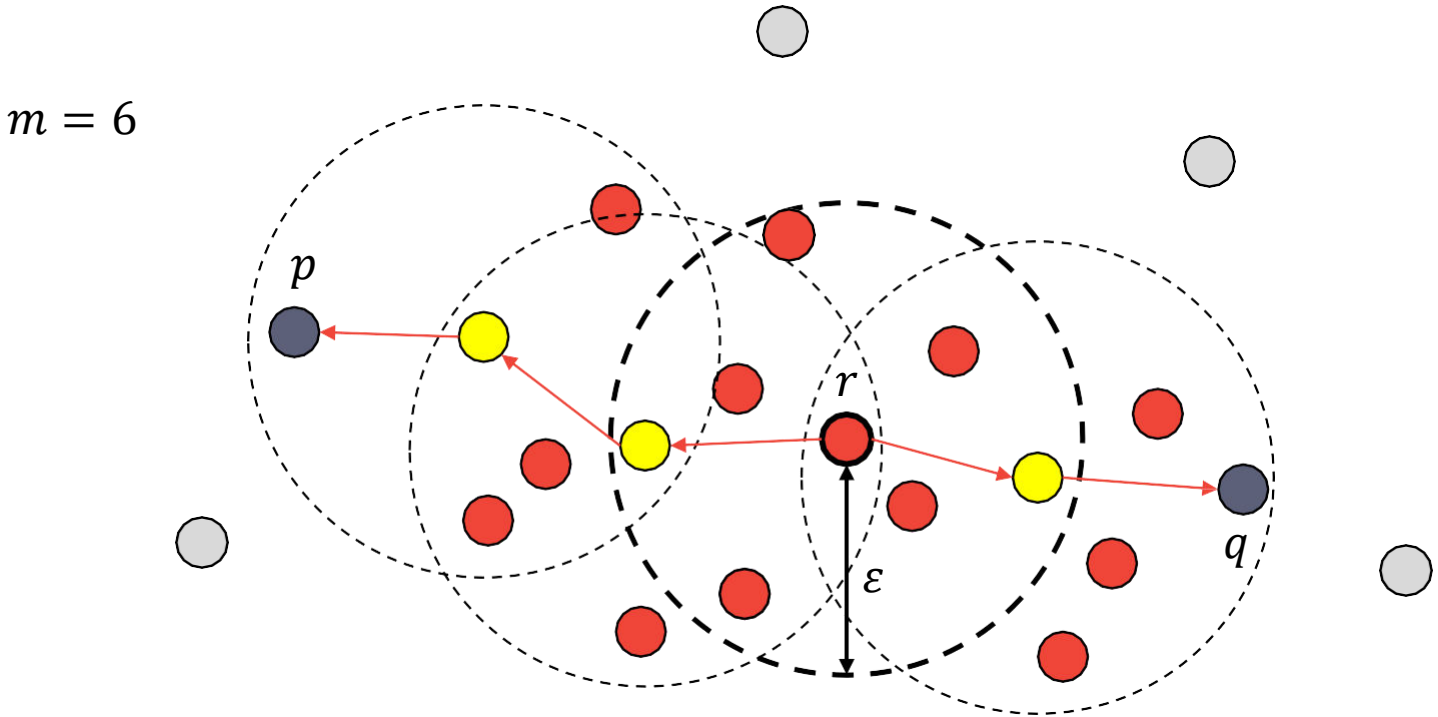
Terminology

- A point q is a **directly density-reachable** from a point p, if p is a core point and q is in the ε-neighborhood of p:

$$|N_\varepsilon(p)| \geq m \; and \; q \in N_\varepsilon(p)$$

- A point q is **density-reachable** from a point p, if there is a chain of points $p_1, p_2, \ldots, p_n \in X$ with $p_1 = p$ and $p_n = q$, such that p_{i+1} is **directly density-reachable** from p_i
- Two points p and q are density-connected if there is a point $r \in X$, such that both p and q are density-reachable from r
- A **cluster** $C \subset X$ is a set of points that satisfies the following conditions:
  - Maximality: $\forall p, q \in X$, if $p \in C$ and q is density-reachable from $p \Longrightarrow q \in C$
  - Connectivity: $\forall p, q \in C$, p and q are density-connected
- A cluster contains both core and border points!

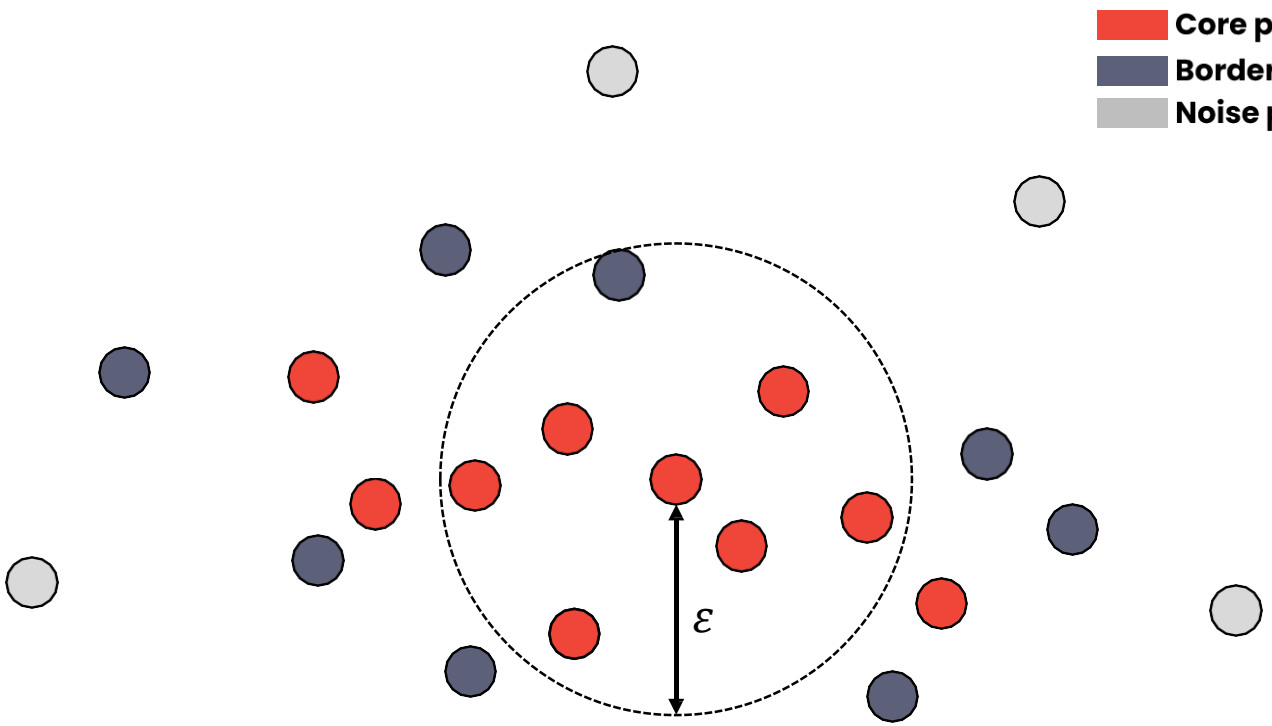
Terminology



- p and q are density-connected

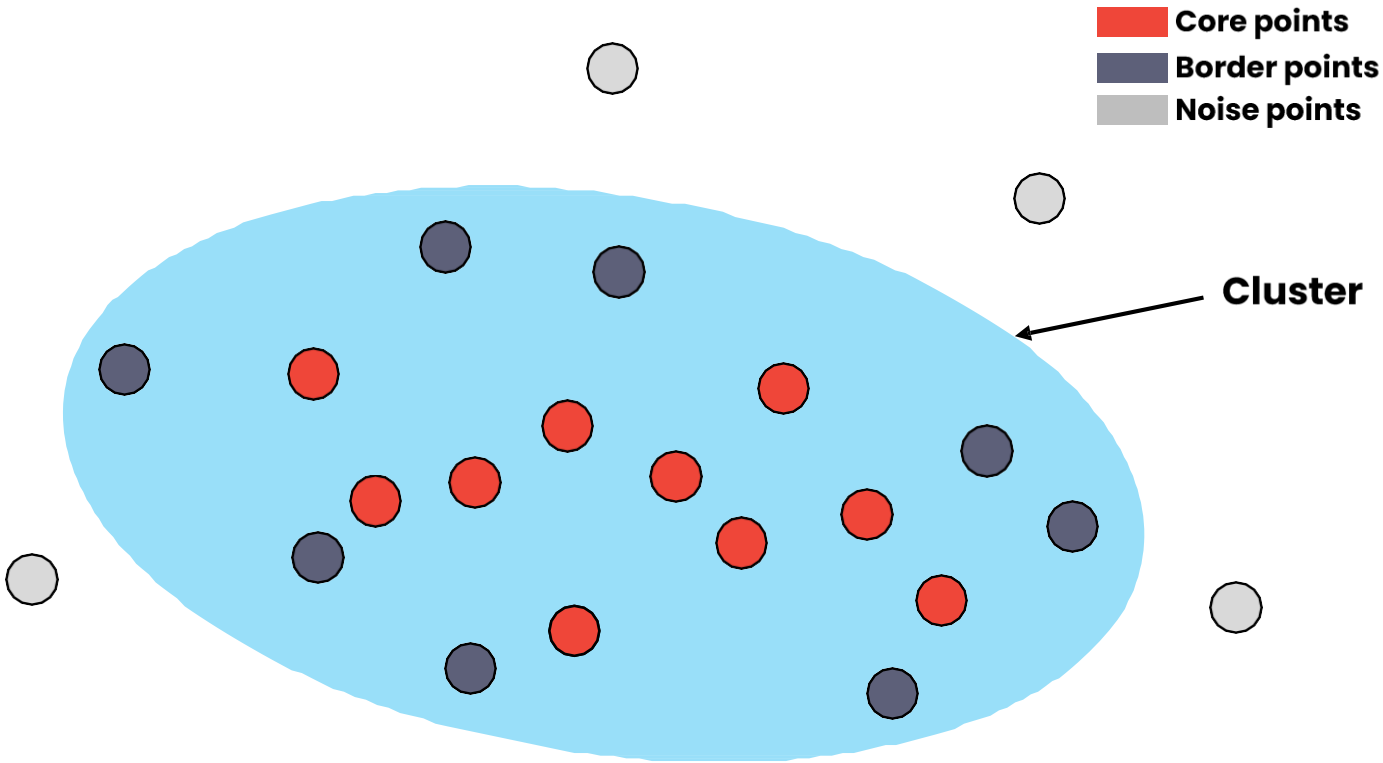
Terminology

$m = 6$



Terminology

$m = 6$



DBSCAN Algorithm (Python)

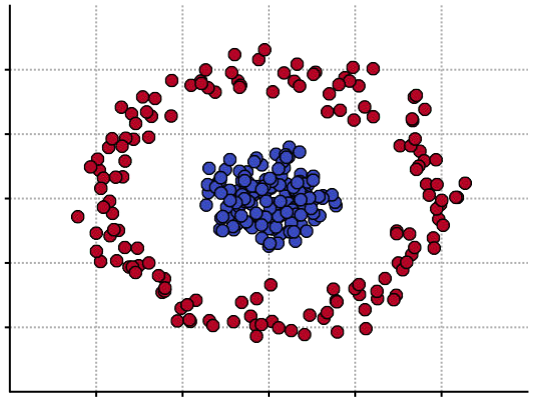
```
current_id = 1
for p in X:
   C[p] = -1 # initialize clustering assignments
for p in X:
   if is_core_point(p) == True:
      C[p] = current_id # label p with a unique cluster ID
      for q in X:
         if is_density_reachable(p,q) and p is not q:
            C[q] = current_id # label q with the same cluster ID as p
      current_id += 1
   else if C[p] == -1: # p has no label
      C[p] = 0 # label p as noise for now (might get relabeled later)
```

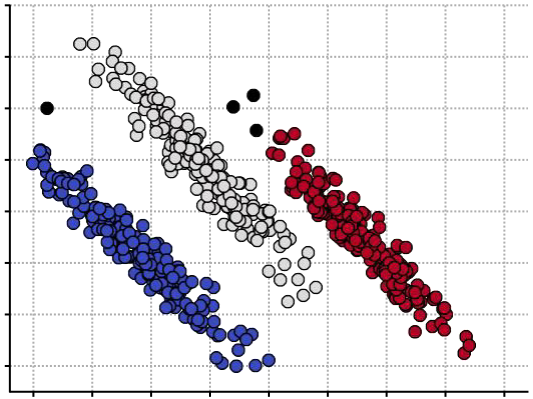
DBSCAN Algorithm in more details (Python)

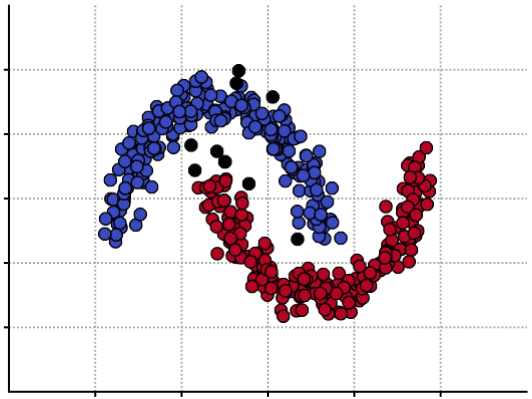
```
def DBSCAN(X, ε, m):
    c = 1                                          # cluster index
    for p in X:
        if p.label is not None:                    # previously processed
            continue
        neighbors = find_neighborhood(p, X, ε)     # find points in ε-neighborhood
        if len(neighbors) < m:
            p.label = "noise"                      # if not core point label as 'noise' (for now)
            continue
        c = c + 1                                  # increment cluster
        p.label = c                                # label first point in the new cluster
        S = neighbors - {p}                        # ε-neighbors of p that we add to the cluster and try to expand
        for q in S:
            if q.label == "noise":
                q.label = c                        # it was labeled as noise, but it is actually a border point
            if q.label is not None:
                continue                           # either border point or in some other cluster
            q.label = c                            # add the point to the cluster
            neighbors = find_neighborhood(q, X, ε) # find ε-neighborhood
            if len(neighbors) >= m:                # check if also core point
                S = S ∪ neighbors
```

DBSCAN results

- DBSCAN can handle non-convex clusters of various shapes
- It does not require the number of clusters to be given as input
- The distance metric can also be considered as a hyperparameter
  - Euclidean distance is the most commonly used

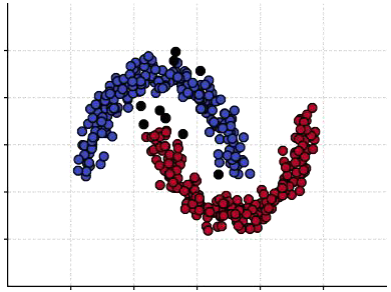






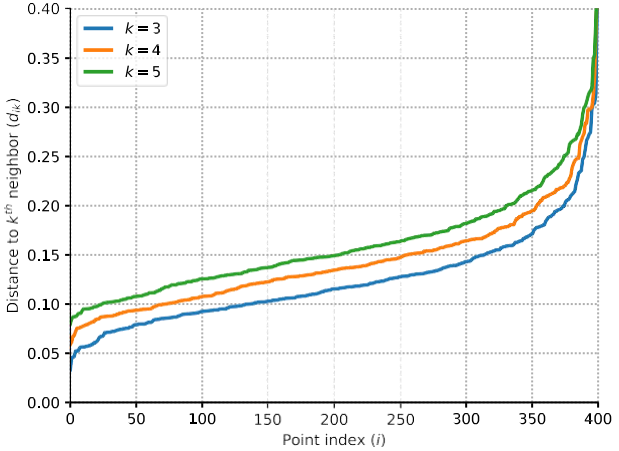
Hyperparameter tuning

- DBSCAN is very sensitive to the hyperparameters ε and m
- Heuristic for choosing ε and m:
  - Choose a number k (e.g. around 4)
  - For each point p_i, compute the distance Δ_{ik} to its k-th nearest neighbor
  - Sort the points by Δ_{ik} and plot the corresponding curve



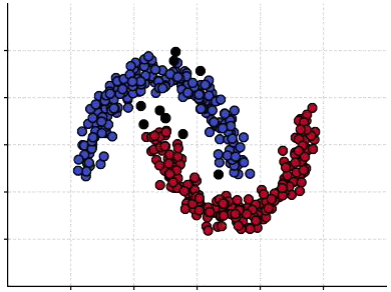
$\varepsilon = ?$

$m = ?$

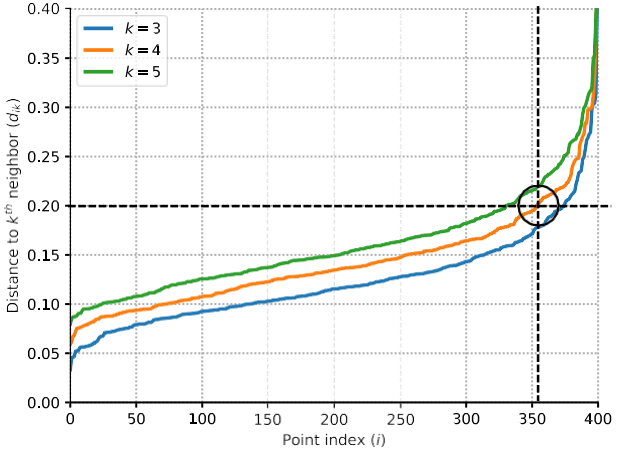


Hyperparameter tuning

- DBSCAN is very sensitive to the hyperparameters ε and m
- Heuristic for choosing ε and m:
  - Choose a number k (e.g. around 4)
  - For each point p_i, compute the distance Δ_{ik} to its k-th nearest neighbor
  - Sort the points by Δ_{ik} and plot the corresponding curve
  - Set $\varepsilon \approx \Delta_{ik}$ for an i for which the curve has large change in slope
  - Set $m = k$
- All points under this threshold will be core points
- It does not work very well for clusters with varying densities

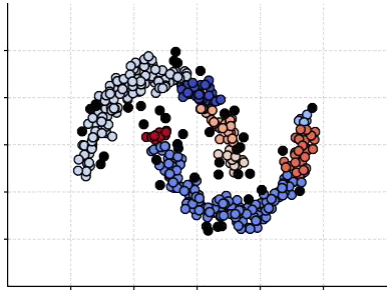


$\varepsilon = 0.2$
$m = 4$



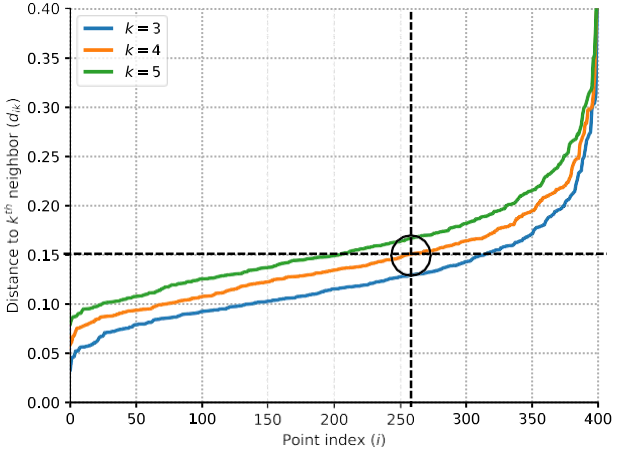
Hyperparameter tuning

- DBSCAN is very sensitive to the hyperparameters ε and m
- Heuristic for choosing ε and m:
  - Choose a number k (e.g. around 4)
  - For each point p_i, compute the distance Δ_{ik} to its k-th nearest neighbor
  - Sort the points by Δ_{ik} and plot the corresponding curve
  - Set $\varepsilon \approx \Delta_{ik}$ for an i for which the curve has large change in slope
  - Set $m = k$
- All points under this threshold will be core points
- It does not work very well for clusters with varying densities



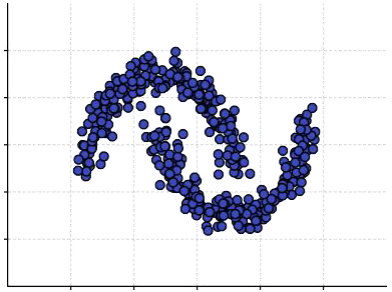
$\varepsilon = 0.15$
$m = 4$

Lower ε produces more clusters



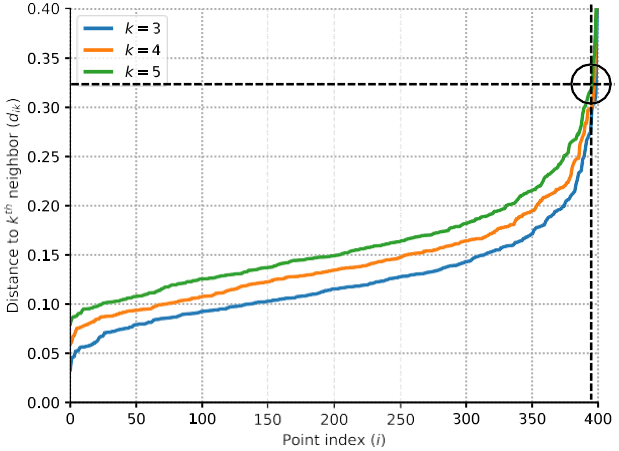
Hyperparameter tuning

- DBSCAN is very sensitive to the hyperparameters ε and m
- Heuristic for choosing ε and m:
  - Choose a number k (e.g. around 4)
  - For each point p_i, compute the distance Δ_{ik} to its k-th nearest neighbor
  - Sort the points by Δ_{ik} and plot the corresponding curve
  - Set $\varepsilon \approx \Delta_{ik}$ for an i for which the curve has large change in slope
  - Set $m = k$
- All points under this threshold will be core points
- It does not work very well for clusters with varying densities



$\varepsilon = 0.32$
$m = 4$

Higher ε produces less clusters



Summary

- **DBSCAN** is a density-based clustering algorithm which groups points that are closely packed together and marks as noise (outliers) points that are in low-density regions
- DBSCAN defines a cluster as a group of points that are density-connect to each other
  - It means that for any pair of points, there is a chain of core points (i.e. points with enough neighbors around them) connecting them
- Different from k-means, DBSCAN:
  - Can find clusters of arbitrary (non-convex) shapes
  - Does not require the number of clusters to be known in advance
- DBSCAN has two hyperparameters, ε (the neighborhood radius around core points) and m (the minimum number of neighbors that define a core point)
  - These parameters are not easy to tune!

Clustering by unmasking

Clustering by unmasking

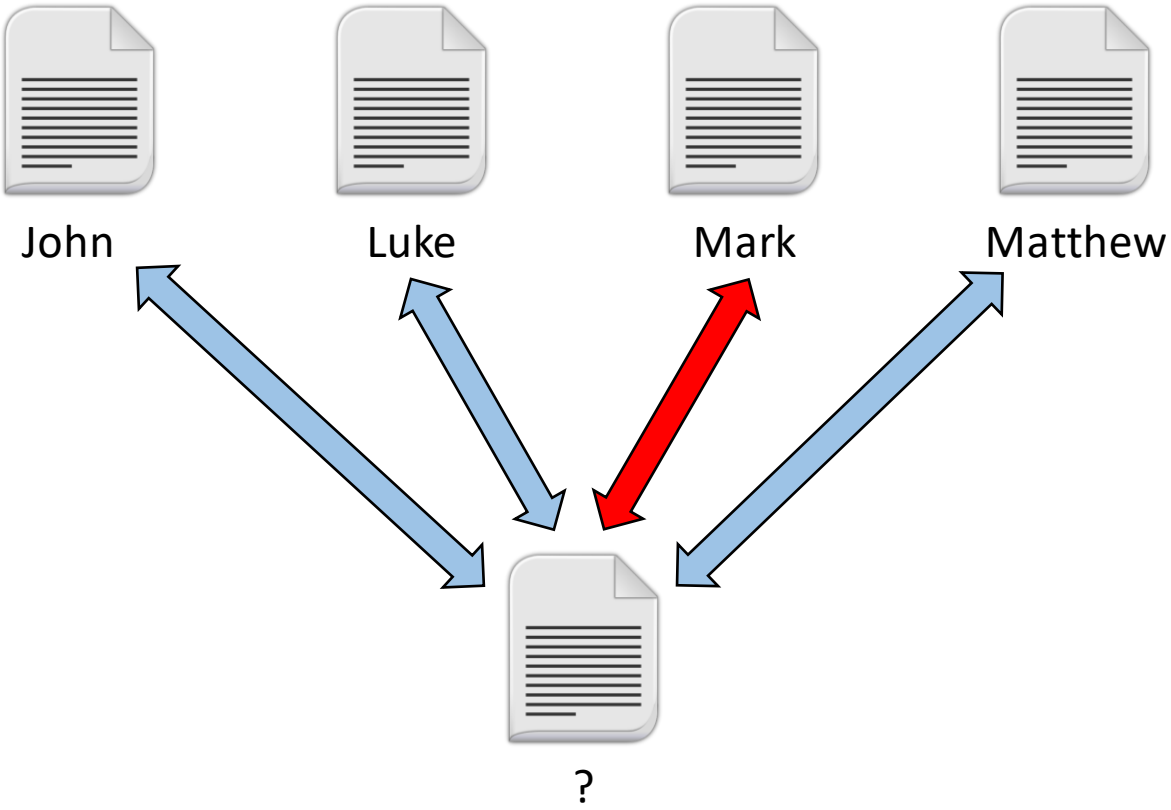
- **Clustering by unmasking** is a novel agglomerative clustering method based on unmasking
- It was introduced in 2019:

Mariana-Iuliana Georgescu, Radu Tudor Ionescu. Clustering Images by Unmasking – A New Baseline. Proceedings of ICIP, pp. 1580–1584, 2019.

- **Unmasking** is a technique that was previously used for authorship verification of text documents:

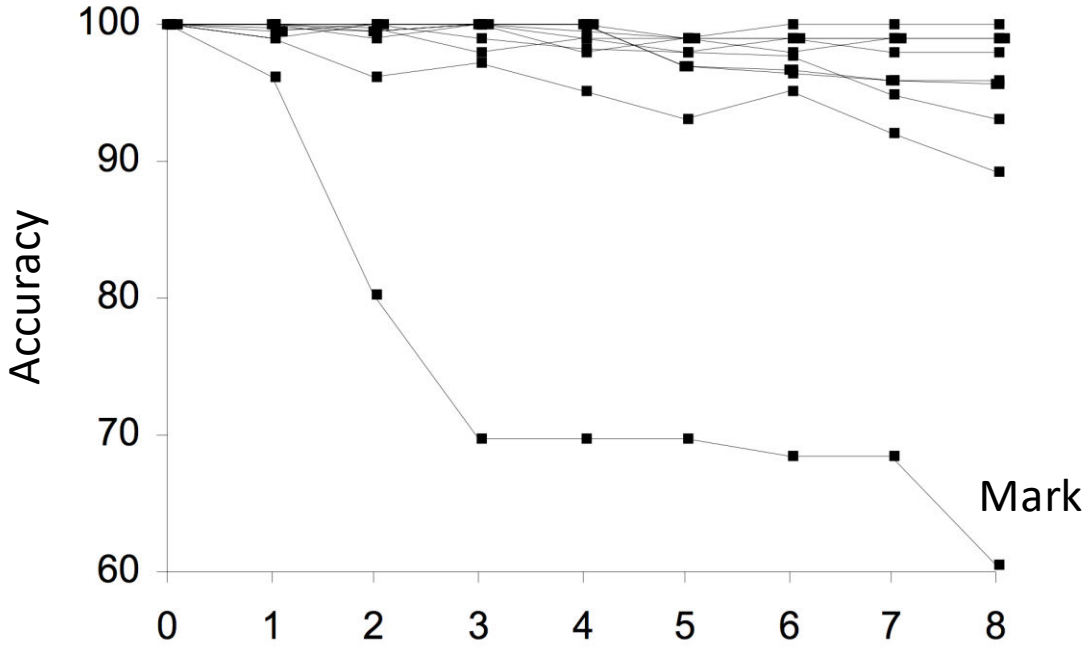
Moshe Koppel, Jonathan Schler, Elisheva Bonchek-Dokow. Measuring Differentiability: Unmasking Pseudonymous Authors. Journal of Machine Learning Research, 8:1261–1276, 2007.

Unmasking for authorship verification



Unmasking for authorship verification

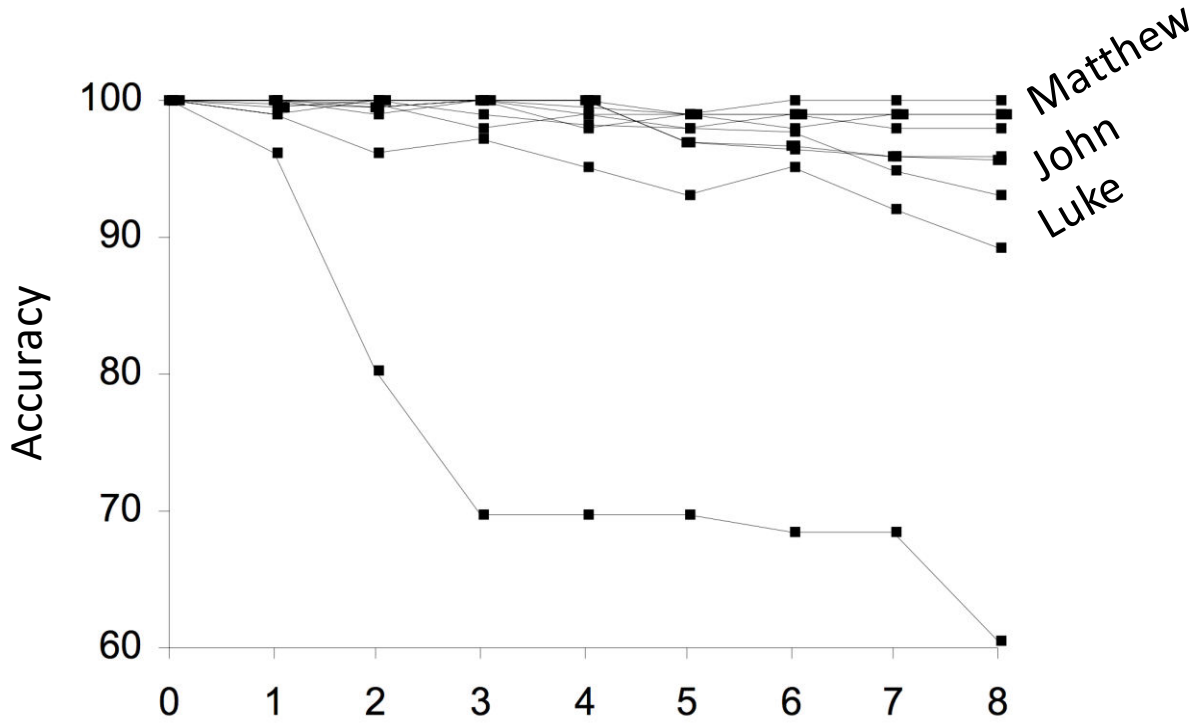
- Unmasking [Koppel et al., JMLR07]: iteratively remove the best features useful for distinguishing between two documents to measure the degradation rate of the cross-validation accuracy



Unmasking for authorship verification

- At first, a bit hard to think of, but consider this:

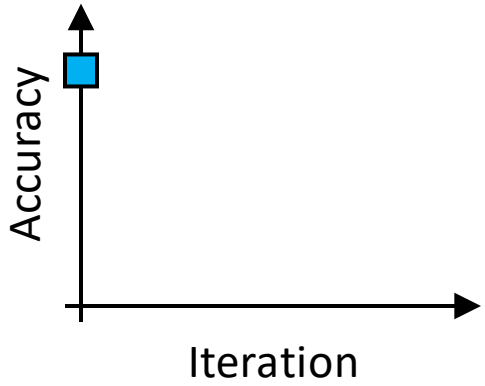
➢ Accuracy stays high if objects are different to their core!



Unmasking – Loop 1.1.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	l_i
x_1	1	5	0	1	2	0	7	1	+1
x_2	2	4	0	2	3	1	5	0	+1
x_3	3	0	3	5	2	1	1	0	-1
x_4	2	1	4	4	3	0	0	2	-1

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
w	+0.5	+6.0	-5.0	-3.0	+0.1	-0.1	+4.0	-0.1



- Given a set of training samples $\{x_1, x_2, \dots, x_n\}$, a linear classifier learns a set of weights w, b such that:

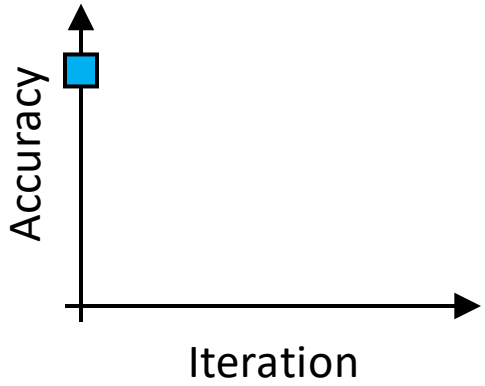
$$sign(w \cdot x + b) = sign(w_1 x_1 + \cdots + w_n x_n + b) = l_i$$

- We evaluate the model and compute the accuracy

Unmasking – Loop 1.2.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	l_i
x_1	1	5	0	1	2	0	7	1	+1
x_2	2	4	0	2	3	1	5	0	+1
x_3	3	0	3	5	2	1	1	0	-1
x_4	2	1	4	4	3	0	0	2	-1

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
w	+0.5	+6.0	-5.0	-3.0	+0.1	-0.1	+4.0	-0.1

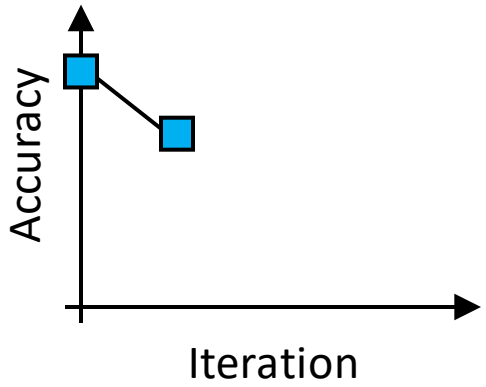


- We eliminate the features with the highest weights in absolute value

Unmasking – Loop 2.1.

	f_1	f_4	f_5	f_6	f_7	f_8	l_i
x_1	1	1	2	0	7	1	+1
x_2	2	2	3	1	5	0	+1
x_3	3	5	2	1	1	0	-1
x_4	2	4	3	0	0	2	-1

	w_1	w_4	w_5	w_6	w_7	w_8
w	-1.5	-4.0	+0.1	-0.1	+5.0	-0.5



- Given a set of training samples $\{x_1, x_2, \dots, x_n\}$, a linear classifier learns a set of weights w, b such that:

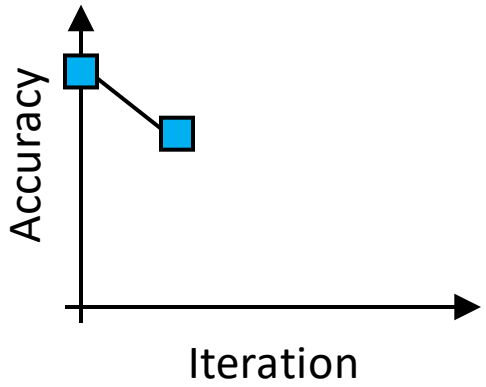
$$sign(w \cdot x + b) = sign(w_1 x_1 + \cdots + w_n x_n + b) = l_i$$

- We evaluate the model and compute the accuracy

Unmasking – Loop 2.2.

	f_1	f_4	f_5	f_6	f_7	f_8	l_i
x_1	1	1	2	0	7	1	+1
x_2	2	2	3	1	5	0	+1
x_3	3	5	2	1	1	0	-1
x_4	2	4	3	0	0	2	-1

	w_1	w_4	w_5	w_6	w_7	w_8
w	-1.5	-4.0	+0.1	-0.1	+5.0	-0.5

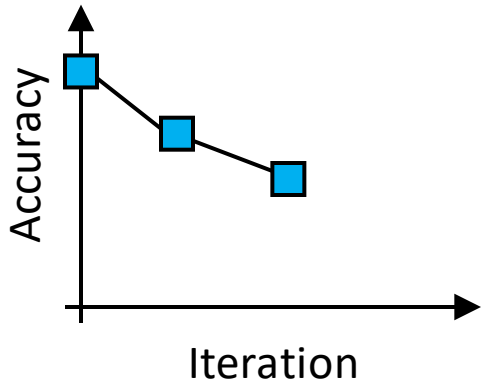


- We eliminate the features with the highest weights in absolute value

Unmasking – Loop 3.1.

	f_1	f_5	f_6	f_8	l_i
x_1	1	2	0	1	+1
x_2	2	3	1	0	+1
x_3	3	2	1	0	-1
x_4	2	3	0	2	-1

	w_1	w_5	w_6	w_8
w	-1.5	+0.1	-0.1	-0.5



- And so on…

Clustering by unmasking (Algorithm)

- Input data and parameters:
  - m – number of training samples
  - k – the number of clusters
  - n – the number of unmasking iterations
  - s – the number of features to be removed at each iteration

Clustering by unmasking (Algorithm)

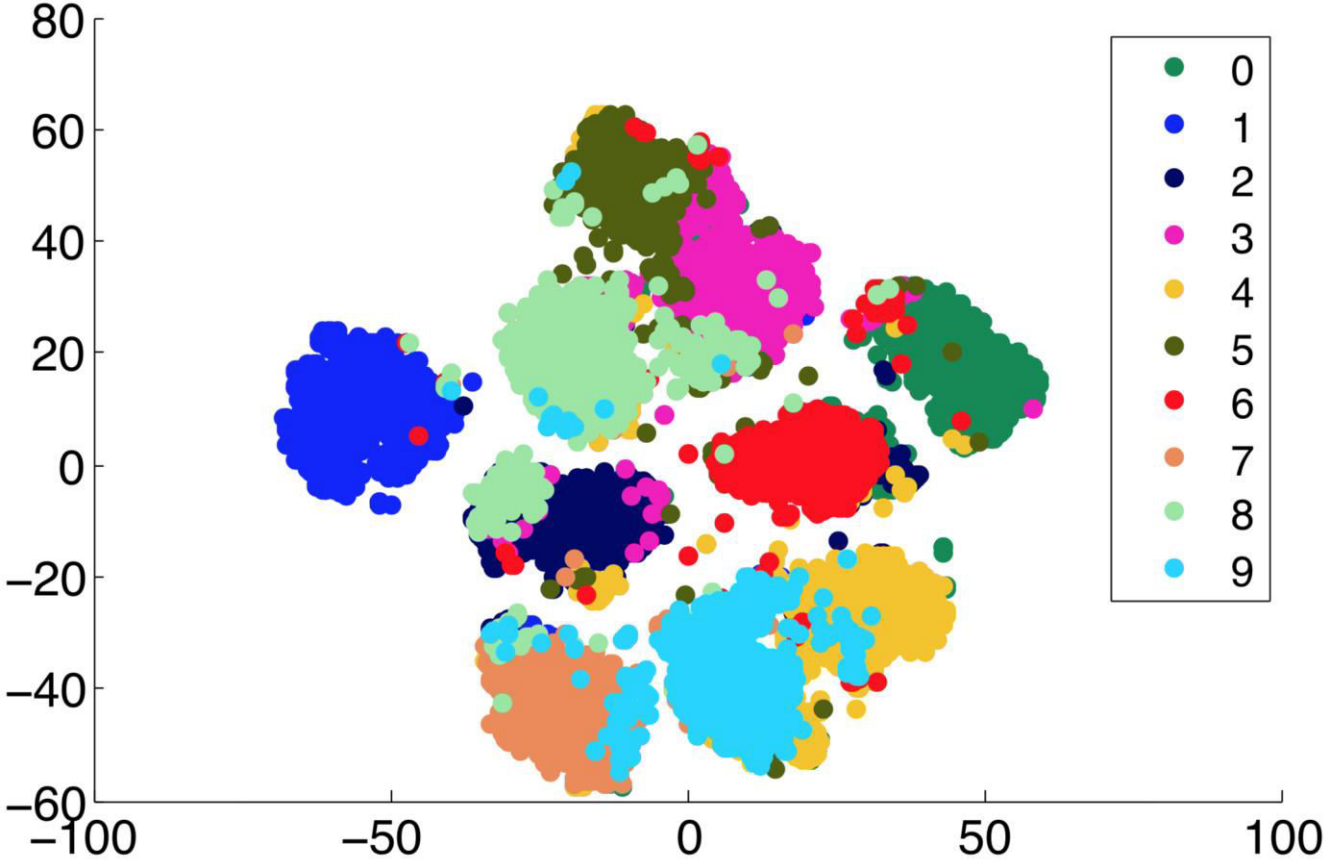
- The algorithm starts with K clusters ($K \gg k$) and executes the following steps:

1. For each pair of clusters i and j, we assume that the samples in cluster i belong to a different class than the samples in cluster j and compute a score that indicates the likelihood of the statement *"clusters i and j should be joined"* = **true**
2. Randomly split the samples in each cluster into a training set and a testing set
3. Train a linear SVM on the training set (until convergence) and evaluate it on the test set, retaining the accuracy rate
4. Sort the weights of the SVM by their absolute values in descending order, take the first s indexes of the sorted list, then remove the corresponding features
5. Repeat steps 3 and 4 for n iterations, retaining the accuracy rate at each iteration
6. Merge each cluster i with the cluster j (using a Greedy approach), such that the score of joining clusters i and j is maximum, $\forall j \in \{1,2,\ldots.,K\}$, with $j \neq i$
7. If the number of clusters k is reached at any point during the merging process, halt the execution and return the current cluster assignments
8. Otherwise, continue by computing the merging scores for the newly-formed clusters

Results on MNIST

Method	ACC	NMI
Random chance	10.00%	-
SVM	94.40%	-
K-means	55.82%	52.18%
IDEC [6]	71.45%	69.40%
Unmasking (n=1)	72.58%	64.99%
Unmasking	81.40%	69.76%

Results on MNIST



Results on UIUCTex and Oxford Flowers

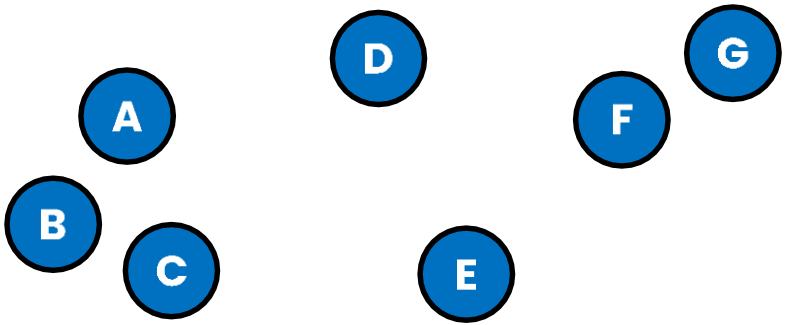
Features	Method	UIUCTex		Oxford Flowers	
		ACC	NMI	ACC	NMI
-	Random chance	4.00%	-	6.67%	-
VGG-f	SVM	97.20%	-	95.50%	-
	K-means	48.20%	70.15%	60.35%	69.55%
	Unmasking (n=1)	19.80%	54.81%	45.50%	62.98%
	Unmasking	61.40%	74.94%	67.50%	75.82%
BOVW	SVM	94.60%	-	80.83%	-
	K-means	25.40%	46.83%	22.10%	22.83%
	Unmasking (n=1)	35.20%	55.30%	12.83%	14.64%
	Unmasking	44.60%	58.81%	25.00%	25.37%
AlexNet	SVM	96.20%	-	81.00%	-
	K-means	36.80%	58.52%	26.89%	30.43%
	Unmasking (n=1)	34.20%	62.07%	9.80%	18.19%
	Unmasking	48.60%	69.78%	33.33%	38.00%

Hierarchical Clustering

Hierarchical Clustering

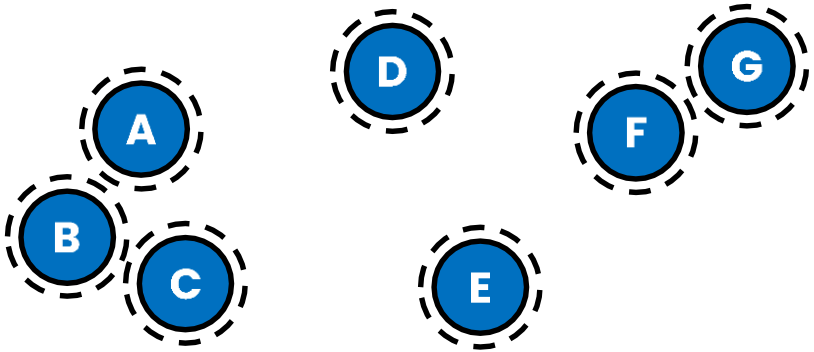
- Hierarchical Clustering is a set of clustering methods that aim at building a hierarchy of clusters
  - A cluster is composed of smaller clusters
- There are two strategies for building the hierarchy of clusters:
  - **Agglomerative** (bottom-up): we start with each point in its own cluster and we merge pairs of clusters until only one cluster is formed
  - **Divisive** (top-down): we start with a single cluster containing the entire set of points and we recursively split until each points is in its own cluster
- The most popular strategy in practical use is bottom-up (agglomerative)!

Agglomerative Hierarchical Clustering



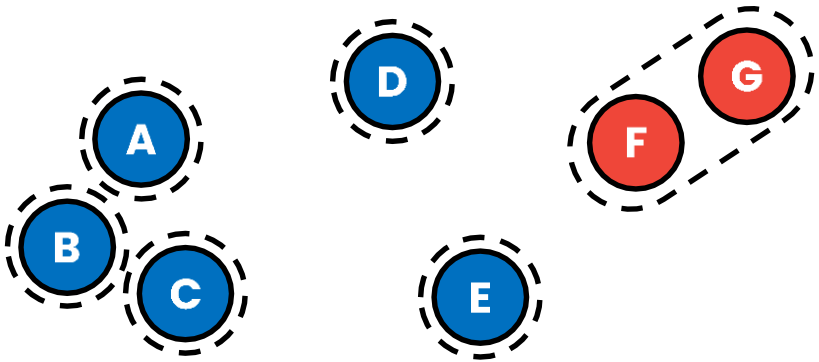
Agglomerative Hierarchical Clustering

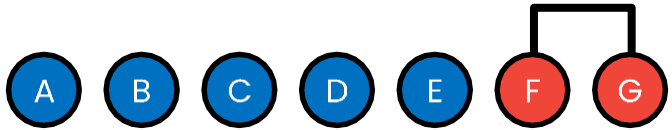
- Each point starts as its own cluster



Agglomerative Hierarchical Clustering

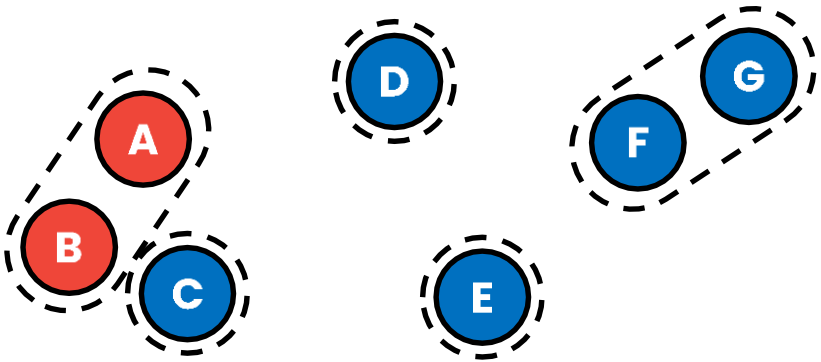
- Each point starts as its own cluster
- At each step, the two most similar clusters are merged

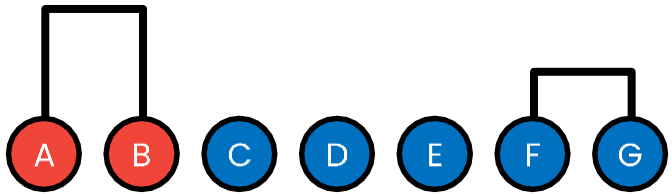




Agglomerative Hierarchical Clustering

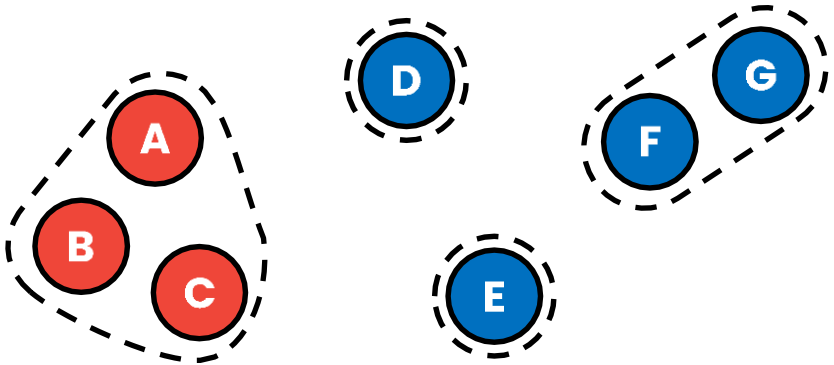
- Each point starts as its own cluster
- At each step, the two most similar clusters are merged

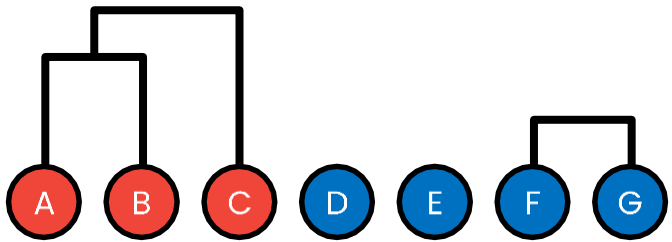




Agglomerative Hierarchical Clustering

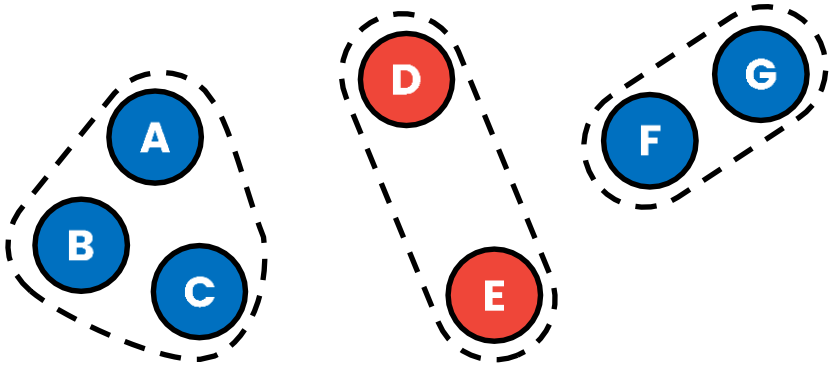
- Each point starts as its own cluster
- At each step, the two most similar clusters are merged

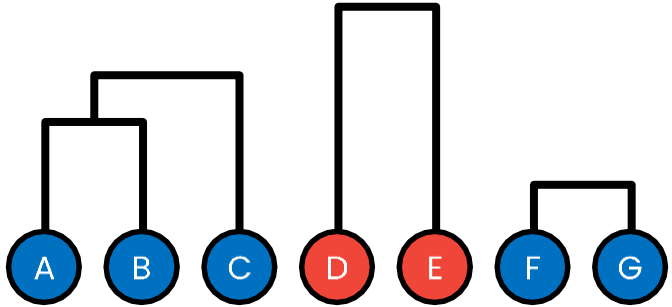




Agglomerative Hierarchical Clustering

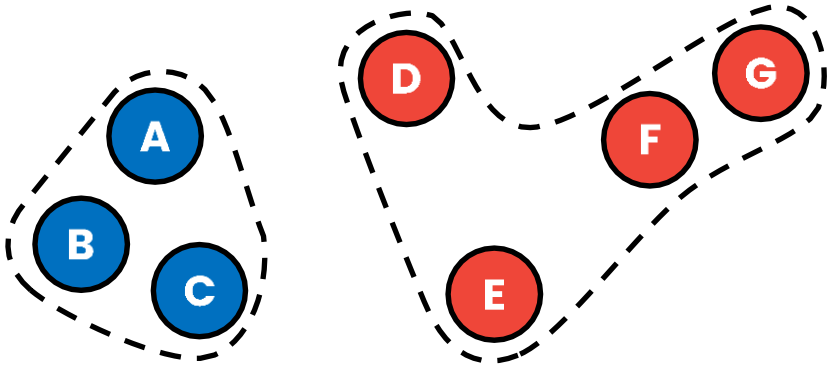
- Each point starts as its own cluster
- At each step, the two most similar clusters are merged

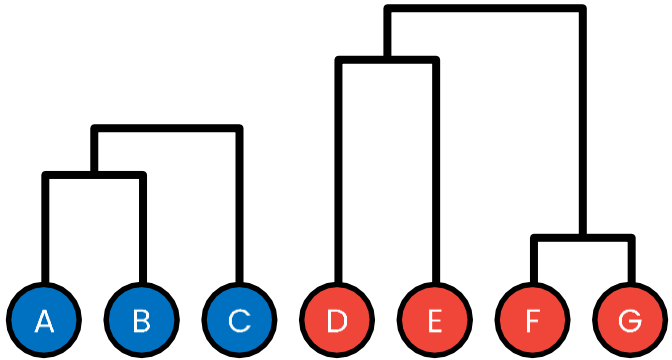




Agglomerative Hierarchical Clustering

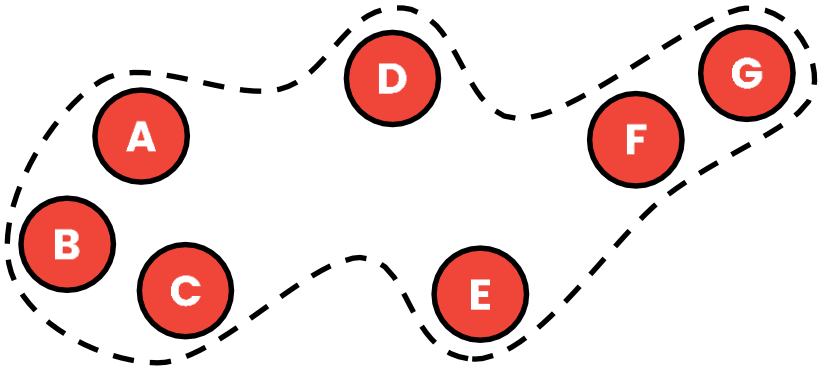
- Each point starts as its own cluster
- At each step, the two most similar clusters are merged

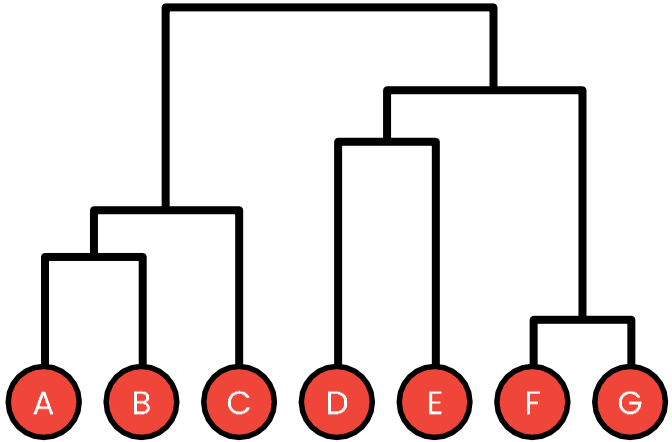




Agglomerative Hierarchical Clustering

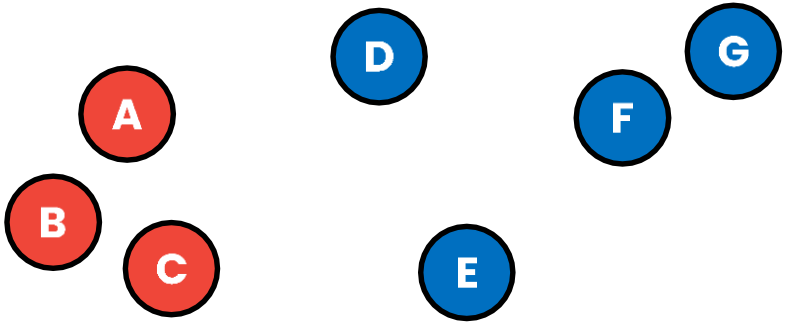
- Each point starts as its own cluster
- At each step, the two most similar clusters are merged



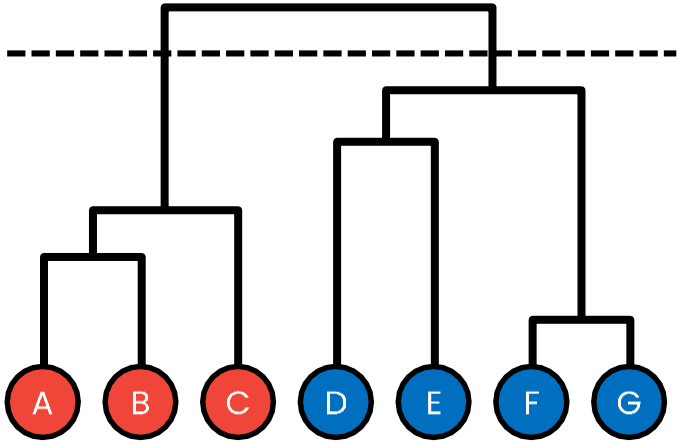


The tree of clusters is called **dendrogram**

Agglomerative Hierarchical Clustering

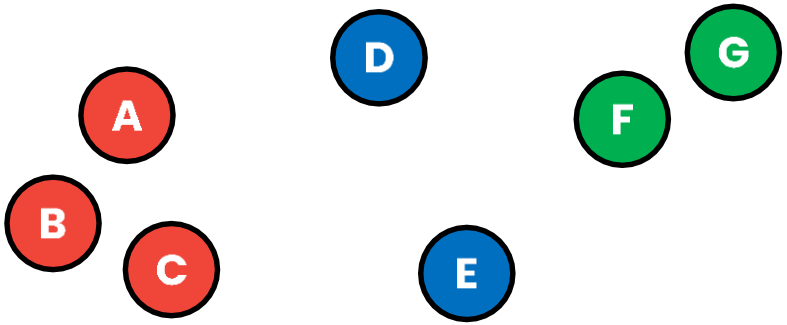


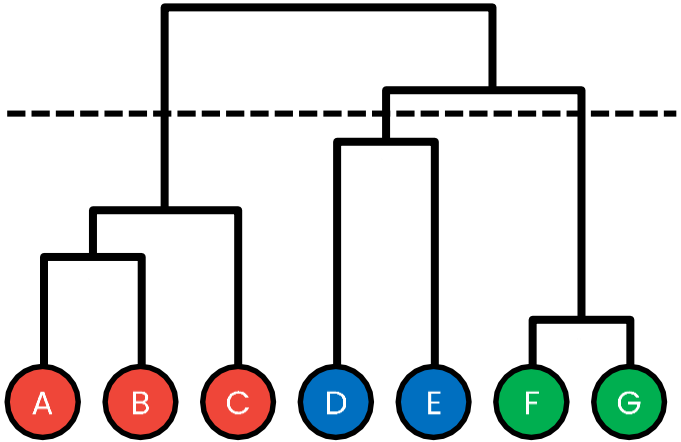
- Each point starts as its own cluster
- At each step, the two most similar clusters are merged
- We can cut at any level to get a clustering



Agglomerative Hierarchical Clustering

- Each point starts as its own cluster
- At each step, the two most similar clusters are merged
- We can cut at any level to get a clustering





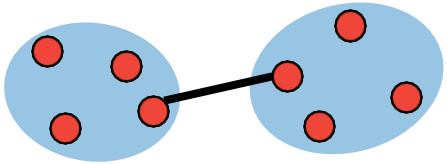
Agglomerative Hierarchical Clustering (Python)

```
def agglomerative_clustering(X = {x_1, x_2, ..., x_n}):
    C = {C_1 = {x_1}, C_2 = {x_2}, ..., C_n = {x_n}} # each point as its own cluster
    steps = [] # steps to recreate the dendrogram
    while len(C) > 1:
        i*, j* = argmax_{i,j} [Δ(C_i, C_j)] # distance between clusters
        C = C − {C_i*, C_j*}
        C = C + {C_i* ∪ C_j*}
        steps.append(i*, j*)
    return steps
```

Linkage criterion

- We need to define a way to measure the distance (linkage criterion) between clusters:

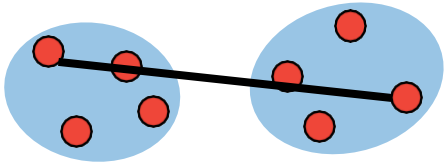
Single linkage



$$\Delta(C_i, C_j) = \min_{x \in C_i, y \in C_j} \{\Delta(x, y)\}$$

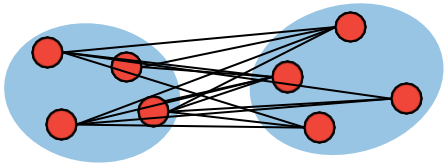
We can use any distance metric

Complete linkage



$$\Delta(C_i, C_j) = \max_{x \in C_i, y \in C_j} \{\Delta(x, y)\}$$

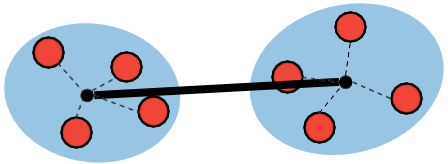
Average linkage



$$\Delta(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} \Delta(x, y)$$

Not all distance metrics have centroids

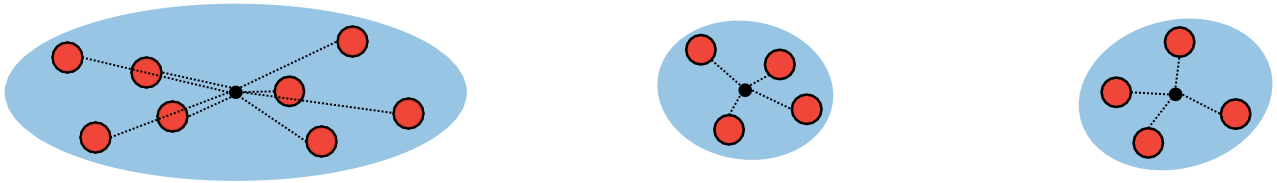
Centroid linkage



$$\Delta(C_i, C_j) = \Delta\left(\frac{\sum_{x \in C_i} x}{|C_i|}, \frac{\sum_{y \in C_j} y}{|C_j|}\right)$$

Linkage criterion

- Ward's criterion is given by the increase in variance due to the merging of two clusters
  - ➢ We link the clusters with the smallest increase in variance!



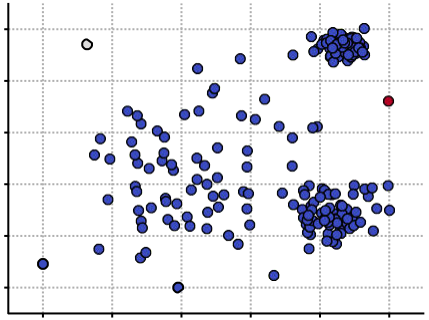
$$\Delta(C_i, C_j) = \frac{1}{|C_i \cup C_j|} \sum_{x \in C_i \cup C_j} \Delta(x, \mu_{C_i \cup C_j}) - \frac{1}{|C_i|} \sum_{y \in C_i} \Delta(y, \mu_{C_i}) - \frac{1}{|C_j|} \sum_{z \in C_j} \Delta\left(z, \mu_{C_j}\right)$$

$$= Var(C_i \cup C_j) - Var(C_i) - Var(C_j)$$

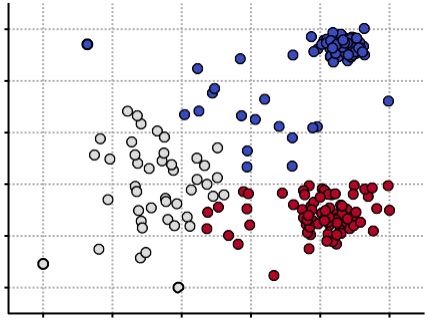
Comparing linkage criteria

- **Single linkage:**
  - Tends to produce long, chain-like clusters
  - The distance between the two farthest elements in a cluster can be large
  - Can handle non-convex clusters
  - Sensitive to noise
- **Complete linkage:**
  - Keeps the points in a cluster close together
  - Tends to produce spherical and compact clusters
  - Less sensitive to noise than single linkage
- **Average linkage:**
  - Compromise between single and complete linkages
  - Produces results more similar to complete linkage
- **Ward's linkage:**
  - Similar results to complete linkage
  - Handles clusters with various densities better than complete linkage

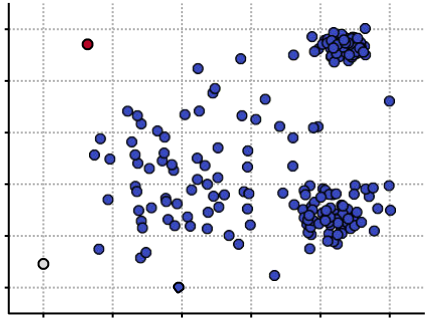
Comparing linkage criteria



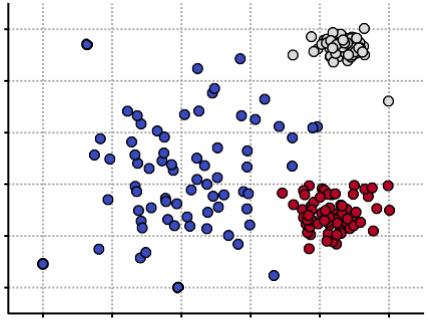
Single linkage



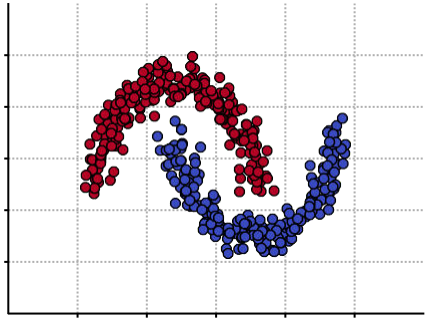
Complete linkage

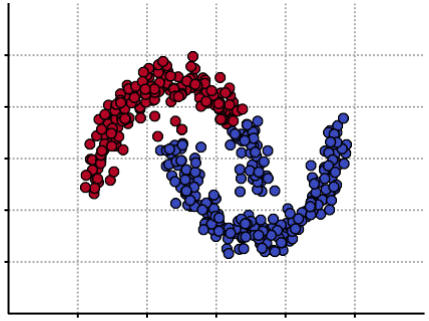


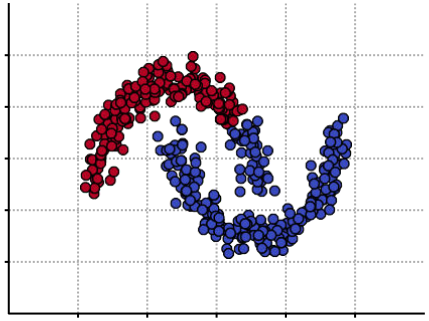
Average linkage

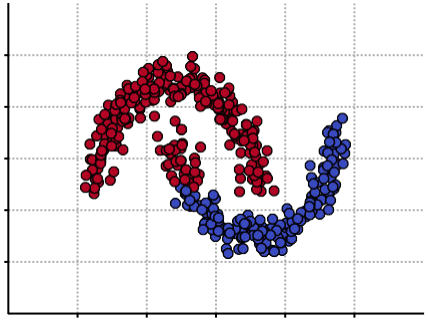


Ward's linkage









Case study: Clustering DNA sequences with Local Rank Distance

Local Rank Distance

- **Local Rank Distance (LRD)** is a new distance measure for strings introduced in [R.T. Ionescu, 2013]
- LRD is inspired from Rank Distance (RD) [L. Dinu & F. Manea, 2006]
- RD measures the global non-alignment score between two sequences
- Example with RD:
  - Let $x = ABAB$, $y = BABB$ be two strings from $\{A, B\}^*$
  - Characters are annotated with indexes in order to eliminate duplicates:

$$\bar{x} = A_1 B_1 A_2 B_2, \bar{y} = B_1 A_1 B_2 B_3$$

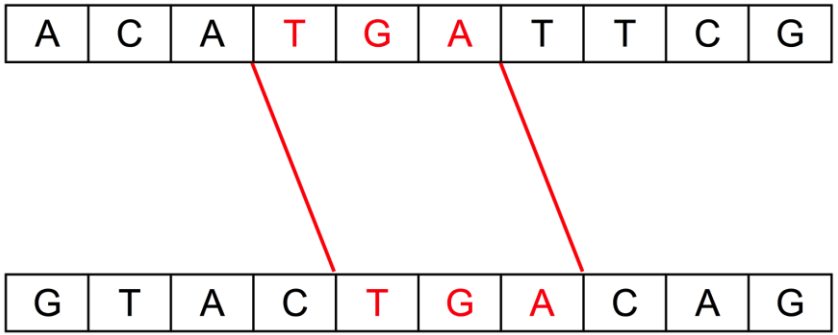
  - Position offsets of identical characters are summed up:

$$\Delta_{RD}(x, y) = |1 - 2| + |2 - 1| + 3 + |4 - 3| + 3 = 9$$

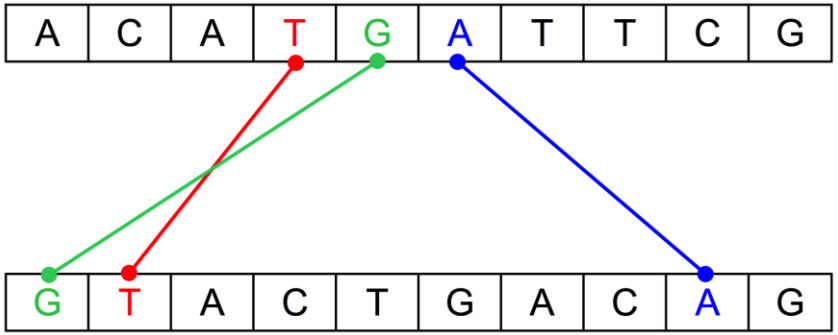
  - Does it seem right to match B_2 in $\bar{x}$ with B_2 in $\bar{y}$ instead of B_3?

Local Rank Distance

- LRD is based on more generic principles than RD:
  - Characters are no longer annotated
  - Substrings (k-mers or n-grams) are used instead of single characters
- LRD measures the local non-alignment score between two sequences



Local Rank Distance



Rank Distance

Efficient Algorithm for LRD

- Given two strings $s, t \in \Sigma^*$, the efficient algorithm to compute $\Delta_{LRD}(s, t)$ works as follows:
  1. Store n-gram positions from t in a hash (inverted index) table h
  2. Search for each n-gram x from s in the inverted index table
  3. Do a binary search in the positional array corresponding $h(x)$ to find the nearest position of x in t
  4. Repeat steps 1–3 by swapping s and t for symmetry, i.e. to make sure that:

$$\Delta_{LRD}(s, t) = \Delta_{LRD}(t, s)$$

- What is the time complexity?
  - $O(|s| \cdot \log|t| + |t| \cdot \log|s|)$,where $|s|$ and $|t|$ are the lengths of s and t, respectively

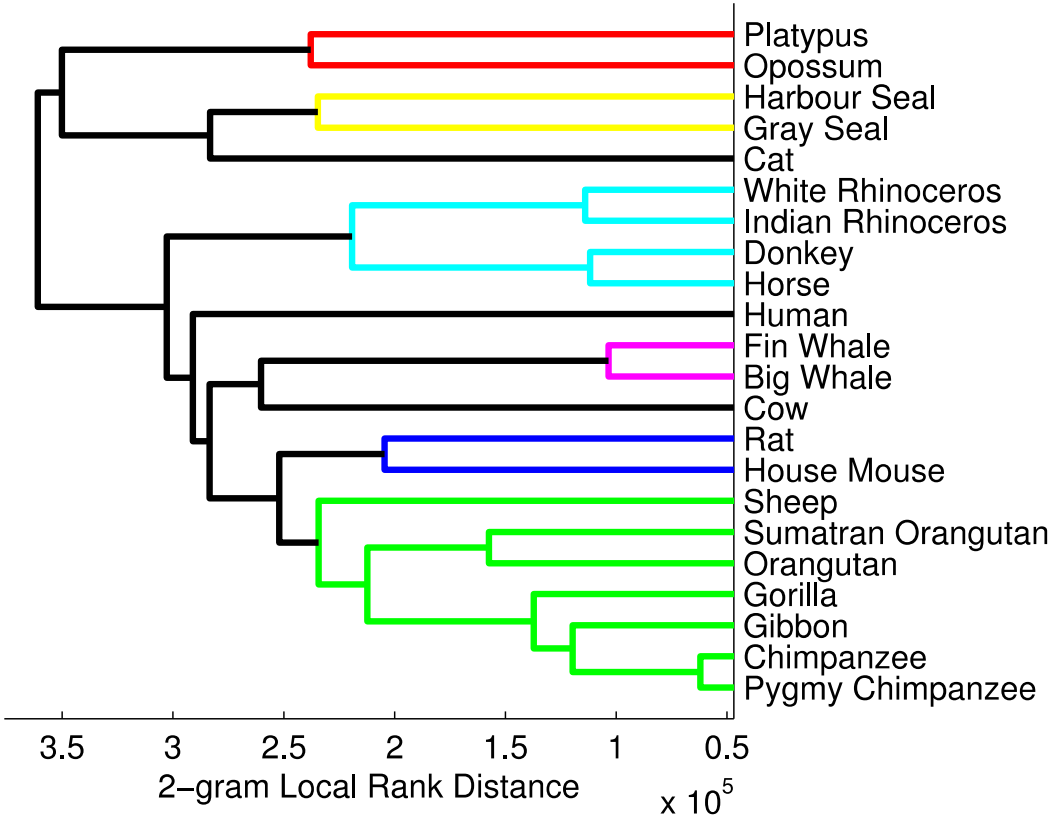
Efficient Algorithm for LRD

- Example with LRD:
  - Let $s = ABAB$, $t = BABBABA$ be two strings from $\{A, B\}^*$
  - The inverted index table with 3-grams from t is:

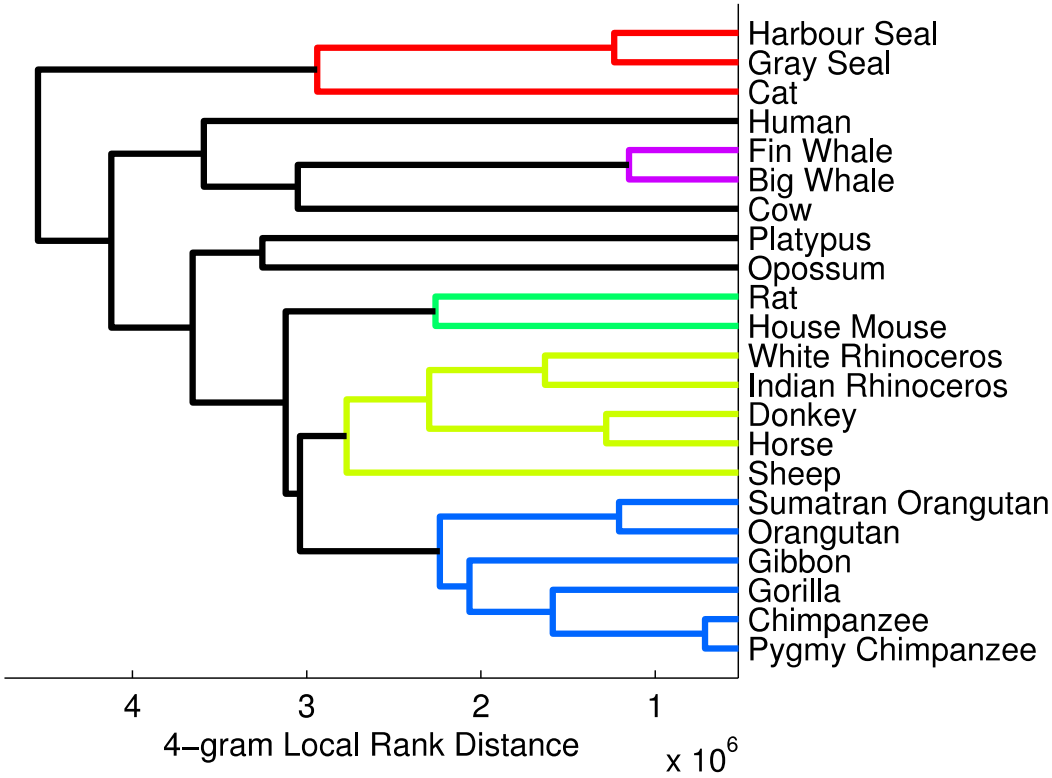
    $$h = \{BAB \to [1,4];\ ABB \to [2];\ BBA \to [3];\ ABA \to [5]\}$$

  - We now take the 3-grams from s and look them up in h:
    1. We find ABA at position 5 in h, so we add $|1 - 5|$ to the distance
    2. We find BAB at positions $[1,4]$ in h
    3. We do a binary search in $[1, 4]$ to find the closest position to 2
    4. We find 1 and we add $|2 - 1|$ to the distance

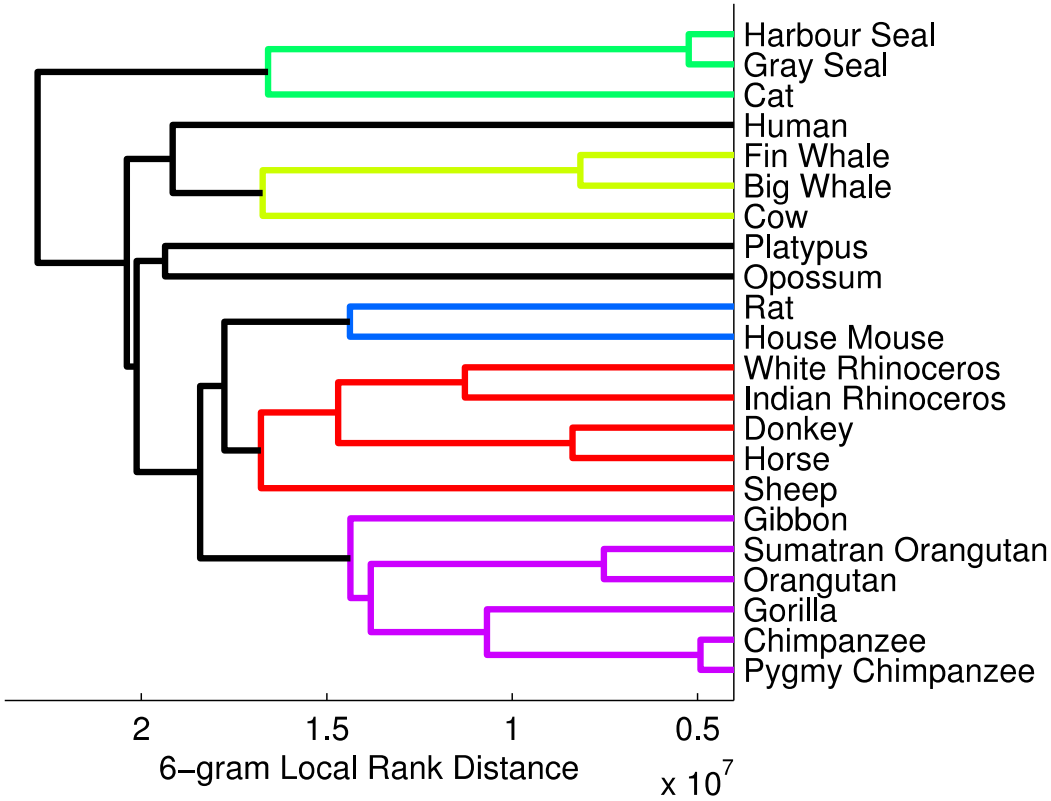
Resulted Dendrograms



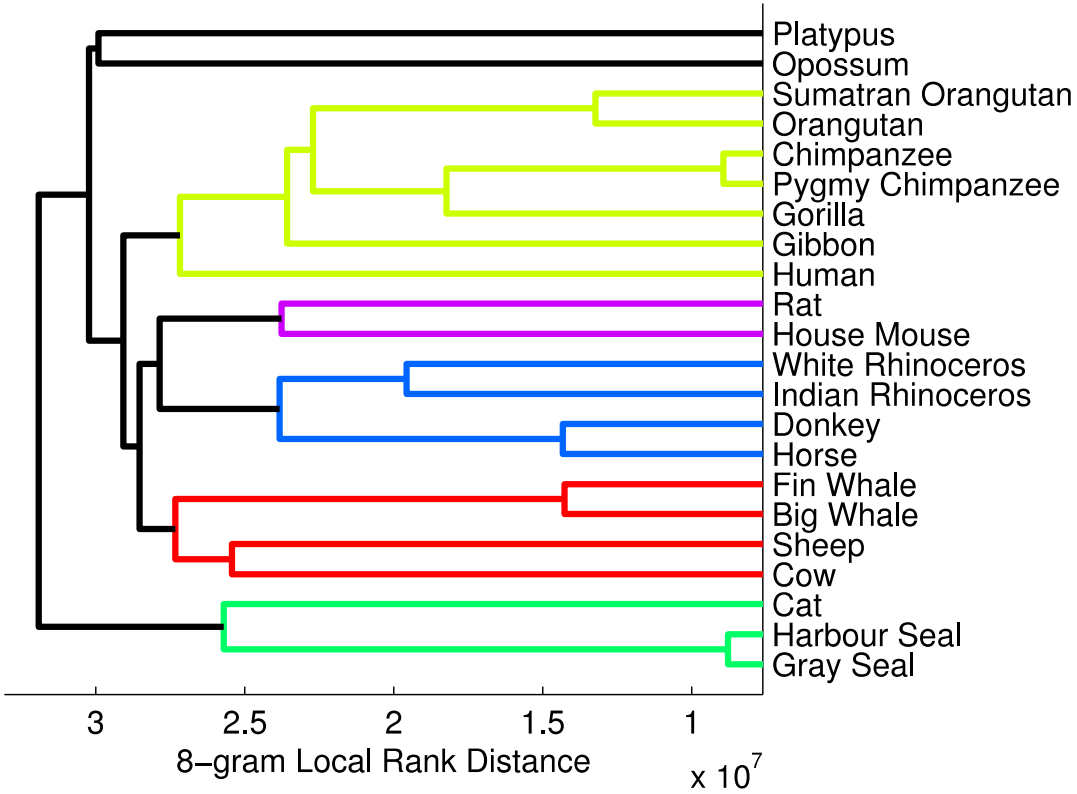
Resulted Dendrograms



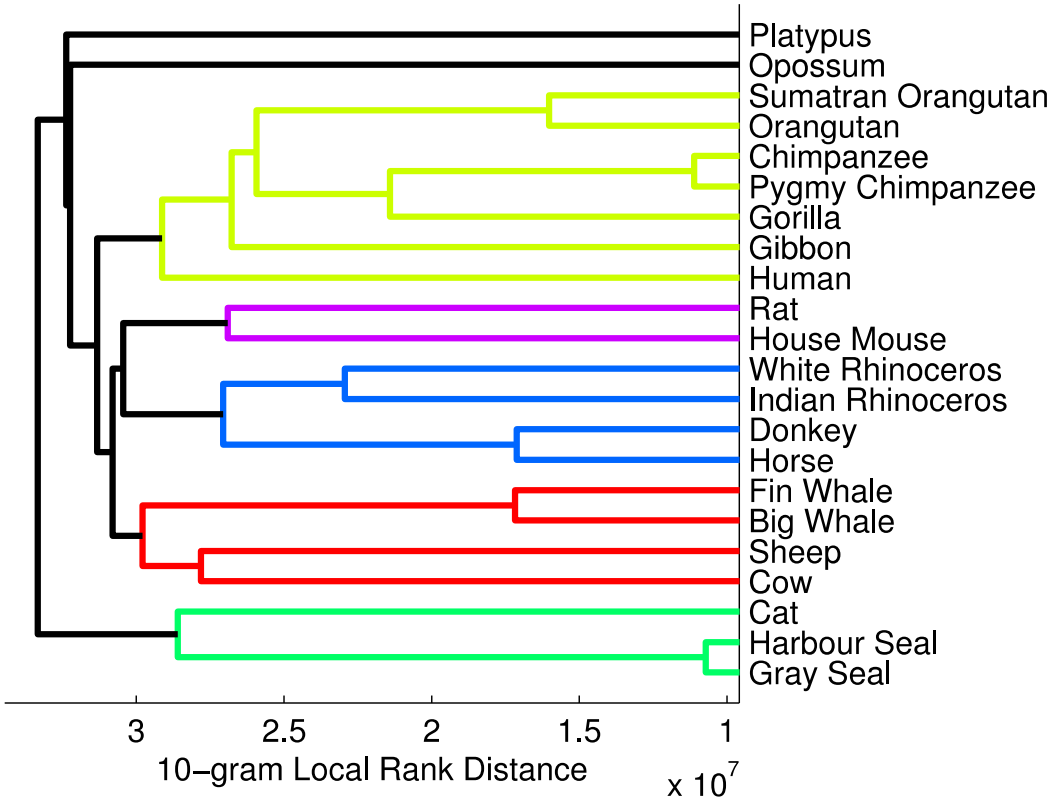
Resulted Dendrograms



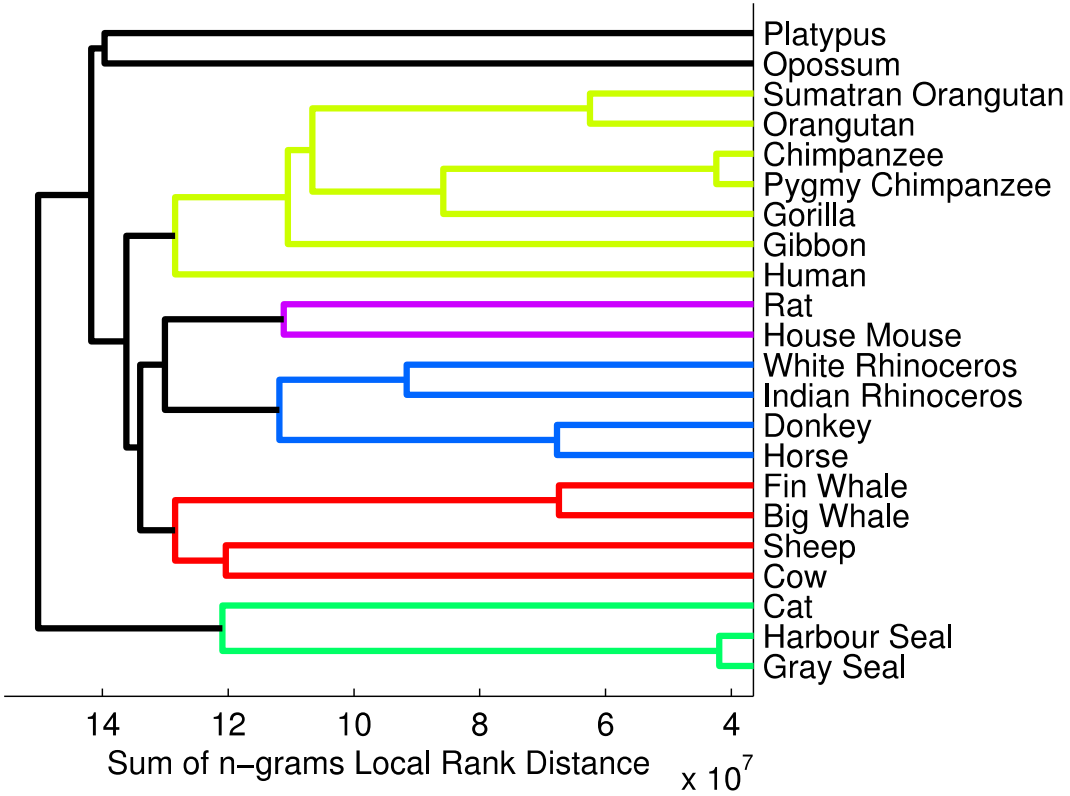
Resulted Dendrograms



Resulted Dendrograms



Resulted Dendrograms



Summary

- **Agglomerative clustering** is a hierarchical clustering algorithm that produces a **dendrogram** of clusters
- It works by starting with each point in its own cluster and by iteratively selecting the two closest clusters and merging them
- The **linkage criterion** (distance between clusters) can be defined in different ways:
  - **single linkage:** considers the distance between closest points
  - **complete linkage:** considers the distance between farthest points
  - **average linkage:** considers the average distance between all pairs of points
  - **Ward's linkage:** considers the increase in variance
- The selected **distance metric** between points and the **linkage criterion** have an impact on the shapes and sizes of the produced clusters!