THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

COMP4434 Big Data Analytics

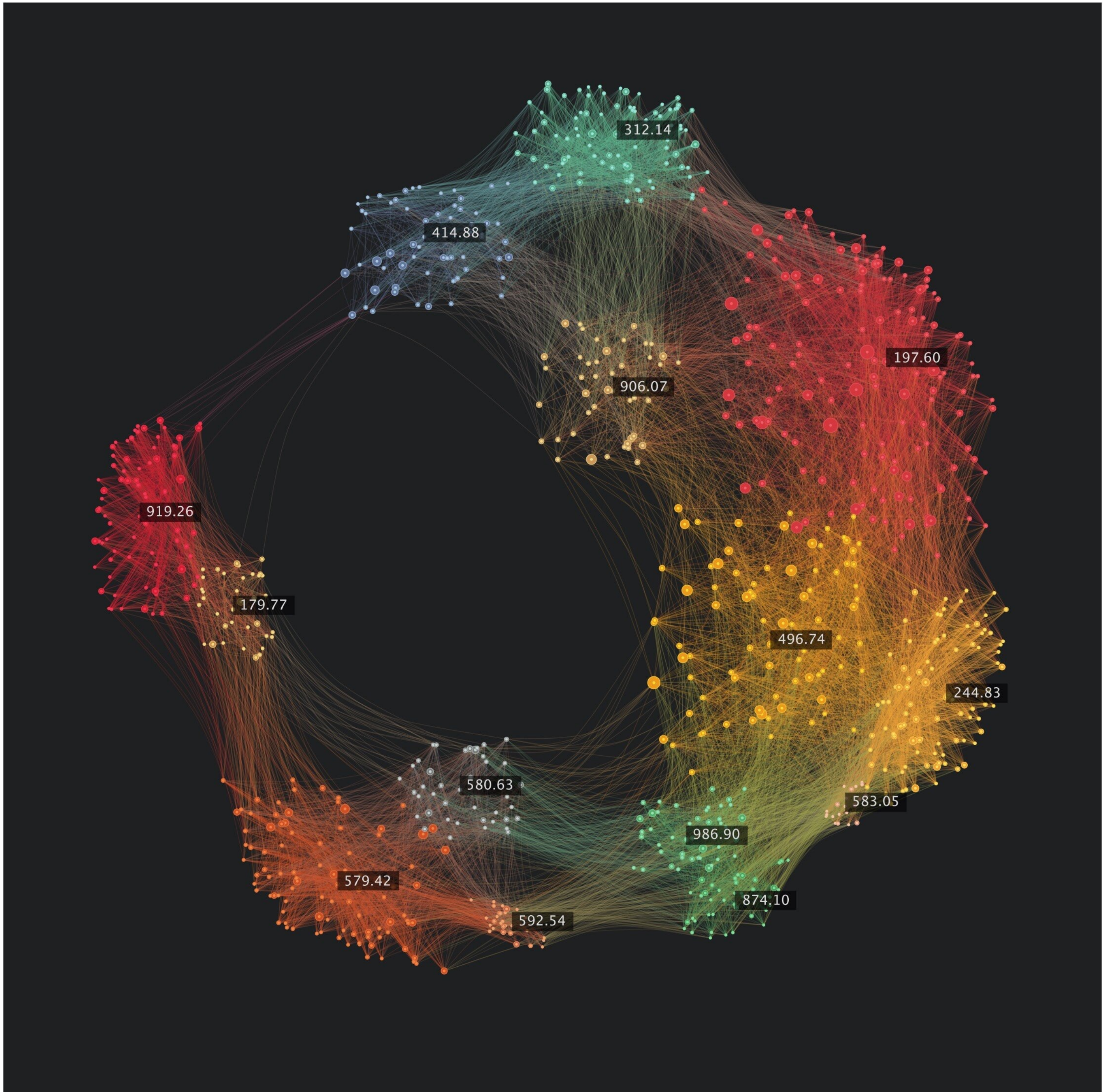
Lecture 5 Clustering & Recommender Systems

HUANG Xiao

xiaohuang@comp.polyu.edu.hk

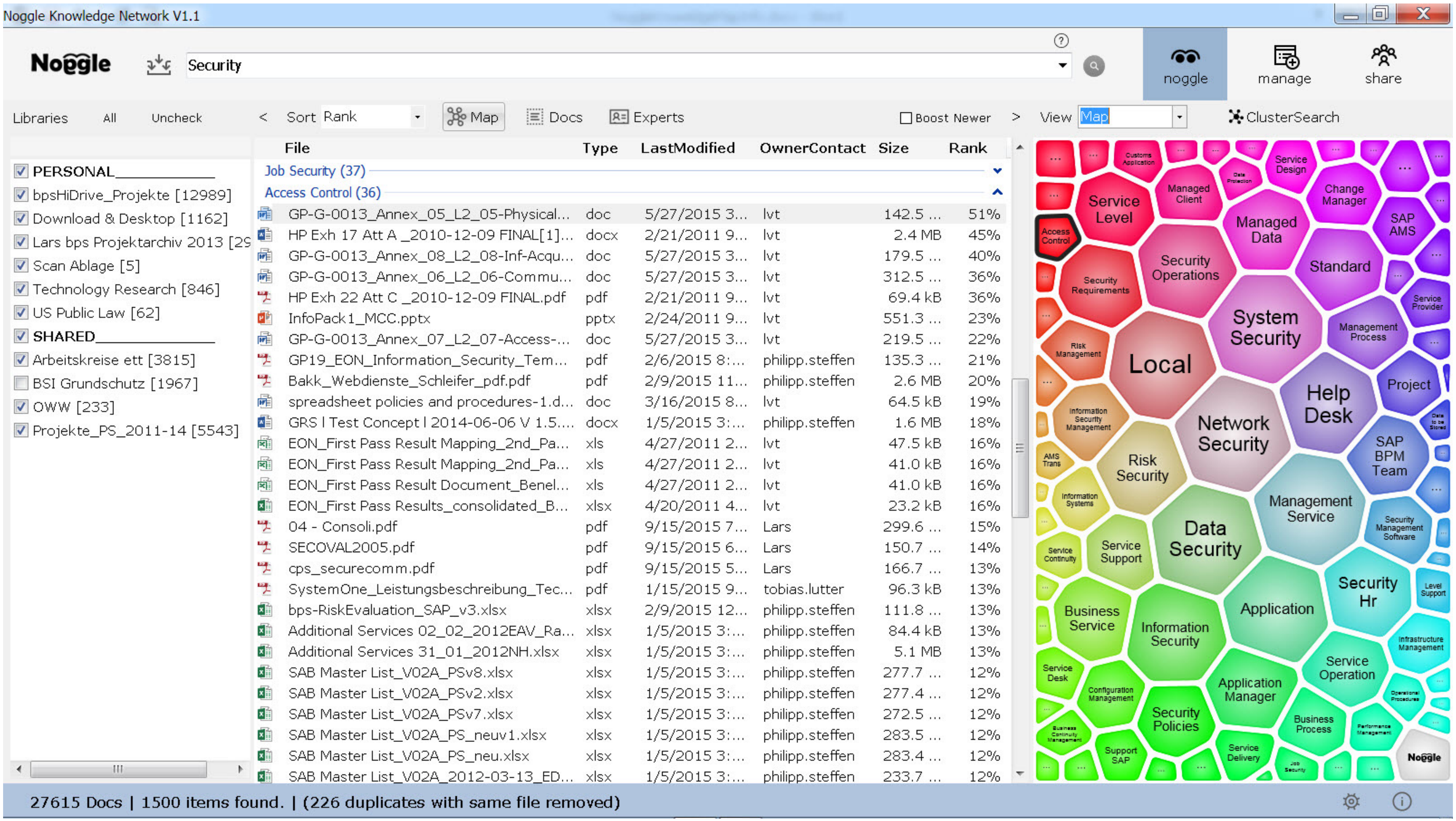
What is Cluster Analysis?

- Cluster: A collection of data objects
  - similar (or related) to one another within the same group
  - dissimilar (or unrelated) to the objects in other groups
- Cluster analysis (or *clustering*, *data segmentation, …*)
  - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters
- Unsupervised learning: no predefined classes (i.e., *learning by observations* vs. learning by examples: supervised)
- Typical applications
  - As a stand-alone tool to get insight into data distribution
  - As a preprocessing step for other algorithms



Given a cloud of data points we want to understand its structure.

Document Clustering



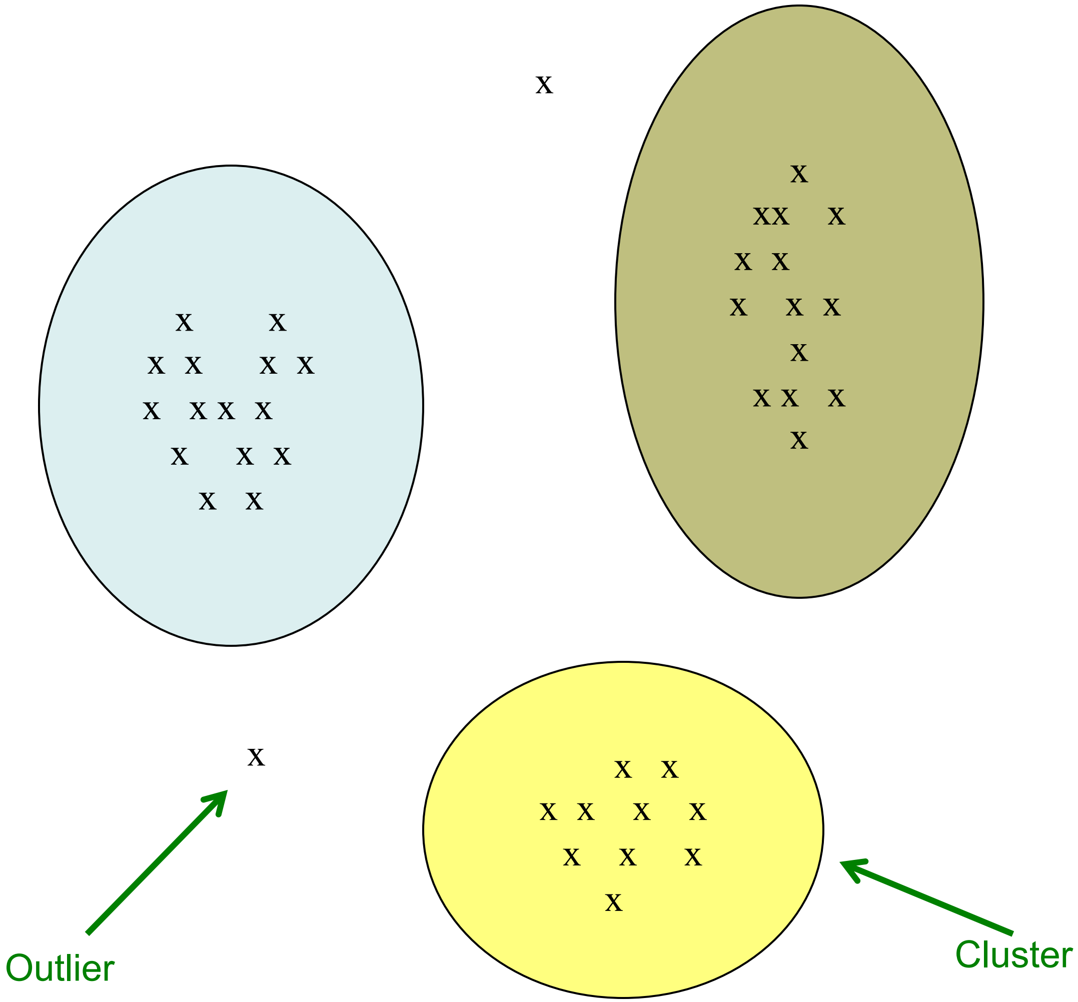
Clustering for Data Understanding & Applications

- **Customer Segmentation**: Businesses use clustering to group customers with similar purchasing behavior. This helps in targeted marketing, personalized recommendations, and product/service customization
- **Image Segmentation**: In computer vision, clustering is used to segment images into regions with similar features. This is useful in object detection, image recognition
- **Anomaly Detection**: Clustering can help identify outliers or anomalies in datasets. This is crucial in fraud detection, network security, and quality control
- **Social Network Analysis**: Clustering can group users with similar connections or behavior in social networks. This is used for community detection and influence analysis

Clustering as a Preprocessing Tool (Utility)

- Summarization:
  - Preprocessing for regression, classification
- Compression:
  - Image processing: vector quantization
- Finding K-nearest Neighbors:
  - Localizing search to one or a small number of clusters
- Outlier detection:
  - Outliers are often viewed as those “far away” from any cluster

Example: Clusters & Outliers



Problem definition of clustering

- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, so that
  - Members of a cluster are close/similar to each other
  - Members of different clusters are dissimilar
- **Usually:**
  - Points are in a high-dimensional space
  - Similarity is defined using a distance measure
    - Euclidean, Cosine, Jaccard, edit distance, …

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

Clustering is a hard problem!

- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy

- Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:** Almost all pairs of points are at about the same distance

Clustering Problem: Music

- **Intuitively: Music divides into categories, and customers prefer a few categories**
  - But what are categories really?

- Represent a song by a set of customers who like it
- Similar songs have similar sets of customers, and vice-versa

Clustering Problem: Music

Space of all songs:

- Think of a space with one dimension for each customer
  - Values in a dimension may be 0 or 1 only
  - A song is a point in this space $(x_1, x_2, \ldots, x_d)$, where $x_i = 1$ iff the i th customer bought the CD
- For Spotify:
  - Spotify lets you discover, organize, and share over 100 million songs, over 5 million podcast titles and 350,000+ audiobooks
  - In 2023, Spotify has 551 million users and 220 million premium subscribers across 184 regions
- **Task:** Find clusters of similar songs

Clustering Problem: Documents

Finding topics:

- Represent a document by a vector $(x_1, x_2, \ldots, x_d)$, where $x_i = 1$ iff the i^{th} word (in some order) appears in the document
  - It actually doesn't matter if d is infinite; i.e., we don't limit the set of words

- **Documents with similar sets of words may be about the same topic**

Similarity is defined using a distance measure

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

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

cosine distance = 1 - cosine similarity

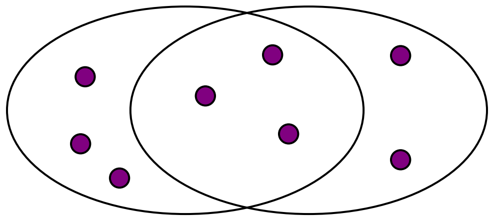
  - Measure similarity by **Euclidean distance**

$$d(\mathbf{A},\mathbf{B}) = \sqrt{\sum_{i=1}^{n}(B_i - A_i)^2}$$

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

Jaccard similarity

- The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union: $\boldsymbol{sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|}$
- **Jaccard distance:** $\boldsymbol{d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|}$



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8

- Document D_1 is a set of its b words
- Equivalently, each document is a 0/1 vector in the space of *k words*
  - Each unique word is a dimension
  - Vectors are very sparse

k-means Clustering Algorithm

- Partitioning method: Partitioning ***n*** objects into a set of ***k*** clusters, such that the sum of squared distances is minimized (where c_i is the centroid or clustroid of cluster C_i)

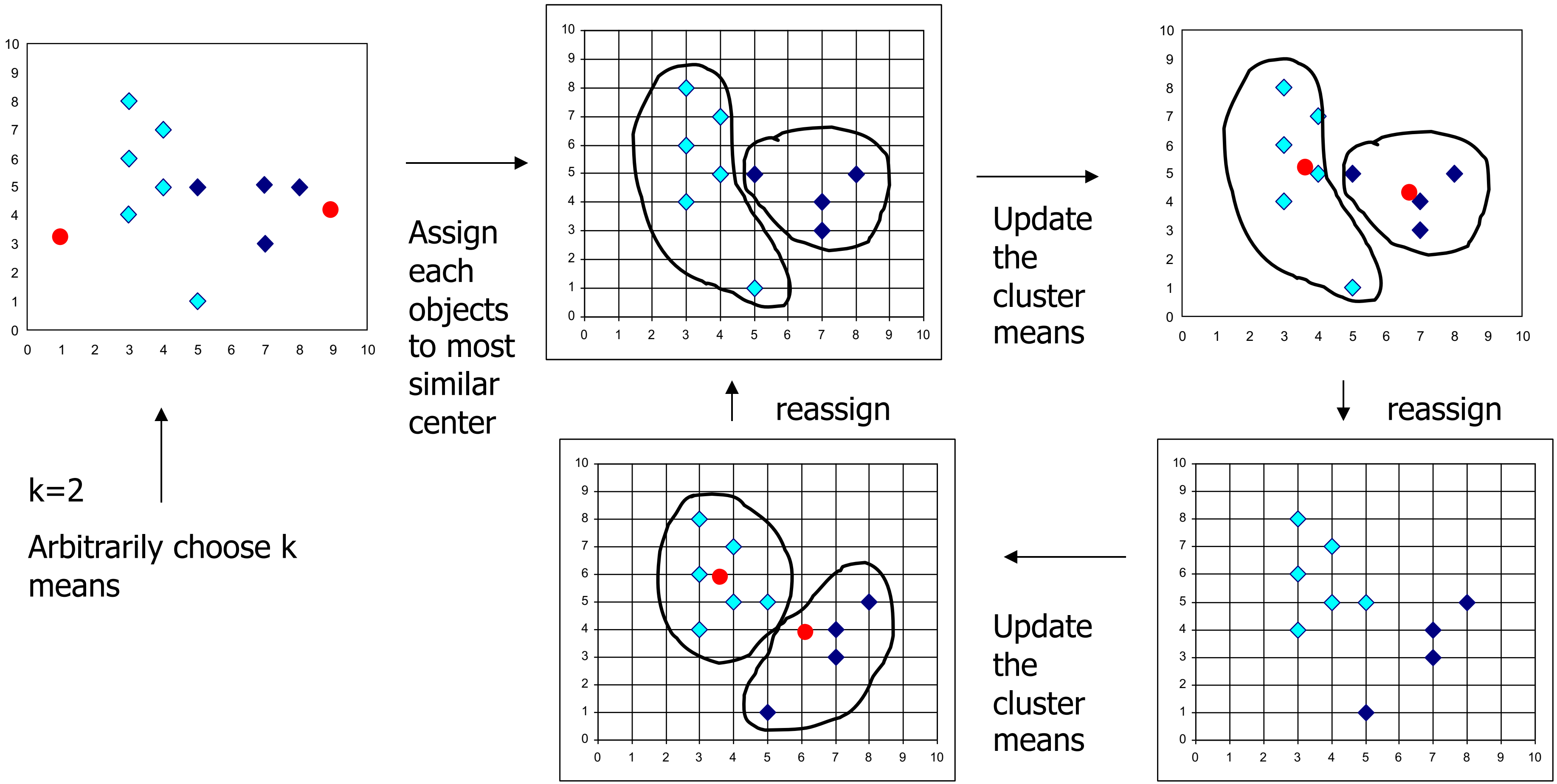
$$E = \Sigma_{i=1}^{k} \Sigma_{p \in C_i} (p - c_i)^2$$

- Given *k*, find a partition of *k clusters* that optimizes the chosen partitioning criterion
  - Global optimal: exhaustively enumerate all partitions
  - Heuristic methods: *k-means* and *k-medoids* algorithms
  - *k-means* (MacQueen'67, Lloyd'57/'82): Each cluster is represented by the center of the cluster
  - *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster

k-means Clustering Algorithm

- Assumes Euclidean space/distance
- Start by picking ***k***, the number of clusters
- Initialize clusters by picking one point per cluster
  - **Example:** Pick one point at random, then ***k*-1** other points, each as far away as possible from the previous points

Demo



Assign each objects to most similar center
Update the cluster means
reassign
reassign
Update the cluster means
k=2
Arbitrarily choose k means

Populating Clusters

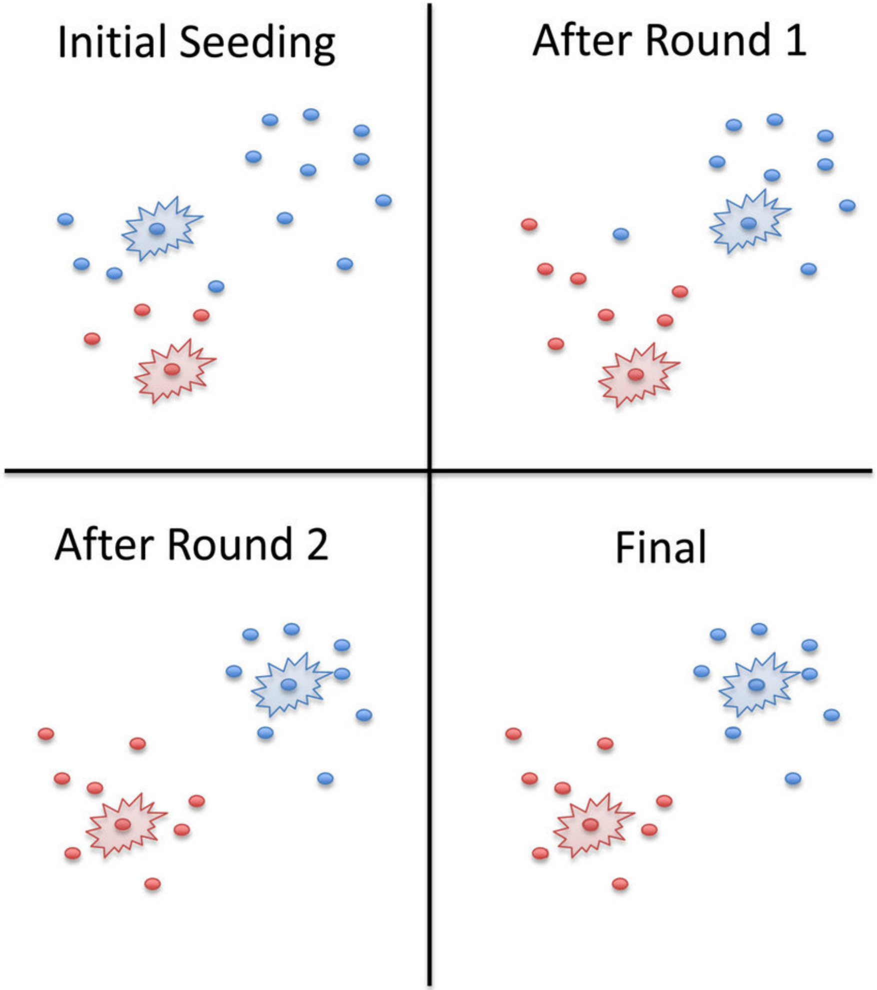
- **1)** For each point, place it in the cluster whose current centroid it is nearest

- **2)** After all points are assigned, update the locations of centroids of the ***k*** clusters

- **3)** Reassign all points to their closest centroid
  - Sometimes moves points between clusters

- **Repeat 2 and 3 until convergence**
  - **Convergence:** Points don't move between clusters and centroids stabilize

Initialization of k-means

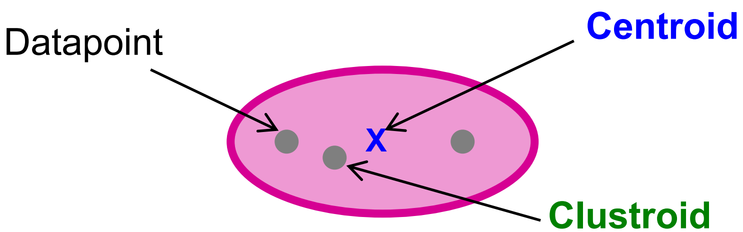


- The way to initialize the centroids was not specified. One popular way to start is to randomly choose k of the examples

- The results produced depend on the initial values for the centroids, and it frequently happens that suboptimal partitions are found. The standard solution is to try a number of different starting points

Centroid & Clustroid

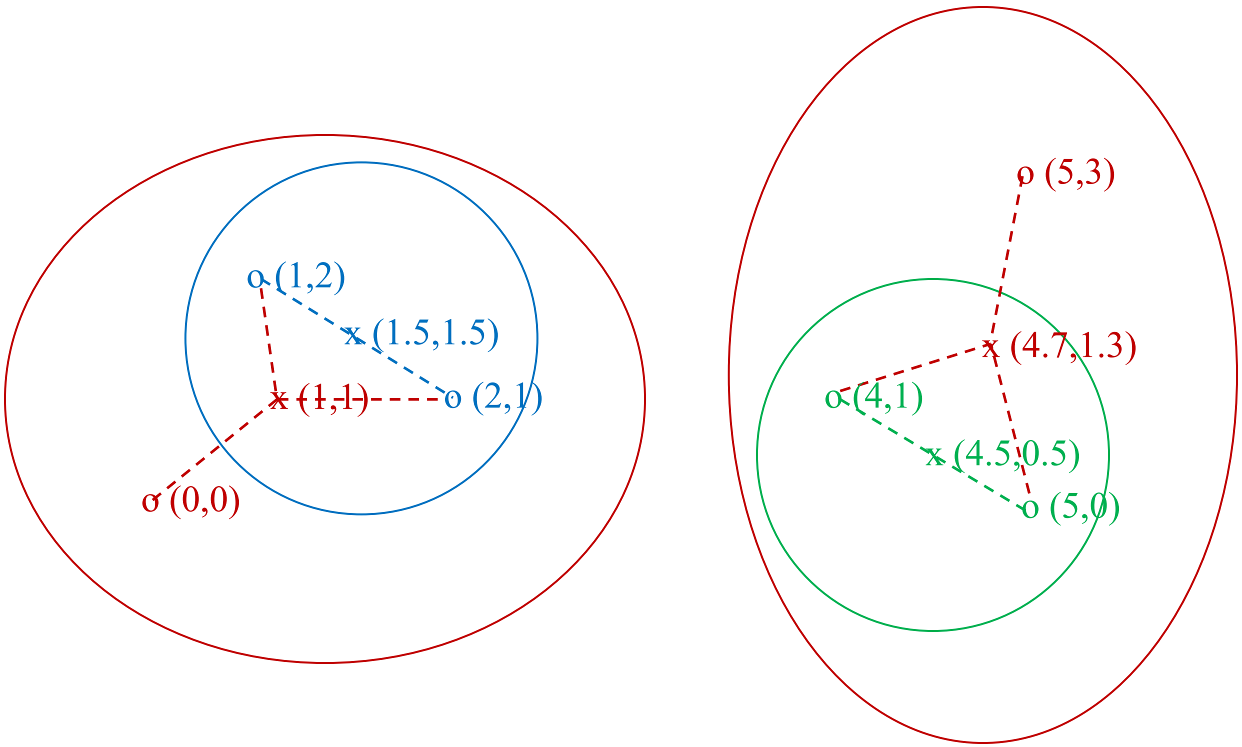
- **Centroid** is the avg. of all (data)points in the cluster. This means centroid is an “artificial” point
- **Clustroid** is an existing (data)point that is “closest” to all other points in the cluster



Cluster on
3 data points

Clustroid

- **Euclidean case:** each cluster has a centroid
  - ***centroid*** = average of its (data) points
  - use the node that is "closest" to the centroid as a clustroid



Data:
o … data point
x … centroid

- **What about the non-Euclidean case?**

Clustroid (non-Euclidean Case)

- Non-Euclidean: The only "locations" we can talk about are the points themselves, i.e., there is no "average" of two points
- *clustroid* = point "***closest***" to other points
- **Possible meanings of "closest":**
  - Smallest average distance to other points
  - Smallest sum of squares of distances to other points, e.g., for distance metric ***d*** clustroid ***c*** of cluster ***C*** is:

$$\min_{c} \sum_{x \in C} d(x,c)^2$$

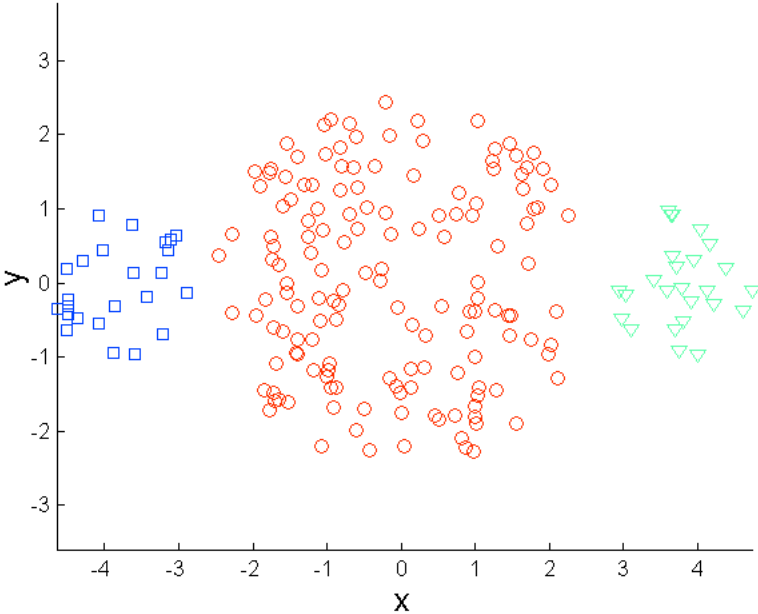
  - Smallest maximum distance to other points

Pros & Cons

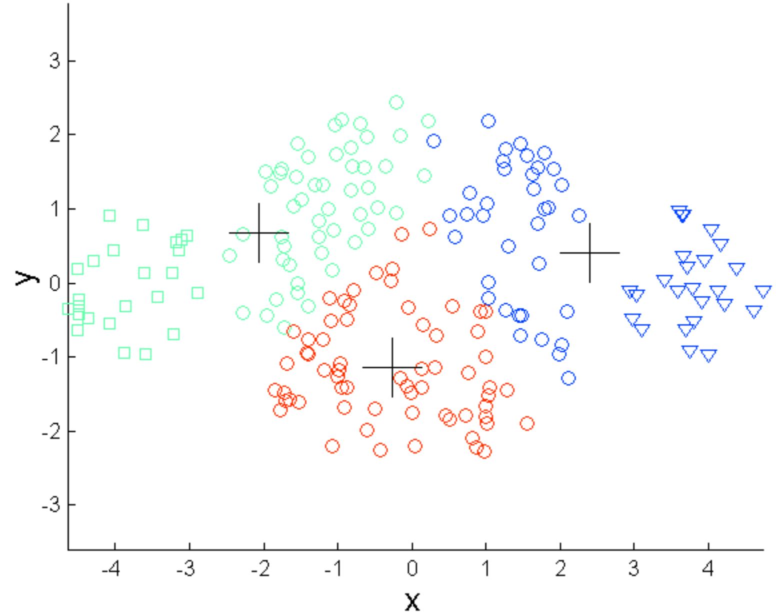
- Simple iterative method
- User provides "K"

- Often too simple ----> bad results
- Difficult to guess the correct "K"
  - We may not know the number of clusters before we want to find clusters
- No guarantee of optimal solution
- Complexity is O(n * K * I * d)
  - n = number of points, K = number of clusters, I = number of iterations, d = number of attributes

Limitations: when clusters are of differing sizes

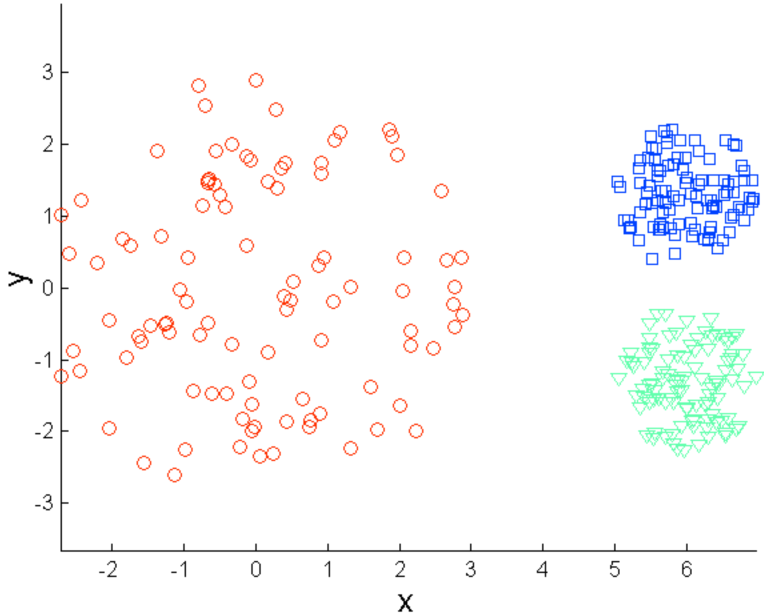


Original Points

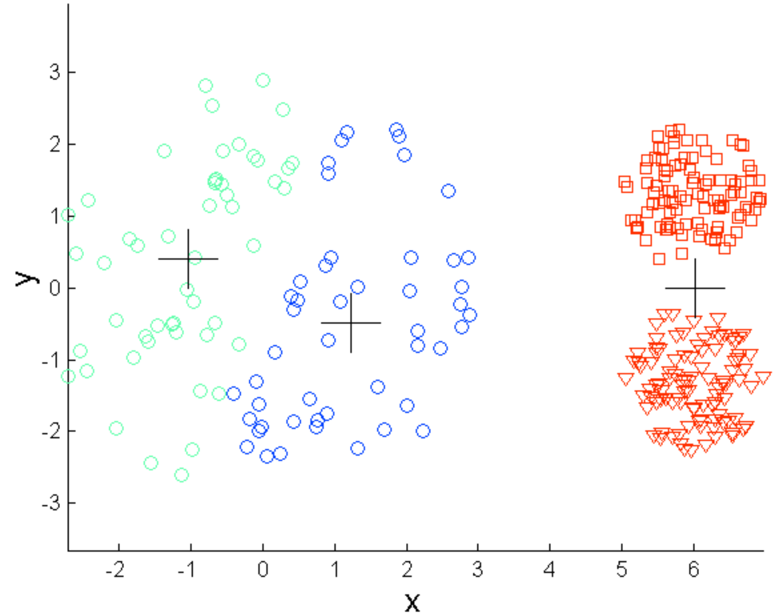


K-means (3 Clusters)

Limitations: when clusters are of differing densities

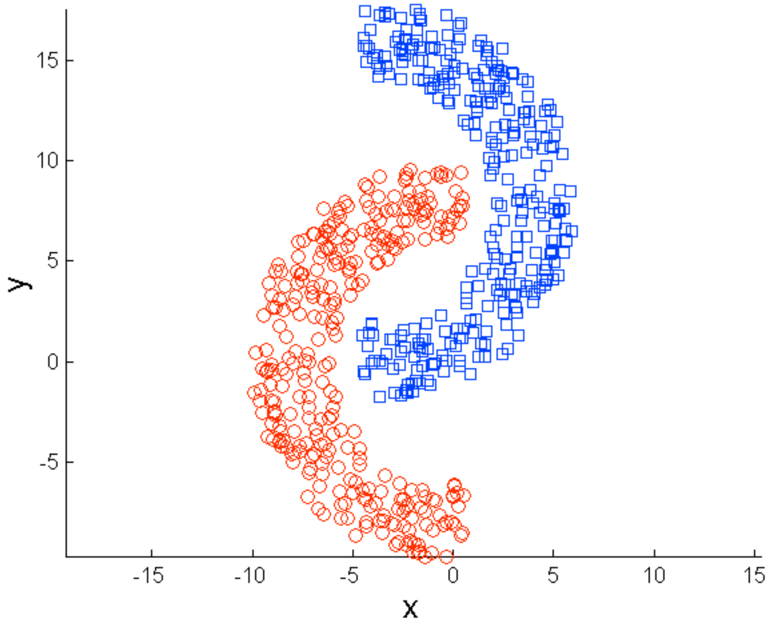


Original Points

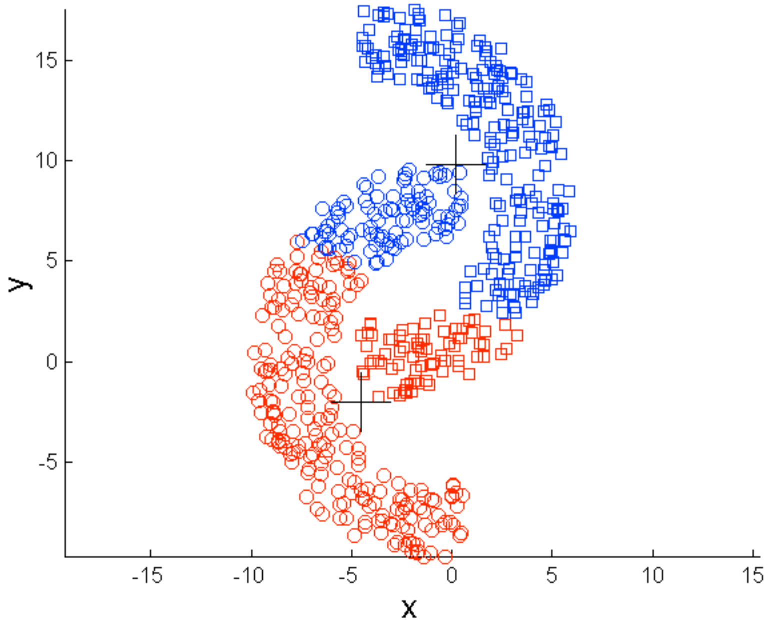


K-means (3 Clusters)

Limitations: when non-globular shapes



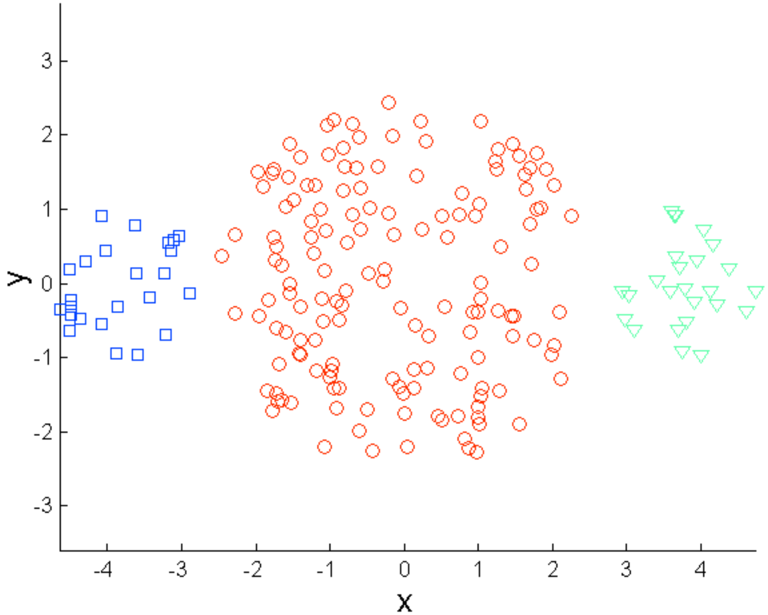
Original Points



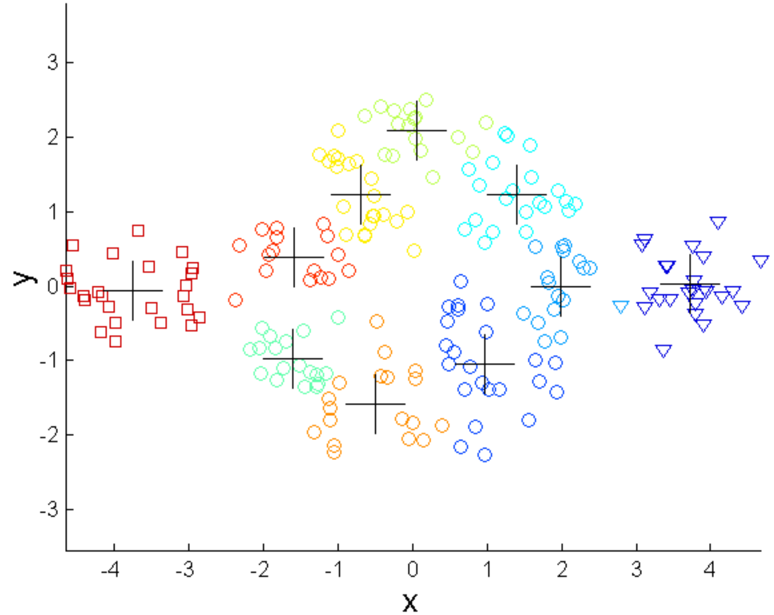
K-means (2 Clusters)

Overcoming K-means Limitations

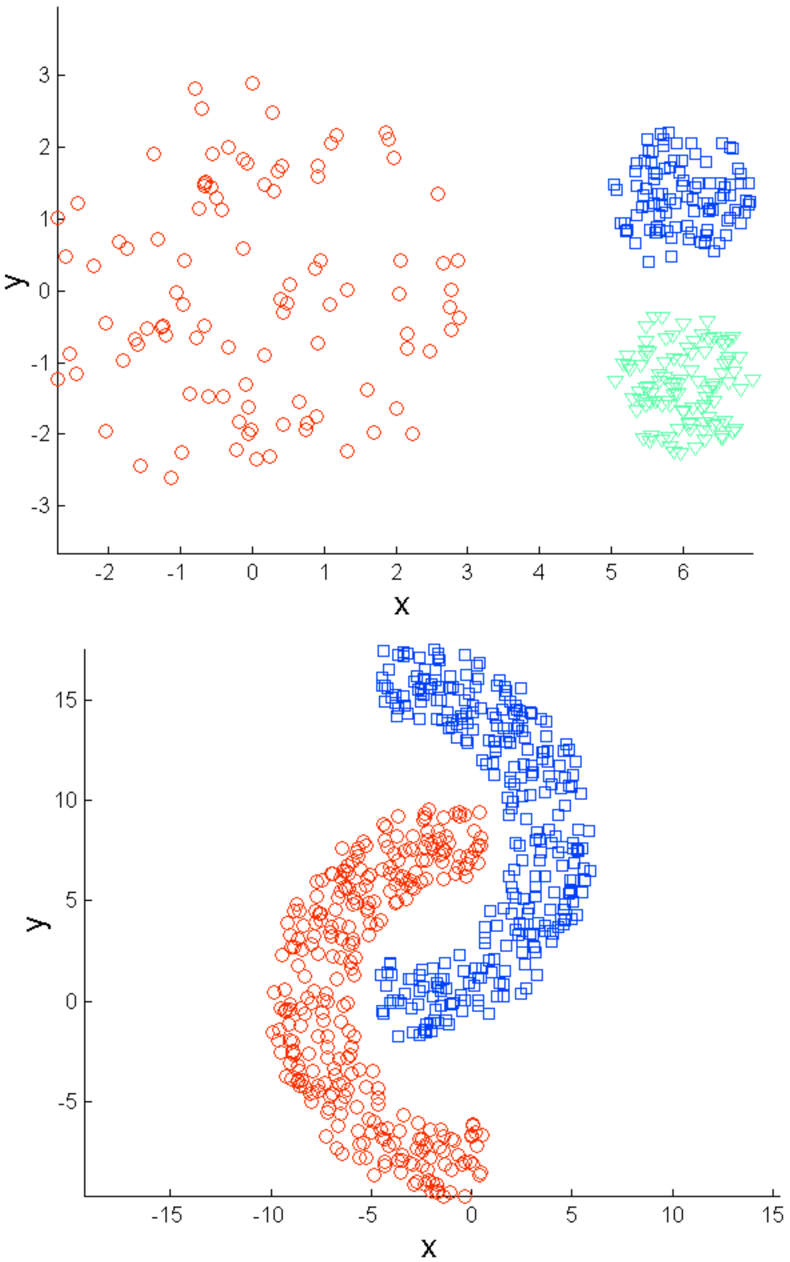
- One solution is to find many clusters
  - each of them represents a part of a natural cluster
  - small clusters need to be put together in a post-processing step



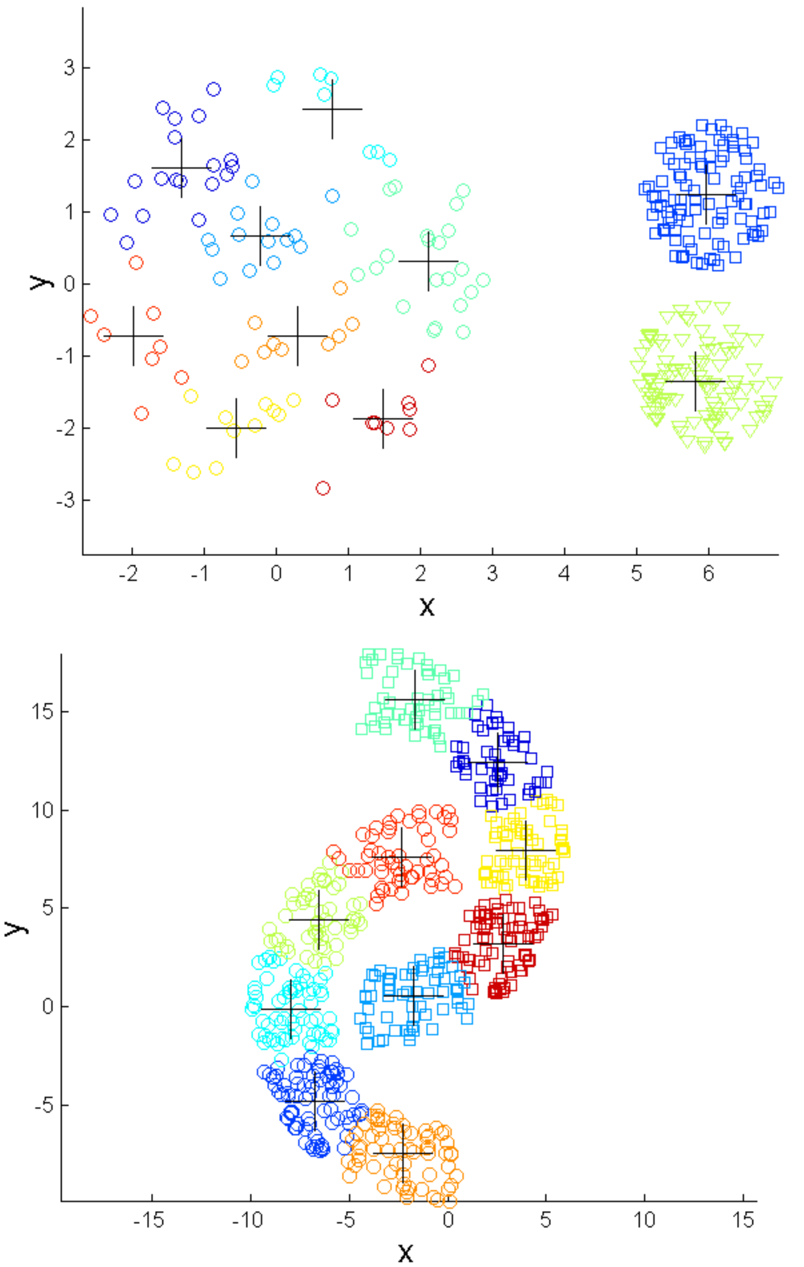
Original Points



K-means Clusters



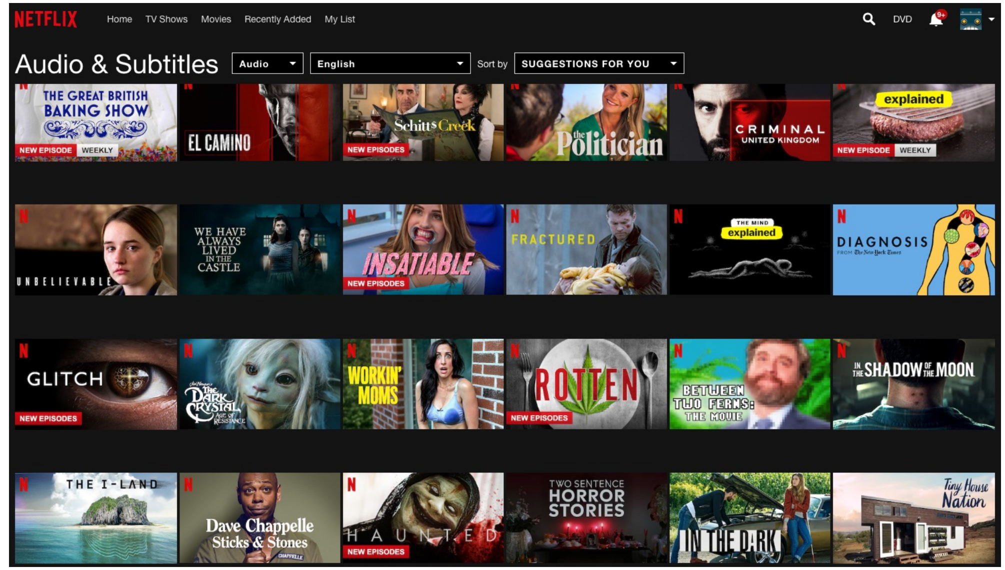
Original Points



K-means Clusters

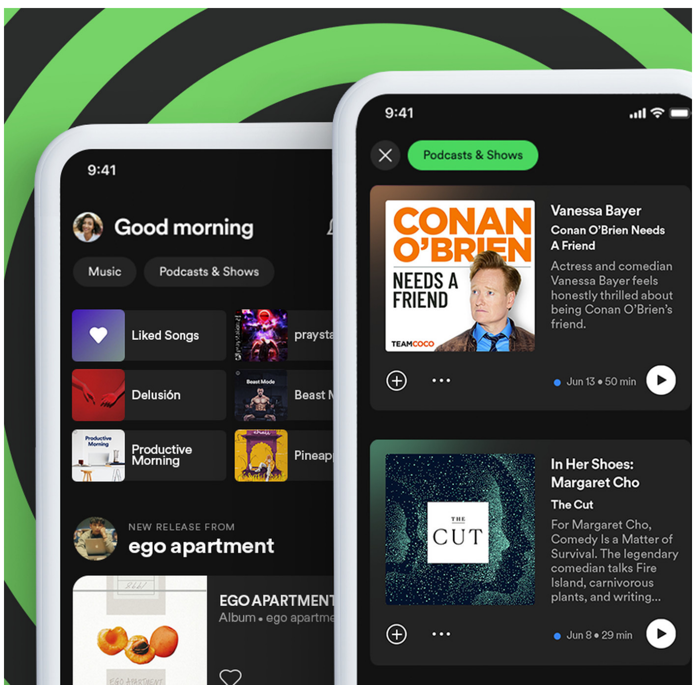
Recommender System Examples

- Amazon, YouTube, Netflix, …
- How to improve users' satisfaction?
- What item for what people?
  - E.g., Recommend movies based on the predictions of user's movie ratings



More Recommender System Examples

- News feed
- Music feed
- Twitter feed

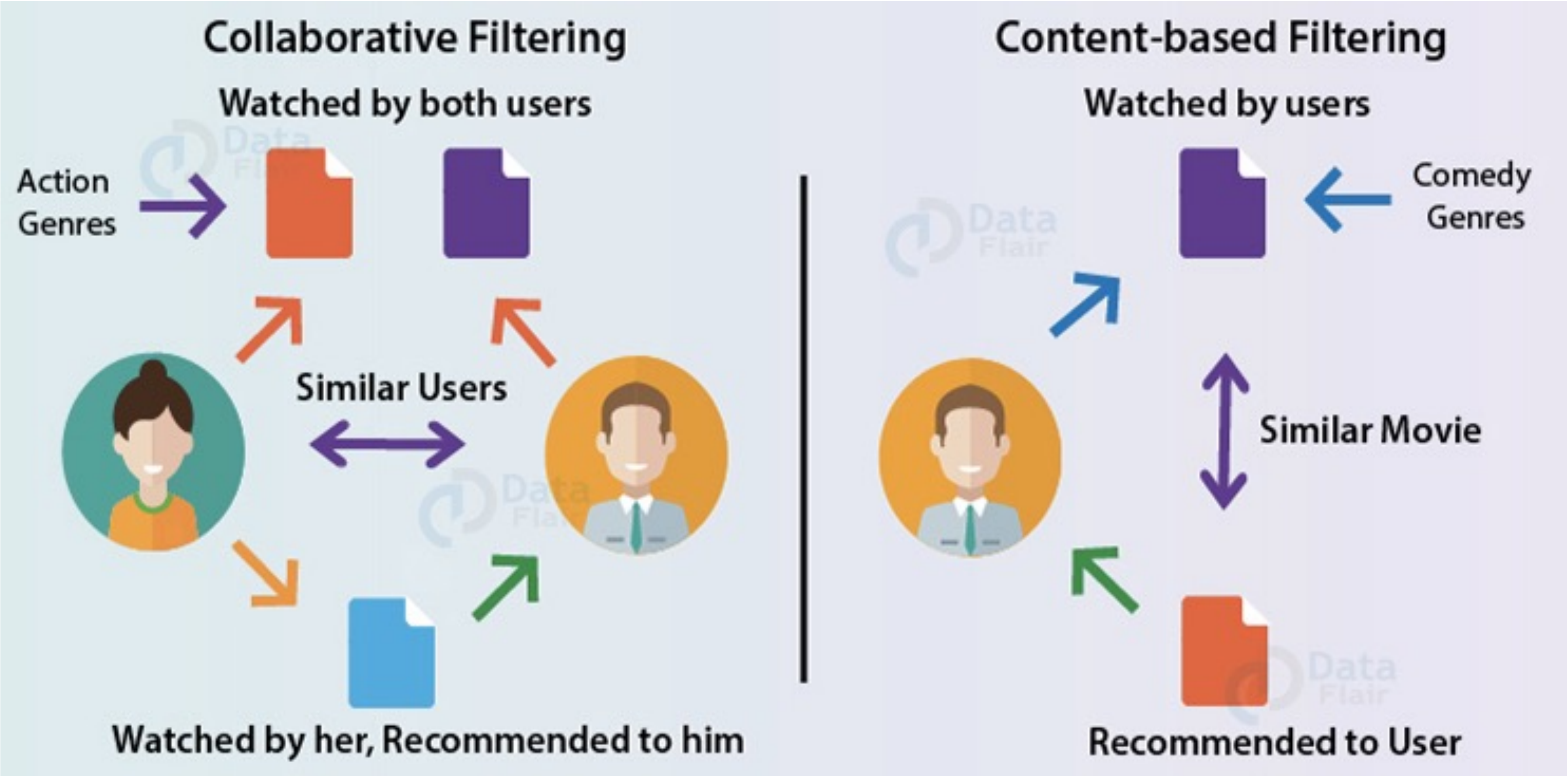


Recommender System Types

- **Content Based (CB):** recommendations are based on the assumption that if in the past a user liked a set of items with particular features, she/he will likely go for the items with similar characteristics.

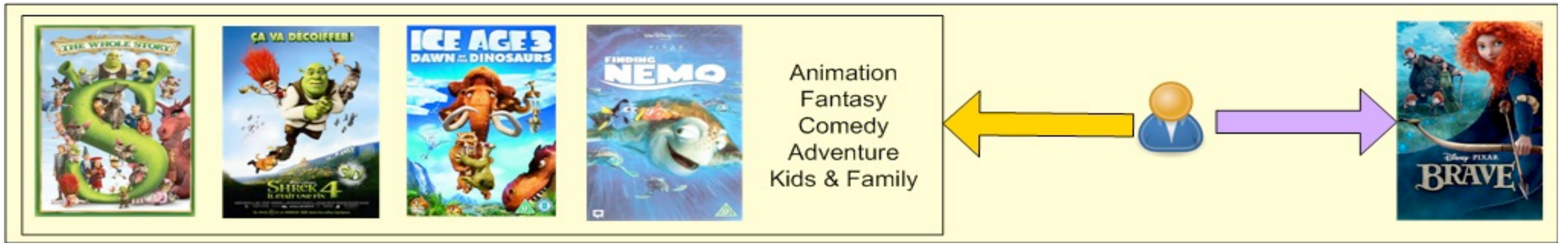
- **Collaborative Filtering (CF):** recommendations are based on the assumption that users having similar history are more likely to have similar tastes/needs.

Recommender System Types

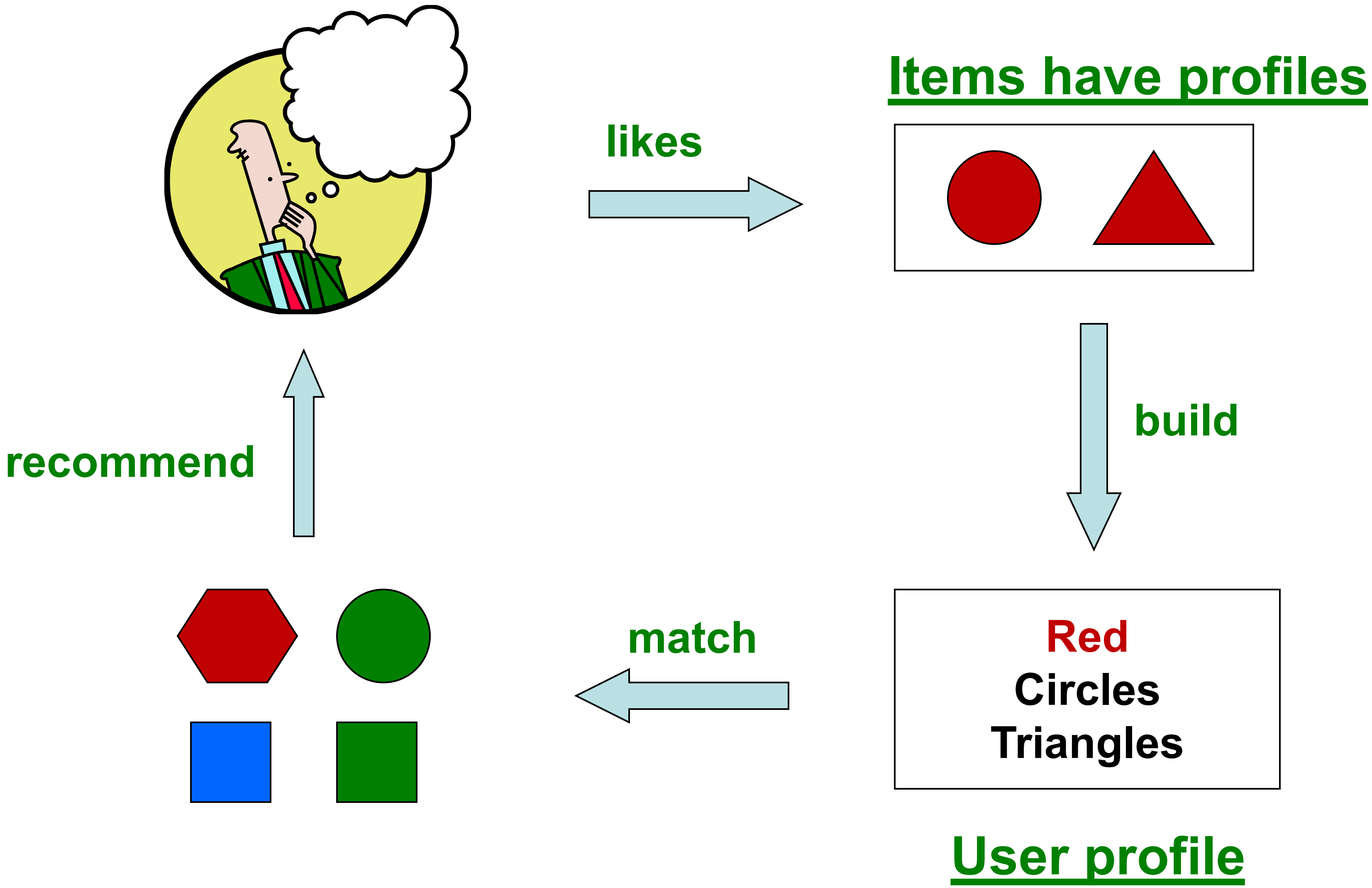


Content-based Recommender Systems

- Give recommendations to a user based on items with “similar” content in user’s profile
- Recommendation is only dependent on particular user’s historical data
- Besides user-item interactions (i.e., ratings), we also have the item feature vectors as the inputs



Plan of Action



Example

Movie	Alice	Bob	Carol	Dave	X1 (Romance)	X2 (KungFu)
Love letter	5	5	0	0	0.9	0
Romancer	5	?	?	0	1	0
Stay with me	?	4	0	?	0.89	0
KungFu Panda	0	0	5	4	0.2	0.9
FightFightFight	0	0	5	?	0.1	1

? not rated yet

- For each item, create an item profile (a set of features)
  - E.g., each movie has genre, author, title, actor, director,…

Symbols: Table

Movie	Alice	Bob	Carol	Dave	X1 (Romance)	X2 (KungFu)
Love letter	5	5	0	0	0.9	0
Romancer	5	?	?	0	1	0
Stay with me	?	4	0	?	0.89	0
KungFu Panda	0	0	5	4	0.2	0.9
FightFightFight	0	0	5	?	0.1	1

$n_m = 5$: number of movies

$n_u = 4$: number of users

$n = 2$: number of movie features

Symbols: Rating

$m^{(1)} = 4$

$r(1,2) = 1, y^{(1,2)} = 5$

$r(3,4) = 0$

Movie	Alice	Bob	Carol	Dave
Love letter	5	5	0	0
Romancer	5	?	?	0
Stay with me	?	4	0	?
KungFu Panda	0	0	5	4
FightFightFight	0	0	5	?

$r(i,j) = 1$ if user j has rated movie i; $y^{(i,j)}$ is the rating

$m^{(j)}$: number of rated movies rated by user j

RMSE

- Compare predictions with known ratings
- My system predicted you would rate
  - The Shawshank Redemption as 4.3 stars
    - In reality, you gave it 5 stars
  - The Matrix with 3.9 stars
    - In reality, you gave it 4 stars

- RMSE = sqrt(1/2 * (4.3 - 5)^2 + (3.9 - 4)^2))

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

How to solve the problem for Alice?

Movie	Alice	Bob	Carol	Dave	X1 (Romance)	X2 (KungFu)
Love letter	5	5	0	0	0.9	0
Romancer	5	?	?	0	1	0
Stay with me	?	4	0	?	0.89	0
KungFu Panda	0	0	5	4	0.2	0.9
FightFightFight	0	0	5	?	0.1	1

Hypothesis For Alice

- Learn parameter $\theta^{(1)} = [\theta_0^{(1)} \theta_1^{(1)} \theta_2^{(1)}]^T$ by solving a Linear Regression problem
- Hypothesis function

$$h_{\theta^{(1)}}(x) = \left(\theta^{(1)} \right)^T x = \theta_0^{(1)} x_0 + \theta_1^{(1)} x_1 + \theta_2^{(1)} x_2$$

- Cost Function

$$J\left(\theta^{(1)}\right) = \frac{1}{2m^{(1)}} \sum_{i:r(i,1)=1} \left(\left(\theta^{(1)} \right)^T x^{(i)} - y^{(i,1)} \right)^2 + \frac{\lambda}{2m^{(1)}} \sum_{k=1}^{n} \left(\theta_k^{(1)} \right)^2$$

$$= \frac{1}{2m^{(1)}} \sum_{i:r(i,1)=1} \left(\sum_{k=0}^{n} \left(\theta_k^{(1)} x_k^{(i)} \right) - y^{(i,1)} \right)^2 + \frac{\lambda}{2m^{(1)}} \sum_{k=1}^{n} \left(\theta_k^{(1)} \right)^2$$

Iteration …

```
class RegularizedLinearRegressionUsingGD:

    def __init__(self, eta=0.01, r= 0.009, n_iterations=10000):
        self.eta = eta
        self.r = r
        self.n_iterations = n_iterations

    def fit(self, x, y):
        self.cost_ = []
        self.w_ = np.array([0.0,0.0,0.0])
        m = x.shape[0]

        for _ in range(self.n_iterations):
            y_pred = np.dot(x, self.w_)
            residuals = y_pred - y

            gradient_vector_w_0 = np.sum(residuals) / m * self.eta
            gradient_vector_w_1 = (np.dot(x[:,1], residuals) + self.r * self.w_[1]) / m * self.eta
            gradient_vector_w_2 = (np.dot(x[:,2], residuals) + self.r * self.w_[2]) / m * self.eta
            self.w_[0] -= gradient_vector_w_0
            self.w_[1] -= gradient_vector_w_1
            self.w_[2] -= gradient_vector_w_2

            cost = np.sum((residuals ** 2)) / (2 * m) + self.r * ((self.w_[1] ** 2) + (self.w_[2] ** 2)) / (2 * m)
            self.cost_.append(cost)

        print('iter {}: w = {} \t cost ={}'.format(_, self.w_, cost))
        return self

    def predict(self, x):
        return np.dot(x, self.w_)
```

Gradient descent update

Alice's Model

```
iter 9999: w = [ 1.95158962  3.16948852 -2.52109055]     cost =0.045574863024676435
predicted response: [ 4.80412929  5.12107814  0.31650583 -0.25255208]
Root mean squared error:  0.05424593597454509
R2 score:  0.9913206502440728
```

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.025 \\ 0.0375 \\ 0 \end{bmatrix} = \cdots = \begin{bmatrix} 1.95 \\ 3.17 \\ -2.52 \end{bmatrix}$$

$$h_{\theta^{(1)}}(x) = \left(\theta^{(1)}\right)^T x = 1.95 + 3.17x_1 - 2.52x_2$$

Rating Prediction for Alice

Movie		X1 (Romance)	X2 (KungFu)	Alice $y^{(i,1)}$
Love letter	$x^{(1)}$	0.9	0	5
Romancer	$x^{(2)}$	1	0	5
Stay with me	$x^{(3)}$	0.89	0	$\boxed{4.77}$
KungFu Panda	$x^{(4)}$	0.2	0.9	0
FightFightFight	$x^{(5)}$	0.1	1	0

- Predict user j rating movie i with $\left(\theta^{(j)}\right)^T x^{(i)}$
- E.g., $\left(\theta^{(1)}\right)^T x^{(3)} = [1.95 \quad 3.17 \quad -2.52]\begin{bmatrix}1\\0.89\\0\end{bmatrix} = 4.77$

General Problem

Movie	Alice $\theta^{(1)}$	Bob $\theta^{(2)}$	Carol $\theta^{(3)}$	Dave $\theta^{(4)}$	X1 (Romance)	X2 (KungFu)
Love letter $x^{(1)}$	5	5	0	0	0.9	0
Romancer $x^{(2)}$	5	?	?	0	1	0
Stay with me $x^{(3)}$	?	4	0	?	0.89	0
KungFu Panda $x^{(4)}$	0	0	5	4	0.2	0.9
FightFightFight $x^{(5)}$	0	0	5	?	0.1	1

- For each user j, learn parameter $\theta^{(j)} \in R^{n+1}$

Problem Formulation

- $r(i,j) = 1$ if user j has rated movie i
- $r(i,j) = 0$ if user j has not rated movie i
- $y^{(i,j)}$: rating by user j on movie i if $r(i,j) = 1$
- n : number of features of a movie
- $\theta^{(j)} \in R^{n+1}$: parameter vector for user j
- $x^{(i)} \in R^{n+1}$: feature vector for movie i
- $m^{(j)}$: number of rated movies rated by user j
- n_u : number of users
- n_m : number of movies

CB Optimization Objective

- Given $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$, to learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} \left(\theta_k^{(j)} \right)^2$$

Movie	$x_1^{(i)}$ (Romance)	$x_2^{(i)}$ (KungFu)	User j $y^{(i,j)}$
Love letter $x^{(1)}$	0.9	0	5
Romancer $x^{(2)}$	1	0	5
Stay with me $x^{(3)}$	0.89	0	?
KungFu Panda $x^{(4)}$	0.2	0.9	0
FightFightFight $x^{(5)}$	0.1	1	0

$$x^{(i)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0.9 & 1 & 0.89 & 0.2 & 0.1 \\ 0 & 0 & 0 & 0.9 & 1 \end{bmatrix}$$

$$\theta^{(j)} = \begin{bmatrix} \theta_0^{(j)} \\ \theta_1^{(j)} \\ \theta_2^{(j)} \end{bmatrix} = ?$$

$$\theta^{(j)} \in R^{n+1}; x^{(i)} \in R^{n+1}, x_0^{(i)} = 1$$

Optimization Objectives

- To learn $\theta^{(j)}$ (parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(\theta_k^{(j)} \right)^2$$

- To learn $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$

$$\min_{\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)} \right)^2$$

CB Gradient Decent Update

- $k = 0$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)}$$

- $k \neq 0$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left(\left(\theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

Pros: Content-based Approach

- **+: No need for data on other users**
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No cold-start item problems
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-based Approach

- **–: Finding the appropriate features is hard**
  - E.g., images, movies, music
- **–: Recommendations for new users**
  - **How to build a user profile?**
- **–: Overspecialization**
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - **Unable to exploit quality judgments of other users**