recommender

January 9, 2016

1 Music Recommender System

Estimated time: 8hrs

1.1 Description

For this project, you are to create a recommender system that will recommend musical artists to a user based on their listening history. Suggesting different songs or musical artists to a user is important to many music streaming services, such as Pandora and Spotify. In addition, this type of recommender system could also be used as a means of suggesting TV shows or movies to a user (e.g., Netflix).

To create this system you will be using Spark and the collaborative filtering technique. The instructions for completing this project will be laid out entirely in this file. You will have to implement any missing code as well as answer any questions.

Submission Instructions: * Add all of your updates to this IPython file and do not clear any of the output you get from running your code. * Upload this file onto moodle.

1.2 Datasets

You will be using some publicly available song data from audioscrobbler, which can be found here. However, we modified the original data files so that the code will run in a reasonable time on a single machine. The reduced data files have been suffixed with `_small.txt` and contains only the information relevant to the top 50 most prolific users (highest artist play counts).

The original data file `user_artist_data.txt` contained about 141,000 unique users, and 1.6 million unique artists. About 24.2 million users' plays of artists are recorded, along with their count.

Note that when plays are scribbled, the client application submits the name of the artist being played. This name could be misspelled or nonstandard, and this may only be detected later. For example, "The Smiths", "Smiths, The", and "the smiths" may appear as distinct artist IDs in the data set, even though they clearly refer to the same artist. So, the data set includes `artist_alias.txt`, which maps artist IDs that are known misspellings or variants to the canonical ID of that artist.

The `artist_data.txt` file then provides a map from the canonical artist ID to the name of the artist.

1.3 Necessary Package Imports

```
In [24]: from pyspark.mllib.recommendation import *
         import random
         from operator import *
         import sys
         from os.path import join, isfile, dirname
         from pyspark import SparkConf, SparkContext
         from operator import add
```

1.4 Loading data

Load the three datasets into RDDs and name them `artistData`, `artistAlias`, and `userArtistData`. View the README, or the files themselves, to see how this data is formated. Some of the files have tab delimeters while some have space delimiters. Make sure that your `userArtistData` RDD contains only the canonical artist IDs.

```
In [25]: def parseUserArtistData(line):
             """
             Parse one line from the file user artist data
             """
             items = line.strip().split(' ')
             return (int(items[0]), int(items[1]), int(items[2]))

         def parseArtistData(line):
             """
             Parse one line from the file artist data
             """
             items = line.strip().split('\t')
             return (int(items[0]), items[1])

         def parseArtistAlias(line):
             """
             Parse one line from the file Artist Alias
             """
             items = line.strip().split('\t')
             return (int(items[0]), int(items[1]))


         def aliasMap(item, aliasDict):
             if item in aliasDict:
                 return aliasDict[item]
             else:
                 return item


         # This fucntion maps the Aliases to actual Artist names and does filteration
         def filterDirtyArtists(userArtistDataDirty, artistAlias):
             #userArtistDataDirtyLocal = userArtistDataDirty.collect()
             artistAliasLocal = artistAlias.collect()
             aliasDict = dict()
             for item in artistAliasLocal:
                 aliasDict[item[0]] = item[1]
             filteredArtists = userArtistDataDirty.map(lambda r: (r[0], aliasMap(r[1], aliasDict), r[2])
             #merged = filteredArtists.map(lambda r: ((r[0], r[1]), r[2])).reduceByKey(add).map(lambda
             #print merged.collect(   )
             #sys.exit(0)
             return filteredArtists


         path1 = "/Users/Himangshu/Desktop/ApacheSpark/Project1/recommender_project/user_artist_data_sma
         path2 = "/Users/Himangshu/Desktop/ApacheSpark/Project1/recommender_project/artist_data_small.t
         path3 = "/Users/Himangshu/Desktop/ApacheSpark/Project1/recommender_project/artist_alias_small.

         artistData = sc.textFile(path2).map(parseArtistData)
```

```
artistAlias = sc.textFile(path3).map(parseArtistAlias)
userArtistDataDirty = sc.textFile(path1).map(parseUserArtistData)
userArtistData = filterDirtyArtists(userArtistDataDirty, artistAlias)
```

1.5 Data Exploration

In the blank below, write some code that with find the users' total play counts. Find the three users with the highest number of total play counts (sum of all counters) and print the user ID, the total play count, and the mean play count (average number of times a user played an artist). Your output should look as follows:

```
User 1059637 has a total play count of 674412 and a mean play count of 1878.
User 2064012 has a total play count of 548427 and a mean play count of 9455.
User 2069337 has a total play count of 393515 and a mean play count of 1519.
```

```
In [26]: #counts code
         userArtistForCount = userArtistData.map(lambda r: (r[0], r[2]))# (uid, count)
         countResults = userArtistForCount.reduceByKey(add)

         artistCounts = userArtistForCount.countByKey()

         countResultsReversed = countResults.map(lambda r: (r[1], r[0])).sortByKey(False)
         topThree = countResultsReversed.take(3)

         for item in topThree:
             user = item[1]
             count = item[0]
             mean = item[0]/artistCounts[user]
             print 'User ' + str(user) + ' has a total play count of ' + str(count) + ' and a mean play
```

```
User 1059637 has a total play count of 674412 and a mean play count of 1878.
User 2064012 has a total play count of 548427 and a mean play count of 9455.
User 2069337 has a total play count of 393515 and a mean play count of 1519.
```

Splitting Data for Testing Use the randomSplit function to divide the data (`userArtistData`) into: * A training set, `trainData`, that will be used to train the model. This set should constitute 40% of the data. * A validation set, `validationData`, used to perform parameter tuning. This set should constitute 40% of the data. * A test set, `testData`, used for a final evaluation of the model. This set should constitute 20% of the data.

Use a random seed value of 13. Since these datasets will be repeatedly used you will probably want to persist them in memory using the cache function.

In addition, print out the first 3 elements of each set as well as their sizes; if you created these sets correctly, your output should look as follows:

```
[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000113, 5)]
[(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000112, 423)]
[(1059637, 1000094, 1), (1059637, 1000130, 19129), (1059637, 1000139, 4)]
19817
19633
10031
```

```
In [27]: # Split train Test validation
         trainData, validationData, testData = userArtistData.randomSplit([2, 2, 1], seed = 13)

         trainData.cache()
         validationData.cache()
```

```
testData.cache()

print trainData.take(3)
print validationData.take(3)
print testData.take(3)

print trainData.count()
print validationData.count()
print testData.count()
```

```
[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000113, 5)]
[(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000112, 423)]
[(1059637, 1000094, 1), (1059637, 1000130, 19129), (1059637, 1000139, 4)]
19817
19633
10031
```

1.6 The Recommender Model

For this project, we will train the model with implicit feedback. You can read more information about this from the collaborative filtering page: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html. The functionn you will be using has a few tunable parameters that will affect how the model is built. Therefore, to get the best model, we will do a small parameter sweep and choose the model that performs the best on the validation set

Therefore, we must first devise a way to evaluate models. Once we have a method for evaluation, we can run a parameter sweep, evaluate each combination of parameters on the validation data, and choose the optimal set of parameters. The parameters then can be used to make predictions on the test data.

1.6.1 Model Evaluation

Although there may be several ways to evaluate a model, we will use a simple method here. Suppose we have a model and some dataset of *true* artist plays for a set of users. The model can be used to predict the top X artist recommendations for a user and compared the artists that were actually played by the user (here, X will be the number of artists in the dataset of *true* artist plays). Then, the fraction of overlap between the top X predictions of the model and the X real artists that the user played can be calculated. This process can be repeated for all users and an average value can be returned.

For example, suppose a model predicted [1,2,4,8] as the top X=4 artists for a user. Suppose, that user actually listened to the artists [1,3,7,8]. Then, for this user, the model would have a score of 2/4=0.5. To get the overall score, this would be performed for all users, with the average returned.

NOTE: when using the model to predict the top-X artists for a user, do not include the artists listed with that user in the training data.

Name your function `modelEval` and have it take a model (the output of ALS.trainImplicit) and a dataset as input. For parameter tuning, the dataset parameter should be set to the validation data (`validationData`). After parameter tuning, the model can be evaluated on the test data (`testData`).

```
In [29]: def modelEval(model, data):
             allArtists = set(userArtistData.map(lambda r: r[1]).collect())

             #print len(allArtists)
             trainingUserArtistPairs = trainData.map(lambda x: (x[0] , x[1]))\
                 .groupByKey().mapValues(list).collect()
             trainUserArtistDict = dict()
             for item in trainingUserArtistPairs:
                 trainUserArtistDict[item[0]] = item[1]
```

```
    dataUserArtistPairs = data.map(lambda x: (x[0] , x[1])).groupByKey().mapValues(list)
    validateDataLocal = dataUserArtistPairs.collect()
    totalPrecision = 0.0
    for item in validateDataLocal:
        originalSet = set(item[1])
        x_t = len(originalSet)
        #print x_t
        #print len(item[1])
        #print "**********"

        currUser = item[0]
        currUserOrigList = set(trainUserArtistDict[currUser])
        candidates = sc.parallelize(allArtists - currUserOrigList)
        #candidates = sc.parallelize([x for x in allArtists if x not in currUserOrigList])
        predictions = model.predictAll(candidates.map(lambda x: (currUser, x))).collect()
        recommendations = sorted(predictions, key=lambda x: x[2], reverse=True)[:x_t]



        predSet = set(sc.parallelize(recommendations).map(lambda r: r.product).collect())
        #print predSet
        commonCount = len(originalSet & predSet)
        currentUserPrecision = float(commonCount) / float(x_t)
        #print currentUserPrecision
        #print "***************"
        totalPrecision += currentUserPrecision
        #sys.exit(0)
    avgPrecision = totalPrecision/len(validateDataLocal)
    #print "Total users :: " + str(len(validateDataLocal))
    return avgPrecision
```

1.6.2 Model Construction

Now we can build the best model possibly using the validation set of data and the `modelEval` function. Although, there are a few parameters we could optimize, for the sake of time, we will just try a few different values for the rank parameter (leave everything else at its default value, **except make `seed` 345**). Loop through the values [5, 10, 15] and figure out which one produces the highest scored based on your model evaluation function.

Note: this procedure may take several minutes to run.

For each rank value, print out the output of the `modelEval` function for that model. Your output should look as follows:

```
The model score for rank 2 is 0.090431
The model score for rank 10 is 0.095294
The model score for rank 20 is 0.090248
```

```
In [30]: ranks = [2, 10, 20]
         #ranks = [5, 10, 15]
         for r in ranks:
             model = ALS.trainImplicit(trainData, rank=r, seed=345)
             rank = modelEval(model, validationData)
             print "The model score for rank " + str(r) + " is " + str(rank)
```

```
The model score for rank 2 is 0.0908903753405
```

```
The model score for rank 10 is 0.0953126426486
The model score for rank 20 is 0.0899507169752
```

Now, using the bestModel, we will check the results over the test data. Your result should be ~0.0507.

```
In [31]: bestModel = ALS.trainImplicit(trainData, rank=10, seed=345)
         rank = modelEval(bestModel, testData)
         print "The Test eval = " + str(rank)
```

```
The Test eval = 0.0512222883501
```

1.7 Trying Some Artist Recommendations

Using the best model above, predict the top 5 artists for user `1059637` using the recommendProducts function. Map the results (integer IDs) into the real artist name using `artistAlias`. Print the results. The output should look as follows:

```
Artist 0: Brand New
Artist 1: Taking Back Sunday
Artist 2: Evanescence
Artist 3: Elliott Smith
Artist 4: blink-182
```

```
In [32]: # making personalised recommendations
         tempRes = bestModel.recommendProducts(1059637, 5)

         artistDataLocal = artistData.collect()
         artistDict = dict()
         for item in artistDataLocal:
             artistDict[item[0]] = item[1]
         count = 0
         for item in tempRes:
             print "Atrist " + str(count) + ": " + str(artistDict[item.product])
             count += 1
```

```
Atrist 0: Brand New
Atrist 1: Taking Back Sunday
Atrist 2: Evanescence
Atrist 3: Elliott Smith
Atrist 4: blink-182
```

```
In [ ]:
```