TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

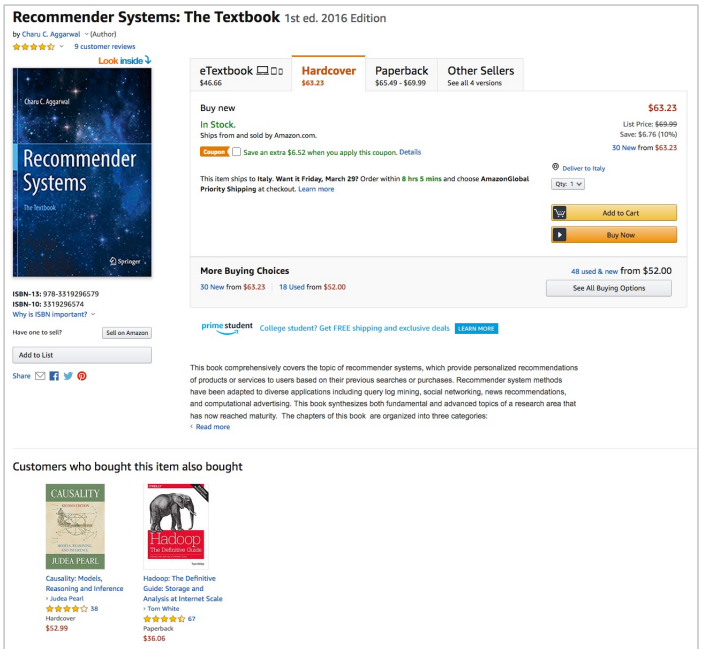
Recommender Systems

Algorithms, Data and Security
A.Y. 2025/26

Valeria Cardellini

Global Governance, 3rd year
Science and Technology Major

Recommendations

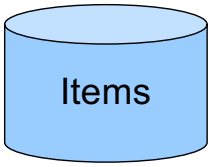


"We are leaving the age of information and entering the age of recommendation." Chris Anderson

Recommendations

Search

Recommendations



Products, web sites, blogs, news items, …

Examples:

Google

amazon.com

NETFLIX

facebook

YouTube

movielens
helping you find the *right* movies

Spotify

A recommender system is able to suggest items to users, predicts user preference, match users with relevant content

From scarcity to abundance

- Shelf space is limited
  - Applies to: physical stores, TV networks, movie theaters
  - 👉 Scarcity of choice
- The Web changes everything
  - Digital platforms: near-zero-cost to store and share information
  - Unlimited catalog size
  - Examples:
    - Amazon: millions of products
    - Netflix: vast movie library
  - 👉Abundance of choice
- More choice requires better filters
  - The solution: recommender systems

The power of recommender systems

- When does a recommender system work well?
  - Helps users find value in a large space of choices

- What is the long tail?
  - A few items are very popular (head)
  - Many items are rare or niche (long tail)

The power of recommender systems

- How Into Thin Air made Touching the Void a bestseller https://www.nytimes.com/2006/08/10/books/10manly.html
  - Joe Simpson's *Touching the Void* was initially a modest success after its 1988 release and was largely forgotten
  - A decade later, the popularity of Jon Krakauer's *Into Thin Air* revived interest in it. Booksellers promoted the two together, boosting sales, and a new edition plus a film adaptation further increased its success
  - The main driver behind its resurgence was Amazon's recommendation algorithm, which suggested it to readers of *Into Thin Air*, creating a powerful feedback loop of sales and positive reviews

Why do we need recommender systems?

- Long tail and digital products
  - Huge catalogs, long tail
- Reduce cognitive load on users
  - Simplify decisions in environments with too many choices
- Enhance user experience
  - Provide personalized and relevant content
- Increase loyalty and volume
- Introduce quality content
- Help with inventory control

Case study: Netflix

- Netflix business started with DVD sales and rental by mail
- Shifted to subscription-based streaming OTT service
- Now produces original content (in-house productions)
- Scale
  - 270+ million subscribers worldwide
  - 9-10% of global Internet traffic… (1 out of 10 bits)

https://www.businessofapps.com/data/netflix-statistics/

Netflix's prize (2006-2009)

- Goal
  - Provide accurate movie recommendations
  - Help users find content they enjoy
  - Maximize customer satisfaction and retention
- The challenge (Oct. 2006)
  - $1,000,000 prize for beating Netflix's Cinematch by >10%
    https://en.wikipedia.org/wiki/Netflix_Prize
- Dataset:
  - 100 million ratings (1999–2005)
  - 480,000+ users
  - 17,770 movies
- Why it was hard
  - Sparse data: most users rated only a few movies
  - Skewed distribution: few users rated many movies (one rated 17,000+)
  - Large scale: huge dataset for 2006

How to measure success?

- Which algorithm predicts ratings closest to what users actually gave?
- Metrics (from most relevant to most tractable)
  - Customer satisfaction: how happy users are with recommendations
  - Prediction effectiveness: how well we predict the ratings users give to movies they actually watched
  - Prediction error
    - Can be calculated for each (*u*,*i*) pair (user *u* and movie *i*) for which we have both prediction and actual rating
    - r_{ui}: actual rating given by user *u* to movie *i*
    - $\hat{r}_{ui}$: predicted rating for user *u* and movie *i*

Prediction error: distance metrics

- Distance metrics measure how far predictions are from actual ratings
- Let's define two distance metrics
  - Hamming distance
  - Root Mean Square Error
- Let's consider a total number of *C* pairs (*u*,*i*)

user *u*
movie *i*

Distance metrics

- Hamming distance
  - Measures disagreement between categorical ratings
  - Definition: number of positions in which two binary strings (of equal length) differ

$$d_H = \sum_{(u.i) \in C} \mathbf{1}_{r_{ui} \neq \hat{r}_{ui}}$$

$\mathbf{1}_{r_{ui} \neq \hat{r}_{ui}}$ = 1 if prediction ≠ actual, else 0

  - Examples
    - 100 and 011 differ in 3 positions: $d_H = 3$
    - 01011 and 11010 differ in 2 positions: $d_H = 2$
  - We count the number of mismatches between predicted and actual ratings

Distance metrics

- Root Mean Square Error (RMSE)
  - Measures numerical distance between predicted and actual ratings
  - Definition: square root of the second sample moment of the differences between observed values r_{ui} and predicted values $\hat{r}_{ui}$, or the quadratic mean of these differences

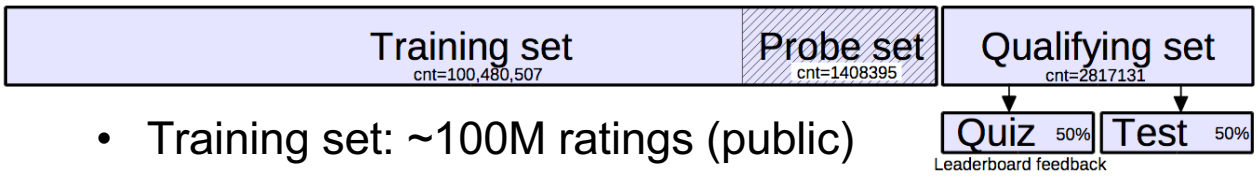
$$RMSE = \sqrt{\sum_{(u,i) \in C} \frac{(r_{ui} - \hat{r}_{ui})^2}{|C|}}$$

  - RMSE ≥ 0 → always non-negative
  - RMSE = 0 → perfect fit to data
  - Smaller RMSE → better recommendations
  - Metric used to evaluate solutions for Netflix's prize

Which recommendation to users?

- Recommender systems usually show only the top-ranked items for each user
  - Example: show top 5 movies with the highest predicted ratings for a user
- Ultimate test of recommendations
  - Does the user watch the recommended items?
  - Does the user enjoy them?
  - Metrics like RMSE or Hamming measure prediction accuracy
  - But real success = actual user engagement and satisfaction

Netflix's prize dataset



- Training set: ~100M ratings (public)
- Probe set: ~1.4M ratings (public)
  - Similar statistical properties to Test and Quiz sets
  - Competitors could use Probe set to validate their algorithms
- Quiz set: ~1.4M ratings (hidden)
  - Algorithms could be submitted once per day
  - RMSE scores on the website were based on Quiz set performance
- Final ranking and prize were based on comparison of RMSE on test set
  - Cinematch's RMSE is 0.9514, new algorithms had to improve by ≥ 10% to win the prize

Netflix's prize: the contest

- Within 1 week, Cinematch was beaten
- By June 2007, over 20,000 teams from 150 countries had registered
- By Sep. 2007, team BellKor made 8.26% improvement
- First place changed several times, 1 year later BellKor got 8.43% improvement
- Leading teams merged, in June 2009 BellKor's Pragmatic Chaos were first to achieve > 10%
  - Test set RMSE equal to 0.8567 after ~3 years

https://www.wired.com/2009/09/bellkors-pragmatic-chaos-wins-1-million-netflix-prize/

Netflix: Predicting the best user ratings

- Netflix offered $1M for the best algorithm, which shows how critical the recommender system was to their business
  - Sparked new research in recommendation algorithms
- Input data used to predict ratings in Netflix prize
  - User-movie ratings
  - Movie attributes: actors, director, genre classification, year of release, etc.

More in general: input data

- What input data can we use to predict user ratings? How input data can be gathered?
- Explicit data: users intentionally provide information about preferences, e.g.,
  - Customer ratings
  - Feedback
  - Demographics
- Implicit data: inferred from monitoring user behavior, e.g.,
  - Purchase history
  - Clicks, browsing history
  - Interaction with products (views, adds to cart, watch time)
- Item information (metadata)
  - Attributes of the items themselves, e.g., product category, description, specifications

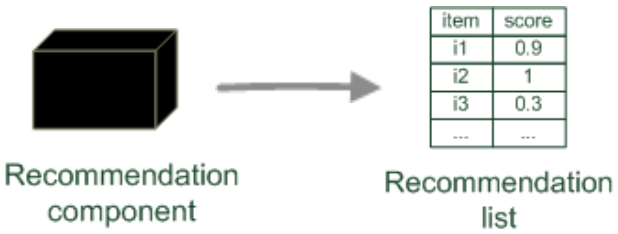
Netflix's prize: Winning algorithm

- Prize winner: ensemble of multiple recommendations models
  - Many different algorithms blended together
  - Principle: combining diverse models often outperforms any single model
- We will study main approaches and a few key methodologies
- Netflix's solution combined:
  - Collaborative filtering
  - Matrix factorization
  - Neighborhood models
  - Other predictive algorithms

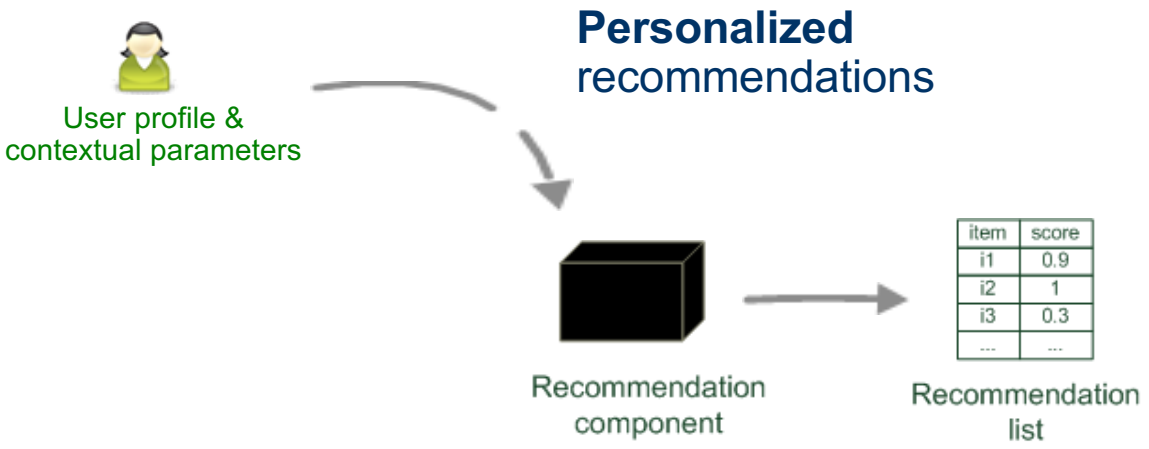
https://dl.acm.org/doi/pdf/10.1145/2843948

Approaches to recommender systems

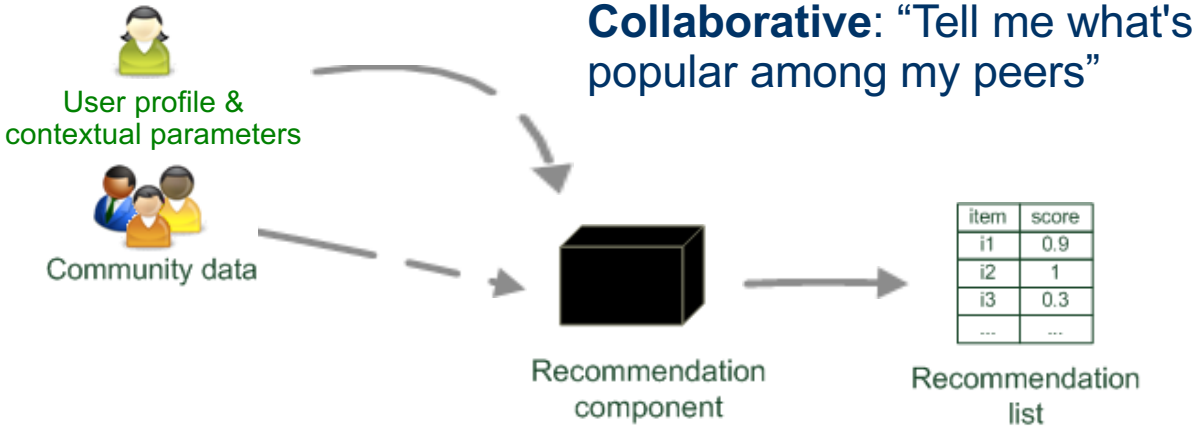
Recommender systems reduce information overload by estimating relevance



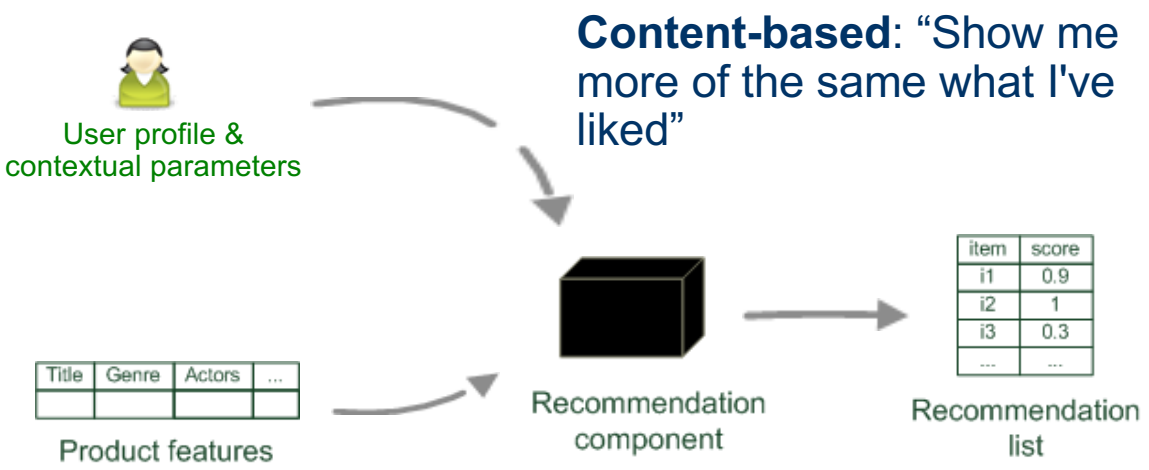
Approaches to recommender systems



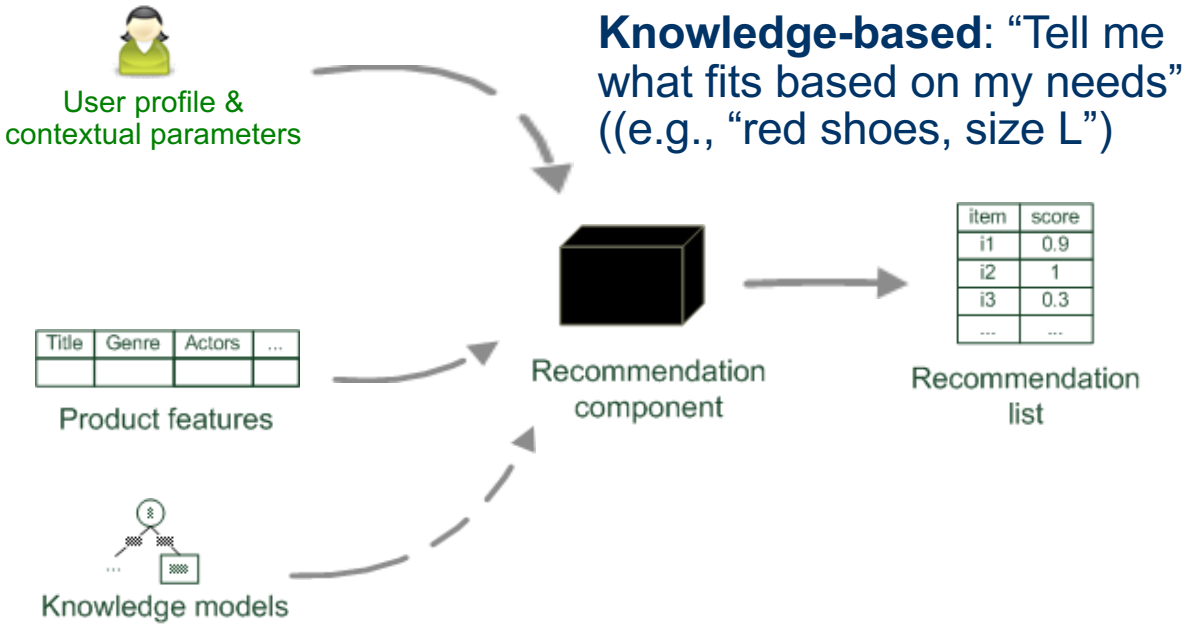
Approaches to recommender systems



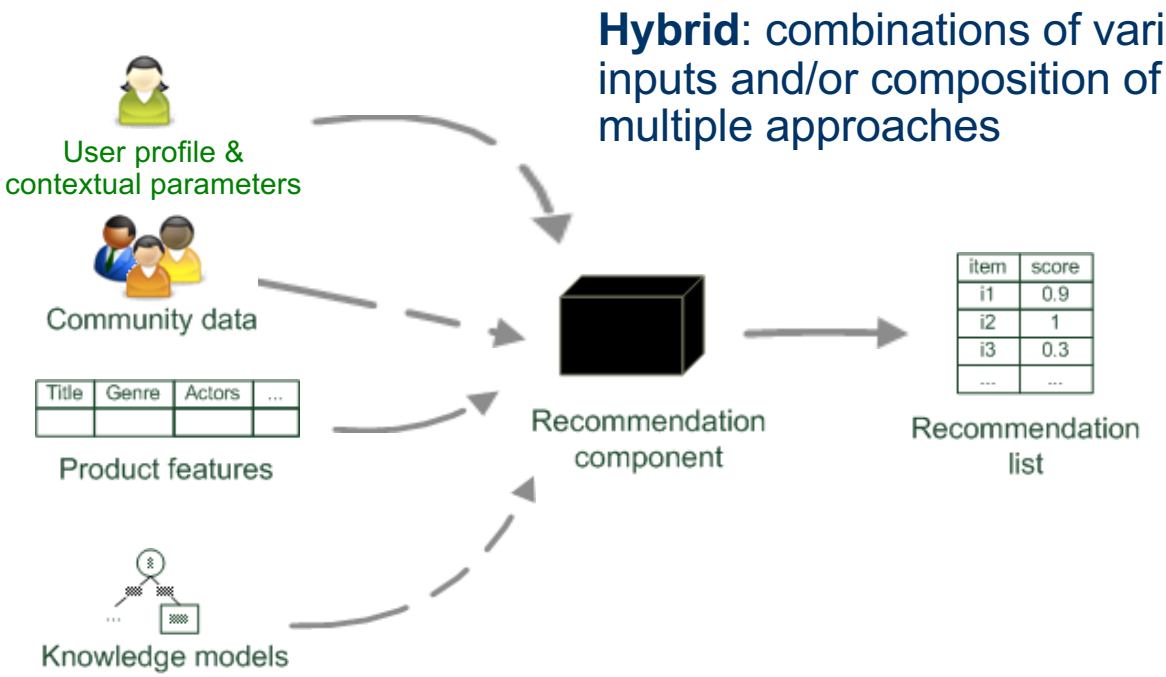
Approaches to recommender systems



Approaches to recommender systems



Approaches to recommender systems



Paradigms of recommender systems

- We analyze:
- Collaborative filtering
  - Make predictions about the interests of a user (filter) by using preferences of many users (collaborative)
  - Assumption: similar users will have similar ratings
- Content-based filtering
  - Recommend items by comparing item content with user profile

Collaborative filtering

- Use other users' ratings to predict a user's rating for an item the user hasn't rated yet
  - Predictions are built upon the known ratings of other users, who have similar ratings with the user
  - E.g., predict Eva's rating for Inception

Ratings matrix

	Forrest Gump	Godfather	Inception	Jaws
Amy	5		4	3
Bob	3	5	2	5
Carl		3	5	4
Dan	4	5	4	
Eva	4	4	?	3

Unknown rating we want to predict

Collaborative filtering

- Let's calculate mean values for ratings

	Forrest Gump	Godfather	Inception	Jaws	Mean rating
Amy	5		4	3	4.00
Bob	3	5	2	5	3.75
Carl		3	5	4	4.00
Dan	4	5	4		4.33
Eva	4	4		3	3.67

User: u

Movie: i

Rating: 1-5

n_u: number of ratings by user u

Mean rating $\mu_u = \frac{1}{n_u} \sum_i r_{ui}$

Collaborative filtering

- Let's measure similarity between users
  - Similarity values range from -1 and 1
    - 1: perfectly similar (same rating behavior)
    - 0: no similarity (ratings uncorrelated)
    - -1: perfectly dissimilar (opposite ratings)
  - Later, we will see how

Similarity matrix

	Amy	Bob	Carl	Dan	Eva
Amy	1	-0.54	0.00	-0.29	0.87
Bob	-0.54	1	-0.82	0.78	-0.32
Carl	0.00	-0.82	1	-0.87	-0.29
Dan	-0.29	0.78	-0.87	1	0.17
Eva	0.87	-0.32	-0.29	0.17	1

Collaborative filtering

	Forrest Gump	Godfather	Inception	Jaws	Mean rating
Amy	5		4	3	4.00
Bob	3	5	2	5	3.75
Carl		3	5	4	4.00
Dan	4	5	4		4.33
Eva	4	4		3	3.67

	Amy	Bob	Carl	Dan	Eva
Amy	1	-0.54	0.00	-0.29	0.87
Bob	-0.54	1	-0.82	0.78	-0.32
Carl	0.00	-0.82	1	-0.87	-0.29
Dan	-0.29	0.78	-0.87	1	0.17
Eva	0.87	-0.32	-0.29	0.17	1

- Can predict Eva's rating for Inception:

  $3.67+[0.87(4-4)-0.32(2-3.75)-0.29(5-4)+0.17(4-4.33)]/(0.87-0.32-0.29+0.17) = 4.17$

- Eva will like Inception more than average

Exploit similarities between users

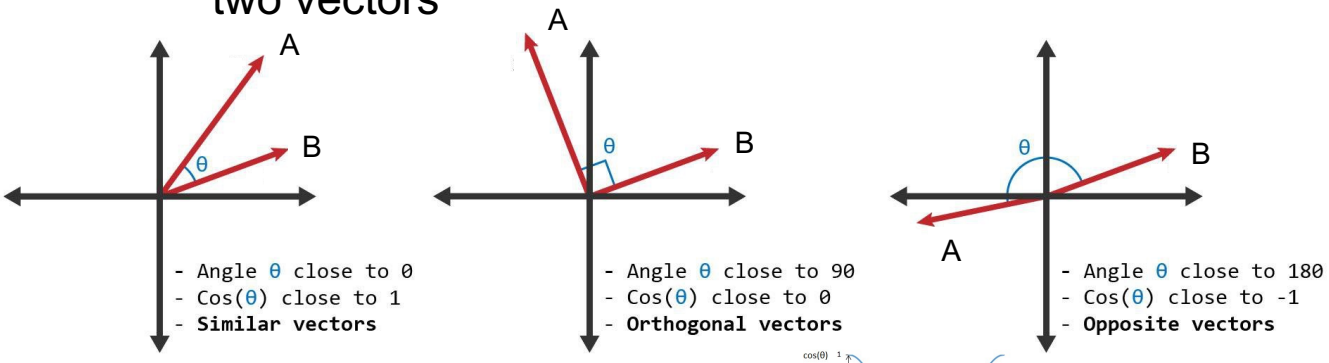
	Forrest Gump	Godfather	Inception	Jaws	Mean rating
Amy	5		4	3	4.00
Bob	3	5	2	5	3.75
Carl		3	5	4	4.00
Dan	4	5	4		4.33
Eva	4	4		3	3.67

	Amy	Bob	Carl	Dan	Eva
Amy	1	-0.54	0.00	-0.29	0.87
Bob	-0.54	1	-0.82	0.78	-0.32
Carl	0.00	-0.82	1	-0.87	-0.29
Dan	-0.29	0.78	-0.87	1	0.17
Eva	0.87	-0.32	-0.29	0.17	1

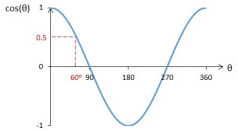
- Consider suggesting “Inception” to Eva, since Amy rated it high, and Eva and Amy have similar preferences
- This technique is called user-based collaborative filtering

Collaborative filtering: measuring similarity

- We consider cosine similarity
  - Represent each user as a vector of ratings in an *n*-dimensional space (*n* = number of items)
  - Measure similarity using cosine of the angle between two user vectors (figure: *n*=2)
  - It measures the similarity in the directions of the two vectors



Cosine function



Cosine similarity

- We consider cosine similarity
  - Cosine of angle θ between two vectors A and B in an *n*-dimensional space
  - Similarity score ranges between -1 and 1
    - cos(0°) = 1: perfectly similar
    - cos(90°) = 0: orthogonal (or uncorrelated)
    - cos(180°) = -1: perfectly dissimilar
- For a pair of users: the larger the score, the closer their taste

$$cosine\ similarity = \cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

  - A_i and B_i are components of vector *A* and *B*, respectively

Cosine similarity: Example

- To deal with biases in the ratings (e.g., a critic always gives low ratings), let's normalize ratings by subtracting the user's average rating from each of their rating
  - Do not include missing ratings in the average

	Forrest Gump	Godfather	Inception	Jaws	Mean rating
Amy	5		4	3	4.00
Bob	3	5	2	5	3.75
Carl		3	5	4	4.00
Dan	4	5	4		4.33
Eva	4	4		3	3.67

	Forrest Gump	Godfather	Inception	Jaws
Amy	5-4=1		4-4=0	3-4=-1
Bob	3-3.75=-0.75	5-3.75=1.25	2-3.75=-1.75	5-3.75=1.25
Carl		3-4=-1	5-4=1	4-4=0
Dan	4-4.33=-0.33	5-4.33=0.67	4-4.33=-0.33	
Eva	4-3.67=0.33	4-3.67=0.33		3-3.67=-0.67

Cosine similarity: Example

Normalized ratings matrix

	Forrest Gump	Godfather	Inception	Jaws
Amy	1		0	-1
Bob	-0.75	1.25	-1.75	1.25
Carl		-1	1	0
Dan	-0.33	0.67	-0.33	
Eva	0.33	0.33		-0.67

Note: after normalization, a negative number means below average rating and a positive number means above average rating given by that user

- Now compute the cosine similarity for each pair (A, B) of users

$$cosine\ similarity = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- E.g., cosine similarity between Amy and Bob:

$$\frac{1*(-0.75)+0*(-1.75)+(-1)*1.25}{\sqrt{1^2+0^2+(-1)^2}\sqrt{(-0.75)^2+1.25^2+(-1.75)^2+1.25^2}} = \frac{-2}{3.6742} = -0.5443$$

- Only include movies that both users have rated

Cosine similarity: Example

Normalized ratings matrix

	Forrest Gump	Godfather	Inception	Jaws
Amy	1		0	-1
Bob	-0.75	1.25	-1.75	1.25
Carl		-1	1	0
Dan	-0.33	0.67	-0.33	
Eva	0.33	0.33		-0.67

- E.g., cosine similarity between Amy and Eva:

$$\frac{1*(0.33)+(-1)*(-0.67)}{\sqrt{1^2+0^2+(-1)^2}\sqrt{0.33^2+0.33^2+(-0.67)^2}} = \frac{1}{1.1547} = 0.8660$$

- We obtain the following similarity values:

Similarity matrix

	Amy	Bob	Carl	Dan	Eva
Amy	1	-0.54	0.00	-0.29	0.87
Bob	-0.54	1	-0.82	0.78	-0.32
Carl	0.00	-0.82	1	-0.87	-0.29
Dan	-0.29	0.78	-0.87	1	0.17
Eva	0.87	-0.32	-0.29	0.17	1

How to predict the rating

- Given the similarity matrix, the predicted rating $\hat{r}_{ui}$ for user *u* on item *i* is given by:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \neq u} s_{uv}(r_{vi} - \mu_v)}{\sum_{v \neq u} s_{uv}}$$

s_{uv}: similarity between u and v
μ_u: mean rating for u

	Forrest Gump	Godfather	Inception	Jaws	Mean rating
Amy	5		4	3	4.00
Bob	3	5	2	5	3.75
Carl		3	5	4	4.00
Dan	4	5	4		4.33
Eva	4	4		3	3.67

	Amy	Bob	Carl	Dan	Eva
Amy	1	-0.54	0.00	-0.29	0.87
Bob	-0.54	1	-0.82	0.78	-0.32
Carl	0.00	-0.82	1	-0.87	-0.29
Dan	-0.29	0.78	-0.87	1	0.17
Eva	0.87	-0.32	-0.29	0.17	1

- Eva's rating for Inception:

$$\hat{r}_{Eva,Inception} = 3.67 + \frac{[0.87(4-4) - 0.32(2-3.75) - 0.29(5-4) + 0.17(4-4.33)]}{(0.87 - 0.32 - 0.29 + 0.17)} = 4.1674$$

More on the predicted rating formula

- Given the similarity matrix, the predicted rating $\hat{r}_{ui}$ for user *u* on item *i* is given by:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \neq u} s_{uv}(r_{vi} - \mu_v)}{\sum_{v \neq u} s_{uv}}$$

s_{uv}: similarity between u and v
μ_u: mean rating for u

- What does this formula mean?
  - μ_u is a baseline predictor for $\hat{r}_{ui}$
  - Weighted average approach: we multiply each normalized rating by similarity (which tells how similar the users are)
  - We sum weighted differences from users, then add to baseline: the heavier the weight, the more the rating would matter

More on the predicted rating formula

- If you get a negative prediction, clip the result to the rating scale used
- Example:
  - If $\hat{r}_{ui}$ = -0.3 and ratings are 1-5 → set to 1 (minimum allowed)

Collaborative filtering: all the steps so far

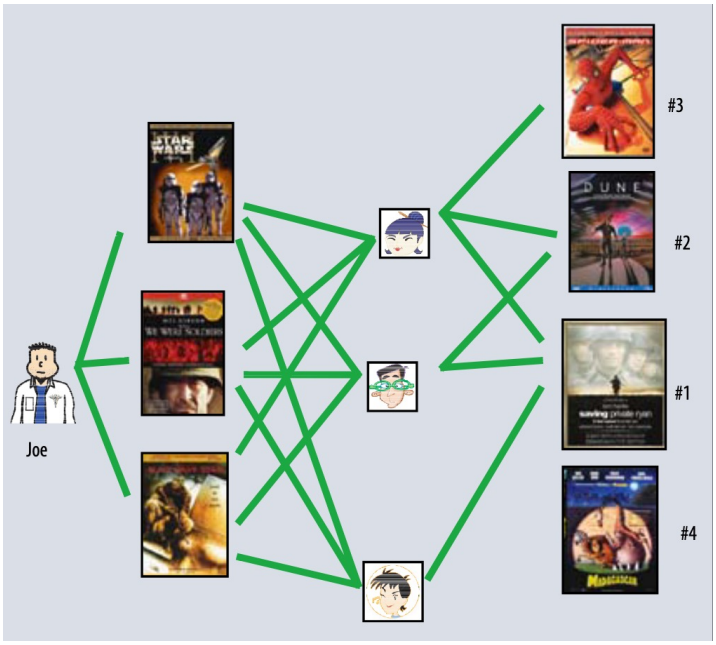
1. Calculate the mean rating for each user
2. Normalize the ratings matrix
3. Compute the cosine similarity for each pair of users
4. Compute the predicted rating(s)

Scaling collaborative filtering: neighborhood

- Real problems are much larger and sparser than our example
- When dealing with large dataset, we do not consider all users (too expensive!) but only like-minded users that is, users with high similarity (neighbors set K) to the target user
  - Find top-K users (neighbors) most similar to the target user u and use only their ratings to predict: weighted average of the ratings given only by similar users

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in K} s_{uv}(r_{vi} - \mu_v)}{\sum_{v \in K} s_{uv}}$$

Considering neighbors: example



- Joe likes the 3 movies on the left
- To make a prediction for him, find similar users who also liked those 3 movies, check what other movies they liked, and rank recommendations based on popularity among similar users
  - All three liked Saving Private Ryan, so that is the first recommendation; two liked Dune, so that is the second, and so on

Exercise: User-based collaborative filtering

- Applying user-based collaborative filtering, predict Eric's rating for Titanic

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall•E
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2	?	3	5	4
Diane	4	3	5	3	

Item-based collaborative filtering

- So far, we considered user-based collaborative filtering
- Collaborative filtering can be *item-based*
  - Assumes that items are similar if they are liked by similar users
- How it works:
  1. Compute similarities between items using ratings
  2. Recommend items that are similar to those a user has previously liked
  - To compute item similarity: start from the normalized ratings matrix, transpose it (swapping rows and columns), and then compute cosine similarity between pairs of items

Item-based collaborative filtering

- Item-based collaborative filtering is scalable and stable because:
  - Items change less frequently than users
  - Item similarities can be precomputed

Exercise: Item-based collaborative filtering

- Applying item-based collaborative filtering, predict U1's rating for M3

	M1	M2	M3	M4
U1	5	4	?	?
U2	5	5	4	?
U3	?	4	5	3
U4	2	?	1	5

Netflix's prize solution

- Netflix's prize solution uses **matrix factorization**: a more complex collaborative filtering algorithm than the user-based one
- Idea: represent users and items in a lower dimensional latent space, exploiting relationships among users and items (that could also be hidden that is, *latent*) to predict ratings
  - Gentle introduction: watch https://www.youtube.com/watch?v=ZspR5PZemcs

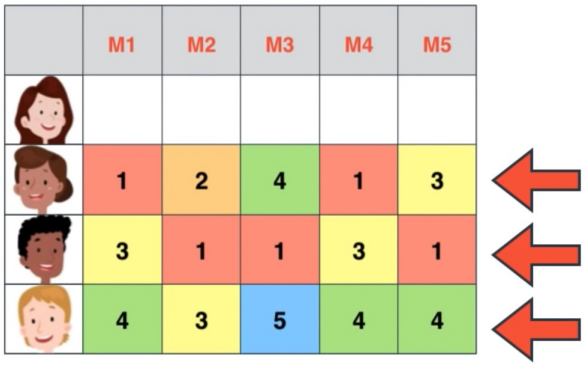
    and read "Recommendation Systems: Collaborative Filtering using Matrix Factorization - Simplified" https://medium.com/sfu-cspmp/recommendation-systems-collaborative-filtering-using-matrix-factorization-simplified-2118f4ef2cd3
  - More details: read "Matrix factorization techniques for recommender systems" https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

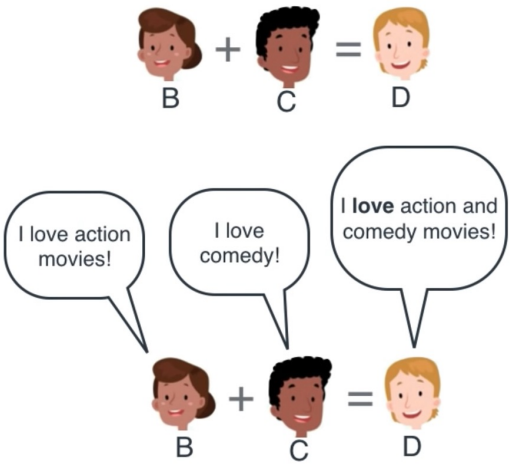
Matrix factorization: dependencies

- Let's analyze some example of row and column dependencies
  - Relationship is easy: two very similar movies

Matrix factorization: dependencies

- Let's analyze some example of row and column dependencies
  - Relationship is less easy





Matrix factorization: dependencies

- Let's analyze some example of row and column dependencies
  - Another example of hidden relationship

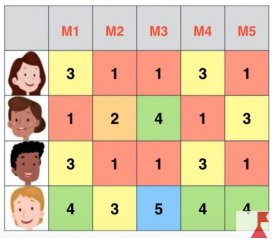
M5 = Average(M2, M3)

M5 = Average(M2, M3)

Matrix factorization: the idea

- We need a mathematical tool to capture all the dependencies
- Idea: decompose the large, sparse R matrix into two smaller, dense matrices P and Q
  - We assume that the rating matrix R can be approximated as the product of these two matrices
  - Matrix factorization because we decompose R into the products of two matrices P and Q

$$R \approx P \times Q = \hat{R} =$$



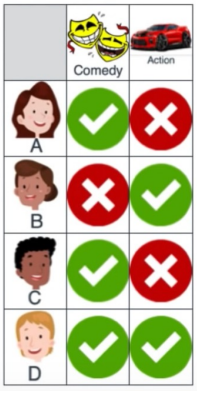
Matrix factorization

- We need a mathematical tool to capture all the dependencies

Movie features Q

	M1	M2	M3	M4	M5
Comedy	3	1	1	3	1
Action	1	2	4	1	3

User features P



	M1	M2	M3	M4	M5
A	3	1	1	3	1
B	1	2	4	1	3
C	3	1	1	3	1
D	4	3	5	4	4

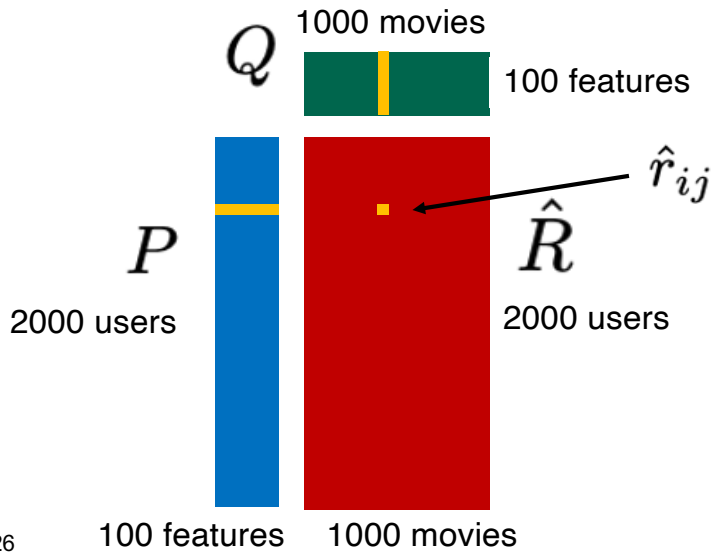
Ratings R ≈ P×Q

Matrix factorization: dot product

- How to get each value of matrix $\hat{R}$? We use the dot product

predicted rating

$$\hat{r}_{ij} = \boldsymbol{p}_i \cdot \boldsymbol{q}_j = \sum_{k=1}^{d} p_{ik} q_{kj}$$



Matrix factorization: dot product

- We can use the entries of the product $P \times Q$ to estimate the corresponding unknown entries (i.e., the predicted ratings) in the matrix $\hat{R}$

Movie features Q

	M1	M2	M3	M4	M5
F1	3	1	1	3	1
F2	1	2	4	1	3

User features P

	F1	F2
A	1	0
B	0	1
C	1	0
D	1	1

	M1	M2	M3	M4	M5
A	3	?	1		1
B	1		4	1	
C	3	1		3	1
D		3		4	4

Matrix factorization: dot product

- Let's use the dot product to predict the unknown ratings
- Predict user A rating for movie M2 using the dot product

Movie features Q

F1	3	1	1	3	1
F2	1	2	4	1	3

$$\hat{r}_{12} = (1,0) \cdot (1,2) = \\ = 1 \times 1 + 0 \times 2 = 1$$

User features P

	F1	F2
A	1	0
B	0	1
C	1	0
D	1	1

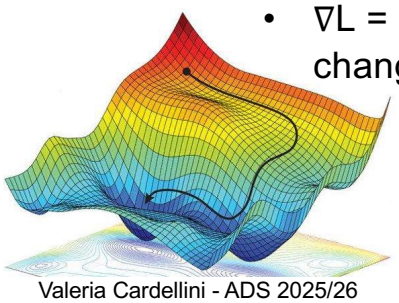
	M1	M2	M3	M4	M5
A	3	1	1	3	1
B	1	2	4	1	3
C	3	1	1	3	1
D	4	3	5	4	4

Matrix factorization: learning P and Q

- How do we find P and Q?
  - Use Machine Learning
- Idea: learn latent factors by fitting known ratings
- Steps
  - Initialize matrices P and Q with random values
  - Compute prediction error (difference between predicted and known ratings)
  - Minimize the error *iteratively* by adjusting some element of P or Q

Matrix factorization: learning P and Q

- How do we find P and Q?
- Optimization method: gradient descent (GD)
  - The gradient indicates the direction in which the error increases the most
  - Update model parameters in the opposite direction to reduce error
  - Gradient descent update rule: $\theta \leftarrow \theta - \eta \nabla L$
    - θ = model parameters (in matrix factorization, P and Q)
    - η = learning rate (controls how big each update step is)
    - ∇L = gradient of the loss function (tells how the error changes with respect to parameters)



"SGD is like a drunkard trying to find his way home."
Geoffrey Hinton

Matrix factorization: learning P and Q

- Initialize matrices P and Q with random values
- Compute the squared error (loss)

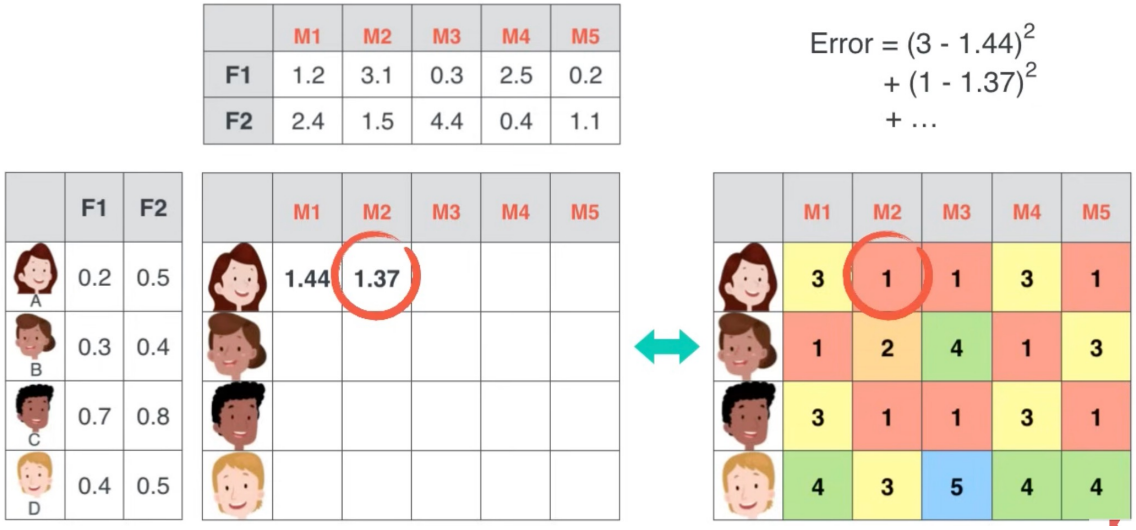
$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = \left(r_{ij} - \sum_{k=1}^{d} p_{ik} q_{kj}\right)^2$$

d = number of latent features

  - The total squared error is a good measure of how close the product PQ is to the given ratings matrix
- Minimize the error *iteratively* by adjusting *some element* of P or Q
  - Gradient descent is stochastic
    - Instead of using all ratings to compute the error, it uses one (or a few) ratings at a time, chosen randomly

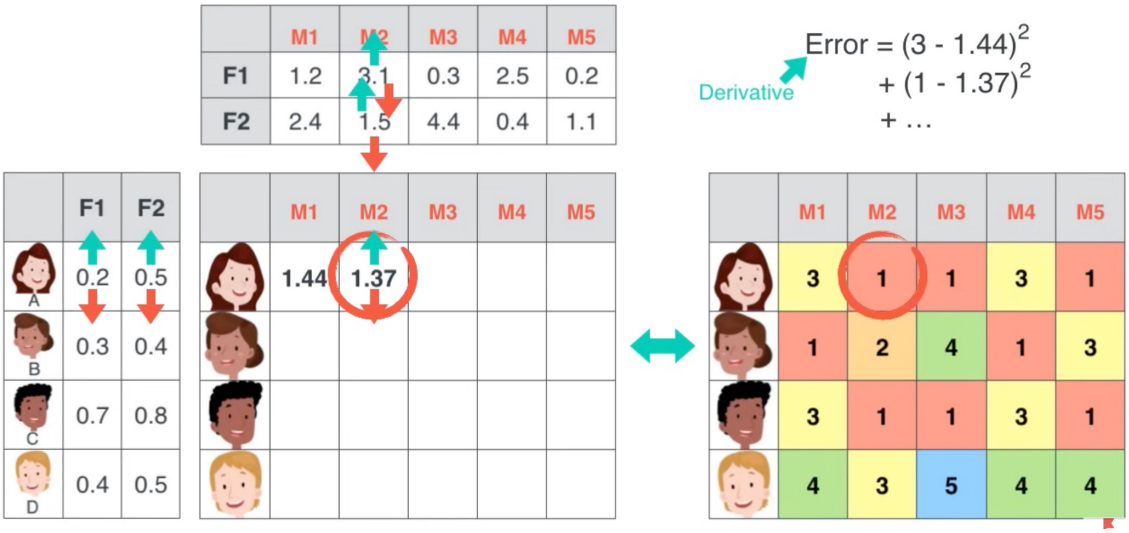
Matrix factorization: learning P and Q

- Consider the example: how to decrease the error?



Matrix factorization: learning P and Q

- Consider the example: how to decrease the error?



Matrix factorization: after learning P and Q

- We can now predict the ratings

	M1	M2	M3	M4	M5
F1	3	1	1	3	1
F2	1	2	4	1	3

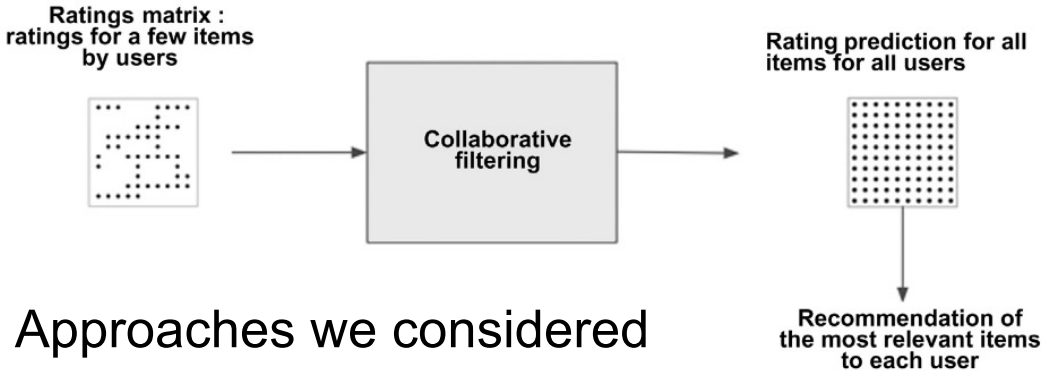
	F1	F2
A	1	0
B	0	1
C	1	0
D	1	1

	M1	M2	M3	M4	M5
A	3	1	1	3	1
B	1	2	4	1	3
C	3	1	1	3	1
D	4	3	5	4	4

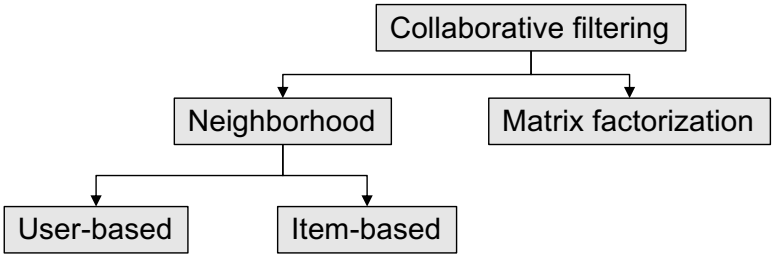
M3

Collaborative filtering: summing up

- Overall picture



- Approaches we considered



Limitations of ratings

- Explicit feedback (ratings) is useful
- But ratings are
  - Sparse: most users don't rate
  - Noisy: ratings may be inconsistent or unreliable
  - Biased: some users always rate high or low; context (mood, time) affects ratings
- Recommendations cannot rely only on ratings
- Need to include implicit feedback, e.g.,
  - Watch time, clicks, replays, skips
  - Add-to-cart, saves, shares

Content-based recommender systems

- Idea: leverage the features (attributes) of items
  - Item similarity is determined by comparing item features
  - Items with similar attributes are more likely to be recommended
- Example: movie features
  1. Cast: users may prefer favorite actors
  2. Director: preference for specific directors
  3. Release year: some prefer classic movies, others only the latest releases
  4. Genre: users may prefer comedies, others dramas or romances

Content-based filtering

- If we know that Amy likes “Men In Black”
  - Directed by Barry Sonnenfeld
  - Classified in the genres of action, adventure, sci-fi and comedy
  - Stars actor Will Smith
- Consider recommending to Amy:
  - Barry Sonnenfeld’s movie “Get Shorty”
  - “Jurassic Park”, which is in the genres of action, adventure, and sci-fi
  - Will Smith’s movie “Hitch”

- This technique is called content-based filtering

https://www.ibm.com/think/topics/content-based-filtering

Collaborative filtering vs. content-based filtering

- Collaborative filtering
  - ✓ Item-domain independent: can predict ratings for items without knowing their features, only user ratings are needed
  - ✗ Data-hungry: requires many ratings to make accurate recommendations
  - ✗ Computationally intensive: millions of items and users
  - ✗ Cold-start problem: difficult to recommend for new users (or items)
    - Mitigation: ask new users to select or rate some items during sign-up to bootstrap recommendations

Collaborative filtering vs. content-based filtering

- Content-based filtering
  - ✗ Item-domain dependent
  - ✓ Data-efficient: can start with little data about a user
  - ✗ Limited in scope: tends to recommend only similar items
  - ✓ No cold-start for users, only needs to know the item features
    - Example: movie X is new
    - Genre = Comedy, Director = Z, Actor = Y
    - User U likes comedies and actor Y
    - CBF can recommend Movie X because features match user preferences

Hybrid recommendation: example

- Step 1: user-based collaborative filtering
  - Find similar users
  - Example: Amy and Eva have similar preferences
- Step 2: content-based filtering
  - Look at items both users liked
  - Example: both liked Forrest Gump
  - Find similar items based on features (e.g., genre)
- Step 3: recommendation
  - Rain Man has similar characteristics to Forrest Gump
  - Recommend Rain Man to both Amy and Eva
  - Even though neither has seen it before

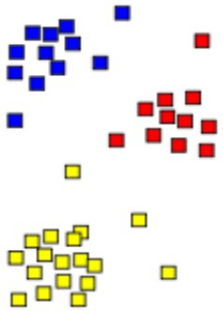
Clustering movies by genres

- Consider the MovieLens dataset
  - A movie recommendation website https://movielens.org/
- Movies in the dataset are classified as belonging to different genres

Unknown	Action	Adventure	Animation	Children's
Comedy	Crime	Documentary	Drama	Fantasy
Film Noir	Horror	Musical	Mystery	Romance
Sci-Fi	Thriller	War	Western	

- Each movie may belong to multiple genres
- How can we find groups of movies with similar sets of genres?

Clustering



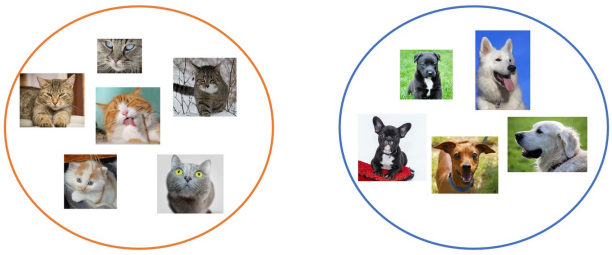
- Clustering is the process of examining a collection of "points," and grouping points into "clusters" based on some distance measure
- Cluster analysis is a type of **unsupervised learning**
  - Goal is to segment data into similar groups rather than making prediction
  - You can also cluster data first and then build a separate predictive model for each cluster instead of the whole dataset
    - Be careful to avoid overfitting your model
    - This works best with large datasets

Break: Machine learning (ML)

- ML is the study of computer algorithms that automatically improve through experience and data
  - «Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed», Arthur Samuel (1959)
- ML approaches
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

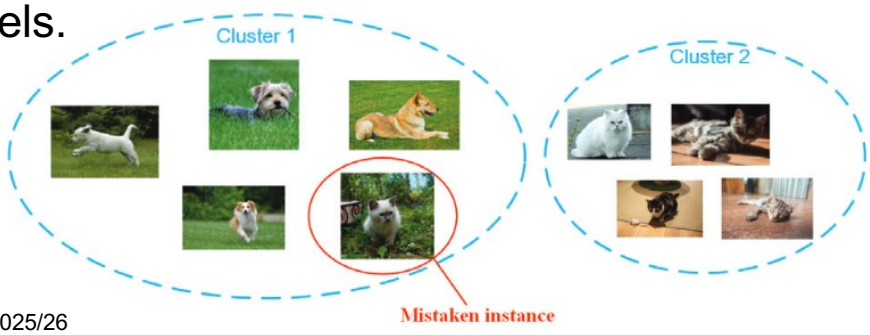
ML approaches: supervised learning

- **Supervised learning**: task of learning a general rule that maps an input to an output based on example input-output pairs
  - It infers a function from *human-labelled training data* (a set of *training examples)*
- Example: image classification
  - Define target classes (e.g., cats and dogs)
  - Train a model using labelled example images to recognize these classes



ML approaches: unsupervised learning

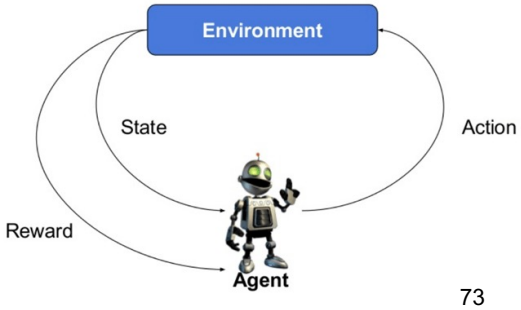
- **Unsupervised learning**: no labelled training data are provided. The ML algorithm find structure in the data on this own
  - The algorithm detects previously undetected patterns in a data with no pre-existing labels and minimal human supervision
- Example: clustering
  - Group similar data points together without prior labels.



ML approaches: reinforcement learning

- **Reinforcement learning**: an intelligent agent lears to take actions in an environment to maximize cumulative reward
  - Differs from supervised learning in not relying on labelled input/output pairs
  - Focus is on balancing *exploration* (discovering new possibilities) and *exploitation* (using existing knowledge)
- Example: AlphaGo
  - Used deep RL

Typical RL scenario

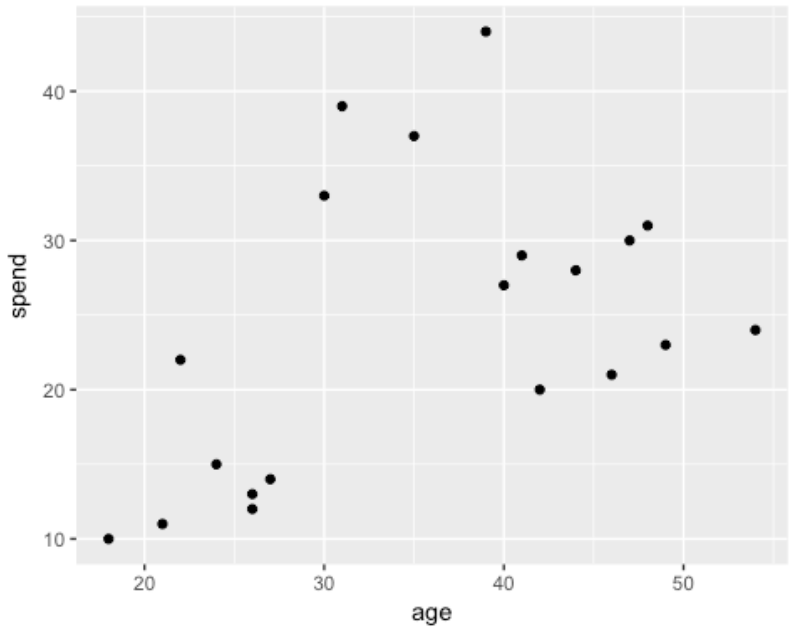


Examples of cluster analysis

- Customer segmentation: group customers based on similar purchasing behavior or demographics
- Stock market clustering: group stocks based on performance metrics (e.g., price movement, volatility)
- Dimensionality reduction: reduce dataset complexity by grouping observations with similar values

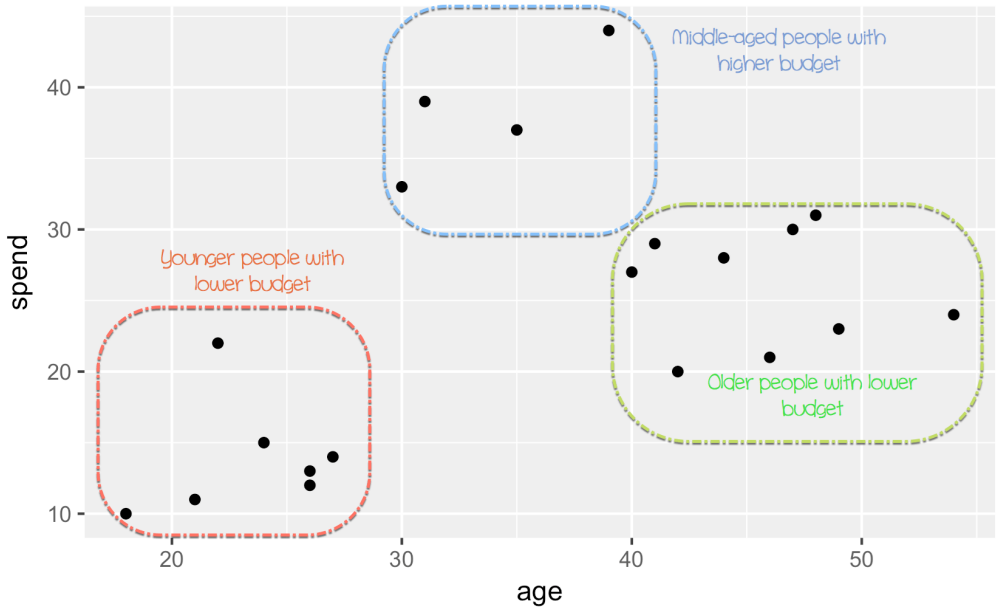
Example of cluster analysis

- Total spend and age of customers



Example of cluster analysis

- Total spend and age of customers



Example of cluster analysis

- Place pizza delivery on the map

Source: https://www.youtube.com/watch?v=IpGxLWOIZy4

Clustering algorithms

- Different algorithms for clustering
  - Vary in how they define clusters and how to find them
- We will study:
  - **Hierarchical clustering**: belongs to hierarchical or agglomerative class of clustering algorithms
  - **K-means**: belongs to point assignment class of clustering algorithms

Distance between points

- We first need to define distance metric between data points
- Most popular is Euclidean distance
  - Distance between two points *p* and *q* is given by:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$

where n is the number of independent variables (dimensions) in the space

Distance between points: Example

- Enconding movie categories
- Toy Story
  - Categories (see slide 63): Animation, Comedy, and Children
  - Encoded as (0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0)
- Batman Forever
  - Categories: Action, Adventure, Comedy, and Crime
  - Encoded as (0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0)

$$d = \sqrt{(0-0)^2 + (0-1)^2 + (0-1)^2 + (1-0)^2 + (1-0)^2 + (1-1)^2 + (0-1)^2 + (0-0)^2 + \ldots + (0-0)^2}$$
$$= \sqrt{0+1+1+1+1+0+1+0\ldots+0} = \sqrt{5}$$

Distance between points

- Other popular distance metrics
- Manhattan distance
  - The sum of absolute differences between coordinates
  - Based on grid-like street geography of Manhattan

$$d_{Manhattan}(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$

- Chebyshev distance
  - Measures the greatest absolute difference between coordinates

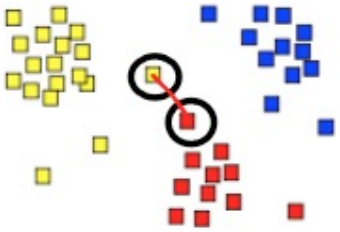
$$d_{Chebyshev}(p,q) = \max_i |p_i - q_i|$$

Distance between cluster

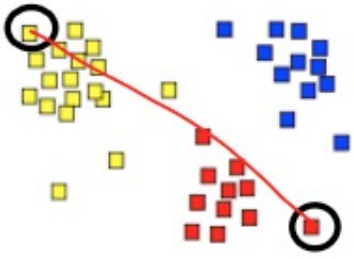
- How to define the distance between two clusters?
- Which are the most representative data points for the cluster between which we can calculate the distance?

Distance between clusters

- Minimum distance
  - Distance between the closest points from each cluster

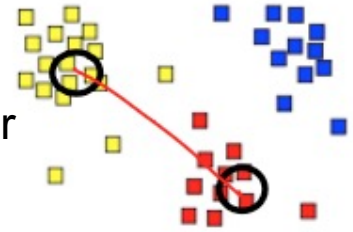


- Maximum distance
  - Distance between the farthest points from each cluster



Distance between clusters

- Centroid distance
  - The distance between the centroids (or centers) of the clusters



- Cluster centroid: the point that represents the mean position of all data points, calculated across all dimensions
  - Example: for points (-1, 10, 3), (0, 5, 2), and (1, 20, 10), the centroid is the mean of the x, y, and z coordinates

    ((-1+0+1)/3, (10+5+20)/3, (3+2+10)/3) = (0, 35/3, 5)
- Note: the centroid does not have to be - and rarely is - any of the original data points

Normalize data

- Distance metrics are highly influenced by scale of data, so it is common to normalize (or standardized) first
- Example with MovieLens dataset: genre data is on the same scale (0 or 1) and normalization is not necessary
- "Box office revenue": we would need to normalize
- Why normalize?
  - Consistency and fair comparison

Hierarchical clustering

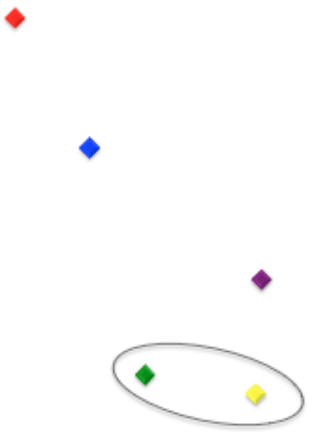
- Bottom-up approach
- Start with every point as its own cluster

5 clusters

Hierarchical clustering

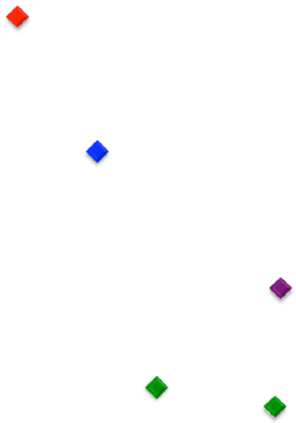
- Combine two closest clusters (Euclidean distance, centroids)

At each iteration, find the two most similar clusters (those with the smallest distance between them) and merge them into a new, larger cluster



Hierarchical clustering

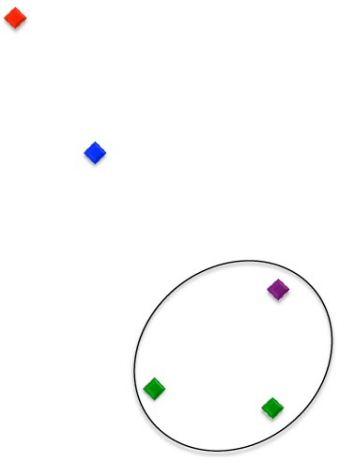
- Combine two closest clusters (Euclidean distance, centroids)



4 clusters

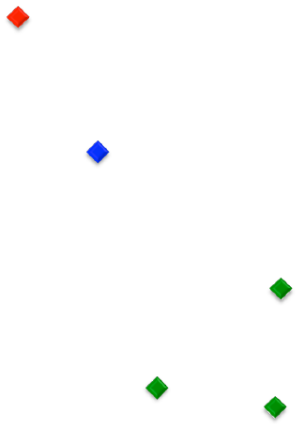
Hierarchical clustering

- Combine two closest clusters (Euclidean distance, centroids)



Hierarchical clustering

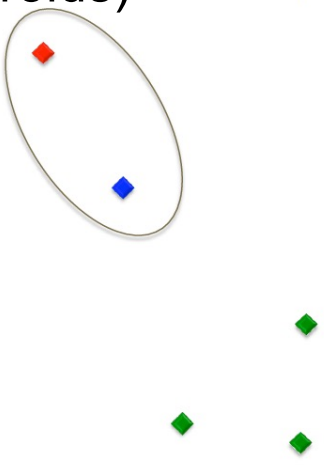
- Combine two closest clusters (Euclidean distance, centroids)



3 clusters

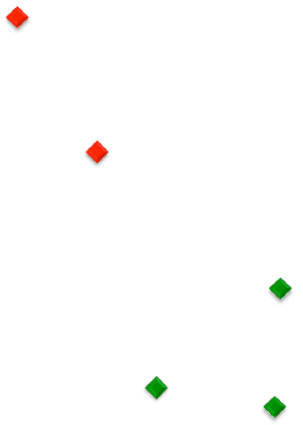
Hierarchical clustering

- Combine two closest clusters (Euclidean distance, centroids)



Hierarchical clustering

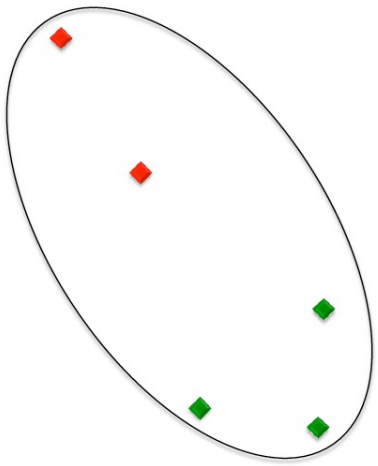
- Combine two closest clusters (Euclidean distance, centroids)



2 clusters

Hierarchical clustering

- Combine two closest clusters (Euclidean distance, centroids)



Hierarchical clustering

- Combine two closest clusters (Euclidean, Centroid)

The algorithm continues until all data points are merged into one single cluster

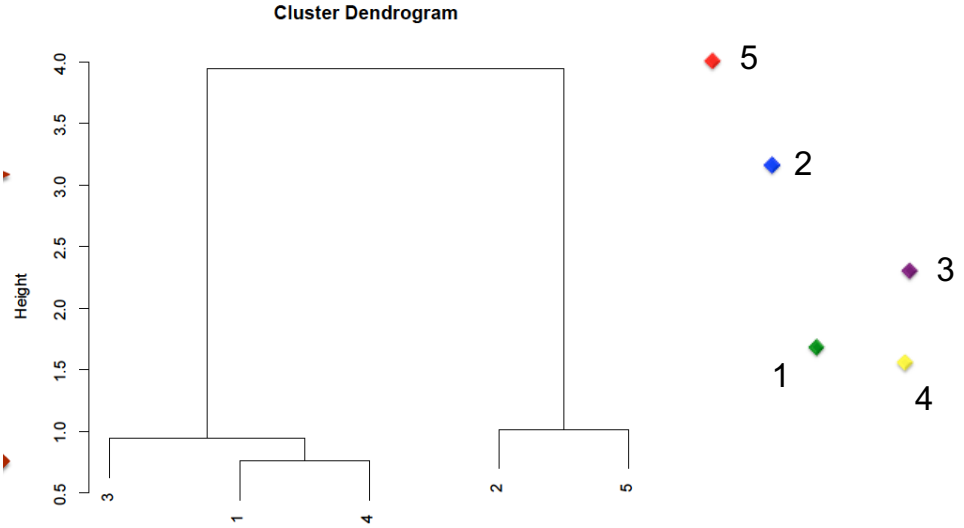
1 cluster

Hierarchical clustering: dendogram

- The result is a tree-like structure called cluster dendogram
  - Show hierarchical relationship between clusters

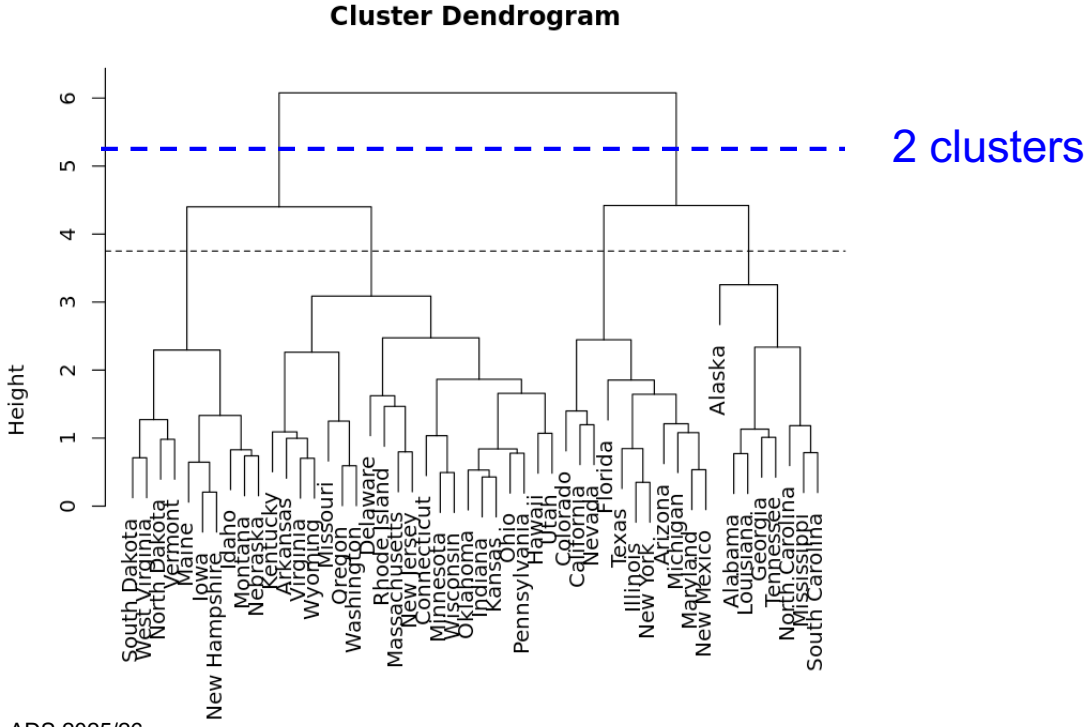
Height of dendrogram: order in which clusters were joined

Height of vertical lines: distance between the points (or clusters) that are merged



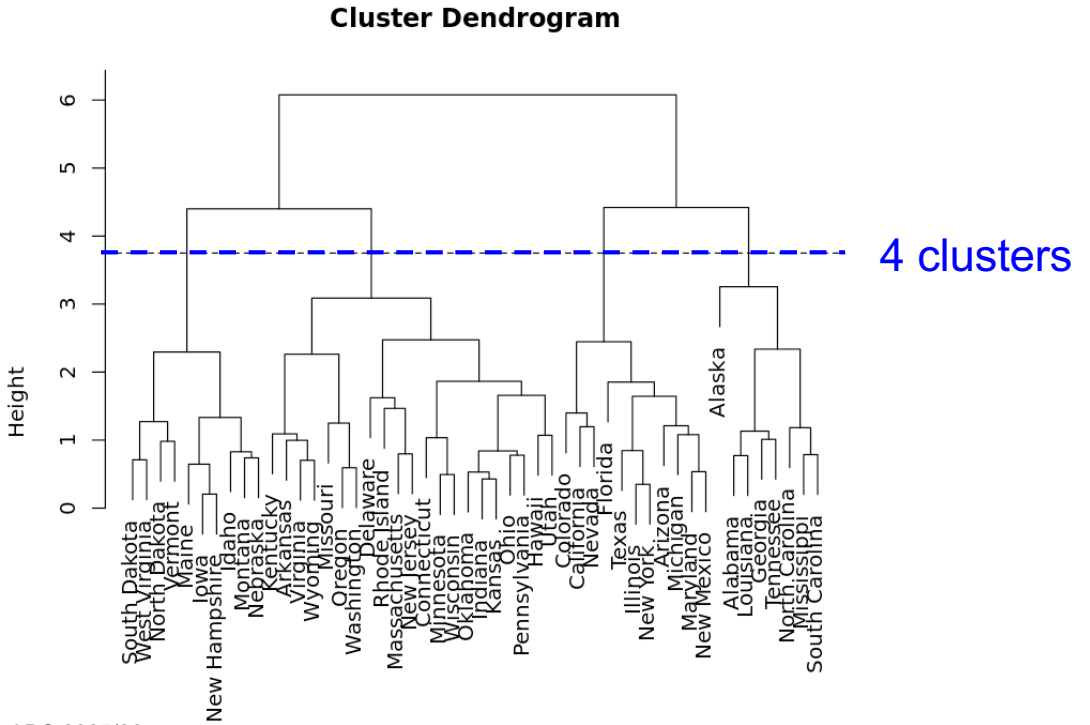
Hierarchical clustering: select clusters

- Example: cluster analysis of US states according to the number of arrests made



Hierarchical clustering: select clusters

- Example: cluster analysis of US states according to the number of arrests made



Hierarchical clustering: pros and cons

✓ Simple to understand and implement

✗ But once two data points or clusters are merged, they cannot be separated

"Once the damage is done, it can never be repaired." (Kaufman, 1990)

k-means clustering

- *k*-means clustering at a glance:
  - Initialize by choosing the desired number of clusters *k*
  - *Iterative* algorithm
  - Each iteration consists of 2 steps:
    1. ***Cluster assignment***
    2. ***Centroid update***

k-means clustering

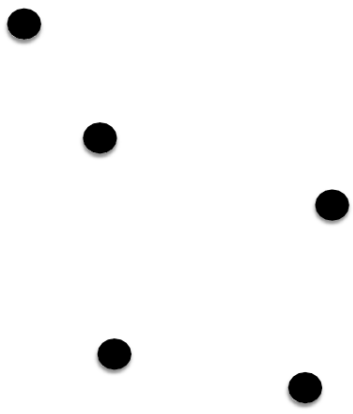
- *k*-means clustering algorithm
  1. Specify the desired number of clusters *k*
  2. Randomly assign each data point to a cluster
  3. Compute the cluster centroids
  4. Re-assign each point to the closest centroid (e.g., using Euclidian distance)
  5. Re-compute the cluster centroids
  6. Repeat steps 4 and 5 until convergence (no further improvement)

- See animation: https://en.wikipedia.org/wiki/File:K-means_convergence.gif

k-means clustering: Example

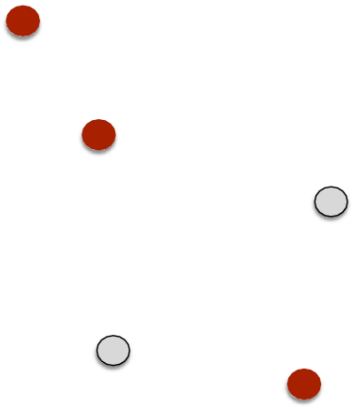
1. Specify the desired number of clusters *k*

Let's choose *k*=2



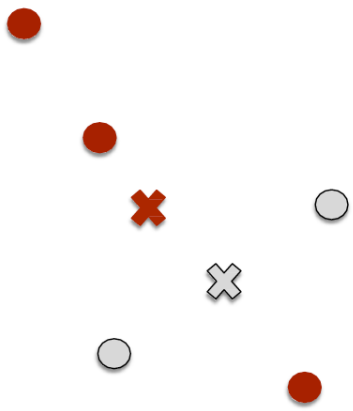
k-means clustering: Example

1. Specify the desired number of clusters *k*
2. Randomly assign each data point to a cluster



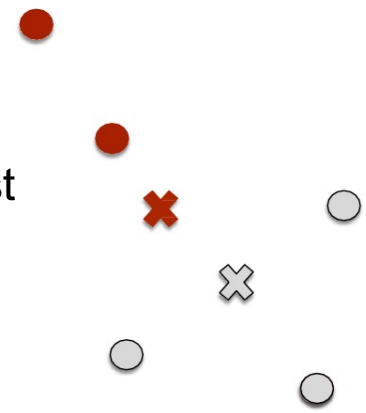
k-means clustering: Example

1. Specify desired number of clusters *k*
2. Randomly assign each data point to a cluster
3. Compute the cluster centroids



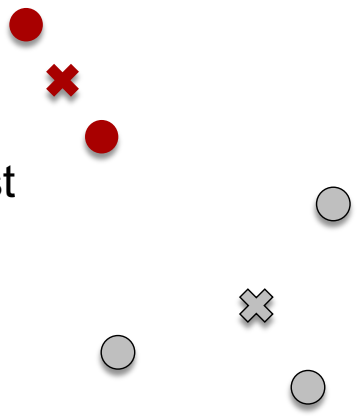
k-means clustering: Example

1. Specify the desired number of clusters *k*
2. Randomly assign each data point to a cluster
3. Compute the cluster centroids
4. Re-assign each point to the closest cluster centroid



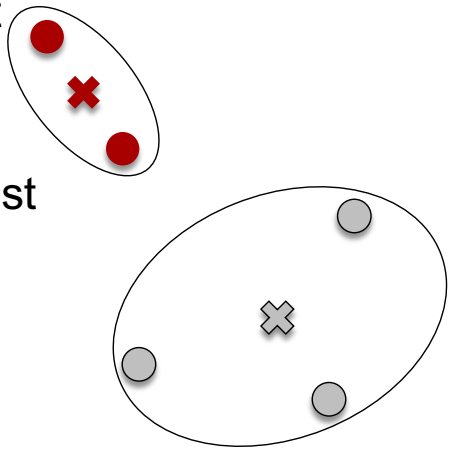
k-means clustering: Example

1. Specify the desired number of clusters *k*
2. Randomly assign each data point to a cluster
3. Compute the cluster centroids
4. Re-assign each point to the closest cluster centroid
5. Re-compute the cluster centroids
6. Repeat steps 4 and 5 until no convergence



k-means clustering: Example

1. Specify the desired number of clusters *k*
2. Randomly assign each data point to a cluster
3. Compute the cluster centroids
4. Re-assign each point to the closest cluster centroid
5. Re-compute the cluster centroids
6. Repeat steps 4 and 5 until no convergence



k-means clustering: when do we stop?

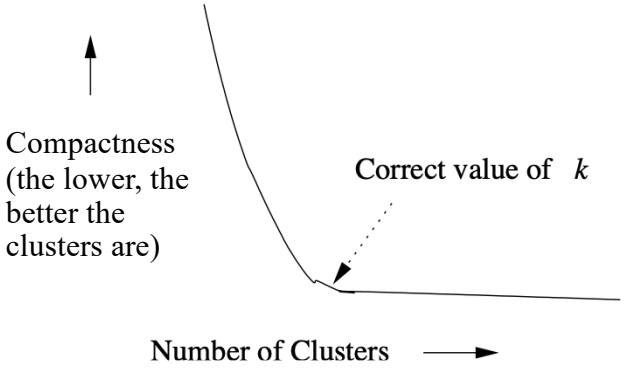
- Set a maximum number of iterations
- Check whether centroids do not change (or change only slightly) between successive iterations
  - See next slide 111, line 11
- In practice, the algorithm stops when either condition is met (logical OR)

k-means clustering: Considerations

1) How to choose *k*? Can exploit some knowledge about the dataset or experiment with different values of *k*
   – Methods: elbow method, silhouette score
2) How to speedup solution? Use a smarter initial partitioning of points into clusters (if you have some knowledge on dataset)
   – See animation https://www.pinecone.io/learn/k-means-clustering
3) How to improve solution quality? Run the *k*-means algorithm multiple times with different initializations and select the best result

How to choose *k*? Elbow method

- Idea: measure a score of clustering quality (e.g., *compactness*) for different values of *k*
- Plot the score a function of *k*
- If the curve resembles an arm, the "elbow" (inflection point) indicates a suitable number of clusters



k-means clustering: Alternative for initialization

- How to choose the initial k centroids?
  - Current approach: randomly assign each data point to a cluster and then compute the initial centroids
- Alternative approach
  - Randomly initialize the cluster centroids
  - Then iterate between cluster assignment and centroid update
  - That is, repeat steps 4 and 5 of slide 100
- This procedure is known as Lloyd's algorithm

k-means: Lloyd's algorithm

1. Randomly choose *k* data points from the dataset and use them as the initial centroids
2. Compute the cluster centroids
3. Re-assign each point to the closest centroid (e.g., using Euclidian distance)
4. Re-compute the cluster centroids
5. Repeat steps 4 and 5 until convergence (no further improvement)

k-means: Lloyd's pseudocode

K-MEANS $(\mathbf{D}, k, \epsilon)$**:**

$t = 0$

Randomly initialize k centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$

repeat

  $t \leftarrow t + 1$

  $C_i \leftarrow \emptyset$ for all $i = 1, \cdots, k$

  // Cluster Assignment Step

  foreach $\mathbf{x}_j \in \mathbf{D}$ **do**

    $i^* \leftarrow \arg\min_i \left\{ \|\mathbf{x}_j - \boldsymbol{\mu}_i^{t-1}\|^2 \right\}$

    $C_{i^*} \leftarrow C_{i^*} \cup \{\mathbf{x}_j\}$ // Assign $\mathbf{x}_j$ to closest centroid

  // Centroid Update Step

  foreach $i = 1, \cdots, k$ **do**

    $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$

until $\sum_{i=1}^{k} \|\boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1}\|^2 \leq \epsilon$

D is the dataset, *k* the number of clusters, ε the stopping threshold

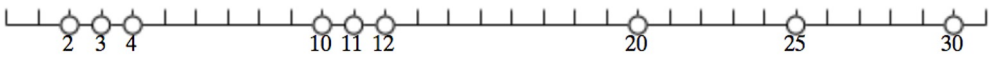
Find the cluster centroid closest to point $\mathbf{x}_j$

Assign point $\mathbf{x}_j$ to the closest cluster C_{i^*}

Update the cluster centroids

Repeat until the change in centroids between consecutive iterations is below the threshold (ε)

k-means clustering (Lloyd): Example

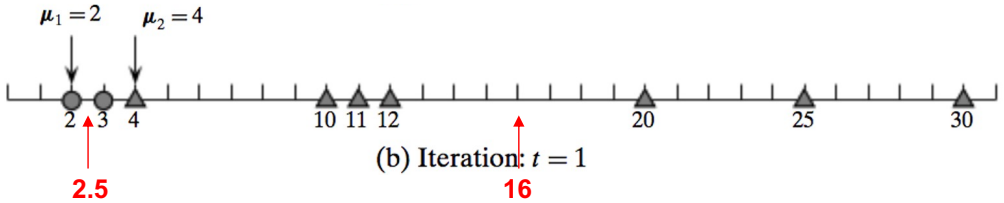


(a) Initial dataset

- Consider 1-dimension data set and set *k*=2 clusters
- Let's randomly initialize the centroids to $\mu_1^0 = 2$ and $\mu_2^0 = 4$
- 1st iteration: assign each point to the closest cluster; get $C_1 = \{2, 3\}$ and $C_2 = \{4, 10, 11, 12, 20, 25, 30\}$
- Recompute the centroids:

  $\mu_1^1 = (2+3)/2 = 5/2 = 2.5$

  $\mu_2^1 = (4+10+11+12+20+25+30)/7 = 112/7 = 16$



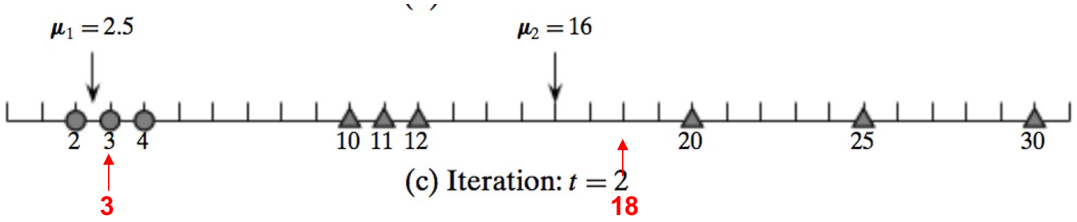
(b) Iteration: $t = 1$

k-means clustering (Lloyd): Example

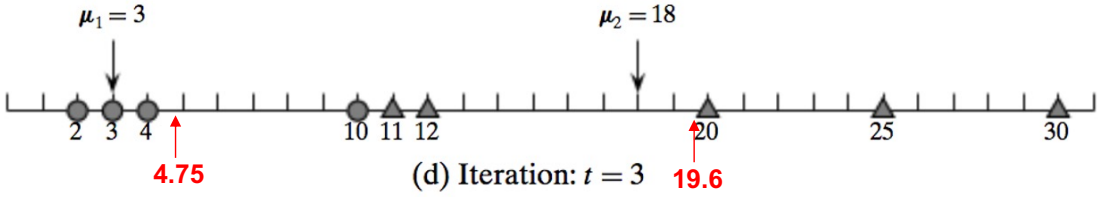
- 2nd iteration: repeat the cluster assignment to get C_1 = {2, 3, 4} and C_2 = {10, 11, 12, 20, 25, 30} and recalculate the new centroids

  μ_1^2 = (2+3+4)/3 = 9/3= 3

  μ_2^2 = (10+11+12+20+25+30)/6 = 108/6= 18

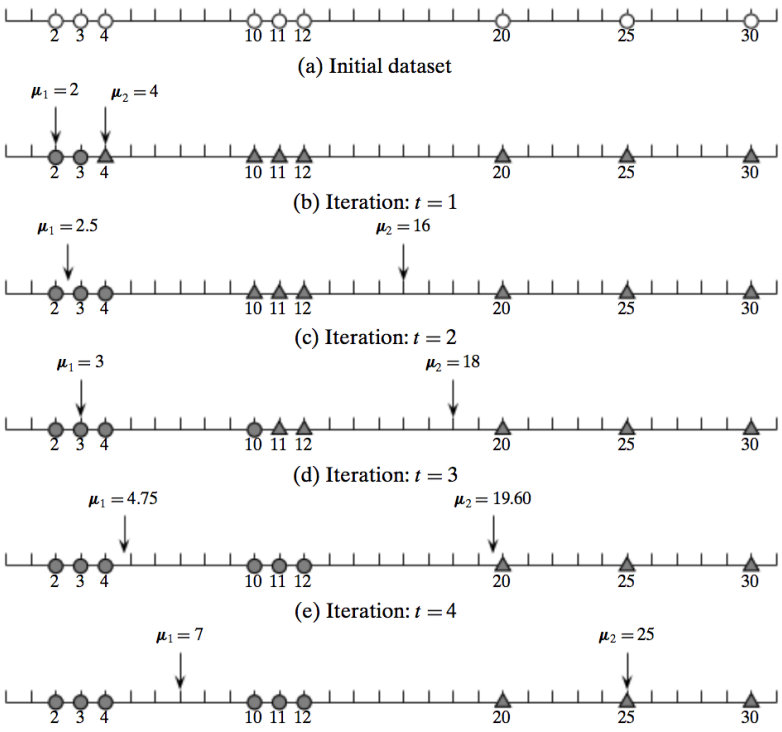


- 3rd iteration: obtain C_1 = {2, 3, 4, 10} and C_2 = {11, 12, 20, 25, 30} and the new centroids μ_1^3 = (2+3+4+10)/4 = 17/4 = 4.75 and μ_2^3 = (11+12+20+25+30)/5 = 19.6



k-means clustering (Lloyd): Example

- We continue until convergence: the final clusters are C_1 = {2, 3, 4, 10, 11, 12} and C_2 = {20, 25, 30}



Using *k*-means in R

- R (programming language) uses by default the efficient algorithm by Hartigan and Wong (1979) that partitions the data points into *k* clusters such that the total within-cluster sum of squares is minimized
  - Total within-cluster sum of squares (WSS): sum of distances between each data point and its cluster centroid, summed over all clusters

$$WSS = \sum_{i=1}^{k} \sum_{j \in C_i} \sum_{l=1}^{n} (x_{jl} - \mu_{jl})^2$$

C_i: set of data points in cluster i
μ_{jl}: coordinate l of the centroid for cluster j

k-means in R

- How to use *k*-means in R?
  - Type `help(kmeans)`

```
kmeans(x, centers, iter.max = 10, nstart = 1,
algorithm = c("Hartigan-Wong","Lloyd","Forgy",
"MacQueen"), trace=FALSE)
```

  - x: numeric data
  - centers: number of clusters (*k*)
  - iter.max: maximum number of iterations allowed
  - nstart: how many random starts (hint: choose a high value, e.g., 20)
  - algorithm: default is Hartigan-Wong

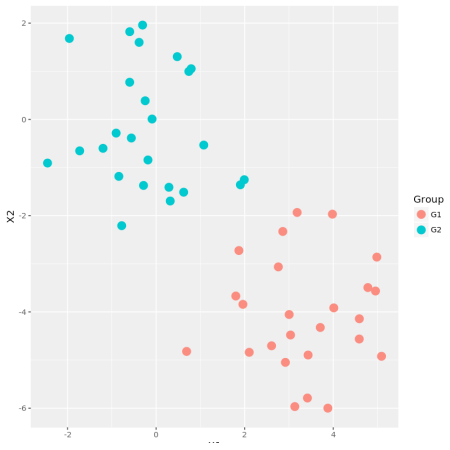
k-means in R: Example

- Use Jupiter Notebook with R https://jupyter.org/try

- Consider a simple dataset with two well-separated clusters
  - Let's generate the data points using normal distribution

k-means in R: Example (*k*=2)

```
#Generate a sample data set with two well-separated clusters
set.seed(2)
f.data <- data.frame("Group"=c(rep("G1",25),rep("G2",25)),
"X1"=c(rnorm(25)+3,rnorm(25)), "X2"=c(rnorm(25)-4,rnorm(25)))

#Plot data set
library(ggplot2)
gg1 <- ggplot(f.data,aes(x=X1,y=X2,color=Group))+geom_point(size=4)
gg1
```



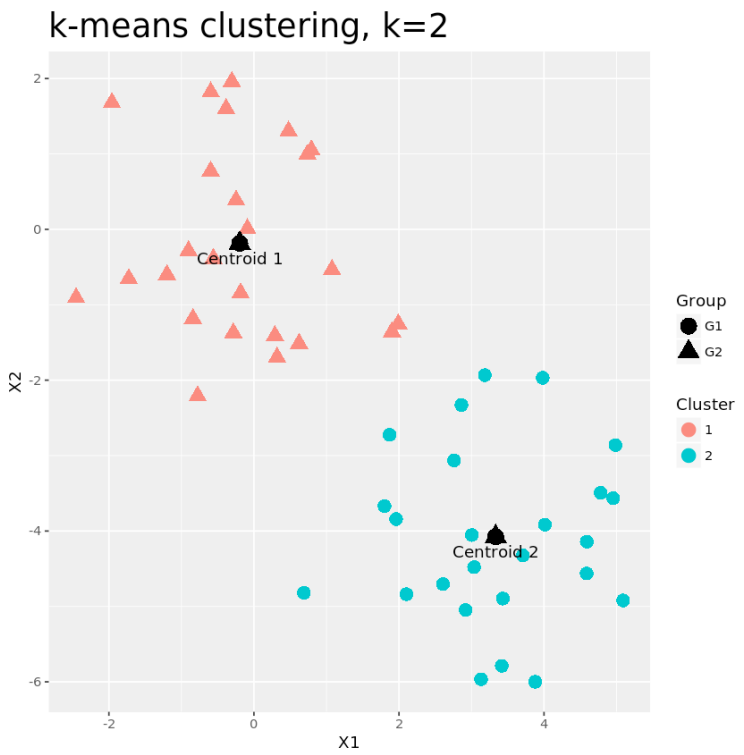
k-means in R: Example (*k*=2)

```
#Run k-means with k=2
km.out <- kmeans(f.data[,2:3],centers=2,nstart=20)
km.out

# Print the vector indicating cluster assignment
km.out$cluster

#Print the centroids
km.out$centers
```

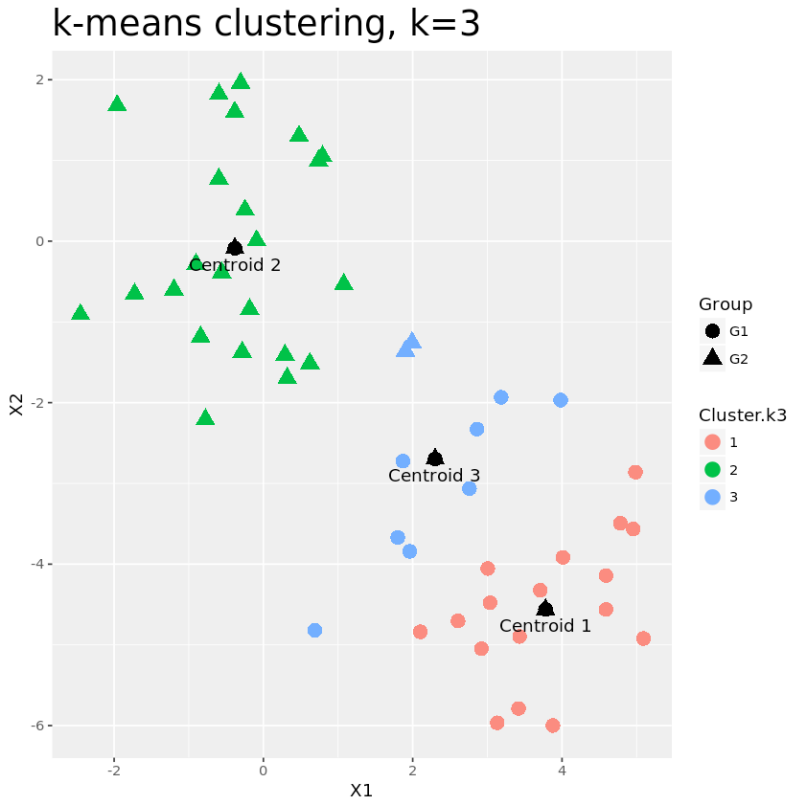

k-means in R: Example (*k*=2)

```
#Prepare the output for plotting
f.data$Cluster <- as.factor(km.out$cluster)
#Plot the clustering result
gg2 <- ggplot(f.data,aes(x=X1,y=X2,color=Cluster,shape=Group)) +
  geom_point(size = 4) + geom_point(aes(x = km.out$centers[1, 1],
  y = km.out$centers[1, 2]), size = 5, color = "black") +
  geom_point(aes(x = km.out$centers[2, 1], y = km.out$centers[2, 2]),
  size = 5, color = "black") +
  annotate("text", x = km.out$centers[1, 1], y = km.out$centers[1, 2] - 0.2,
  label = "Centroid 1") + annotate("text", x = km.out$centers[2, 1],
  y = km.out$centers[2, 2] - 0.2, label = "Centroid 2") +
  ggtitle("k-means clustering, k = 2") +
  theme(plot.title = element_text(size = rel(2)))
gg2
```

k-means in R: Example (*k*=2)

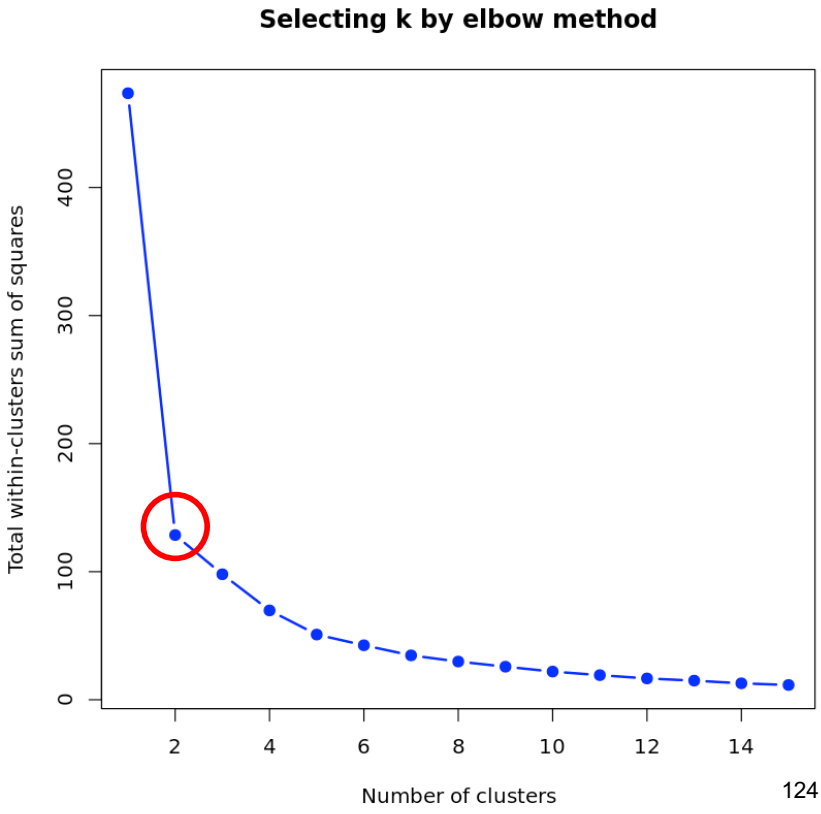


k-means in R: Example (*k*=3)



Elbow method: Example

- Compute WSS, the total within-cluster sum of squares, and use the elbow method
- WSS measures cluster compactness and we aim to minimize it (but it cannot be 0 unless every point is its own cluster)
  - As *k* increases, WSS tends to 0



References

- J. Leskovec, A. Rajaraman and J. Ullman, Recommendation Systems, ch. 9 of "Mining of Massive Datasets" book http://infolab.stanford.edu/~ullman/mmds/ch9.pdf
- J. Leskovec, A. Rajaraman and J. Ullman, Clustering, ch. 7 of "Mining of Massive Datasets" book http://infolab.stanford.edu/~ullman/mmds/ch7.pdf
- Some videos:
  - Overview of Recommender Systems, Stanford University https://www.youtube.com/watch?v=1JRrCEgiyHM
  - L. Serrano, Clustering: K-means and Hierarchical https://www.youtube.com/watch?v=QXOkPvFM6NU
  - V. Lavrenko, Agglomerative Clustering: How It Works https://www.youtube.com/watch?v=XJ3194AmH40
  - V. Lavrenko, K-means Clustering: How It Works https://www.youtube.com/watch?v=_aWzGGNrcic