Introduction to Data Science and Analytics (DSC510)

Data Clustering

Machine learning

- **Supervised:** We are given input/output samples (x, y) which we relate with a function y = f(x). We would like to "learn" f, and evaluate it on new data. Types:
  - **Classification:** y is discrete (class labels).
  - **Regression:** y is continuous, e.g. linear regression.

- **Unsupervised:** Given only samples x of the data, we compute a function f such that y = f(x) is a "simpler" representation.
  - Discrete y: **clustering**
  - Continuous y: **dimensionality reduction** (e.g., matrix factorization, unsupervised neural networks)

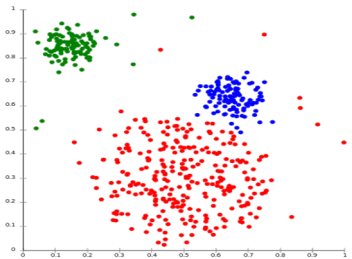
The clustering problem

Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, such that

- members of a cluster are close (i.e., similar) to each other
- members of different clusters are far apart from each other

Usually:

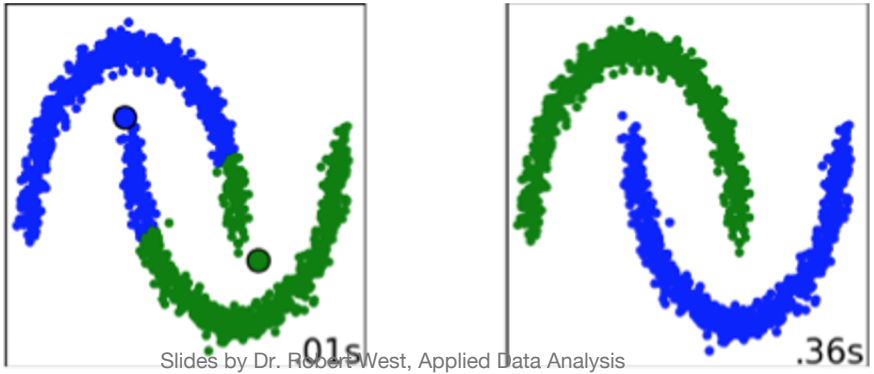
- Points are in a high-dimensional space
- Similarity is defined via a distance measure
  - Euclidean, cosine, Jaccard, edit distance, …



Characteristics of clustering methods

Quantitative: scalability (many samples), dimensionality (many features)

Qualitative: types of features (numerical, categorical, etc.), type of shapes (polyhedra, hyperplanes, manifolds, etc.)

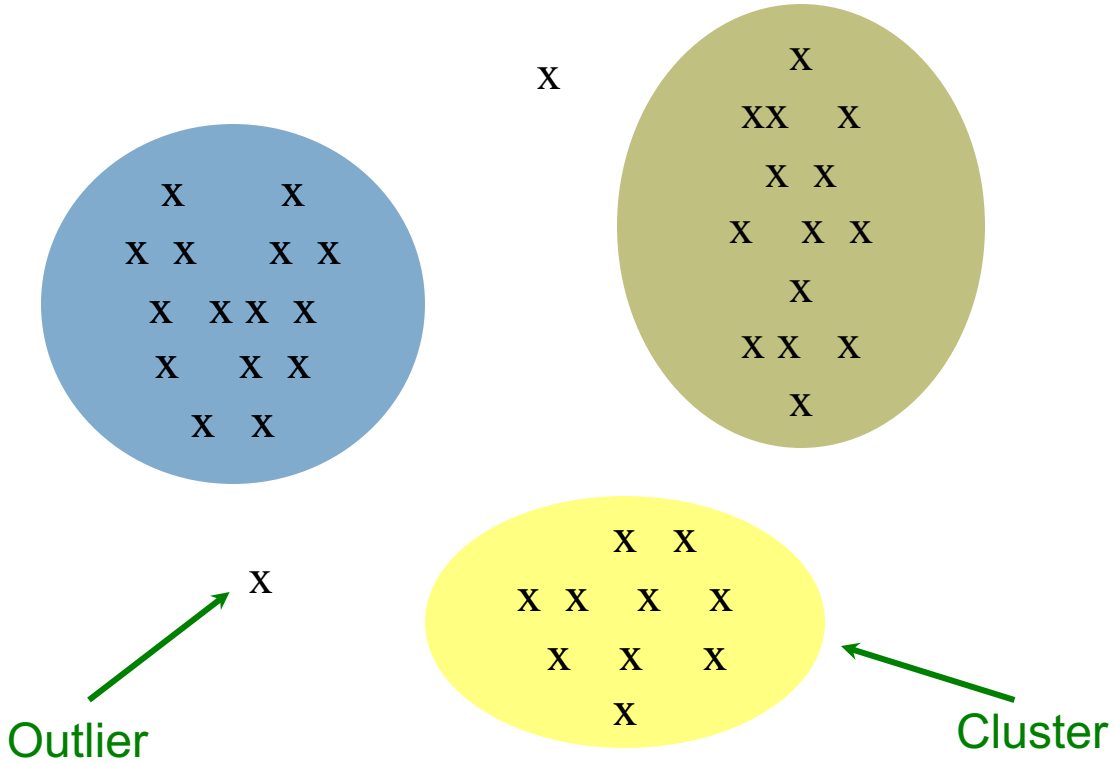


Characteristics of clustering methods

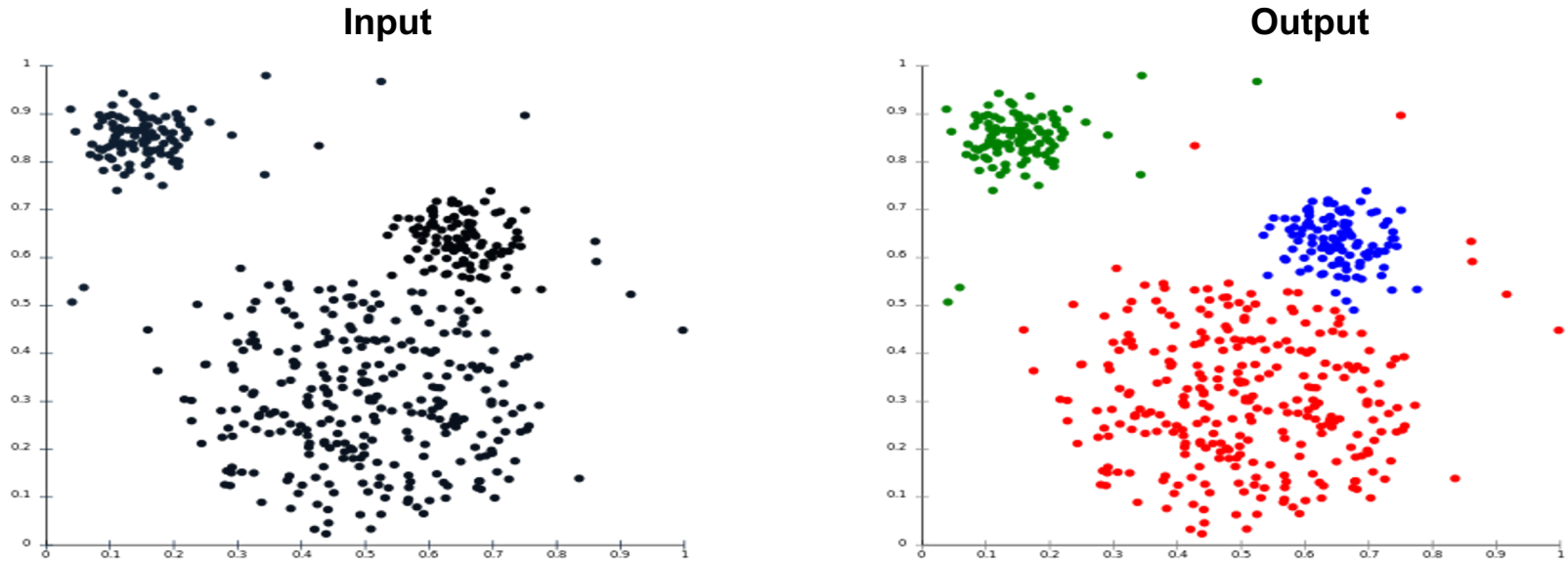
Robustness: sensitivity to noise and outliers, sensitivity to the processing order

User interaction: incorporation of user constraints (e.g., number of clusters, max size of clusters), interpretability and usability

Example: clusters & outliers



A typical clustering example



Note: Above is 2D; real scenarios often much more high-dimensional, e.g., 10,000-dimensional for 100x100 images.

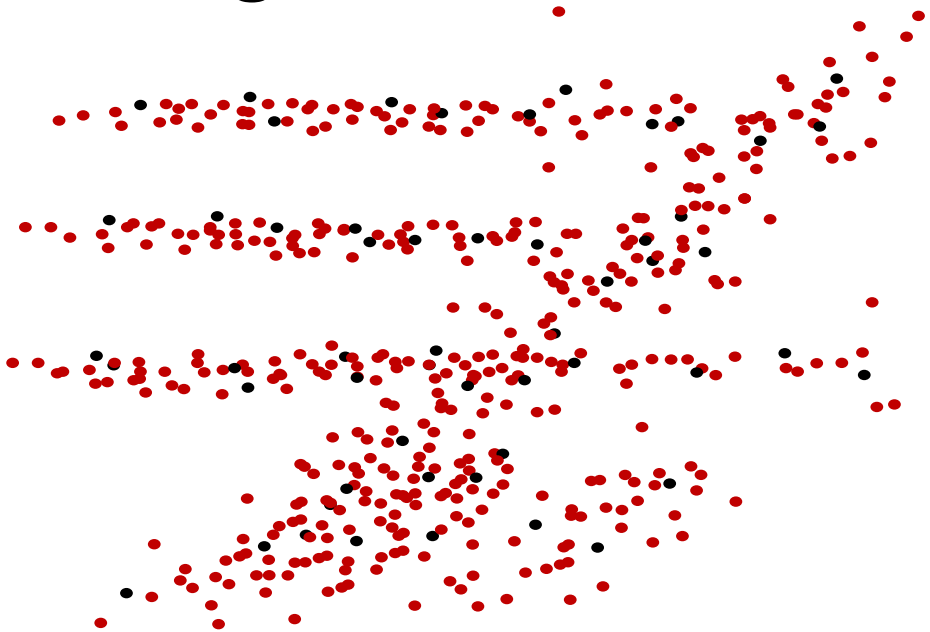
Some use cases for clustering

- Data exploration (especially for high-dimensional data, where visualization fails)
- Partitioning of data for more fine-grained subsequent analysis
- Marketing: building personas
- Supporting data labeling for supervised learning
- Data compression (next slide)
- …

Clustering for segmentation

Task: break an image into regions of points with similar features (Brox and Malik, ECCV 2010).

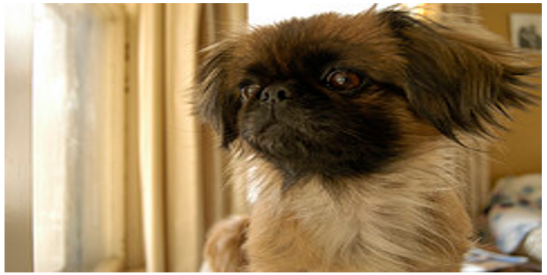
Clustering for condensation/compression



Here we don’t require that clusters extract meaningful structure, but that they give a coarse-grained version of the data.

Beware of “cluster bias”!

- Human beings conceptualize the world through categories represented as *exemplars* (Rosch 1973, Estes 1994).

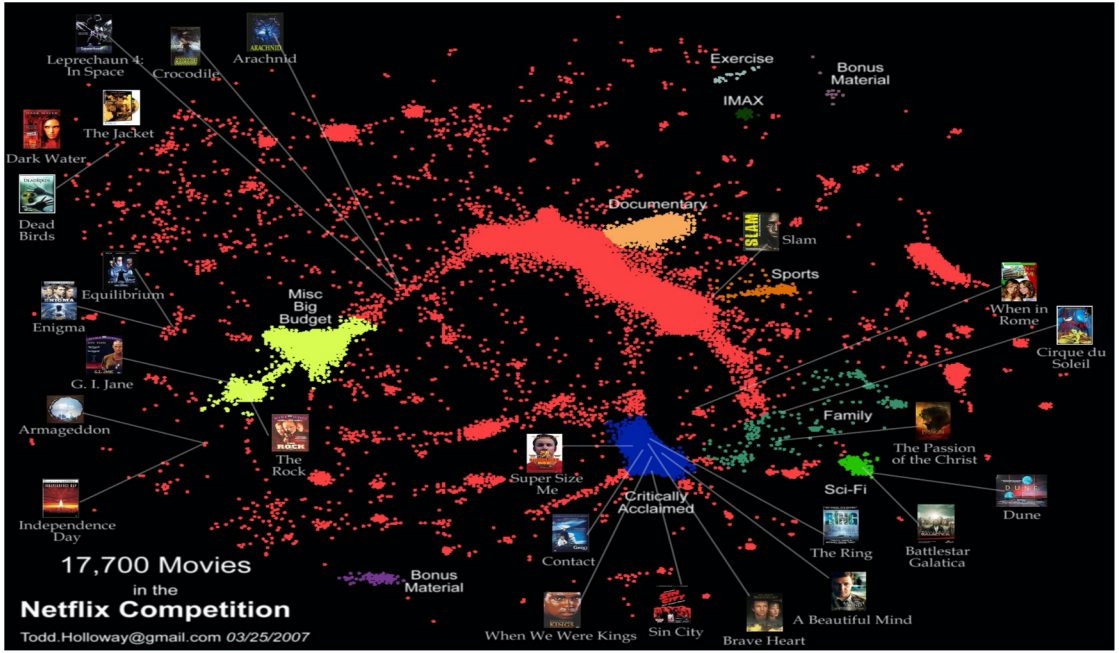




- We tend to see cluster structure whether it is there or not.
- Works well for dogs, but…

“Cluster bias”

- **Clustering is used more than it should be**, because people assume an underlying domain has discrete classes in it
  - Especially true for characteristics of people, e.g., Myers-Briggs personality types like “ENTP”.
- In reality the underlying data is usually **continuous**.
- In such cases, continuous models (e.g., matrix factorization, “soft” clustering, k-NN) tend to do better (cf. next slide)

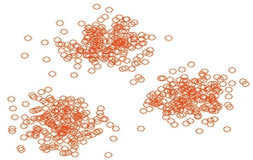
Netflix



- More of a continuum than discrete clusters
- Other methods (e.g., matrix factorization, k-NN) may do better than discrete cluster models

Terminology

- **Hierarchical clustering:** clusters form a tree-shaped hierarchy. Can be computed bottom-up or top-down.
- **Flat clustering:** no inter-cluster structure.



- **Hard clustering:** items assigned to a unique cluster.
- **Soft clustering:** cluster membership is a probability distribution over all clusters

Clustering is a hard problem!

Why is it hard?

Clustering in 2 dimensions looks easy

Clustering small amounts of data looks easy

And in these special cases, it actually is often easy, but...

… many applications involve not 2, but 10 or 10,000 dimensions (and large amounts of data)

High-dimensional spaces look different: Almost all pairs of points are at about the same distance

("curse of dimensionality")

Clustering problem: galaxies

A catalog of 2 billion “sky objects” represents objects by their radiation in 7 dimensions (frequency bands)

Problem: Cluster into similar objects, e.g., galaxies, nearby stars, quasars, etc.

Sloan Digital Sky Survey [link]

Clustering problem: music CDs

Intuitively: Music divides into categories, and customers prefer a few categories

- But what are categories really?
- —> take a data-driven approach!

Represent a CD by a set of customers who bought it ("collaborative filtering")
Similar CDs have similar sets of customers, and vice-versa

Clustering problem: music CDs

Space of all CDs:

Think of a space with one dimension for each customer

- Values in a dimension may be 0 or 1 only
- A CD is a point in this space $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i-th customer bought the CD

For Amazon, the dimensionality is tens of millions

Task: Find clusters of similar CDs

Clustering problem: documents

Finding topics:

Represent a document by a vector (x_1, x_2,…, x_k), where x_i = 1 iff the *i*-th word appears in the document (in any position)

Idea: documents with similar sets of words are about same topic

Cosine, Jaccard, Euclidean distances

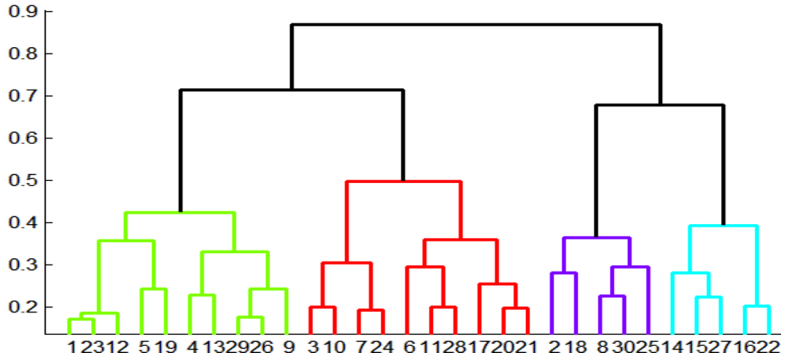
In both examples (CDs, documents) we have a choice when we thinking of data points as sets of features (users, words):

- **Sets as vectors:**
  - Measure similarity by **Euclidean distance**
  - Measure similarity by the **cosine distance**
- **Sets as sets:** Measure similarity by the **Jaccard distance**

Overview: Methods of clustering

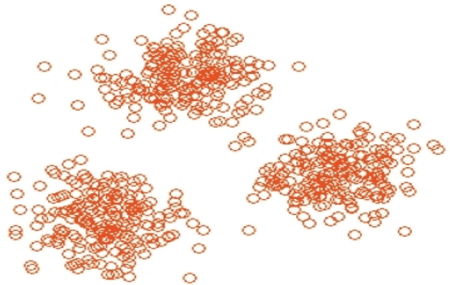
Hierarchical:

- **Agglomerative** (bottom up):
  - Initially, each point is a cluster
  - Repeatedly combine the two "nearest" clusters into one
- **Divisive** (top down):
  - Start with one cluster and recursively split it



Point assignment:

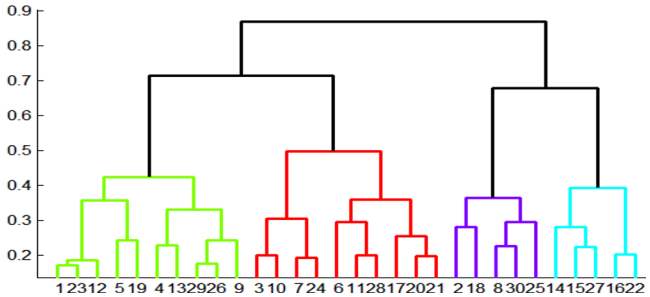
- Maintain a set of clusters
- Points belong to "nearest" cluster



Agglomerative hierarchical clustering

Key operation:

Repeatedly combine two nearest clusters



Three important questions:

- **1)** How to represent a cluster of more than one point?
- **2)** How to determine the "nearness" of clusters?
- **3)** When to stop combining clusters?

Agglomerative hierarchical clustering

Key operation: Repeatedly combine two nearest clusters

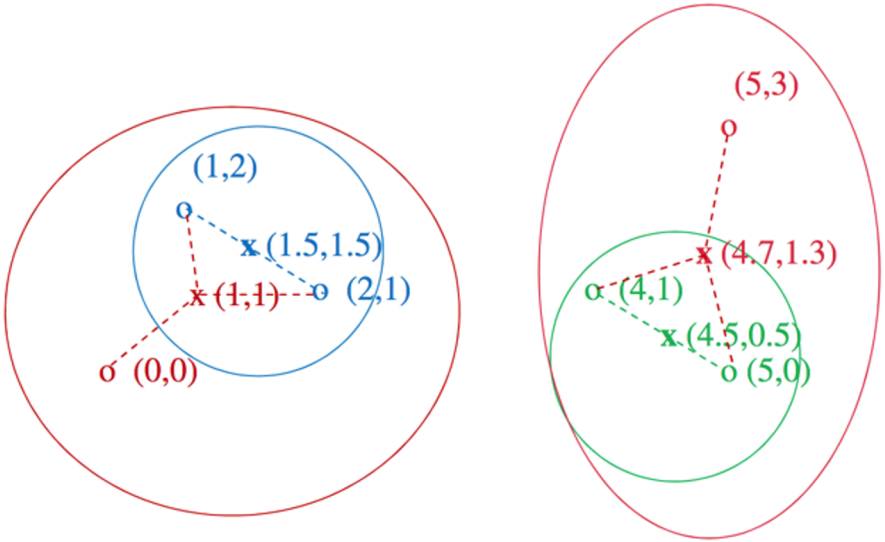
(1) How to represent a cluster of many points?

- Euclidean case: each cluster has a **centroid** = average of its points
- What about non-Euclidean case?

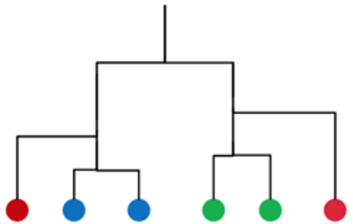
(2) How to determine "nearness" of clusters?

- Euclidean case: measure cluster distances by distances of centroids
- What about non-Euclidean case?

Example: Hierarchical clustering



Data:
o … data point
x … centroid



Dendrogram

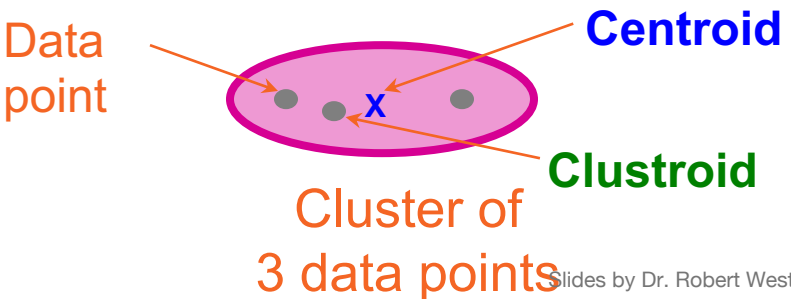
Non-Euclidean case: clustroids

(1) How to represent a cluster of many points?

clustroid = point **"closest"** to other points

Possible meanings of "closest":

- Smallest maximum distance to other points
- Smallest average distance to other points (a.k.a. **medoid**)
- Smallest sum of squares of distances to other points



Centroid is the avg. of all (data)points in the cluster. This means centroid is an "artificial" point.

Clustroid is an **existing** (data)point that is "closest" to all other points in the cluster.

Defining “nearness” of clusters

(2) How do you determine the “nearness” of clusters?

- **Approach 1:**

  Intercluster distance = minimum of the distances between any two points, one from each cluster; or average of distances; or distance between centroids/clustroids; etc.

- **Approach 2:**

  Pick a notion of “**cohesion**” (“tightness”) of clusters

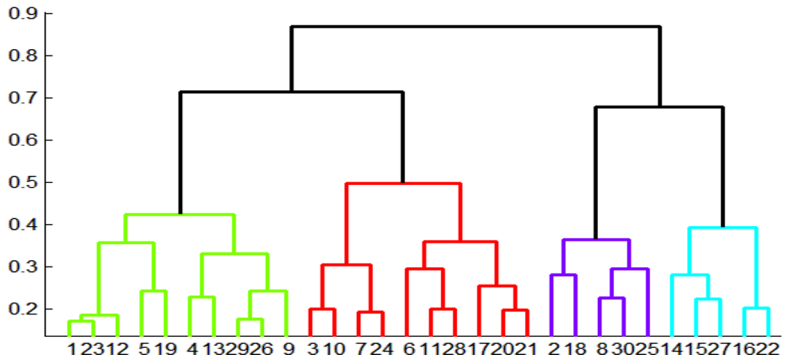
  - Nearness of clusters = cohesion of their *union*

Cohesion

Approach 2.1: Use the **diameter** of the merged cluster = maximum distance between points in the cluster

Approach 2.2: Use the **average distance** between points in the cluster

How many branching points are there in a dendrogram for a dataset with *N* data points?



Implementation

Naïve implementation of hierarchical clustering:

- At each step, compute pairwise distances between all pairs of clusters, then merge
- $O(N^3)$, where N is the number of data points

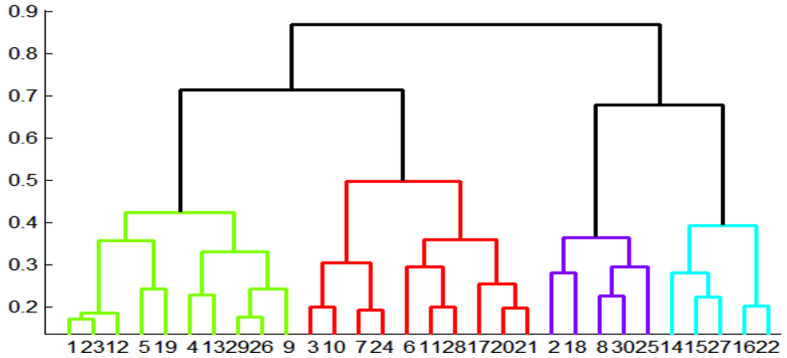
Careful implementation using priority queue can reduce time to $O(N^2 \log N)$

- Still too expensive for really big datasets that do not fit in memory

Overview: Methods of clustering

Hierarchical:

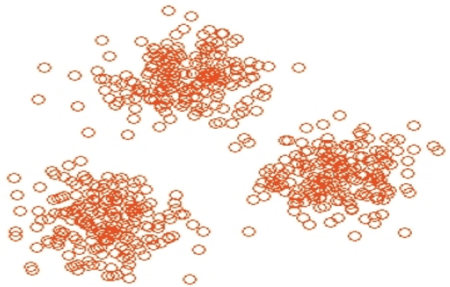
- **Agglomerative** (bot
  - Initially, each cluster
  - peatedly e the two
  - est” rs into one
- **Divi** own):
  - Start one cluster and recursively split it

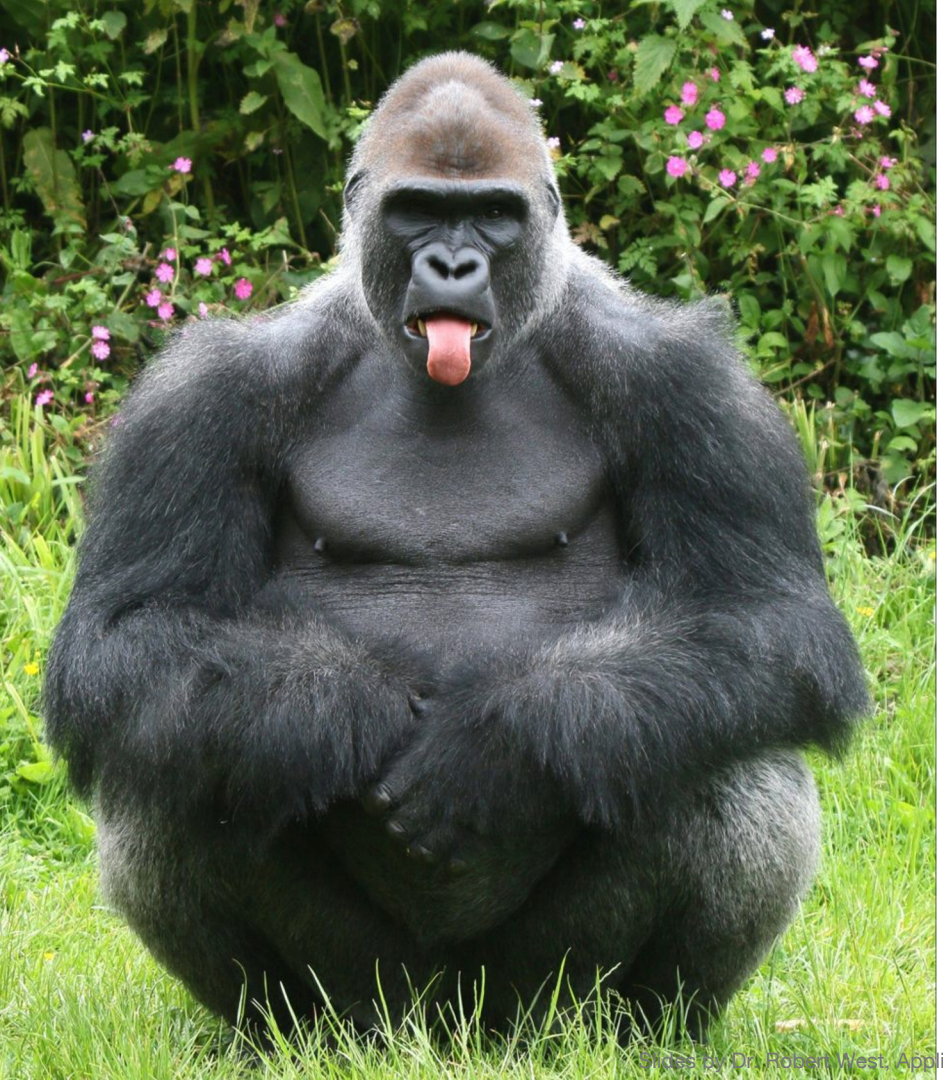


Point assignment:

- Maintain a set of clusters
- Points belong to “nearest” cluster

NEXT





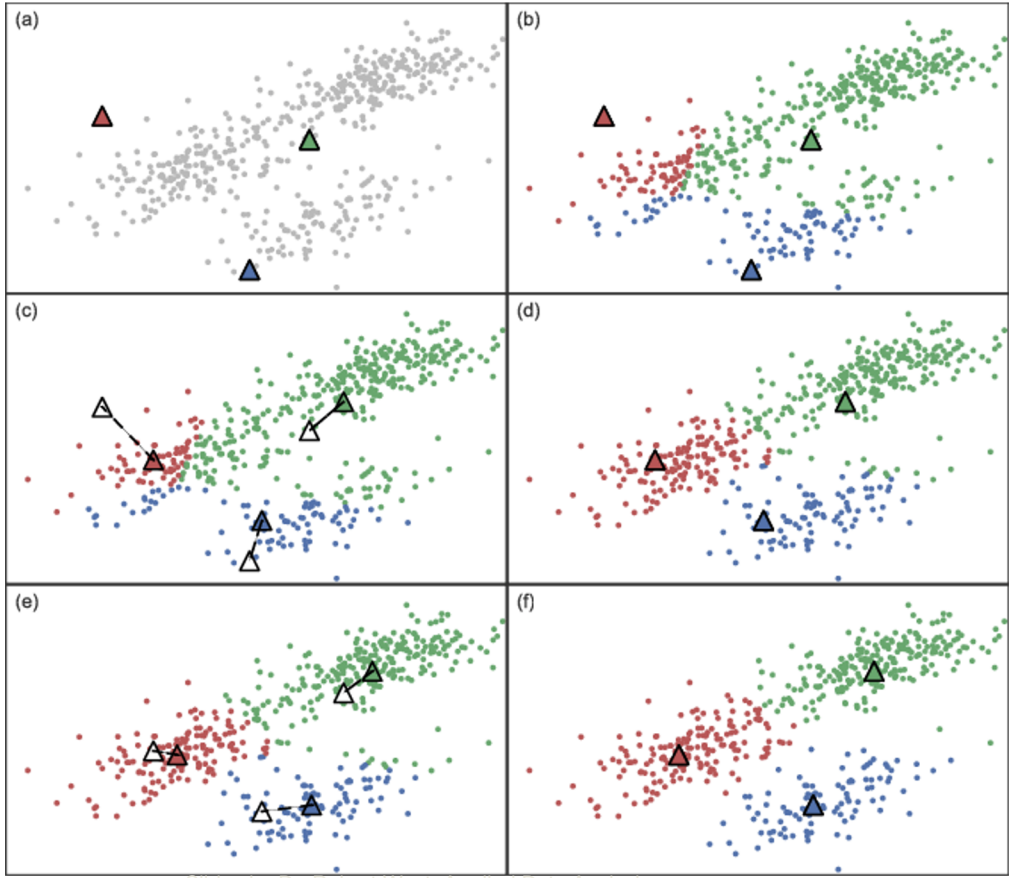
K-means

The gorilla among the point-assignment clustering algorithms

K-means clustering

- Goal: assign each data point to one of *k* clusters such that the total distance of points to their centroids is minimized
- Solved by a simple greedy algorithm (Lloyd's algorithm):
- Locally minimize the "distance" (usually squared Euclidean distance) from data points to their respective centroids:
  - **Find the closest cluster centroid** for each item, and assign it to that cluster.
  - **Recompute the cluster centroid** (the mean of items in the cluster) for each cluster.

K-means clustering



K-means clustering

How long to iterate?

- For fixed number of iterations
- or until no change in assignments
- or until only small change in cluster “tightness” (sum of [squared] distances from points to centroids)

K-means initialization

We need to pick some points for the first round of the algorithm:

- **Random sample:** Pick a random subset of k points from the dataset.
- **K-Means++:** Iteratively construct a random sample with good spacing across the dataset.

Note: Finding an optimal k-means clustering is NP-hard. The above help avoid bad configurations.

K-means++ [link]

Start: Choose first cluster center at random from the data points

Iterate:

- For every remaining data point x, compute the distance D(x) from x to the closest previously selected cluster center.
- Choose a remaining point x randomly with probability proportional to $D(x)^2$, and make it a new cluster center.

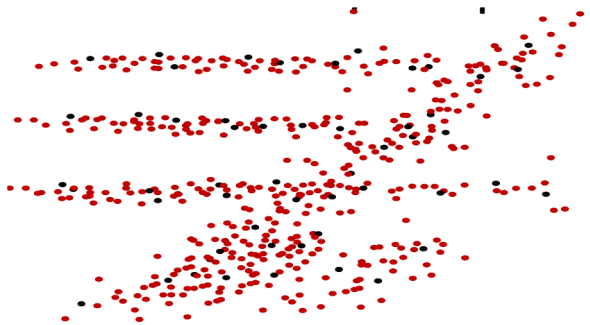
Intuitively, this finds a sample of widely-spaced points from dense regions of the data space, avoiding "collapsing" of the clustering into a few internal centers.

K-means++

- The exact algorithm is as follows:
  1. Choose one center uniformly at random among the data points.
  2. For each data point *x* not chosen yet, compute D(*x*), the distance between *x* and the nearest center that has already been chosen.
  3. Choose one new data point at random as a new center, using a weighted probability distribution where a point *x* is chosen with probability proportional to $D(x)^2$.
  4. Repeat Steps 2 and 3 until *k* centers have been chosen.
  5. Now that the initial centers have been chosen, proceed using standard *k*-means clustering.

K-means properties

- It's a greedy algorithm with random setup – **solution isn't optimal** and varies significantly with different initial points.
- Very simple convergence proofs.
- **Performance is O(nk) per iteration** — not bad, and can be heuristically improved.
  n = number of points in the dataset, k = number clusters
- Many variants, e.g.
  - Fixed-size clusters
  - Soft clustering
- Works well for data condensation/compression.



K-means drawbacks

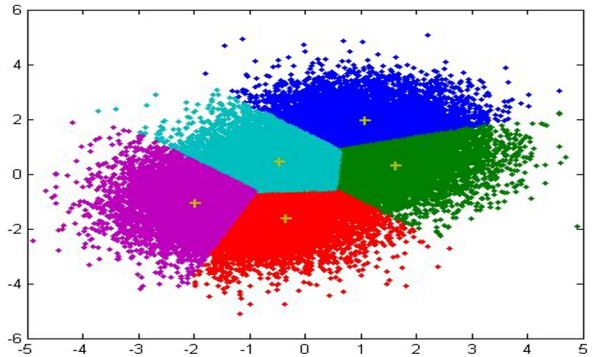
Often terminates at a **local optimum** (mitigated by smart initialization such k-means++, or by re-running multiple times with different initializations)

Need a notion of **mean**

Need to specify **k** (number of clusters) in advance

Doesn't handle **noisy data and outliers** well

Clusters **only** have **convex shapes**



How to choose k?

Run k-means for k = 1, 2, 3, …

For each data point i, compute "silhouette" $s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$

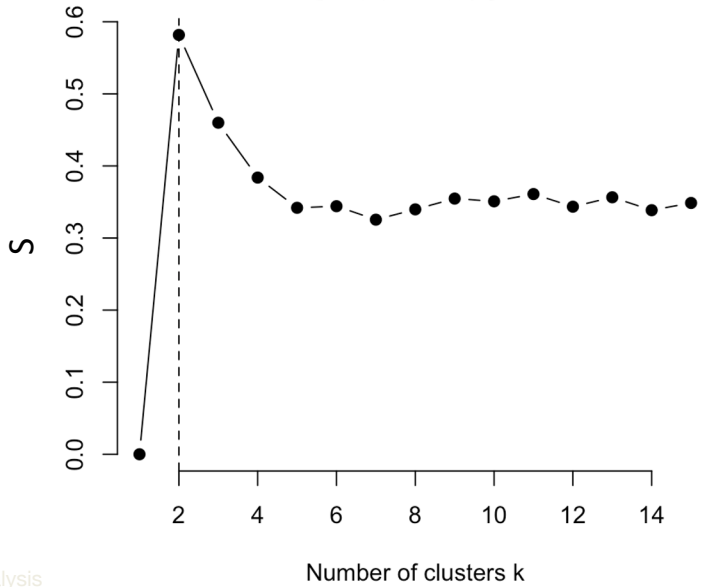
S = average of s(i) over all i

Plot S against k

Pick k for which S is greatest

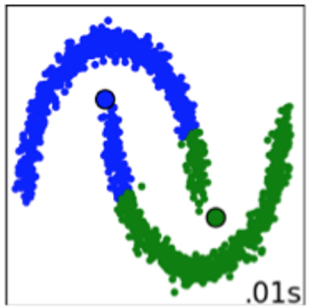
b(i): avg. distance to points in closest other cluster

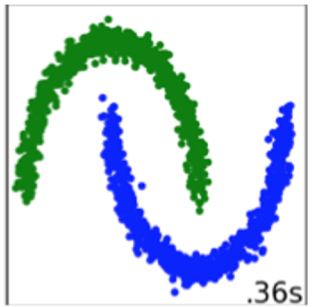
a(i): avg. distance to points in own cluster



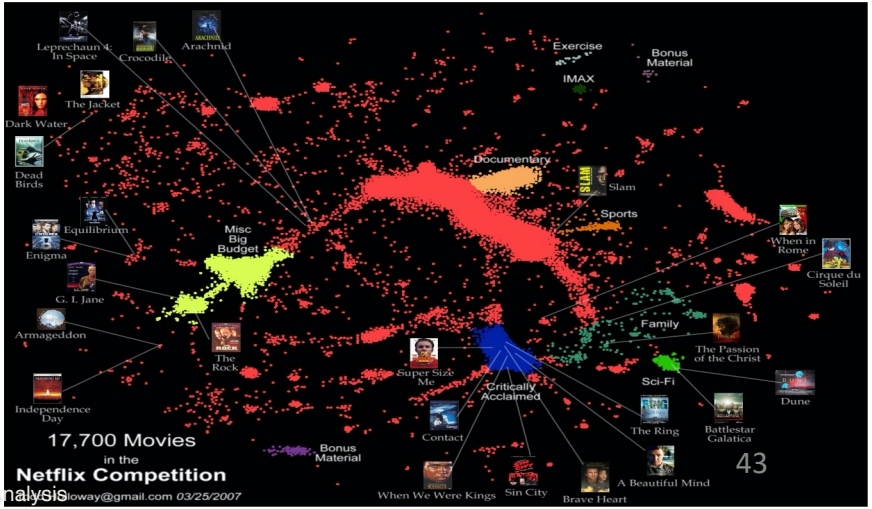
DBSCAN

- "**D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise"
- Motivation: Centroid-based clustering methods like k-means favor clusters that are spherical, and have great difficulty with anything else



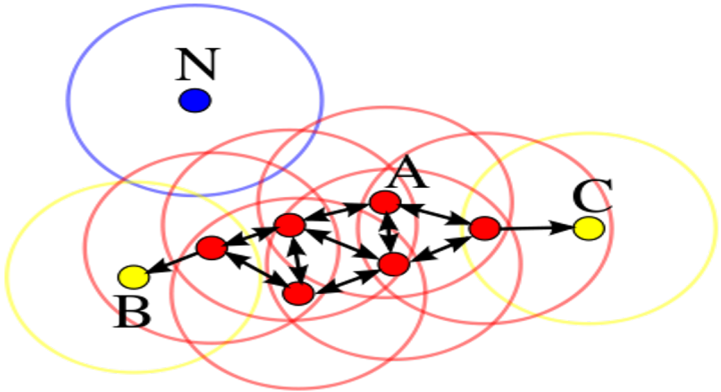


- But with real data we have:



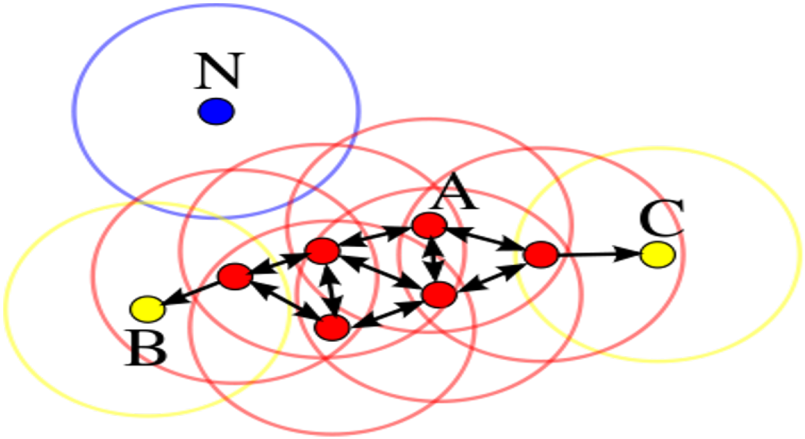
DBSCAN

- DBSCAN performs density-based clustering, and follows the shape of dense neighborhoods of points.



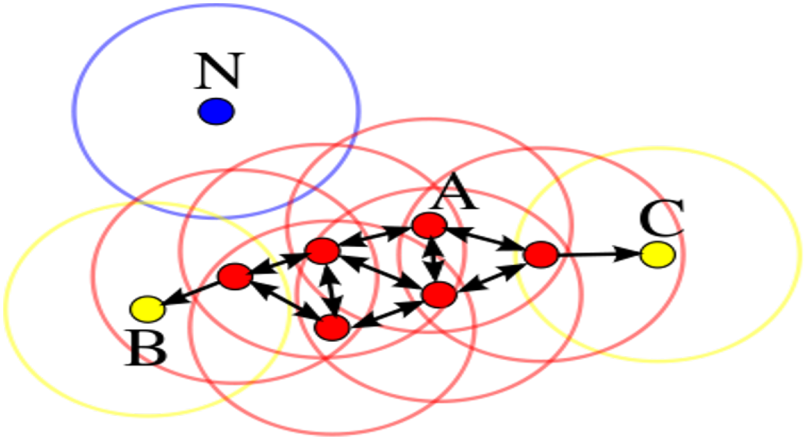
- **Def.:** **core points** have at least *minPts* neighbors in a sphere of diameter ε around them.
- The red points here are core points with at least *minPts* = 3 neighbors in an ε-sphere around them.

DBSCAN

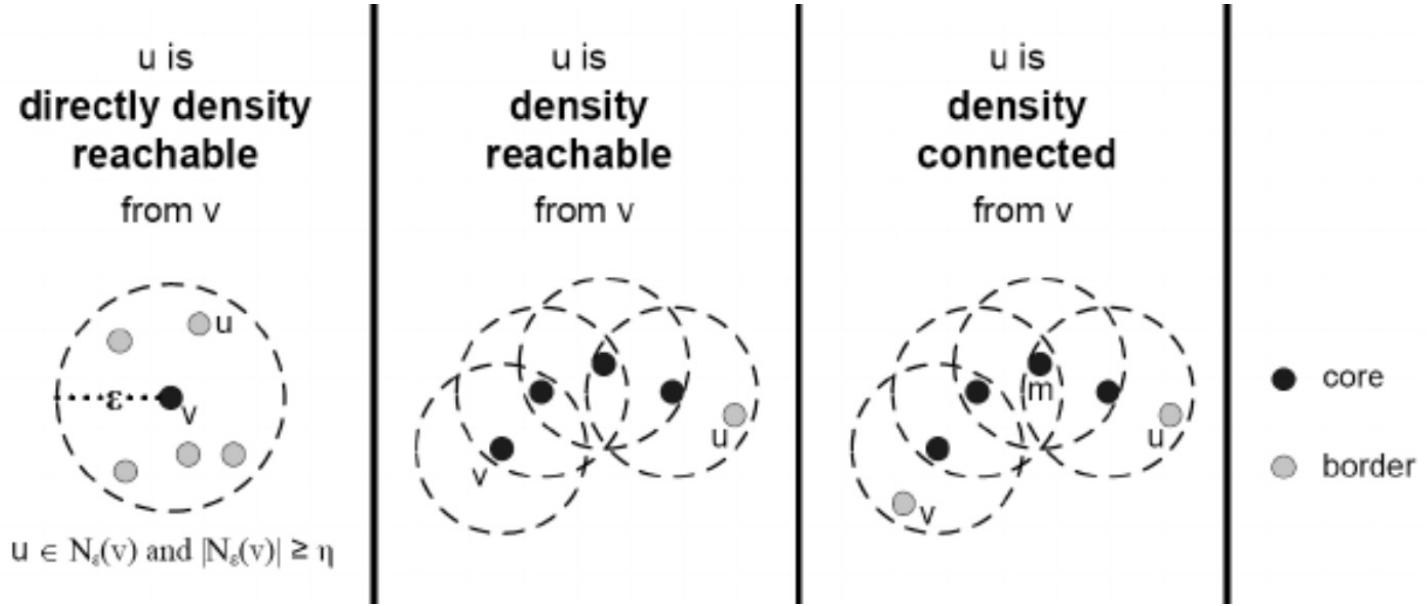


- More **definitions (!):**
  - Core points can **directly reach** neighbors in their ε-sphere
  - From non-core points, no other points can be reached
  - Point *q* is **density-reachable** from *p* if there is a series of points $p = p_1, \ldots, p_n = q$ such that p_{i+1} is directly reachable from p_i
  - All points not density-reachable from any other points are **outliers**

DBSCAN clusters



- Even more **definitions:**
  - Points *p*, *q* are **density-connected** if there is a point *o* such that both *p* and *q* are density-reachable from *o*.
  - A **cluster** is a set of points which are **mutually density-connected**.
  - That is, if a point is density-reachable from a cluster point, it is part of the cluster as well.
  - In the above figure, red points are mutually density-reachable; B and C are density-connected; N is an outlier.

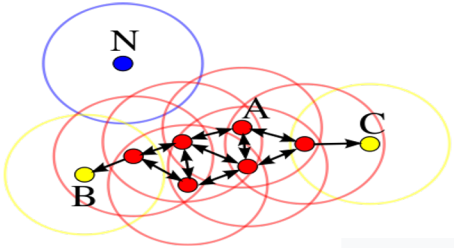

u is
directly density
reachable
from v
u
ε
v
u ∈ N_ε(v) and |N_ε(v)| ≥ η
u is
density
reachable
from v
v
u
u is
density
connected
from v
m
u
v
core
border

DBSCAN clusters

- The DBSCAN algorithm can be abstracted into the following steps:[4]
  1. Find the points in the ε (eps) neighborhood of every point, and identify the core points with more than minPts neighbors.
  2. Find the connected components of *core* points on the neighbor graph, ignoring all non-core points.
  3. Assign each non-core point to a nearby cluster if the cluster is an ε (eps) neighbor, otherwise assign it to noise.
- A naive implementation of this requires storing the neighborhoods in step 1, thus requiring substantial memory. The original DBSCAN algorithm does not require this by performing these steps for one point at a time.

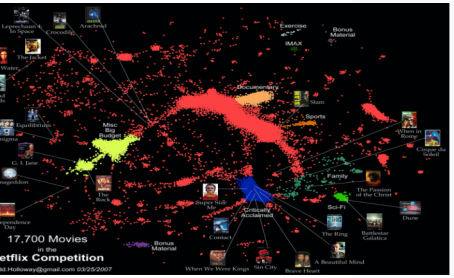
DBSCAN Advantages

- DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means.
- DBSCAN can find arbitrarily-shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced.
- DBSCAN has a notion of noise, and is robust to outliers.
- DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.)
- DBSCAN is designed for use with databases that can accelerate region queries, e.g. using an R* tree.
- The parameters minPts and ε can be set by a domain expert, if the data is well understood.



DBSCAN algorithm

```
DBSCAN(DB, dist, eps, minPts) {
   C = 0                                                  /* Cluster counter */
   for each point P in database DB {
      if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
      Neighbors N = RangeQuery(DB, dist, P, eps)          /* Find neighbors */
      if |N| < minPts then {                              /* Density check */
         label(P) = Noise                                 /* Label as Noise */
         continue
      }
      C = C + 1                                           /* next cluster label */
      label(P) = C                                        /* Label initial point */
      Seed set S = N \ {P}                                /* Neighbors to expand */
      for each point Q in S {                             /* Process every seed point */
         if label(Q) = Noise then label(Q) = C            /* Change Noise to border point */
         if label(Q) ≠ undefined then continue            /* Previously processed */
         label(Q) = C                                     /* Label neighbor */
         Neighbors N = RangeQuery(DB, dist, Q, eps)       /* Find neighbors */
         if |N| ≥ minPts then {                           /* Density check */
            S = S ∪ N                                     /* Add new neighbors to seed set */
         }
      }
   }
}
```



DBSCAN performance

- DBSCAN uses all-pairs point distances, but using an efficient indexing structure, each RangeQuery (for finding neighbors within ε-sphere) takes only O(log n) time
- The algorithm overall can be made to run in **O(n log n)**
- Fast neighbor search becomes progressively harder (higher constants) in higher dimensions

Model-based clustering

- Assume data generated from **k** probability distributions
- ***Goal:*** find the distribution parameters
- ***Algorithm:*** Expectation Maximization (EM)
- ***Output:*** Distribution parameters and a **soft** assignment of points to clusters

Model-based clustering

- Assume **k** probability distributions with parameters: $(\theta_1,\ldots,\theta_k)$
- Given data **X**, compute $(\theta_1,\ldots,\theta_k)$ such that $Pr(X|\theta_1,\ldots,\theta_k)$ [likelihood] or $ln(Pr(X|\theta_1,\ldots,\theta_k))$ [loglikelihood] is maximized.
- Every point $x \epsilon X$ need not be generated by a single distribution but it can be generated by multiple distributions with some probability [soft clustering]

Expectation-maximization algorithm

- Iterative procedure to compute the ***Maximum Likelihood (ML)*** estimate – even in the presence of missing or hidden data
- **EM** consists of two steps:
  - **Expectation step:** the (missing) data are estimated given the observed data and current estimates of model parameters
  - **Maximization step:** The likelihood function is maximized under the assumption that the (missing) data are known

EM Algorithm

- Initialize **k** distribution parameters **(θ_1,…, θ_k)**; Each distribution parameter corresponds to a cluster center

- Iterate between two steps

  - **E**xpectation step: (probabilistically) assign points to clusters

  - **M**aximation step: estimate model parameters that maximize the likelihood for the given assignment of points

EM Algorithm

- Initialize **k** cluster centers
- Iterate between two steps
  - **E**xpectation step: assign points to clusters

$$\Pr(x_i \in C_k) = \Pr(x_i \mid C_k) \Big/ \sum_j \Pr(x_i \mid C_j)$$

$$w_k = \frac{\sum_i \Pr(x_i \in C_k)}{n}$$

  - **M**aximation step: estimate model parameters

$$r_k = \frac{1}{n} \sum_{i=1}^{n} \frac{\Pr(x_i \in C_k)}{\sum_k \Pr(x_i \in C_j)}$$

What Is A Good Clustering?

- Internal criterion: A good clustering will produce high quality clusters in which:
  - the intra-class (that is, intra-cluster) similarity is high
  - the inter-class similarity is low
  - The measured quality of a clustering depends on both the document representation and the similarity measure used

Cluster Validity

- For cluster analysis, the question is how to evaluate the “goodness” of the resulting clusters?
- But “clusters are in the eye of the beholder”!
- Then why do we want to evaluate them?
  - To avoid finding patterns in noise
  - To compare clustering algorithms
  - To compare two sets of clusters
  - To compare two clusters

Different Aspects of Cluster Validation

1. Determining the clustering tendency of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
2. Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels.
3. Evaluating how well the results of a cluster analysis fit the data *without* reference to external information.

   - Use only the data

4. Comparing the results of two different sets of cluster analyses to determine which is better.
5. Determining the 'correct' number of clusters.

For 2, 3, and 4, we can further distinguish whether we want to evaluate the entire clustering or just individual clusters.

External criteria for clustering quality

- Quality measured by its ability to discover some or all of the hidden patterns or latent classes in gold standard data
- Assesses a clustering with respect to ground truth … requires *labeled data*
- Assume documents with C gold standard classes, while our clustering algorithms produce K clusters, $\omega_1, \omega_2, \ldots, \omega_K$ with n_i members.

Evaluating clusters

- Function **H** computes the cohesiveness of a cluster (e.g., smaller values larger cohesiveness)
- Examples of cohesiveness?
- Goodness of a cluster **c** is **H(c)**
- **c** is better than **c'** if **H(c) < H(c')**

Evaluating **clusterings** using cluster cohesiveness?

- For a clustering **C** consisting of **k** clusters $c_1,\ldots,c_k$
- $H(C) = \Phi_i H(c_i)$
- What is Φ ?

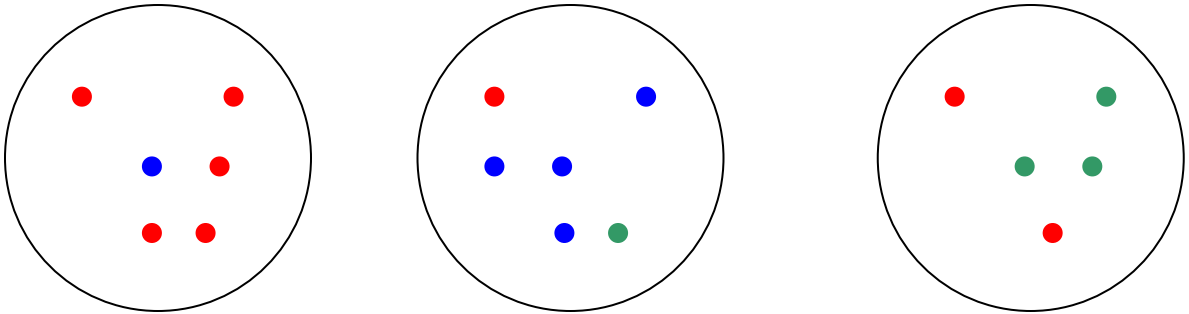
External Evaluation of Cluster Quality

- Simple measure: <u>purity</u>, the ratio between the dominant class in the cluster π_i and the size of cluster ω_i

$$Purity(\omega_i) = \frac{1}{n_i} \max_j (n_{ij}) \quad j \in C$$

- Biased because having *n* clusters maximizes purity
- Others are entropy of classes in clusters (or mutual information between classes and clusters)

Purity example



Cluster I

Cluster II

Cluster III

Cluster I: Purity = 1/6 (max(5, 1, 0)) = 5/6

Cluster II: Purity = 1/6 (max(1, 4, 1)) = 4/6

Cluster III: Purity = 1/5 (max(2, 0, 3)) = 3/5

Cluster separation?

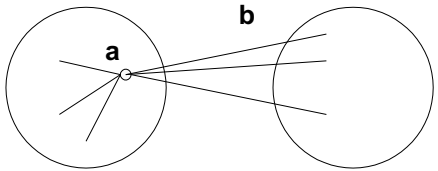
- Function **S** that measures the separation between two clusters c_i, c_j
- Ideas for $S(c_i,c_j)$?
- How can we measure the goodness of a clustering $C = \{c_1,\ldots,c_k\}$ using the separation function **S**?

Silhouette Coefficient

- Silhouette Coefficient combines ideas of both cohesion and separation, but for individual points, as well as clusters and clusterings
- For an individual point, *I*
  - ***a*** = average distance of ***i*** to the points in the same cluster
  - ***b*** = min (average distance of ***i*** to points in another cluster)
  - silhouette coefficient of **i**:

    s = 1 – a/b if **a < b**

  - Typically between 0 and 1.
  - The closer to 1 the better.



- Can calculate the Average Silhouette width for a cluster or a clustering

Perceptron. Consider the following Boolean function:

x_1	x_2	$y = \neg x_1 \cup x_2$
0	0	1
0	1	1
1	0	0
1	1	1

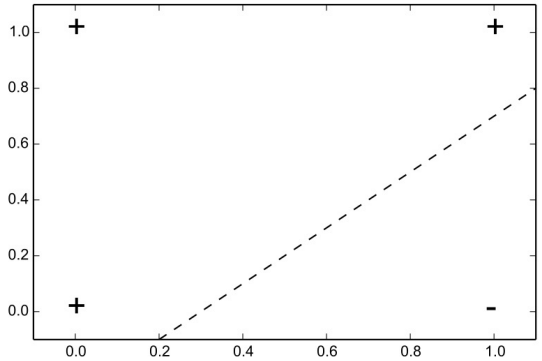
Can this function be represented by a perceptron? Explain your answer.

Perceptron. Consider the following Boolean function:

x_1	x_2	$y = \neg x_1 \cup x_2$
0	0	1
0	1	1
1	0	0
1	1	1

Can this function be represented by a perceptron? Explain your answer.

Solution: Yes, because the function is linearly separable.



If we run K-Means clustering three times, and the generated labels are exactly equal all three times, then the locations of the generated cluster centers are also exactly equal all three times.

◯ True ◯ False

Assuming no two points have the same distance, the cluster labels computed by K-means are always the same for a given dataset.

◯ True ◯ False

Dimensionality reduction can be used as pre-processing for machine learning algorithms like decision trees, neural networks etc.
True False

Making a decision tree deeper will assure better fit but reduce robustness.
True False

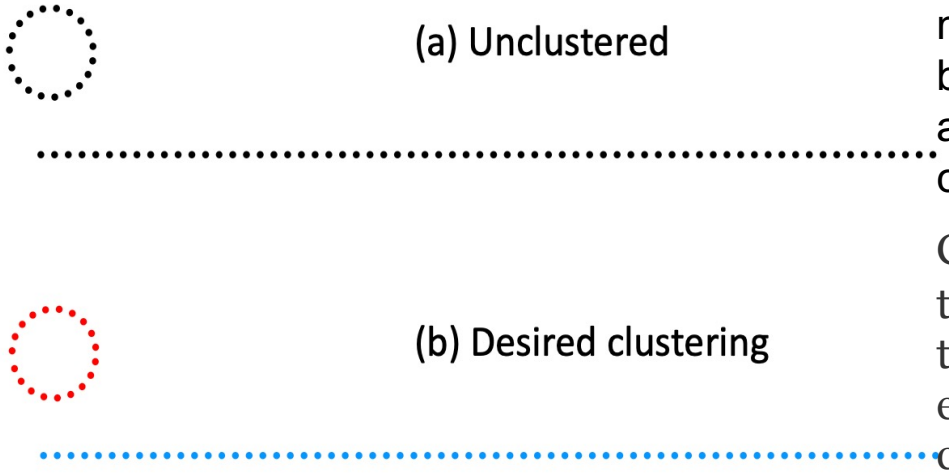
Performing K-nearest neighbors with K = N yields more complex decision boundaries than 1-nearest neighbor.
True False

K-means is a kind of agglomerative clustering
True False

[3 pts] You've just finished training a decision tree for spam classification, and it is getting abnormally bad performance on both your training and test sets. You know that your implementation has no bugs, so what could be causing the problem?

- ◯ Your decision trees are too shallow.
- ◯ You need to increase the learning rate.
- ◯ You are overfitting.
- ◯ All of the above.

[4 pts] Which of the following methods will cluster the data in panel (a) of the figure below into the two clusters (red circle and blue horizontal line) shown in panel (b)? Every dot in the circle and the line is a data point. In all the options that involve hierarchical clustering, the algorithm is run until we obtain two clusters.



Single linkage uses the minimum distance between two clusters as a metric for merging clusters

Complete linkage is the distance between the most distant elements from each class

○ A: Hierarchical agglomerative clustering with Euclidean distance and complete linkage

○ B: Hierarchical agglomerative clustering with Euclidean distance and single linkage

○ C: Hierarchical agglomerative clustering with Euclidean distance and centroid linkage

○ D: k-means clustering with $k = 2$

[3 pts] Averaging the output of multiple decision trees helps _____.

Increase bias

Increase variance

Decrease bias

Decrease variance

[4 pts] Consider the following dataset: $A = (0, 2)$, $B = (0, 1)$ and $C = (1, 0)$. The k-means algorithm is initialized with centers at A and B. Upon convergence, the two centers will be at

A and C

C and the midpoint of AB

A and the midpoint of BC

A and B

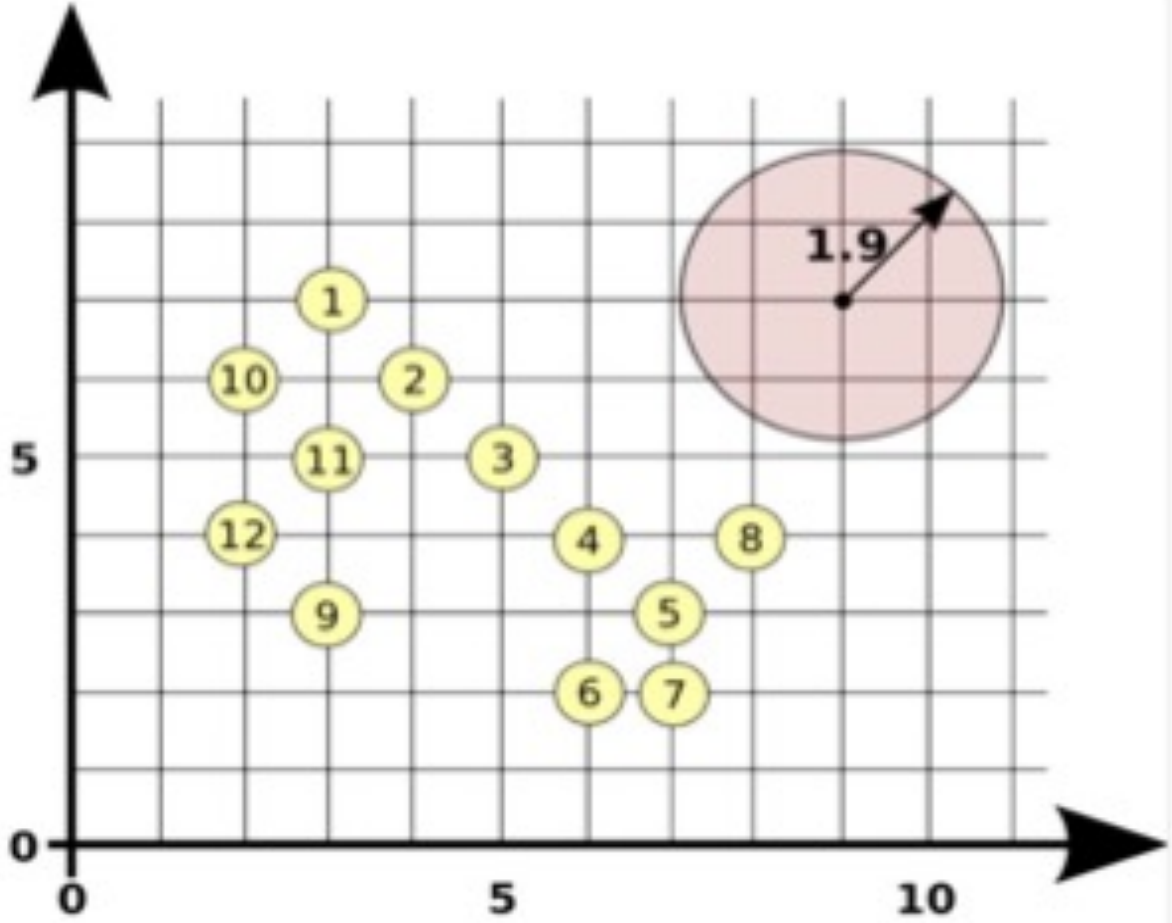
Most machine learning approaches use training sets, test sets and validation sets to derive models. Describe the role each of the three sets plays.

Most machine learning approaches use training sets, test sets and validation sets to derive models. Describe the role each of the three sets plays!

Training set: used to learn the model

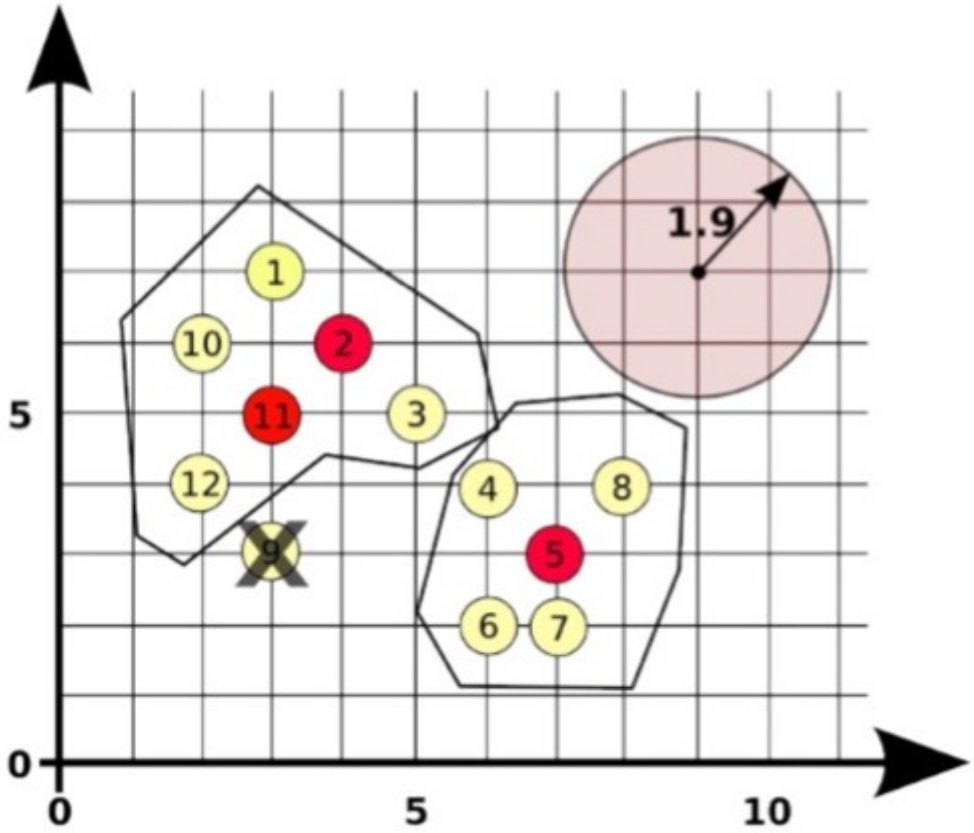
Test set: used to evaluate the model, particularly its accuracy

Validation set: used to determine the "best" input parameter(s) for the algorithm which learns the model; e.g. parameters which control the degree of pruning of a decision tree learning algorithm.



Apply DBSCAN Algorithm with radius 1.9 and MinPts=4 (3 neighbors + the point we are considering as center for computing the density).

1) Indicate if a point is a *core, border* or *noise* point.
2) Indicate the clusters obtained



Red points are *cores*, yellow points are *borders*. *Noise points* are eliminated.