CS535 Big Data

4/13/2020  Week 12-A Sangmi Lee Pallickara

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

Topics of Today's Class
- Part 1: Collaborative Filtering with the case study of Item-to-Item CF
- Part 2: Collaborative Filtering with the case study of Latent Factor CF
- Part 3: Evaluating Recommendation Systems

FAQs
- Wednesday (4/15) is the GEAR Session IV presentation
- Discussion will be available on 4/15, 16, and 17
- Watch video clips on Canvas ➔ Assignments ➔ Echo360

Recommendation System
- Amazon.com uses recommendations as a targeted marketing tool
  - Email campaigns
  - Most of their web pages
- Amazon.com uses recommendations in their email campaigns
- Amazon.com's recommendations are targeted and personalized

Amazon.com: Item-to-item collaborative filtering
What if they use a Traditional CF

- Build a utility matrix
  - N-dimensional vector of items per user regarding their ratings
  - Where N is the number of distinct catalog items
  - Positive for purchased or positively rated items
  - Negative for negatively rated items

- To compensate for the best-selling items
  - Multiplies the vector components by the inverse frequency
  - Making less well-known items more relevant

What if they use a Traditional CF

- Find out similar users
  - Cosine similarity between the vectors
  - E.g. user A and B
  - \[ \text{Cosine Similarity}(A,B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \]

- Select items within the group of items purchased by the similar users
  - E.g. Rank each item according to how many similar customers purchased it

- Highly ranked item(s) will be recommended

What if they use a Traditional CF

- For N items (in the catalog) and M users

  - Worst case
    - \( O(MN) \)

  - Average customer vector is extremely sparse
    - \( O(M) \)
    - Most of scanning will be approximately \( O(M) \)
    - There are a few customers who have purchased or rated a significant percentage of the catalog
    - Therefore, the final performance of the algorithm is approximately \( O(M+N) \)

What if they use a Traditional CF

- Dimensionality reduction
  - Reducing M by randomly sampled customers or discarding customers with few purchases
  - Reducing N by discarding very popular or unpopular items

  - What will be the problem of above approaches?

- 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

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

### Part 1: tracking co-occurrence items

<table>
<thead>
<tr>
<th>Purchase record for the user U={I, I, I}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purchase record for the user U={I, I, I}</td>
</tr>
<tr>
<td>Purchase record for the user U={I, I, I}</td>
</tr>
<tr>
<td>Purchase record for the user U={I, I, I}</td>
</tr>
</tbody>
</table>

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

### Part 2: Computing similarity between items

- Using cosine measure
  - Each vector corresponds to an item
  - Item A and B (rather than customers)
  
  $$\text{Cosine}_\text{Similarity}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

http://www.cs.colostate.edu/~cs535
Creating a similar-item table

- Generating the co-occurrence table is extremely computing intensive
  - \( O(N^2M) \) in the worst case
  - Where \( N \) is the number of items and \( M \) is the number of users
- Average case is closer to \( O(NM) \)
  - Most customers have very few purchases
- Sampling customers who purchase best-selling titles reduces runtime even more
  - With little reduction in quality
  - Offline computation

Generating the final recommendation

- Using a similar-items table
  - Algorithm finds items similar to each of the user’s purchases and ratings, aggregates those items
  - Recommends the most popular or correlated items
  - Quick, depending only on the number of items the user purchased or rated

Scalability

- Amazon.com has around 606 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

GEAR Session 4. Large Scale Recommendation Systems and Social Media
Lecture 2. Large Scale Recommendation Systems
Recommendation Systems
Recommending Music and the Audioscrobbler Dataset
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, Jzone, mpg321, Mume, Rhapsody, YME, Soundbridge, VLC...

Audioscrobbler 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

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

Collaborative filtering

• 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
    - Rates an item
    - Search history
    - Favorite item
    - Wish list
  - Implicit data collection
    - Viewing items
    - 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

Simplified illustration of the latent factor approach

<table>
<thead>
<tr>
<th>Nancy</th>
<th>Area 1</th>
<th>Jennifer</th>
<th>Area 3</th>
<th>Tom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area 2</td>
<td>Bob</td>
<td>Area 4</td>
<td>Fast and Furious</td>
<td></td>
</tr>
</tbody>
</table>

How do we model this?

- User and product data in a large matrix $A$
- Row $i$ and column $j$
- If user $i$ has played product $j$
- The columns correspond to the latent factors

Creating user and artist matrices

- Two matrices
  - Matrix $X$ for user
    - Each value corresponds to a latent feature in the model
  - Matrix $Y$ 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 $X$ and $Y$ completes estimation of the entire, dense user-product interaction matrix

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

- $A = X^T Y$ generally no solution
- If $X$ and $Y$ are not large enough

Goal
- Finding the best $X$ and $Y$

Alternating Least Squares (ALS)

- Alternating least squares algorithm to compute $X$ and $Y$
  - Spark MLlib’s ALS implementation

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

  - 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

- Takes advantage of the sparsity of the input data
- Easy to apply data parallelism
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 artistByID = rawArtistData.map{ line =>
  val (id, name) = line.span(_!='	')
  id.toInt, name.trim
}

val artistByID = rawArtistData.flatMap{ line =>
  val (id, name) = line.span(_ != 't')
  if (name.isEmpty) { None }
  else { try {
    Some((id.toInt, name.trim))
  } catch {
    case e: NumberFormatException => None
  }}
}

Extracting names

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

Building Model

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

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()
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

Building the ALS model
- Constructs model as a MatrixFactorizationModel

Retrieving some feature vectors
- Array of 10 numbers

Spot Checking Recommendations
- To see if the artist recommendations for user(2093760) makes any intuitive sense
- To see five recommendations for this user (ID: 2093760)
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
- Scala’s BinaryClassificationMetrics
- Computes AUC per user and averages the result
- Generating mean AUC

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

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

```scala
val allData = buildRatings(rawUserArtistData, bArtistAlias)
val (trainData, cvData) = allData.randomSplit(Array(0.9, 0.1))
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>Test example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td></td>
</tr>
<tr>
<td>Experiment 2</td>
<td></td>
</tr>
<tr>
<td>Experiment 3</td>
<td></td>
</tr>
<tr>
<td>Experiment 4</td>
<td></td>
</tr>
</tbody>
</table>
```

- Total number of examples

**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

\[
E = \frac{1}{k} \sum E_i
\]

**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 values guard 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