https://i0.wp.com/thedatascientist.com/wp-content/uploads/2018/05/recommender_systems.png

Lecture 3: Scalable Matrix Factorisation for Collaborative Filtering in RecSys

COM6012: Scalable ML by Haiping Lu

YouTube Playlist: https://www.youtube.com/c/HaipingLu/

Week 3 Contents / Objectives

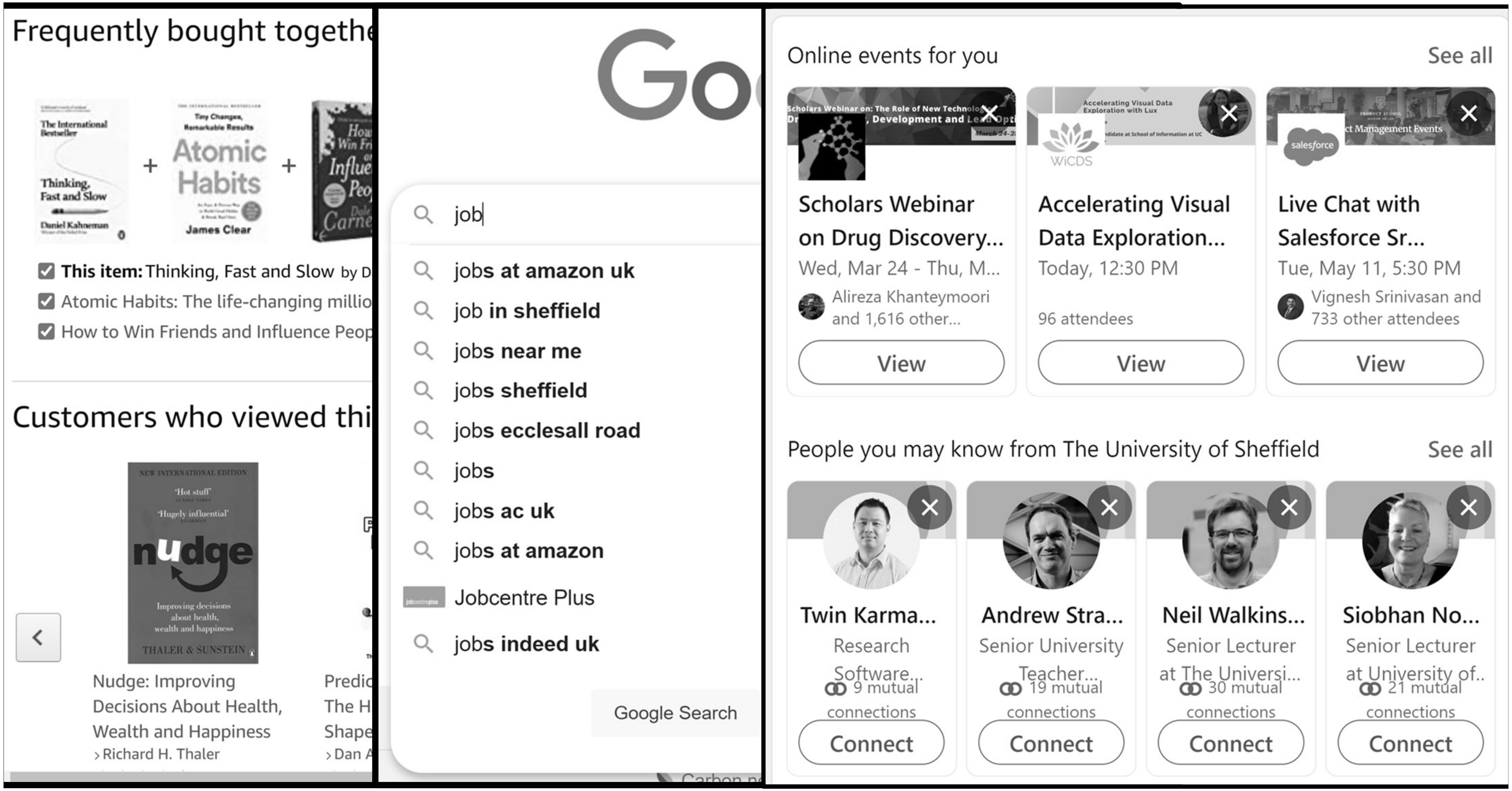
- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

Week 3 Contents / Objectives

- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

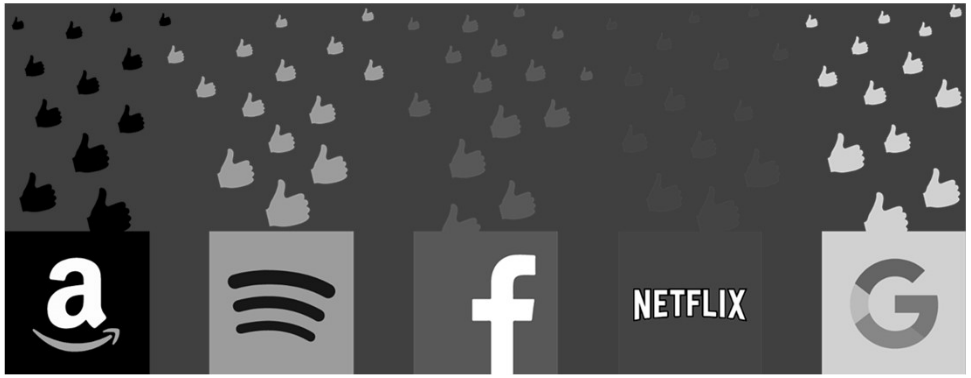
Many Decisions to Make

Recommendations Everywhere



Recommender Systems (RecSys)

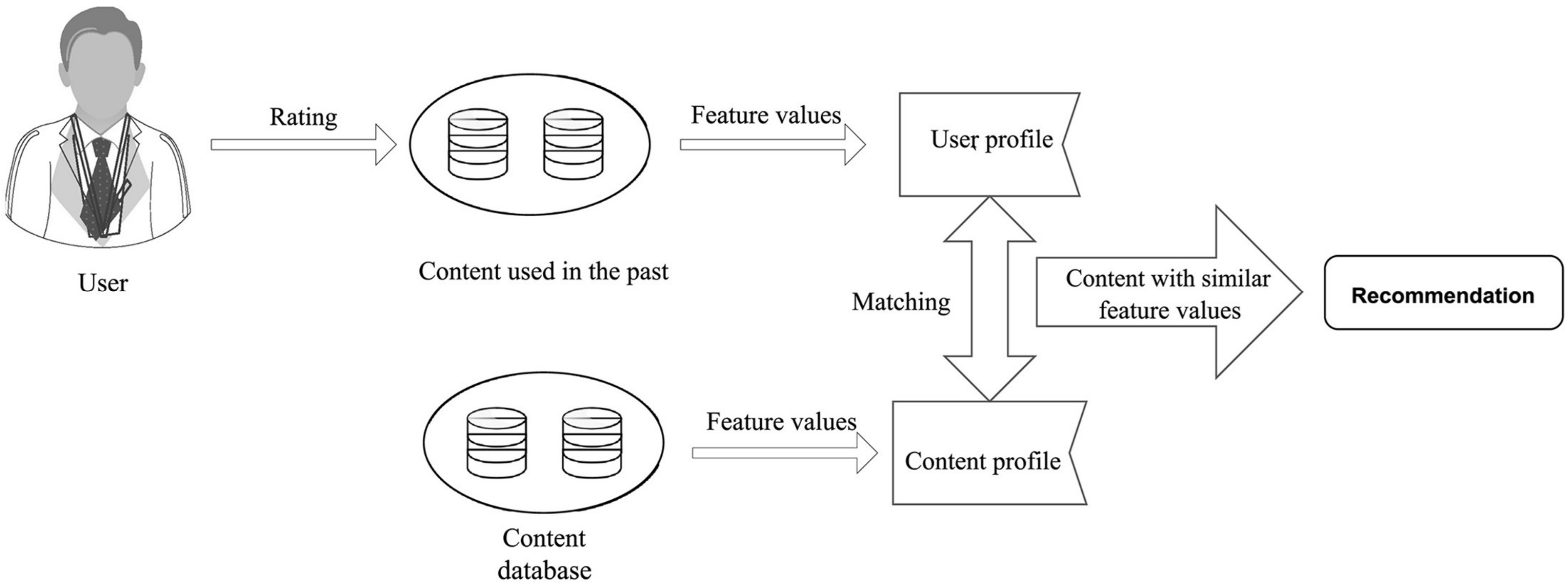
- Predict relevant items for a user, in a given context
- Predict to what extent these items are relevant
- A ranking task (searching as well)
- Implicit, targeted, **intelligent** advertisement
- Effective, popular marketing



Two Classes of RecSys

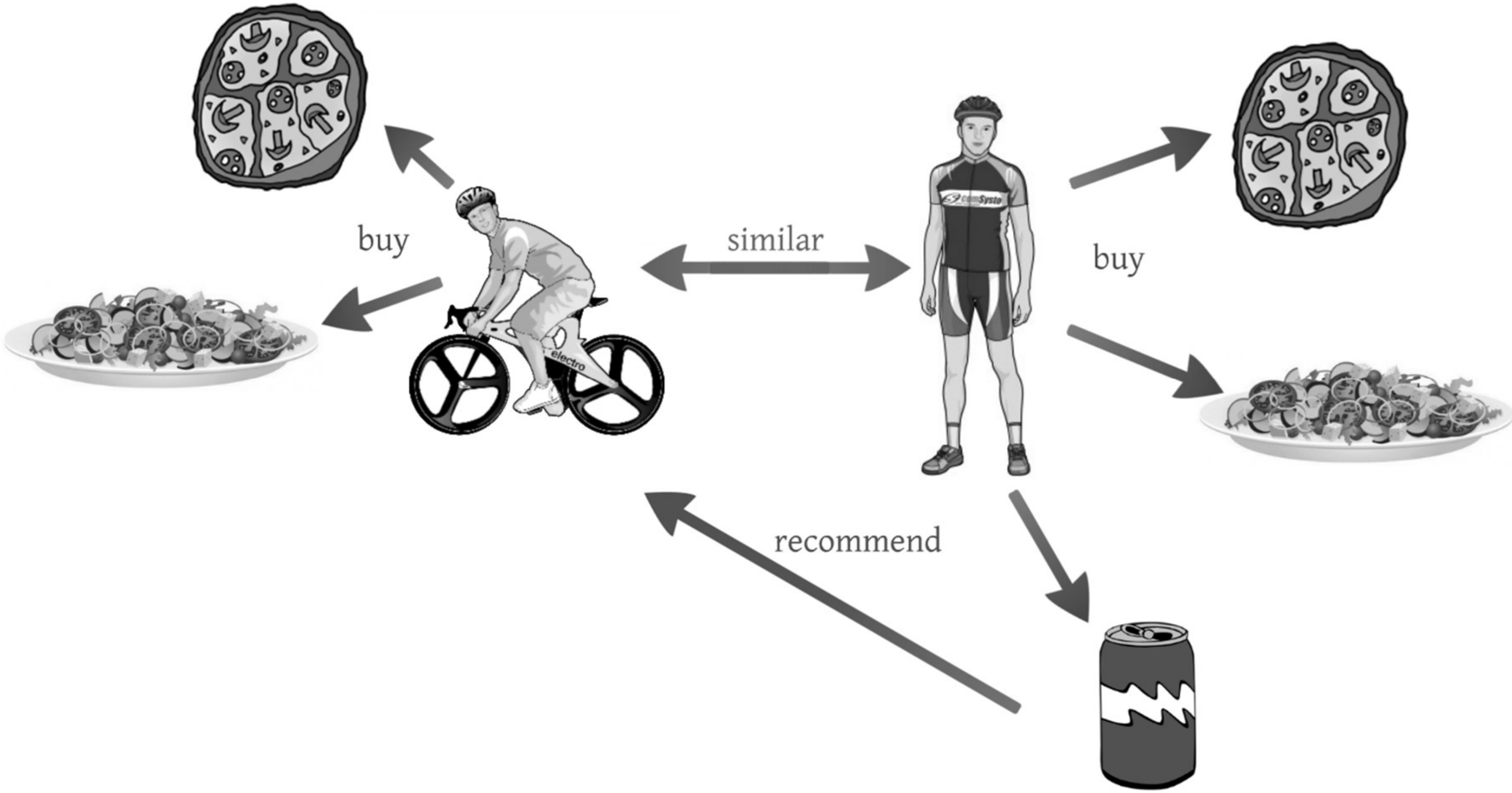
- Content-based recommender systems
- Collaborative filtering recommender systems

Content-based RecSys



dac4058-fig-0010-m.jpg (2128×789) (wiley.com)

Collaborative Filtering RecSys



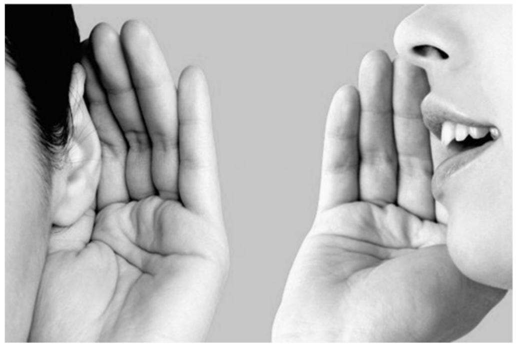
https://miro.medium.com/max/2656/1*6_NlX6CJYhtxzRM-t6ywkQ.png

Week 3 Contents / Objectives

- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

What is Collaborative Filtering?

- Information filtering based on past records
  - Electronic word of mouth marketing
  - Turn visitors into customers (e-Salesman)
- Components
  - Users (customers): who provide ratings
  - Items (products): to be rated
  - Ratings (interest): core data



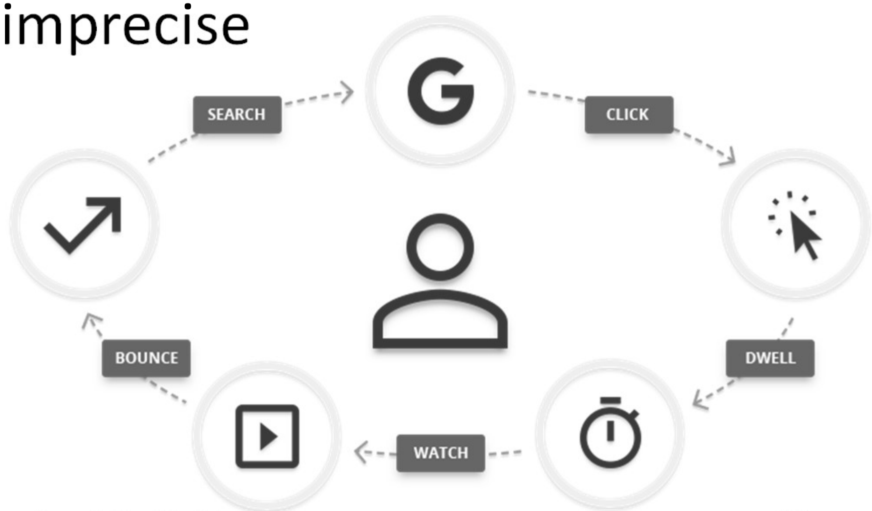
John	5	1	3	5
Tom	?	?	?	2
Alice	4	?	3	?

Collaborative Filtering (CF)

- Objective: predict how well a user will like an unrated item, given past ratings for a community of users
- How does CF work?
  - Input: many users' ratings for many items
  - Model: similar users ← ratings strongly correlate
  - Recommend items rated highly by similar users

Explicit vs Implicit Ratings

- Explicit (direct): users indicate levels of interest
  - Most accurate descriptions of a user's preference
  - Challenging in collecting data
- Implicit (indirect): observing user behavior
  - Can be collected with little or no cost to user
  - Ratings inference may be imprecise



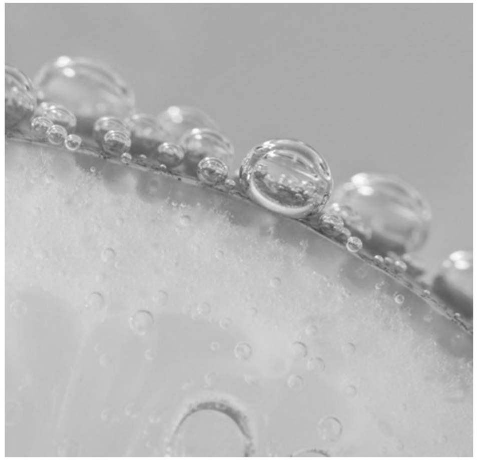
Rating Scales

- Scalar ratings
  - Numerical scales
  - 1-5, 1-7, etc.

- Binary ratings
  - Agree/Disagree, Good/Bad, etc.

- Unary ratings
  - Presence/absence of an event, e.g., purchase/browsing history, search patterns, mouse movements
  - No info about the opposite ≠ 0

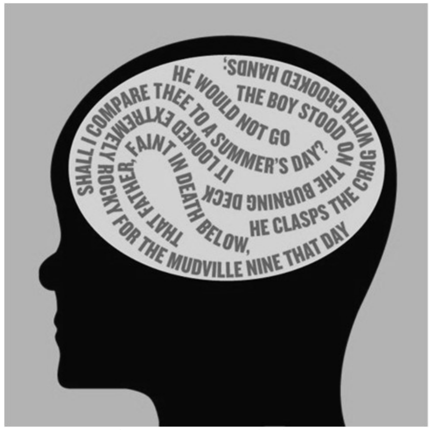
CF Preferences



- Many users, many items, many ratings
- Users rate multiple items
- Other users with similar needs/tastes
- Item evaluation requires personal taste
- Taste persists

CF Methods

- Memory-based: predict using past ratings directly
  - Weighted ratings given by other similar users
  - User-based & item-based (non-ML)
- Model-based: model users based on past ratings
  - Predict ratings using the learned model



iforget-465.jpg (465×465) (newyorker.com)

neural-header.jpg (756×503) (utsouthwestern.edu)

Prediction Accuracy

- Mean absolute error (MAE)

$$MAE = \frac{\sum_{i,j} |p_{i,j} - r_{i,j}|}{n}$$

- Normalized MAE

$$NMAE = \frac{MAE}{r_{max} - r_{min}}$$

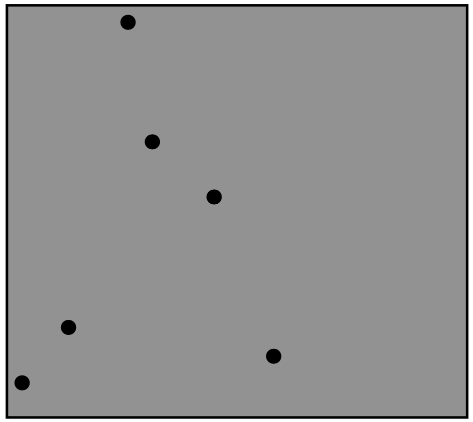
- Root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i,j}(p_{i,j} - r_{i,j})^2}$$

Challenges

- Cold Start
  - New user
    - Rate some initial items
    - Non-personalized rec.
    - Describe tastes
    - Demographic info
  - New item
    - Randomly selecting items
    - Content analysis, metadata (non-CF)
- Sparsity: sparse user-item matrix
- Scalability: millions of users and items

isbil+2.jpg (490×303) (squarespace-cdn.com)



Week 3 Contents / Objectives

- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

Matrix Factorisation (MF) for CF

- Characterise items/users by vectors of factors learned from the rating matrix user x item
- High correlation between item and user factors → good recommendation
- Flexibility: incorporate implicit feedback, temporal effects, and confidence levels

John	5	1	3	5
Tom	?	?	?	2
Alice	4	?	3	?

Basic MF Model

- Map users & items to a joint latent factor space of dimensionality k
  - Item i → vector q_i: the extent to which the item possesses those k factors
  - User u: vector p_u: the extent of interest the user has on those k factors
- User-item interactions: the user’s overall interest in the item’s characteristics
  - Inner product $q_i^T p_u$: predicted user u‘s rating of item i

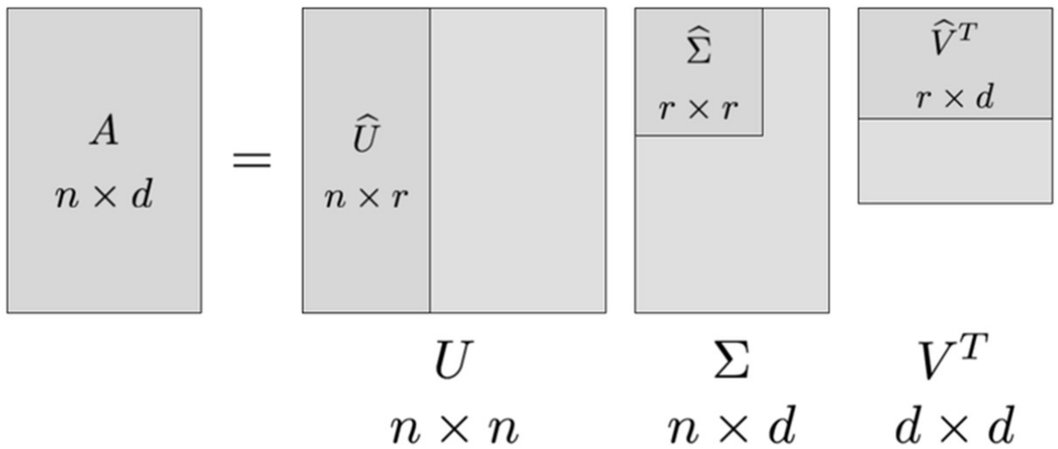
$$\hat{r}_{u,i} = q_i^T p_u$$

How to Learn the MF Model

- To learn: item factors $\{q_i\}$ and user factors $\{p_u\}$
- Factorisation assuming full rating matrix
  - Factorise rating matrix R using <u>SVD</u> to obtain P, S, Q

$$R = PSQ^T$$

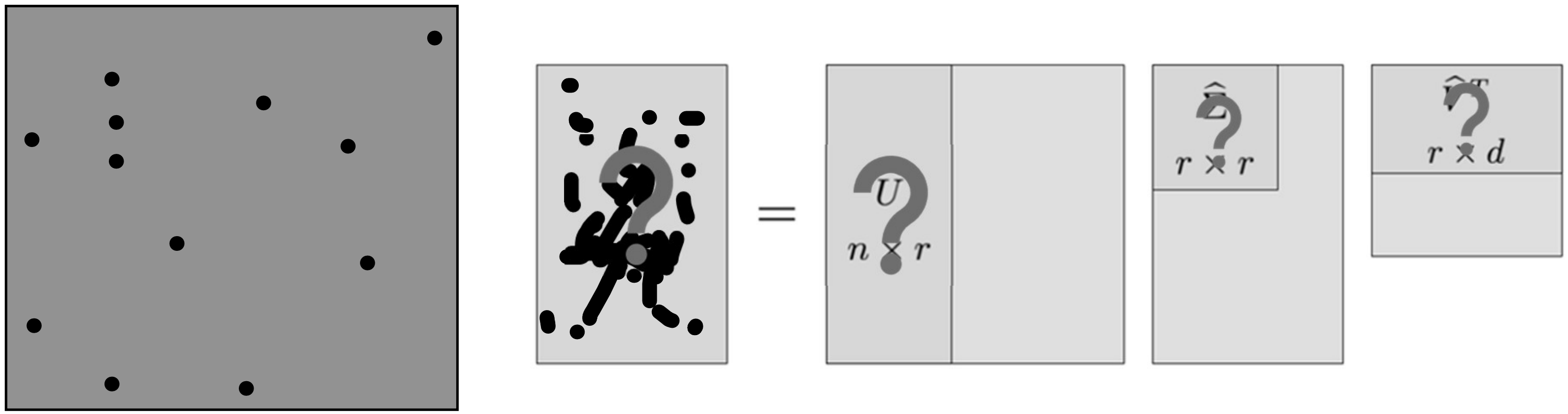
  - Reduce the matrix S to dimension k, i.e. S_k
  - $P \rightarrow P_k$ and $Q \rightarrow Q_k$: $P_k S_k \rightarrow \hat{P}$, and $S_k Q_k^T \rightarrow \hat{Q}^T$
  - uth row of $\hat{P} \rightarrow p_u$, ith column of $\hat{Q}^T \rightarrow q_i$



A: R
U: P
Σ: S
V: Q
r: k

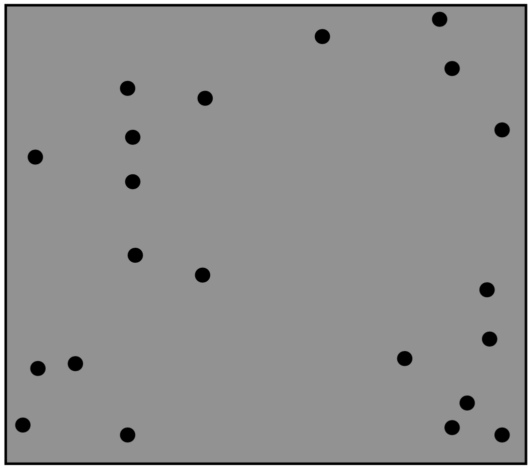
Challenges in MF for CF

- High portion of missing values caused by sparseness in the user-item rating matrix
- Conventional SVD is undefined when knowledge about the matrix is incomplete



How to Fill Missing Values

- Imputation: fill in missing ratings using the average ratings for user and item
- Problems
  - Expensive: significantly increases the amount of data
  - Inaccurate imputation might distort the data



MF with Missing Values

- Modelling directly the observed ratings only
  - Avoid overfitting through a regularised model
  - Minimize the regularised squared error on the set of known ratings to learn the factor vectors p_u and q_i

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2)$$

  - κ: the training set of the (u,i) pairs with known ratings
  - λ: the regularisation parameter

Alternating Least Squares for MF-CF

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2)$$

- Both p_u and q_i are unknown (non-convex function)
- Fix one of them → quadratic with optimal solution
- Alternating Least Squares (ALS): alternate between fixing q_i s and fixing p_u s
  - Fix P (p_us) as $\hat{P}$ to recompute q_i s by solving a least-squares problem $\| R - PQ^T \|_F$ (Frobenius norm)

$$R = \hat{P}Q^T \Rightarrow Q^T = (\hat{P}^T\hat{P})^{-1}\hat{P}^T R$$

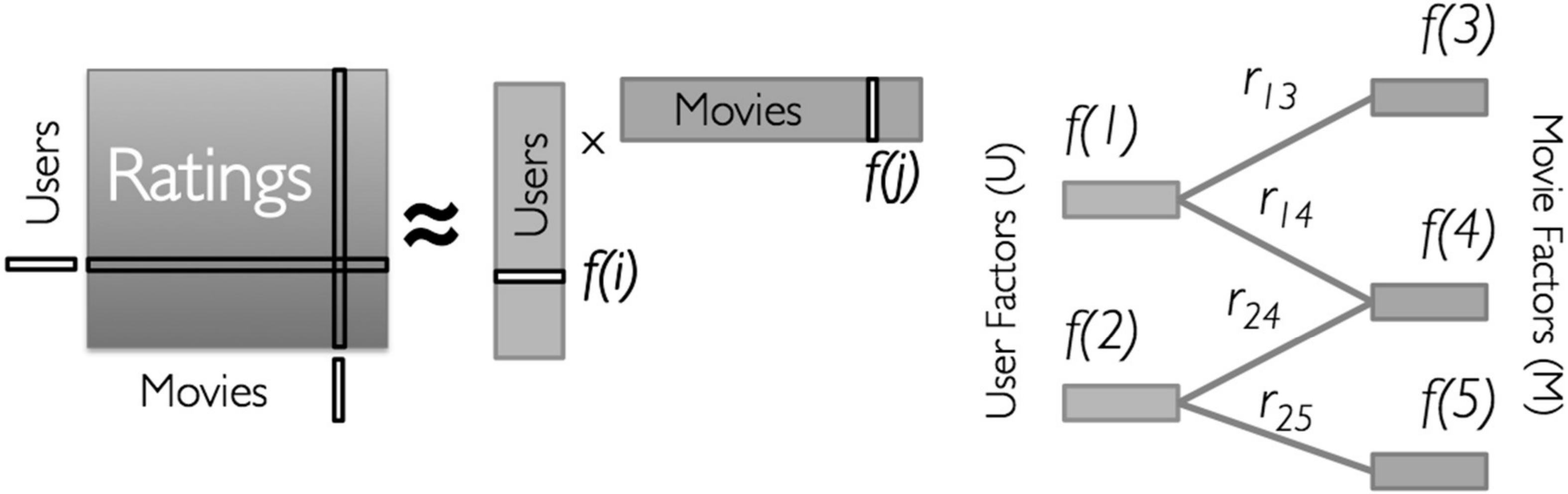
  - Fix Q as $\hat{Q}$, we have

$$P = R\hat{Q}(\hat{Q}^T\hat{Q})^{-1}$$

  - Random initialisation to start this iteration

MF for Movie Recommendation

Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

Figure frospark-training/matrix_factorization.png at master · databricks/spark-training (github.com)

Week 3 Contents / Objectives

- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

Key in Scalable ML

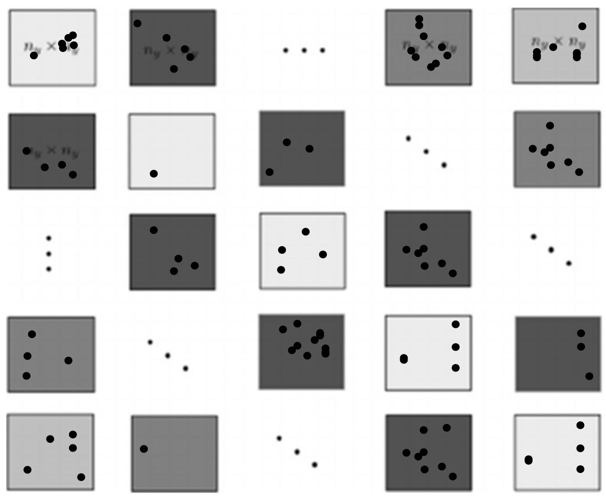
- Computation and storage should be linear (in n, d)
  → Low-cost computation (time + space)

- Perform parallel and in-memory computation
  → Many working + reduce disk I/O

- Minimise network communication → Reduce overhead in parallelisation, not the more the better

Blocked Implementation of ALS

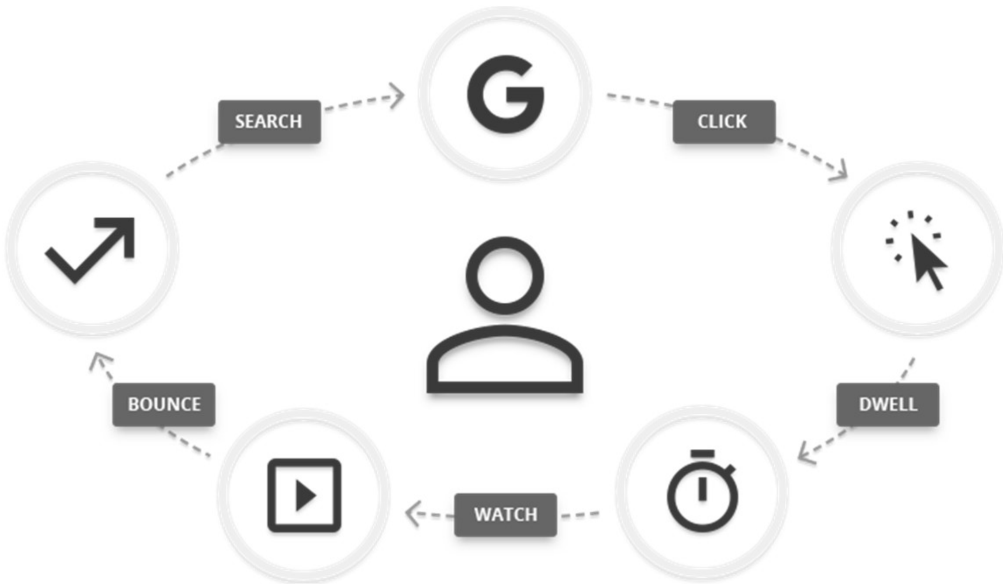
- Group users and items into blocks
  - Reduce communication: only send one copy of each user vector to each item block on each iteration, and only for the item blocks that need that user's feature vector
  - Pre-compute info: out-links of each user (which blocks of items it will contribute to); in-links for each item (which of the feature vectors it receives from each user block it will depend on)



Color-online-A-symmetric-block-Toeplitz-matrix-Each-block-is-also-a-symmetric-Toeplitz.png (488×369) (researchgate.net)

Implicit Feedback Modelling

- Implicit feedback: views, clicks, purchases, likes, shares
  - Rating $r =$ strength in observations of user actions (#clicks, viewing duration) → confidence level in observed user preference
  - Construct a preference matrix *P*: e.g. 1 if $r > 0$ and 0 if $r = 0$
  - Factorisation of *P* → latent factors to predict the preference of a user for an item (details in an ICDM08 paper)



The ALS API in Spark

- numUserBlocks/numItemBlocks: the number of blocks the users/items will be partitioned into to parallelize computation (defaults to 10)
- rank: the number of latent factors in the model (defaults to 10)
- regParam: the regularization parameter in ALS (defaults to 1.0)
- implicitPrefs: whether to use the explicit feedback ALS variant or one adapted for implicit feedback data (defaults to false: explicit ratings)
- alpha: the baseline confidence of implicit feedback (defaults to 1.0)
- nonnegative: whether to use nonnegative constraints (defaults to false)
- coldStartStrategy: “drop”→ drop any rows in the DataFrame of predictions that contain NaN values (defaults to "nan“: assign NaN to a user and/or item factor is not present in the model)
- blockSize: the size of the user/product blocks in the blocked implementation of ALS to reduce communication

```scala
  def train[ID: ClassTag]( // scalastyle:ignore
      ratings: RDD[Rating[ID]],
      rank: Int = 10,
      numUserBlocks: Int = 10,
      numItemBlocks: Int = 10,
      maxIter: Int = 10,
      regParam: Double = 0.1,
      implicitPrefs: Boolean = false,
      alpha: Double = 1.0,
      nonnegative: Boolean = false,
      intermediateRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
      finalRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
      checkpointInterval: Int = 10,
      seed: Long = 0L)(
      implicit ord: Ordering[ID]): (RDD[(ID, Array[Float])], RDD[(ID, Array[Float])]) = {

    require(!ratings.isEmpty(), s"No ratings available from $ratings")
    require(intermediateRDDStorageLevel != StorageLevel.NONE,
      "ALS is not designed to run without persisting intermediate RDDs.")

    val sc = ratings.sparkContext

    // Precompute the rating dependencies of each partition
    val userPart = new ALSPartitioner(numUserBlocks)
    val itemPart = new ALSPartitioner(numItemBlocks)
    val blockRatings = partitionRatings(ratings, userPart, itemPart)
      .persist(intermediateRDDStorageLevel)
    val (userInBlocks, userOutBlocks) =
      makeBlocks("user", blockRatings, userPart, itemPart, intermediateRDDStorageLevel)
    userOutBlocks.count()    // materialize blockRatings and user blocks
```

CF in Spark ML

- Scala code (1800+ lines)
- Documentation: Collaborative Filtering in Spark
- DataBricks movie recommendations tutorial
- DataBricks: founded by the creators of Apache Spark
  - Their latest packages at their GitHub page
  - Databricks community edition: 15GB memory free

References

- Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 8 (2009): 30-37 (Yahoo & AT&T)
- Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets." *Eighth IEEE International Conference on Data Mining*, 2008
- Charu C. Aggarwal, *Recommender Systems: The Textbook*, April 2016