Large-scale Matrix Factorization for Music Recommendations

Background

In this assignment, your goal will be to predict personalized music recommendations. Let's say you want to know what would be the most likely choice of songs that you'd listen to based on your ratings of the music that you are listening to right now. How would we do that? Somehow recommending music based on user ratings seems inexplicable, but recommender systems do exceedingly well in identifying music based on our ratings.

We will use non-negative matrix factorization (NMF) as our choice of algorithm to build our recommender system. Most recommender systems rely on user ratings, i.e., they explicitly allow users to rate their choice of song/ movie on their website (e.g., Netflix, Amazon, etc.). However, we are going to rely on how often a song was played rather than the explicit rating provided by users. This is because, listeners rate music far less frequently than playing music. Thus, the dataset can be quite large, can cover more users/artists and contains richer information than a rating dataset. This type of data is called implicit feedback data since the user-artist relationships are implied as a side effect of their actions, and not an explicit rating/thumbs up option.

What to expect? What to hand in?

For this assignment, you can collaborate, but your reports and code must be your own. The total points on this assignment is 100, it is worth 15% of your overall grade.

Your write up should consist of the following subsections: (1) Collaborators, (2) Approach, and (3) Results. I expect that you will typeset your report. If you do not typeset your report, I will deduct 10 points from the final grade.

Deadlines

Deadlines are firm. You have up to three (3) "dog days" total (for the entire semester). Please email me if you need to use your dog days. Please submit your assignment by **11.59 PM on Mar 19, 2018**.

Programming

You are welcome to use either Scala or Python for your programming environment. I have tried to provide starting points for code snippets that will be a general framework for setting up your respective runs. You are welcome to use these as you wish, but note that the code has been provided only for illustrative purposes.

Email me and Yongli Zhu (@vols.utk.edu) with the following attachment [lastname]-HW2-submit.tgz on or before Mar 19, 2018. On the subject line, please include the following tag- [COSC-526: HW2].

Dataset

The dataset we will be using is distributed by last.fm in 2005 and can be found online as a compressed archive at http://bit.ly/1KiJdOR. Download the archive and you will find several files. The main dataset that we will be using is the user artist data.txt file. It has over 140,000 users, 1.5 million unique artists and roughly 24 million users' information about how often they played a given song/artist.

The dataset also gives the names of each artist by ID in the artist data.txt file. When the data was recorded, the names of the artists could be misspelled or non-standard, and this can be detected only later. For example, 'The Smiths,' 'Smiths The' and 'he smiths' may appear as distinct artist IDs, but they refer to the same artist. So, the dataset also includes artist alias.txt dataset, which maps the artist IDs that are known to be misspelled or variants to the canonical ID of the artist.

Preparing the data

In this section, we will try to understand some basics about processing the dataset. Note that the computations used in this assignment will take unusually large amounts of memory. You may need to specify flags such as `--driver-memory 6g` to have enough memory to complete the computations.

As you know, the alternating least squares (ALS) for NMF implementation requires numeric IDs for users and items, and further requires non-negative 32-bit integers as IDs. This means IDs larger than `Integer.MAX_VALUE` (2147483647) cannot be used. Does the data conform to this requirement? [*Hint:* Use the `stats()` command on the variable where you are storing the user-artist data.]

You also need to write code that will allow you to map the artist names that correspond to the numeric IDs. The first one will take the artist_data.txt file and map the names in that file with the IDs found in the user_artist_data.txt file. The second piece of code will allow you to map the artist_alias.txt file so that you can get the misspelled or non-standard names to be mapped to the artist's canonical names. [*Hint:* Use the `flatMap` function for the first one, and the `Map` function for the second one.]

Building a first model for NMF [20 points]

Although the dataset is now ready to use with the Spark MLLib NMF implementation (called `ALS` in Spark MLLib), there are two small transformations that you need to do on the data. First, you should apply the aliases dataset so that all of the artist IDs are converted to a canonical ID (if a different name exists). Second, we need to convert the data into a `Rating` object, which is what the ALS implementation requires.

Build an initial model for ALS using the `trainImplicit` function assuming you created a variable called `trainData` in the above step.

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

Note that the four other arguments in the `trainImplicit` function are hyper-parameters that can affect the quality of recommendations. We will work with these parameters in a later part of our assignment.

Further, these types of models in Scala (or Python) can take more memory. In particular, the model consists of 10 values for each user and artist, and in this case, we have more than 1.7 million of them! Thus, the model will have large user-feature and artist-feature matrices as RDDs of their own.

Spot checking recommendations [20 points]

Do these artist recommendations make any intuitive sense? Let's take users such as 2093760, 2093765, 2093771. Extract the IDs (and names, using the maps that you defined previously) of the artists that these users have listened to. Also, for the same users, print the recommendation estimates for the artists from the ALS recommender. Are the recommendations making sense?

Using the Cross-validation loop to study performance of our recommender [20 points]

Now, we will try to put our code into a cross-validation loop using the recommender we built above. This listing below gives a vignette on how to compute the MSE value based on the predictions obtained from NMF.

```
val ratesAndPreds = ratings.map { case Rating(user, artist, rate) =>
  ((user, artist), rate)
}.join(predictions)
val MSE = ratesAndPreds.map { case ((user, artist), (r1, r2)) =>
  val err = (r1 - r2)
  err * err
}.mean()
println("Mean Squared Error = " + MSE)
```

Using a 90%-10% training-testing split on the input data (i.e., the user-artist input matrix), build the NMF model and report MSE values for our recommender.

Playing with the hyper-parameters [20 points]

The hyper-parameters we used before include the following:

1. `rank = 10`: The number of latent factors in the model, or equivalently, the number of columns k in the user-feature, feature-artist matrices.
2. `iterations = 10`: The number of iterations that the factorization runs.
3. `lambda = 0.01`: A standard overfitting parameter.

4. `alpha = 1`: Controls the relative weights of the observed versus unobserved user-artist interactions in the factorization.

Now, run the model against a set of hyper-parameter choices including: `rank = 10` and `50`, `lambda = 1` and `0.0001`, `alpha = 1.0` and `40.0`, and report the MSE values for the same. We will keep the iterations to be only 10 in all the cases.

For which parameter settings, do you find that the MSE values are lower? Report your analyses based on the results that you observe from running the NMF under the cross-validation loop.