CS535 Big Data

PART B. GEAR SESSIONS
SESSION 4: LARGE SCALE RECOMMENDATION SYSTEMS AND SOCIAL MEDIA

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs
- Today is the last day of discussion period for Session III on Piazza
- Watch video clips on Canvas ➔ Assignments ➔ Echo360
- Feedbacks are available in Canvas
- Please arrange a meeting if needed

Topics of Today's Class
- Part 1: Distributed implementation of Triplets View in GraphX
  - Recommendation Systems
- Part 2: Introduction and Content based recommendation systems
- Part 3: Collaborative Filtering (Case study of Amazon's Item-to-Item model and Netflix' Latent Factor Model)

Efficient lookup of edges
- Edges within a partition are clustered by source vertex id using a compressed sparse row (CSR) representation and hash-indexed by their target id
- CSR with an example:
  - With a sparse n x m matrix
  - Using three (1 dimensional) arrays (V, ColInd, RowEnds)
  - Y = [[0 0 0] [5 0 0] [0 0 0] [0 6 0]]
  - ColInd = [0 1 2 1]
  - RowEnds = [0 0 2 3 4]
  - Index in Ind where the given row starts

Index Reuse
- GraphX inherits the immutability of Spark
  - All graph operators logically create new collections rather than destructively modifying existing ones
- Derived vertex and edge collections can often share indices to reduce memory overhead and improve local performance
  - Hash index on vertices can enable fast aggregation and resulting aggregates share the index with the original vertices
  - Faster Joins
    - Vertex collections sharing the same index can be joined by a coordinated scan
    - Without requiring any index lookups
    - Index reuse reduces the per-iteration runtime of PageRank on the twitter graph by 59% (GraphX paper)
  - Operators that do not modify the graph structure (e.g. mapV) automatically preserve indices
  - Faster joins
  - Operators that do not modify the graph structure (e.g. map) automatically preserve indices

http://www.cs.colostate.edu/~cs535

Spring 2020 Colorado State University, page 1
### Implementing the Triplets View

- **Triplets view**
  - Three way join between the source and destination vertex properties and the edge properties

- **Vertex Mirroring**
- **Multicast Join**
- **Partial Materialization**
- **Incremental View Maintenance**

#### Vertex Mirroring
- Join requires data movement
- Vertex and edge property collections are partitioned independently
- Three-way join
  - Shipping the vertex properties across the network to the edges
  - Setting the edge partitions as the join sites

- **Observation 1:** Real-world graphs commonly have orders of magnitude more edges than vertices
- **Observation 2:** A single vertex may have many edges in the same partition
  - Enabling substantial reuse of the vertex property

#### Multicast Join
- **Broadcast join**
  - All vertices are sent to each edge partition
- **Multicast join**
  - Each vertex property is sent only to the edge partitions that contain adjacent edges
  - Join site information is stored in the routing table
    - Co-partitioned with the vertex collection
    - Routing table is associated with the edge collection
    - Routing table is constructed lazily upon first instantiation of the triplets view
- **Example**
  - Per-city partitioning scheme on the Facebook social network graph
  - 50.5% reduction in query time

#### Partial Materialization
- Local joins at the edge partitions
- Mirrored vertex properties are stored in local hash maps on each edge partition
- Referenced when the triplets are constructed

#### Incremental View Maintenance
- Iterative graph algorithms often modify only a subset of the vertex properties in each iteration
- **Incremental view maintenance**
  - To avoid unnecessary movement of unchanged data
    - After each graph operation
      - You can track which vertex properties have changed since the triplets view was last constructed
      - When the triplets view is accessed next time
        - Only the changed vertices are re-routed to their edge-partition join sites
        - Local mirrored values of the unchanged vertices are reused

#### Query Optimizations for the mrTriplets operator
- **Filtered Index Scanning**
  - myTriplets operator logically involves a scan of the triplets view to apply user-defined map function
    - As iterative graph algorithms converge, the working sets tend to shrink
    - Map function skips many Triplets
  - **Active set**
    - Map function only need to operate on triplets containing active vertices
    - Defined by the application specific predicate
    - E.g. connected component analysis
    - Indexed scan for the triplets view
    - Application expresses the current active set by restricting the graph using subgraph operator
    - Filter the triplets using this vertex predicate
Query Optimizations for the mrTriplets operator

- **Automatic Join Elimination**
  - Some operations on triplets view may access only one of the vertex properties or none at all.
  - For example, counting the degrees of each vertex.
  - GraphX uses a JVM’s bytecode analyzer to inspect user-defined functions at runtime.
  - If only one property is referenced and the triplets view has not been already materialized:
    - GraphX rewrites the query plan for generating the triplets view.
    - From three-way join to a two-way join.
  - If none of the vertex properties are referenced:
    - GraphX eliminates the join entirely.

Additional Optimizations

- **Memory-based Shuffle**
  - Spark’s default shuffle implementation materializes the temporary data to disk.
  - GraphX modified the shuffle phase to materialize map outputs in memory and remove this temporary data using a timeout.
- **Batching and Columnar Structure**
  - In the join code, batch a block of vertices routed to the same target join site and convert the block from row-oriented format to column-oriented format.
  - Apply the LZF compression algorithm on these blocks to send them.
- **Variable Integer Encoding**
  - While GraphX uses 64-bit vertex ids, most of ids are smaller than $2^{64}$.
  - GraphX uses a variable-encoding scheme.
  - Uses only first 7 bits to encode the value.

GEAR Session 4. Large Scale Recommendation Systems and Social Media

Lecture 1. Large Scale Recommendation Systems

Recommendation Systems: Introduction

The long tail phenomenon

- Distribution of numbers with a portion that has a large number of occurrences far from the “head” or central part of the distribution.
  - The vertical axis represents popularity.
  - The items are ordered on the horizontal axis according to their popularity.
  - The long-tail phenomenon forces online institutions to recommend items to individual users.

Recommendation systems

- Seek to predict the “rating” or “preference” that a user would give to an item.

http://www.cs.colostate.edu/~cs535
Applications of Recommendation Systems
- Product recommendations
  - Amazon or similar online vendors
- Movie recommendations
  - Netflix offers its customers recommendations of movies they might like
- News articles
  - News services have attempted to identify articles of interest to readers based on the articles that they have read in the past
  - Blogs, YouTube

Types of Recommendation Systems
- Random prediction algorithm
  - Randomly chooses items from the set of available items and recommends them to the users
  - Accuracy of this algorithm is poor
- Frequent sequence
  - Uses the frequent pattern to recommend other items
- Content based algorithms
  - Based on properties of items
  - Similarity of items is determined by measuring the similarity in their properties
  - Collaborative Filtering algorithms (CF)
  - Based on the relationship between users and items
  - Similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items
- Serendipitous recommendation systems
  - Assumes that the user may want to be surprised with something unexpected
  - From the results of existing recommendation systems, SR increases diversity and novelty

Content-Based Recommendations
- Focuses on properties of items
  - Similarity of items is determined by measuring the similarity in their properties

Item Profiles
- A record or collection of records representing important characteristics of the item
  - E.g. the features of a movie
    - The set of actors of the movie (Some viewers prefer movies with their favorite actors)
    - The director
    - The year in which the movie was made
    - The genre or general type of movie
    - Other features: manufacturer, screen size, etc.

Discovering Features of Documents
- Some items have features that are not immediately apparent to the systems
  - E.g. document collections and images
    - E.g. News articles
      - Suggesting articles on topics a user is interested in
      - Possible features
        - \( n \) words with the highest TF-IDF scores
        - \( n \) percentage of word with the highest TF-IDF scores
  - To measure the similarity
    - Jaccard distance or Cosine distance

GEAR Session 4. Large Scale Recommendation Systems and Social Media
Lecture 1. Large Scale Recommendation Systems
Recommendation Systems: Content-based Recommendations

http://www.cs.colostate.edu/~cs535
Obtaining Item Features from Tags of Images

- Crowd sourcing
  - Inviting users to tag the items
  - del.icio.us: earlier attempt to tag massive amount of data
  - Yahoo: invited user to tag Web pages

- Disadvantage
  - Are users willing to take the trouble to create the tags?
  - Erroneous tags can bias the system

Representing Item Profiles

- Goal
  - Create both an item profile consisting of feature-value pairs and a user profile summarizing the preferences of the user

- Example
  - Word vector (with 0's and 1's)
    - 1 represents the occurrence of a high TF-IDF word in the document
    - 0 represents the occurrence of a low TF-IDF word in the document

Representing Item Profiles

- Suppose the only features of movies are the set of actors and the average rating
- Consider two movies with five actors each
- Two of the actors are in both movies
- Example
  - Movie A: [1, 1, 0, 1, 1]
  - Movie B: [1, 1, 0, 1, 1, 1]
  - Cosine similarity between above vectors

User Profiles

- Using the utility matrix representing the connection between users and items
  - Example: "Find user's preference for movies with a specific actor!"

  - Suppose user U gives an average rating of 3
    - Three movies with Julia Roberts as an actor, and ratings of 3, 4, and 5
    - The user profile for U has, in the component for Julia Roberts, the average of (2 − 4), (3 − 4), and (5 − 4), that is, −2/3

Classification Algorithms

- Decision tree
  - A collection of nodes, arranged as a binary tree
  - The leaves render decisions
    - In this case, the decision would be "likes" or "doesn't like"
  - Each interior node is a condition on the objects being classified
Collaborative Filtering

- Identifies similar users and recommending what similar users like
- Instead of using features of items to determine their similarity
- Focus on the similarity of the user rating for two items
- Users are similar if their rating vectors are close according to some distance measure
- Jaccard or cosine distance
- Recommendation for a user \( U \) is made by looking at the users that are most similar to \( U \)
- Recommending items that these users like

Measuring Similarity? -- Jaccard Similarity Coefficient

<table>
<thead>
<tr>
<th>SW Episode VII</th>
<th>SW Episode VIII</th>
<th>SW Episode IX</th>
<th>Frozen I</th>
<th>Frozen II</th>
<th>Joker</th>
<th>Average</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewer A</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td>( J(A, B) = 1/2 \times 100)</td>
</tr>
<tr>
<td>Reviewer B</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td>( J(A, C) = 2/2 \times 100)</td>
</tr>
<tr>
<td>Reviewer C</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td>( J(A, D) = 1/2 \times 100)</td>
</tr>
</tbody>
</table>

Jaccard Similarity Coefficient

- If the utility matrix only reflects purchases of the movie, this can be useful
- If utilities are more detailed ratings, the Jaccard distance loses important information

Measuring Similarity? -- Cosine Similarity

<table>
<thead>
<tr>
<th>SW Episode VII</th>
<th>SW Episode VIII</th>
<th>SW Episode IX</th>
<th>Frozen I</th>
<th>Frozen II</th>
<th>Joker</th>
<th>Average</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewer A</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td>( \cos(A, B) = 20/ (\sqrt{40} \times \sqrt{40}) = 0.37)</td>
</tr>
<tr>
<td>Reviewer B</td>
<td>4.66</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td>( \cos(A, C) = 20/ (\sqrt{16} \times \sqrt{16}) = 0.50)</td>
</tr>
<tr>
<td>Reviewer C</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td>( \cos(A, D) = 20/ (\sqrt{4} \times \sqrt{4}) = 0.50)</td>
</tr>
</tbody>
</table>

Cosine Similarity

- Clustering Users and Items
  - It is hard to detect similarity among either items or users
  - We have little information about user-item pairs in the sparse utility matrix

- Clustering items based on the series

http://www.cs.colostate.edu/~cs535
Clustering Users and Items

- Find the clusters to which user and items belong
- Estimate entries based on the user-item relationship
- If the entry is empty, find the most similar item group

<table>
<thead>
<tr>
<th>SW Episode</th>
<th>Episode 1</th>
<th>Episode 2</th>
<th>Episode 3</th>
<th>Episode 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewer A</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Reviewer B</td>
<td>4.66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reviewer C</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Reviewer D</td>
<td>4</td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

This material is built based on:
- Greg Linden, Brent Smith, and Jeremy York, "Amazon.com Recommendations, Item-to-Item Collaborative Filtering" IEEE Internet Computing, 2003
- Amazon.com uses recommendations as a targeted marketing tool
  - Email campaigns
  - Most of their web pages

Item-to-item collaborative filtering

- It does NOT match the user to similar customers
- Item-to-item collaborative filtering
  - Matches each of the user’s purchased and rated items to similar items
  - Combines those similar items into a recommendation list

Determining the most-similar match

- The algorithm builds a similar-items table
  - By finding items that customers tend to purchase together
- How about building a product-to-product matrix by iterating through all item pairs and computing a similarity metric for each pair?
  - Many product pairs have no common customer
  - If you already bought a TV today, will you buy another TV again today?

http://www.cs.colostate.edu/~cs535
Calculating the similarity between a single product and all related products:

For each item in product catalog, I1
- For each customer C who purchased I1
  - Record that a customer purchased I1 and I2
- For each item I2 purchased by customer C
- Compute the similarity between I1 and I2

Co-Similarity(A,B) = cos(A,B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}

Option 1. Using co-occurrence matrix
- If an item has been purchased by the same user together many times, it is considered as a "similar" item

Option 2. Using cosine measure
- Each vector corresponds to an item rather than a customer
  - M dimensions correspond to customers who have purchased that item

Creating a similar-item table
- Similar-items table is extremely computing intensive
  - \(O(NM)\) in the worst case
  - \(O(N)\) in the average case
    - Where \(N\) is the number of items and \(M\) is the number of users

- Most customers have very few purchases
- Sampling customers who purchase best-selling titles reduces runtime even more
  - With little reduction in quality

Cooccurrence matrix

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Purchase record for the user \(U=\{ I1, I2, I3, I4, I5 \}\)
- Purchase record for the user \(U=\{ I1, I2, I3, I4 \}\)
- Purchase record for the user \(U=\{ I2, I3, I4 \}\)
- Purchase record for the user \(U=\{ I2, I3 \}\)
- Purchase record for the user \(U=\{ I1, I2 \}\)
- Purchase record for the user \(U=\{ I1, I2, I3 \}\)

Cosine similarity (\(I1, I5\)) = \frac{I1 \cdot I5}{\|I1\| \cdot \|I5\|}

\frac{0 \cdot 1}{\sqrt{5} \cdot \sqrt{5}} = 0/5
Scalability

- Amazon.com has around 110 million active customers (244 million total customers) and several million catalog items.
- Traditional collaborative filtering does little or no offline computation.
- Online computation scales with the number of customers and catalog items.

Key scalability strategy for Amazon recommendations

- Creating the expensive similar-items table offline.
- Online component
  - Looking up similar items for the user’s purchases and ratings.
  - Scales independently of the catalog size or the total number of customers.
- It is dependent only on how many titles the user has purchased or rated.

Recommendation quality

- The algorithm recommends highly correlated similar items.
- Recommendation quality is excellent.
- Algorithm performs well with limited user data.

Dataset

- Audioscrobbler dataset
  - 2002, Richard Jones
  - Collecting and analyzing user’s songs to generate recommendation
  - Started with support for Winamp and XMMS
  - iTunes, Winamp, Windows Media Player, Foobar, iPod, Amarok, Rhythmbox, mpd, XBox media center, Slimserver, Jidora, mpg321, Mauve, Rhapsody, YME, Soundbridge, VLC…

Dataset

- Confined rating system
  - “Bob rates Coldplay 3.5 stars.”
  - Users rate music far less frequently than they play music.

- Audioscrobbler dataset
  - “Bob played Coldplay track.”
  - Each individual data carries less information.

- Implicit feedback
  - User-artist connections are implied as a side effect of other actions.

http://www.cs.colostate.edu/~cs535

Spring 2020 Colorado State University, page 9
## Dataset
- 141,000 unique users
- 1.6 million unique artists
- 24.2 million user's plays of artist are recorded
- User_artist_data.txt
- On average, each user has played songs from about 171 artists (out of 1.6 M)
- Extremely sparse dataset

## Netflix Prize
- The Netflix Prize challenge concerned recommender systems for movies (October, 2006)
- Netflix released a training set consisting of data from almost 500,000 customers and their ratings on 18,000 movies.
- More than 100 million ratings
- The task was to use these data to build a model to predict ratings for a hold-out set of 3 million ratings

## GEAR Session 4: Large Scale Recommendation Systems and Social Media
Lecture 1. Large Scale Recommendation Systems

### Recommendation Systems

#### Collaborative Filtering: Latent Factor Model
- Collects and analyzes a large amount of information on users' behaviors, activities or preferences and predicts what users will like based on their similarity to other users
- Explicit data collection
  - Rate an item
  - Search history
  - Favorite item
  - Wish list
- Implicit data collection
  - Viewing times
  - Tracking online purchases
  - Analyzing the user's social network

#### Collaborative filtering
- Two users may share similar tastes because they are the same age
  - It is NOT an example of collaborative filtering
- Two users may both like the same song because they play many other same songs
  - It is an example of collaborative filtering
- Algorithm that learns without access to user or artist attributes

#### Latent-Factor model
- Tries to explain observed interactions between large numbers of users and products through a relatively small number of unobserved, underlying reasons
- Within the music business context,
  - Why millions of people buy a particular few of thousands of possible albums by describing users and albums for tens of genres and tastes that are not directly observable

http://www.cs.colostate.edu/~cs535
Simplified illustration of the latent factor approach

How do we model this?
- User and product data in a large matrix $\mathbf{A}$
  - Row $i$ and column $j$
  - If user $i$ has played product $j$
- The $k$ columns correspond to the latent factors

Creating user and artist matrices
- Two matrices
  - Matrix $\mathbf{X}$ for users
    - Each value corresponds to a latent feature in the model
  - Matrix $\mathbf{Y}$ for products
    - Each value corresponds to a latent feature in the model
- Rows express how much users and products associate with these latent features
- Product of $\mathbf{X}$ and $\mathbf{Y}$
  - Complete estimation of the entire, dense user-product interaction matrix

Computational challenge
- $\mathbf{A} = \mathbf{X} \mathbf{Y}^T$ generally no solution
- $\mathbf{X}$ and $\mathbf{Y}$ are not large enough

Goal
- Finding the best $\mathbf{X}$ and $\mathbf{Y}$

Alternating Least Squares (ALS)
- Alternating least squares algorithm to compute $\mathbf{X}$ and $\mathbf{Y}$
- Spark MLlib’s ALS implementation

Step 1
- $\mathbf{Y}$ is not known
  - Initialized to a matrix with randomly chosen row vectors
  - Then simple linear algebra gives the best $\mathbf{X}$, given $\mathbf{Y}$ and $\mathbf{A}$
    - $\mathbf{A} = \mathbf{X} \mathbf{Y}^T$
    - $\mathbf{X} = \frac{\mathbf{A}}{\mathbf{Y}^T}$
- Equality cannot achieved exactly
  - The goal becomes to minimize $\| \mathbf{A} \|_F^2 - \| \mathbf{X} \mathbf{Y}^T \|_F^2$
  - The sum of squared differences between the two matrices’ entries

http://www.cs.colostate.edu/~cs535
Alternating Least Squares (ALS)

- Step 2.
  - Repeat similar sequence as step 1 to compute $Y$ from the $X$ (from step 1)
- Step 3.
  - Repeat similar sequence as step 1 to compute $X$ from the $Y$ (from step 2)

... $X$ and $Y$ do eventually converge to good (acceptable) solutions

Preparing the Data

- Files are available at /user/ds/
- Spark MLlib's ALS implementation
  - Requires numeric IDs for users and items
  - Nonnegative 32-bit integers
  - An ID larger than Integer.MAX_VALUE cannot be used

```
val rawUserArtistData = sc.textFile("hdfs:///user/ds/user_artist_data.txt")
rawUserArtistData.map(_.split(' ')(0).toDouble).stats()
rawUserArtistData.map(_.split(' ')(1).toDouble).stats()
```

Maximum user IDs: 24443548
Maximum artist IDs: 2147483647
No additional transformation will be needed

Extracting names

- artist_data.txt
- Artist ID and name separated by a tab

```
val rawArtistData = sc.textFile("hdfs:///user/ds/artist_data.txt")
val artistByNumber = rawArtistData.map { line =>
  val (id, name) = line.split(_='.').map(_.trim)
  (id.toInt, name.trim)
}
```

- Straightforward parsing of the file into (Int, String) tuples will fail

Extracting names

- Scala’s Option class
  - Option represents a value that might only optionally exist

```
val artistById = rawArtistData.flatMap { line =>
  val (id, name) = line.split(_='.').map(_.trim)
  if (name.length > 0) Some((id.toInt, name.trim))
  else None
}
```
Building a First Model

- Two transformations are required
  - Aliases dataset should be applied to convert all artist IDs to a canonical ID
  - The data should be converted to a Rating object
- User-product value data

```
import org.apache.spark.mllib.recommendation._
val bArtistAlias = sc.broadcast(artistAlias)
val trainData = rawUserArtistData.map(line =>
  val Array(userID, artistID, count) = line.split(' ').map(_.toInt)
  val finalArtistID = bArtistAlias.value.getOrElse(artistID, artistID)
  Rating(userID, finalArtistID, count)
).cache()
```

Building the ALS model

- Constructs model as a MatrixFactorizationModel

```
val model = ALS.trainImplicit(trainData, 10, 5, 0.01, 1.0)
```

Spot Checking Recommendations

- To see if the artist recommendations for user(2093760) makes any intuitive sense

```
val rawArtistsForUser = rawUserArtistData.map(_.split(' ')).filter {
  case Array(_, _, _) =>
    user.toInt = 2093760
}
val existingProducts = rawArtistsForUser.map{
  case Array(_, artist, _) =>
    artist.toInt
}.collect().toSet
artistByID.filter{
  case (id, name) =>
    existingProducts.contains(id)
}.values.collect().Foreach(println)
...```

Broadcast variables

- For the case that many tasks (from different closures) need access to the same (immutable) data structure
- Extends normal handling of task closures
- Caching data as raw Java objects on each executor
- Caching data across multiple jobs and stages
- Spark will send, and hold in memory, just one copy for each executor in the cluster
- Saves network traffic and memory
Spot Checking Recommendations

- To see five recommendations for this user (ID: 2093760)

```scala
val recommendations = model.recommendProducts(2093760, 5)
recommendations.foreach(println)
```

Evaluating the Recommendation Model

What is a “good” recommendation?

- “a popular artist”?
- “artists the user has listened to”?
- “artists the user will listen to”?

Preparing data for evaluation

- To perform a meaningful evaluation, some of the artist play data can be set aside
  - Hidden from the ALS model building process
- The held-out data can be used as a collection of good recommendations for each user
- Compute the recommender’s score

AUC metric

- Rank 1.0 is perfect, 0.0 is the worst
- Receiver Operating Characteristic (ROC)
  - Based on the rank used to decide final recommendations
- Area Under the Curve (AUC) of ROC may be used as the probability that a randomly chosen good recommendation ranks above a randomly chosen bad recommendation
- Spark's BinaryClassificationMetrics
  - Computes AUC per users and averages the result
  - Generating mean AUC

MAP metric

- Mean average precision
- Focuses on the top recommendations

http://www.cs.colostate.edu/~cs535
Computing AUC

- 90% of the data is used for training and the remaining 10% for validation

```scala
import org.apache.spark.rdd._

def areaUnderCurve(positiveData: RDD[Rating], bAllItemIDs: Broadcast[Array[Int]], predictFunction: (RDD[(Int, Int)] => RDD[Rating])) = {
...
}

val allData = buildRatings(rawUserArtistData, bArtistAlias)

val Array(trainData, cvData) = allData.randomSplit(Array(0.9, 0.1))

trainData.cache()

val allItemIDs = allData.map(_.product).distinct().collect()

val bAllItemIDs = sc.broadcast(allItemIDs)

val model = ALS.trainImplicit(trainData, 10, 5, 0.01, 1.0)

val auc = areaUnderCurve(cvData, bAllItemIDs, model.predict)
```

k-Fold Cross-validation

- Create a k-fold partition of the dataset
- For each of the k experiments use k-1 folds for training
- The remaining fold for testing

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Experiment 1</th>
<th>Experiment 2</th>
<th>Experiment 3</th>
<th>Total number of examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

True error estimate

- k-fold cross validation is similar to random subsampling
- The advantage of k-Fold Cross validation
- All the examples in the dataset are eventually used for both training and testing
- The true error is estimated as the average error rate

\[ E = \frac{1}{K} \sum_{i=1}^{K} E_i \]

Hyperparameter selection

- `MatrixFactorizationModel`  
  - `ALS.trainImplicit()`  
  - `rank = 10`  
  - The number of latent factors in the model  
  - The number of columns, k
  - `iterations = 5`  
  - The number of iterations that the factorization runs
  - `lambda = 0.1`  
  - A standard overfitting parameter
  - Higher value guards against overfitting
  - Values that are too high will decrease the factorization's accuracy
  - `alpha = 1.0`  
  - Controls the relative weight of observed versus unobserved user-product interactions in the factorization

http://www.cs.colostate.edu/~cs535
<table>
<thead>
<tr>
<th>Questions?</th>
<th></th>
</tr>
</thead>
</table>
