Lecture 10

Scalable Collaborative Filtering via Matrix Factorisation

Haiping Lu

http://www.dcs.shef.ac.uk/~haiping

COM6012 Scalable Machine Learning

Spring 2018

Week 10 Contents

- **Recommender Systems & Collaborative Filtering**
- Matrix Factorisation for Collaborative Filtering
- Collaborative Filtering in Spark
- Final Remarks

Recommender Systems

- Implicit, targeted, **intelligent** advertisement

- Online stores: effective, popular marketing

Tasks of a Recommender System

- **Predict** relevant/useful/interesting items for given user (in a given context)
- **Predict** to what extent these items are relevant/useful/interesting
- A **ranking** task (**searching** as well)

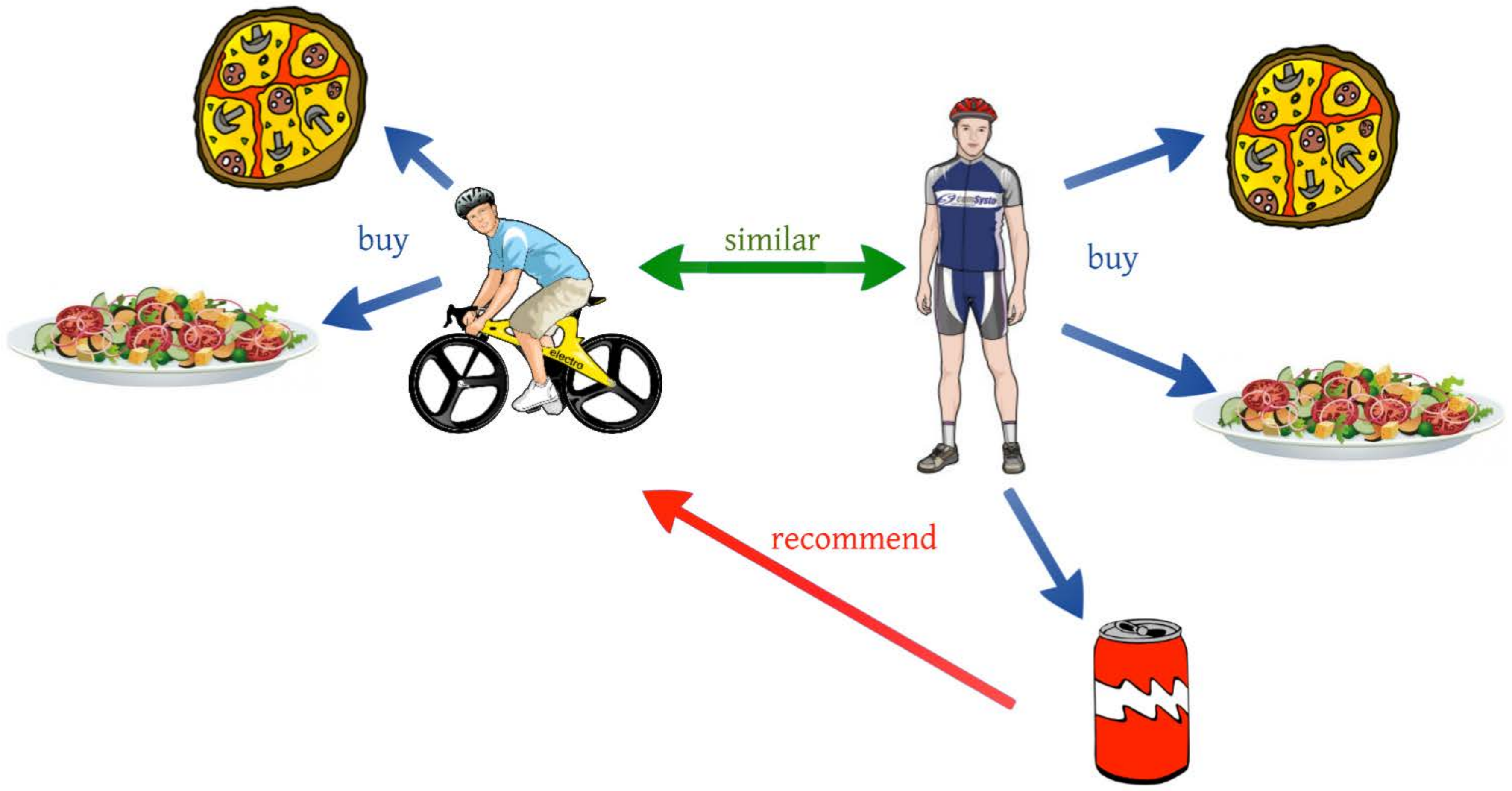
  "Rank items so that the items that are most relevant/useful/interesting for a given user (in a given context) appear at the top of the ranking"

Two Basic Classes of RecSys

- **Collaborative filtering** systems

- **Content-based** recommender systems

What is Collaborative Filtering?

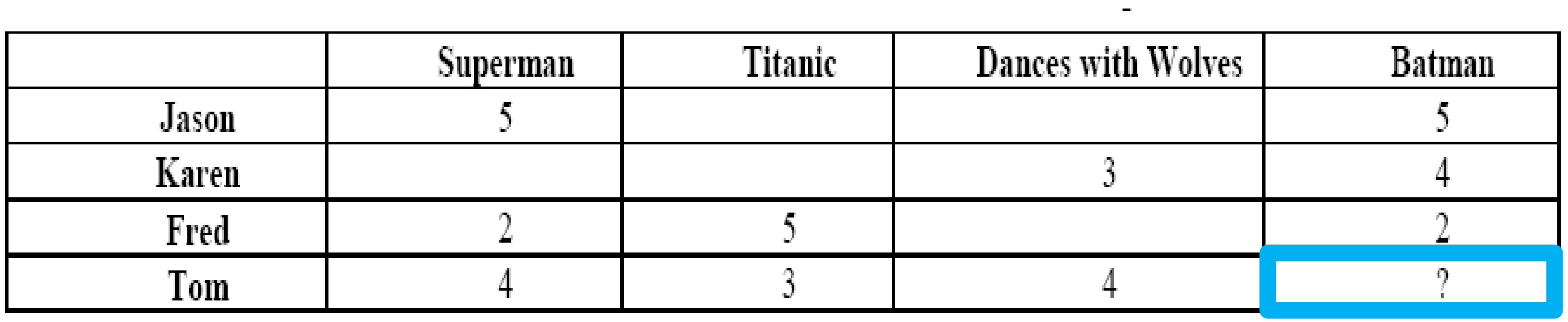


What is Collaborative Filtering?

- Information filtering based on past records
- Recommender system: predict, recommend
- Powerful/**Intelligent** marketing tool
  - Electronic **Word of Mouth** marketing
  - Turn visitors into customers (e-Salesman)
- Components
  - User (customer): who provides ratings
  - Items (product): to be rated
  - Ratings (interest)

What is Collaborative Filtering?

- **Predict** how well a user will like an item that he **has not rated** given a set of historical preference judgments (ratings) for a community of users
- Matrix completion



	Superman	Titanic	Dances with Wolves	Batman
Jason	5			5
Karen			3	4
Fred	2	5		2
Tom	4	3	4	?

Predict this

What is Collaborative Filtering?

- Maintain a database of many users' ratings of a variety of items
- For a given user, find other similar users whose ratings strongly correlate with the current user
- Recommend items rated highly by similar users, but not rated by the current user
- Used by almost all existing online stores

Rating Types

- Explicit ratings
  - Users rate themselves for an item
  - Most accurate descriptions of a user's preference
  - Challenging in collecting data
- Implicit ratings
  - Observations of user behavior
  - Can be collected with little or no cost to user
  - Ratings inference may be imprecise

Rating Scales

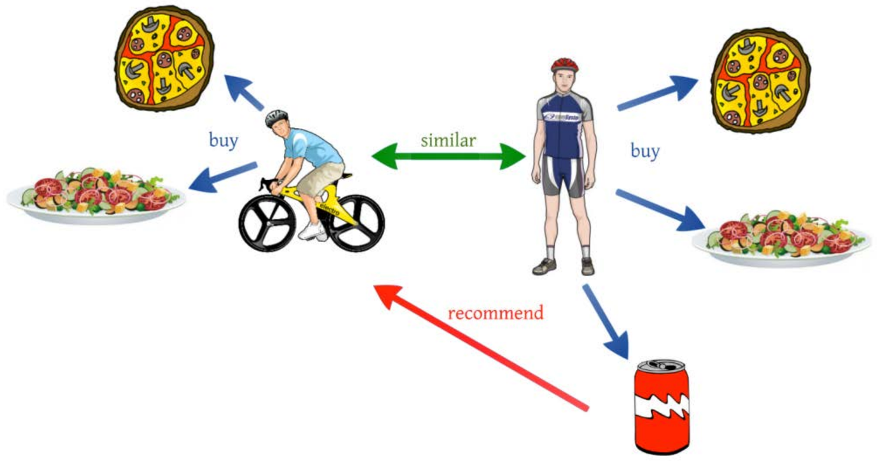
- Scalar ratings
  - Numerical scales
  - 1-5, 1-7, etc.
- Binary ratings
  - Agree/Disagree, Good/Bad, etc.
- Unary ratings
  - Good, Purchase, etc.
  - Absence of rating indicates no information

Preferred Application Domains

- Many items
- Many ratings
- Many more users than items recommended
- Users rate multiple items
- For each user of the community, there are other users with common needs or tastes
- Item evaluation requires personal taste
- Items persists
- Taste persists
- Items are homogenous

Collaborative filtering

- Input: user ratings for items
- Output: recommended items to a user **u1**:
    - Items liked by users who are similar to **u1**
    - similar users = users that like similar items



Collaborative Filtering Methods

- Memory-based
  - Predict ratings based on past ratings
    - Weighted ratings given by other (neighbor) users
    - User-based & item-based
- **Model-based**
  - Model users based on past ratings
  - Predict ratings using these models

Week 10 Contents

- Recommender Systems & Collaborative Filtering
- **Matrix Factorisation for Collaborative Filtering**
- Collaborative Filtering in Spark
- Final Remarks

Matrix Factorisation Methods

- Characteristic
  - Characterise both items and users by vectors of factors inferred from item rating patterns
  - High correspondence between item and user factors leads to a recommendation
  - Flexible: allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels
- Rely on matrix input data
  - One dimension representing user
  - The other representing items

Two Data Types

- High-quality explicit feedback
  - Include explicit input by users regarding their interest in products
  - We refer to explicit user feedback as ratings
  - Usually sparse matrix, since any single user is likely to have rated only a small percentage of possible items
- Implicit feedback
  - Indirectly reflect opinion by observing user behaviour
    - Purchase history, browsing history, search patterns, mouse movements
  - Usually denote the presence or absence of an event
  - Typically represented by a densely filled matrix

Basic Matrix Factorisation Model

- Map both users and items to a joint **latent factor** space of dimensionality *k*
- User-item interactions are modelled as inner products in that space
- Each item *i* is associated with a vector q_i ,and each user *u* is associated with a vector p_u,
    - q_i measures the extent to which the item possesses those factors
    - p_u measures the extent of interest the user has in items
    - The resulting dot product $q_i^T p_u$ captures the interaction between user *u* and item *i* – the user's overall interest in the item's characteristics

Basic Matrix Factorisation Model

- The user *u*'s rating of item *i*, which is denoted by r_{ui}, leading to the estimate

$$\hat{r}_{ui} = q_i^T p_u.$$

- The major challenge is computing the mapping of each item and user to factor vectors q_i, p_u
- We can capture the latent relationships between users and items
- We can produce a low-dimensional representation of the original rating matrix

Basic Matrix Factorisation Model

- Factor rating matrix *R* using SVD obtain *Q*, *S*, *P*
- Reduce the matrix *S* to dimension *k*
- Compute two resultant matrices: $Q_k S_k (q^T)$ and $S_k P_k (p)$
- These resultant matrices can now be used to compute the recommendation score for any user and item
- We can simply calculate the dot product of the *i*th row of *q* and the *u*th column of *p*

$$\hat{r}_{ui} = q_i^T p_u.$$

Challenges in Recommender Systems via MF

- 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
- Carelessly addressing only the relatively few known entries is highly prone to overfitting

How to Fill Missing Values

- Earlier systems relied on imputation to fill in missing rating and make the rating matrix dense, such as using the average ratings for user and item
- Problems
  - Imputation can be very expensive as it significantly increases the amount of data
  - Inaccurate imputation might distort the data

Matrix Factorisation with Missing Values

- Modelling directly the observed ratings only
  - Avoid overfitting through a regularized model
  - To learn the factor vectors(p_u and q_i), the system minimizes the regularised squared error on the set of known ratings:

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

  - The goal is to generalize those previous ratings in a way that predicts future unknown ratings
  - The constant λ controls the extent of regularisation

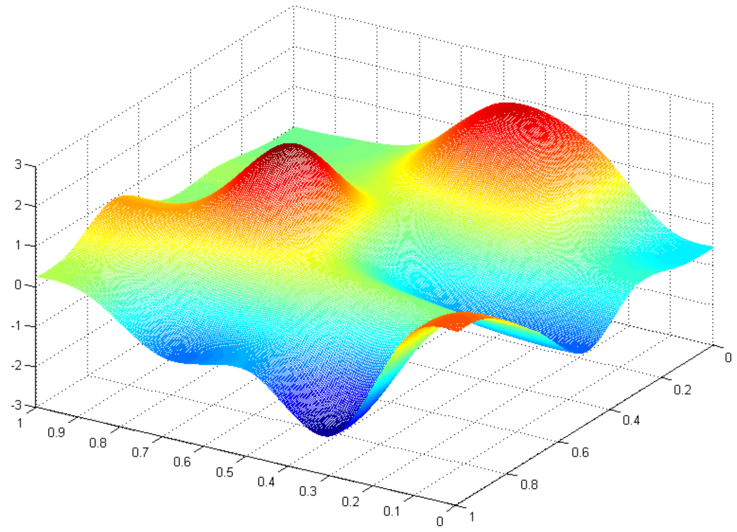
Learning Algorithms

- Stochastic gradient descent
  - Incremental learning
  - For each given training case, the system predicts r_{ui} and computes the associated prediction error

$$e_{ui} \overset{def}{=} r_{ui} - q_i^T p_u.$$

  - Learning rule

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$



Alternating Least Squares

- Because both q_i and p_u are unknowns, the object function is not convex
- However, if fixing one of the unknowns, the optimisation problems become quadratic and can be solved optimally
- ALS techniques rotate between fixing the q_i's and fixing the p_u's
- When all p_u's are fixed, the system recomputes the q_i's by solving a least-squares problem

$$\|R - PQ^T\|_F$$

Alternating Least Squares

- We can fix the matrix *P* as some matrix $\hat{P}$, such that minimization problem would be equivalent to

$$R = \hat{P} Q^T$$

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

- Analogously, we can fix Q as $\hat{Q}$

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

ALS vs SGD

- ALS is more scalable than SGD
  - Parallelization: in ALS, the system computes each q_i independently of the other item factors and computes each p_u independently of the other user factors. This gives rise to potentially massive parallelization of the algorithm
- ALS for systems centred on implicit data
  - Because the training set cannot be considered sparse, looping over each single training case—as gradient descent does—would not be practical. ALS can efficiently handle such cases

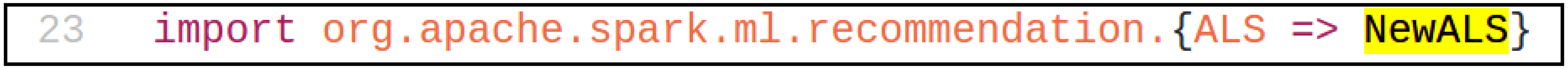
Week 10 Contents

- Recommender Systems & Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- **Collaborative Filtering in Spark**
- Final Remarks

CF in MLlib (notebook)

- Code:
  https://github.com/apache/spark/blob/v2.1.0/mllib/src/main/scala/org/apache/spark/ml/recommendation/ALS.scala

  Call ALS in ML (as NewALS)



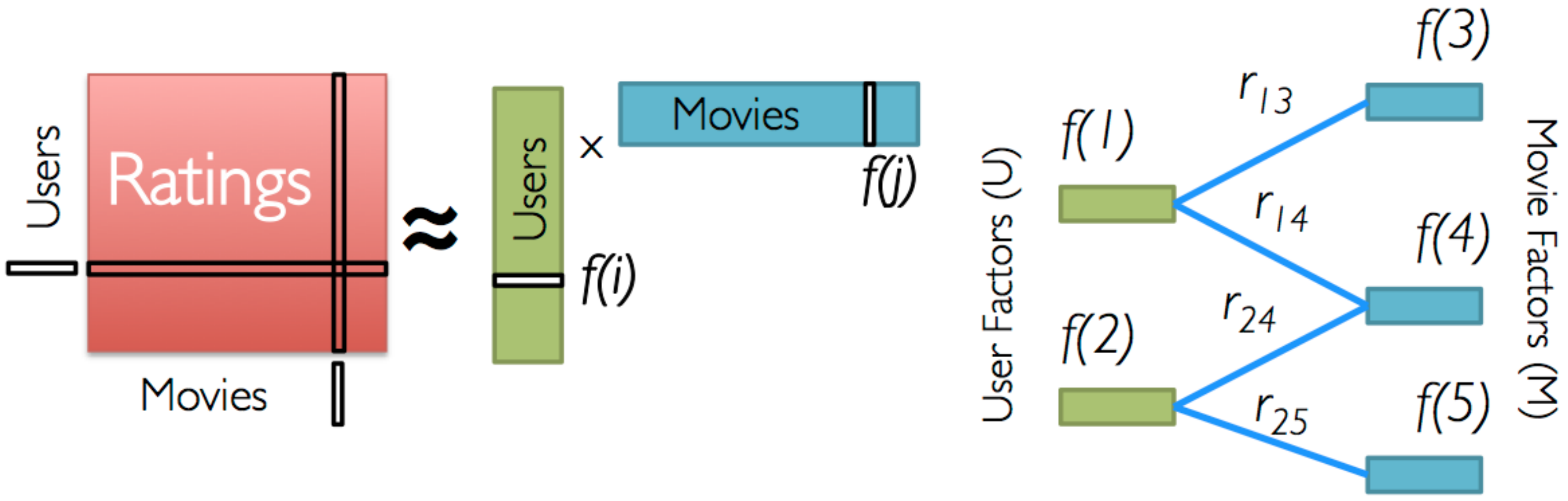
```scala
import org.apache.spark.ml.recommendation.{ALS => NewALS}
```

- Documentation:
  https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS
  https://spark.apache.org/docs/2.1.0/mllib-collaborative-filtering.html

- Tutorial: personalized movie recommendation with spark.mllib in Spark Summit 2014
  https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html

CF in MLlib (notebook)

Low-Rank Matrix Factorization:



Iterate:

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

CF in MLlib (notebook)

- *numBlocks:* the number of blocks used to parallelize computation (set to -1 to auto-configure)
- *rank*: the number of latent factors in the model
- *iterations*: the number of iterations of ALS to run (typically converges to a reasonable solution in <=20 iterations)
- *lambda*: specifies the regularization parameter in ALS
- *implicitPrefs*: specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data
- *alpha*: a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations

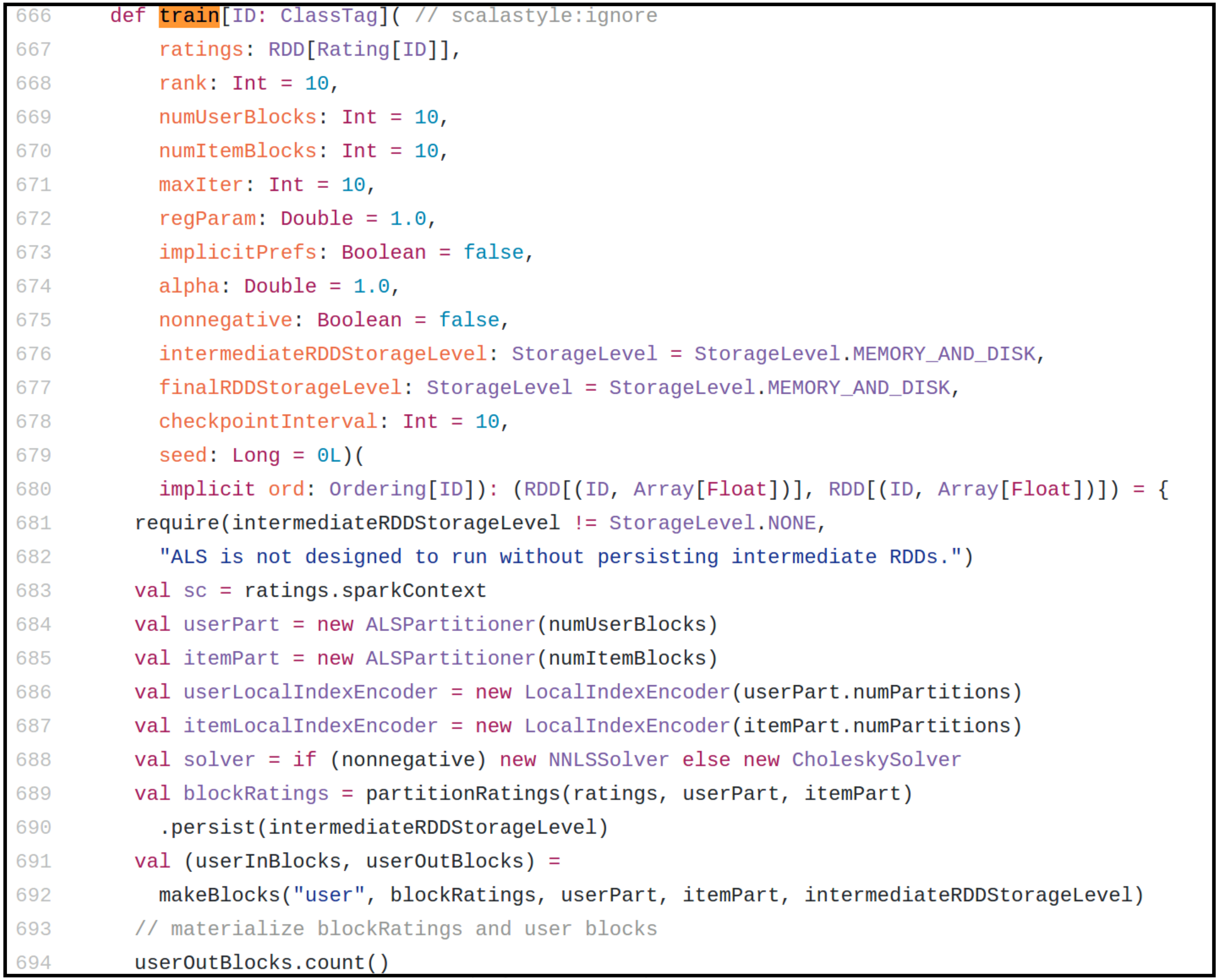
CF in ML

- Code: https://github.com/apache/spark/blob/v2.1.0/mllib/src/main/scala/org/apache/spark/ml/recommendation/ALS.scala
- Documentation: https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.ml.recommendation.ALS https://spark.apache.org/docs/2.1.0/ml-collaborative-filtering.html

- New: Nonnegative matrix factorisation
  - Constraint: factor matrices are nonnegative

CF in ML

- *numBlocks*: the number of blocks the users and items will be partitioned into in order to parallelize computation (defaults to 10)
- *rank:* the number of latent factors in the model (defaults to 10)
- *maxIter:* the maximum number of iterations to run (defaults to 10)
- *regParam*: the regularization parameter in ALS (defaults to 1.0)
- *implicitPrefs*: specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data (defaults to false which means using explicit feedback)
- *alpha:* a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations (defaults to 1.0)
- *nonnegative*: whether or not to use nonnegative constraints for least squares (defaults to false)



```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 = 1.0,
      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(intermediateRDDStorageLevel != StorageLevel.NONE,
      "ALS is not designed to run without persisting intermediate RDDs.")
    val sc = ratings.sparkContext
    val userPart = new ALSPartitioner(numUserBlocks)
    val itemPart = new ALSPartitioner(numItemBlocks)
    val userLocalIndexEncoder = new LocalIndexEncoder(userPart.numPartitions)
    val itemLocalIndexEncoder = new LocalIndexEncoder(itemPart.numPartitions)
    val solver = if (nonnegative) new NNLSSolver else new CholeskySolver
    val blockRatings = partitionRatings(ratings, userPart, itemPart)
      .persist(intermediateRDDStorageLevel)
    val (userInBlocks, userOutBlocks) =
      makeBlocks("user", blockRatings, userPart, itemPart, intermediateRDDStorageLevel)
    // materialize blockRatings and user blocks
    userOutBlocks.count()
```

Blocked Implementation of ALS

- Group the two sets of factors ("users" & "items") into blocks
- Reduce **communication** by only sending 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
- Need to pre-compute info about the ratings matrix to determine the "**out-links**" of each user (which blocks of items it will contribute to) and "**in-link**" information for each item (which of the feature vectors it receives from each user block it will depend on).
- This allows us to send only an array of feature vectors between each user block and item block, and have the item block find the users' ratings and update the items based on these messages.

Explicit vs. Implicit Feedback

- What if we only have access to implicit feedback (e.g. views, clicks, purchases, likes, shares etc.).
- ICDM2008: CF for Implicit Feedback Datasets
- Treat the data as numbers representing the strength in observations of user actions (e.g, #clicks, the cumulative duration spent viewing a movie), related to the level of confidence in observed user preferences. The model then tries to find latent factors that can be used to predict the expected preference of a user for an item.
- Low-rank approximations for a(nother) preference matrix \`P\` where the elements of \`P\` are 1 if r is greater than 0 and 0 if r is less than or equal to 0. The ratings then act as 'confidence’

References

- http://ieeexplore.ieee.org/document/5197422/
- http://ieeexplore.ieee.org/document/4781121/

Week 10 Contents

- Recommender Systems & Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Collaborative Filtering in Spark
- **Final Remarks**

Key in Scalable ML

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

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

- Minimize Network Communication
  → Overhead in parallelisation, not the more the better

Developing Your Software

- Do not work on the full data immediately, get a small subset or a reduced version to study, develop, debug, and test. In this way, you can develop your software much faster, made/find mistakes much earlier
- Break down big/difficult problem into smaller/easier substeps (avoid blackbox debugging)
- Be structured, organised, and logical
- Keep good documentation and learn how to search (the correct keywords) for help online

Life is Short

The World is Big

Be Scalable

Keep Learning