Collaborative filtering [1/2]

- 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 [2/2]

- 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

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 $i$ columns correspond to the latent factors

http://www.cs.colostate.edu/~cs535
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 $Y$
  - Complete estimation of the entire, dense user-product interaction matrix

Computational challenge

- $A \times X Y$ 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 $A$
- $A Y = X Y (Y^T Y)^{-1}$

Equality cannot achieved exactly

- The objective is non-convex
- Gradient descent can be used as an approximate approach
- Many iterations and expensive computing cost
- If we fix the set of variables $X$ and treat them as constants
- The objective becomes a convex function of $Y$ and vice versa

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

Alternating Least Squares (ALS)

- Takes advantage of the sparsity of the input data
- Easy to apply data parallelism
GEAR Workshop | Advanced Big Data Analytics Case Study
Recommendation Systems
Building a model with Spark MLlib

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

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

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

Extracting names
- Artist ID and name separated by a tab

```scala
val rawArtistData = sc.textFile("hdfs:///user/ds/artist_data.txt")
val artistByID = rawArtistData.map(line =>
  val (id, name) = line.span(_!='	')
  (id.toInt, name.trim)
)
```

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

```scala
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
    }
  }
)
```

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

```scala
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

http://www.cs.colostate.edu/~cs535
Building the ALS model
- Constructs model as a MatrixFactorizationModel

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

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 user and averages the result
  - Generating mean AUC

MAP metric
- Mean average precision
  - Focuses on the top recommendations

Evaluating recommendation model
- GEAR Workshop I | Advanced Big Data Analytics Case Study
  - Recommendation Systems

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

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

```python
import org.apache.spark.rdd._
def areaUnderCurve(positiveData: RDD[Rating], allItemIDs: 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()
cvData.cache()
val allItemIDs = allData.map(_.product).distinct().collect()
bAllItemIDs = sc.broadcast(allItemIDs)
val model = ALS.trainImplicit(trainData, 10, 5, 0.01, 1.0)
val auc = areaUnderCurve(cvData, bAllItemIDs, model.predict)
```

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

\(k\)-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

```
def predictMostListened(sc: SparkContext, train: RDD[Rating]) = {
    val bListenCount = sc.broadcast(train.map(r => (r.product, r.rating)).reduceByKey(_ + _).collectAsMap())
    allData.map{ case (user, product) => Rating(user, product, bListenCount.value.getOrElse(product, 0.0)) }
    }
val auc = areaUnderCurve(cvData, bAllItemIDs, predictMostListened(sc, trainData))
```

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
Anomaly Detection

• Finding unusual things
• Case A: If we already know what "anomalous" meant for a dataset
  - Applying this rule
• Case B: Being "anomalous" is not defined
• Applications
  - Fraud detection, detect network attacks, discover problems in servers or other sensor-equipped machinery

Network Intrusion

• Cyber attacks
  - Attempt to exploit flaws in networking software to gain unauthorized access to a computer
  - Some exploit behaviors follow known patterns
  - E.g. accessing every port on a machine in rapid succession
• What about unknown pattern?
  - K-means can cluster connections based on statistics about each of them
  - Results cluster can collectively define types of connections that are like past connections
  - Anything not close to a cluster could be anomalous

Regression Vs. Classification to understand normal behaviors

• Dividing an ecommerce site's customers by their shopping habits and tastes
• Input features
  - Purchases
  - Clicks
  - Demographic information
• Output
  - Groupings of customers
  - E.g. fashion-conscious buyers, price-sensitive bargain hunters
• Can you determine this target label for each new customers?

Unsupervised learning

• Predicting without any target value
  - Because none is available
• What types of input are likely to occur and what types are not

K-means Clustering

• Unsupervised learning
• One of the most widely used clustering algorithm
• Detects k clusters in a dataset
  - k is a hyperparameter of the model
  - Choosing a good value for k is important

http://www.cs.colostate.edu/~cs535
KDD Cup 1999 Data Set

- The KDD Cup
  - Annual data mining competition hosted by a special interest group of the ACM
  - It was like Kaggle
- Summary information (pre-processed)
  - 73MB, 4,884 connections
  - Medium size data
- 38 Attributes
  - Number of bytes sent
  - TCP errors
  - etc.

KDD Cup - Data

- Each connection is one line of CSV-formatted data

KDD Cup - Workshops II

- Vassilvitskii Bahman Workshop II

\[ \text{Complexity} \]

- Finding optimal solution for k-means clustering problem for observations in d dimensions
- \( O(nkd^2) \), where \( n \) is the number of entities to be clustered
- For unknown \( k \) and \( d \)
- Heuristic approach (e.g. Lloyd’s algorithm)

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

- \( O(nkd^2) \), where \( n \) is the number of entities to be clustered
- For unknown \( k \) and \( d \)

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]

\[ \text{Finding optimal solution for k-means clustering problem for observations in d dimensions} \]
k-means clustering with Spark

```scala
import org.apache.spark.mllib.clustering.

val kmeans = new KMeans()

val model = kmeans.run(data)

model.clusterCenters.foreach(println)

val clusterLabelCount = labelsAndData.map{
  case(_,datum) =>
    val cluster = model.predict(datum)
    (cluster, label) =>
      println(s"$cluster %1s $label %18s $count %8s")
}
```

Choosing k

```scala
def distance(a : Vector, b : Vector) =
  math.sqrt(a.toArray.zip(b.toArray).map{p => p._1 * p._2}.map{d => d * d}.sum)

def distToCentroid(datum : Vector, model : KMeansModel) = {
  val cluster = model.predict(datum)
  val centroid = model.clusterCenters(cluster)
  distance(centroid, datum)
}
```

Choosing k

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

def clusteringScore(data : RDD[ resolve to Vector], k : Int) = {
  val kmeans = new KMeans()
  kmeans.setK(k)
  val model = kmeans.run(data)
  data.map{datum => distToCentroid(datum, model)}.mean()
}
```

```scala
(5 to 40 by 5).map{k =>
  (k, clusteringScore(data, k))
}.foreach(println)
```

Questions?