CS5304 Assignment 2 - Large Scale Recommendation System

Group Members

Kuo-wei Tseng (kt535)
Chong-yee Gan (fc297)

Instructor

Giri Iyengar

Table of Contents

Section 1: ALS recommendation system on MovieLens

1.1 MovieLens 10 Million Dataset

The 10 Million MovieLens Dataset was first downloaded from grouplens.org. For this experiment, the ratings.dat file was used to create the recommendation system, whilst the movies.dat file was used to display the recommended movies as strings, based on their Movie IDs. The structure of the files shown in Table 1.1.

	ratings.dat	**movies.dat**
Heading	UserID::MovieID::Rating::Timestamp	MovieID::Title::Genres
Number of samples/rows	10,000,054	10,681
Encoding	UTF-8	UTF-8

Table 1.1 Summary Statistics of Dataset

Each UserID and MovieID were unique to each individual user and movie respectively. The IDs were consistent between both data files, meaning that a MovieID of value n in ratings.dat represented the same MovieLens movie in movies.dat.

1.1.1 Model Evaluation

The ratings.dat file was first parsed as an RDD using pyspark. The RDD was then sorted by timestamp using the sortBy function, before being partitioned into 60-20-20 train-validate-test sets. This was done to prevent overfitting of the model - a model should be trained with the most recent user ratings to predict past user ratings in the testing set, and not vice versa.

The data sets were converted into pyspark's Rating structure - Rating(user=int i, product=int j, rating=float k) - before being trained using pyspark's ALS package. In order to choose the best parameters for the ALS model, different latent factors and regulation parameters were used to train the model, and their corresponding root mean square errors were recorded:

```
No. of latent factors: 10        Regulation Parameter: 0.01
RMSE = 0.994771720874


No. of latent factors: 10        Regulation Parameter: 0.1
RMSE = 0.832598364153


No. of latent factors: 10        Regulation Parameter: 1.0
RMSE = 1.97907550365


No. of latent factors: 10        Regulation Parameter: 10.0
RMSE = 13.644303884


No. of latent factors: 20        Regulation Parameter: 0.01
RMSE = 1.06994780361


No. of latent factors: 20        Regulation Parameter: 0.1
RMSE = 0.834236910709


No. of latent factors: 20        Regulation Parameter: 1.0
RMSE = 1.97907550525


No. of latent factors: 20        Regulation Parameter: 10.0
RMSE = 13.644303884


No. of latent factors: 30        Regulation Parameter: 0.01
RMSE = 1.1039384923
No. of latent factors: 30        Regulation Parameter: 0.1
RMSE = 0.834441417826


No. of latent factors: 30        Regulation Parameter: 1.0
RMSE = 1.97907550497


No. of latent factors: 30        Regulation Parameter: 10.0
RMSE = 13.644303884


No. of latent factors: 40        Regulation Parameter: 0.01
RMSE = 1.13017770616


No. of latent factors: 40        Regulation Parameter: 0.1
RMSE = 0.834586002726


No. of latent factors: 40        Regulation Parameter: 1.0
RMSE = 1.97907550405


No. of latent factors: 40        Regulation Parameter: 10.0
RMSE = 13.644303884


No. of latent factors: 50        Regulation Parameter: 0.01
RMSE = 1.13999888188


No. of latent factors: 50        Regulation Parameter: 0.1
RMSE = 0.834550962195
```

```
No. of latent factors: 50        Regulation Parameter: 1.0
RMSE = 1.97907550339


No. of latent factors: 50        Regulation Parameter: 10.0
RMSE = 13.644303884
```

Figure 1.1 RMSE of ALS Model on Validation Set with Different Latent Factors and Regulation Parameters

Based on the results in Figure 1.1, the best model - with latent factor of 10 and regulation parameter of 0.1 - was used to test the accuracy of our model on the testing set. The results were shown below:

```
No. of latent factors: 10        Regulation Parameter: 0.1
RMSE = 0.844157036741
```

Figure 1.2 RMSE of Best Performing ALS Model on Testing Set

1.1.2 Recommendation System

Using the ALS Model generated in Section 1.1.1, a movie recommendation system was built using pysparks's recommendProducts function. However, as the recommended movies should not be ones that have already been rated/watched by a user, recommended products were first filtered using a dictionary of movies rated by respective users. The results were shown below:

```
Rating(user=62510, product=42783, rating=5.068078596544477)
Rating(user=62510, product=33264, rating=4.841301731740355)
Rating(user=62510, product=32657, rating=4.788054686800457)
Rating(user=62510, product=4454, rating=4.722393769511833)
Rating(user=62510, product=61742, rating=4.721271023162554)
```

Figure 1.3 Top 5 Recommended Movies

The final recommended movieIDs were then converted into movie titles using the movies.dat file:

```
Shadows of Forgotten Ancestors (1964)
Satan's Tango (Sátántangó) (1994)
Man Who Planted Trees, The (Homme qui plantait des arbres, L') (1987)
More (1998)
Maradona by Kusturica (2008)
```

Figure 1.4 Top 5 Recommended Movies by Title

1.2 MovieLens 22 Million Dataset

The larger 22 Million MovieLens dataset was downloaded and used to build the same recommendation system. This dataset consisted of the same structure as those in Section 1.1, but with 22 million rows instead of 10 million.

1.2.1 Model Evaluation

As with Section 1.1.1, the 22 Million MovieLens dataset was first sorted by time before being split into 60-20-20 partitions. The only change made in this session was to remove the header of the RDD using a filter function.

The model was trained using the training set created from the 22 Million Dataset with the same "best-performing" model parameters - latent factors of 10, regulation parameter of 0.1. The results were shown below:

```
No. of latent factors: 10        Regulation Parameter: 0.1
RMSE = 0.738841244577
```

Figure 1.5 RMSE of Best Performing ALS Model on 22 Million MovieLens Dataset's Testing Set

1.2.2 Recommendation System

Similar to Section 1.1.2, a recommendation system was built using parameters of the best-performing ALS model. The recommended movies were then filtered similarly, with results:

```
Rating(user=73725, product=142891, rating=6.0114232865355515)
Rating(user=73725, product=150900, rating=5.855408786545276)
Rating(user=73725, product=144202, rating=5.84263406216658)
Rating(user=73725, product=149268, rating=5.824363774857004)
Rating(user=73725, product=148857, rating=5.824363774857004)
```

Figure 1.6 Top 5 Recommended Movies using 22M MovieLens Dataset

The movieIDs were then converted to titles in the same way:

```
The Legend of Paul and Paula (1973)
Top Gear: The Worst Car In the History of the World (2012)
Catch That Girl (2002)
The Pied Piper (1942)
"Christmas
```

Figure 1.7 Top 5 Recommended Movies by Title using 22M MovieLens Dataset

Section 2: Preprocessing MovieLens Dataset

2.1 Bias Removal

As equations in reference paper Collaborative Filtering Recommender Systems, we compute the user bias and the item bias to do the feature normalization to make our ALS model more robust.

$$b_{user} = \frac{1}{|I_u|}\sum_{i \in I_u}(r_{u,i} - \mu) \qquad b_{item} = \frac{1}{|U_i|}\sum_{u \in U_i}(r_{u,i} - b_u - \mu)$$

```
#compute Global mean
score_mean = train.map(lambda data: data[2]).mean()

#compute user bias
user_bias = train.map(lambda data: (data[0], data[2] -
score_mean)).groupByKey().map(lambda data: (data[0],
sum(data[1])/len(data[1]))).collectAsMap()

#compute item bias
movie_bias = train.map(lambda data: (data[1], data[2] - score_mean
-  user_bias[data[0]])).groupByKey().map(lambda data: (data[0],
sum(data[1])/len(data[1]))).collectAsMap()

#remove user bias
train_rm_user = train.map(lambda rating: Rating(rating.user,
rating.product, rating.rating - user_bias[rating.user]))

#remove both user and item bias
train_rm_movie = train.map(lambda rating: Rating(rating.user,
rating.product, rating.rating - user_bias[rating.user] -
movie_bias[rating.product]))
```

2.2 Dimensionality Reduction

We tried two methods in this section, SVD and K-means clustering. However, we don't have time to do further experiment on that, but we do attach the test code in our submission.

2.2.1 Singular value decomposition

Singular value decomposition is commonly used in machine learning in dimension reduction. By using SVD, we can decompose User-Item matrix into U, Σ, V where U is more user-aspect eigen-vector, V is item-aspect eigen-vector and Σ is related to the corresponding eigen-value. Therefore we can reduce the user space by projecting it on first half column of U and reduce the item space by projecting it on first half row of V. Because the User-Item space are "huge" but "sparse", we can use scipy.sparse function to do the SVD faster and more efficient.
For example we have User-Item matrix X which is a M(user)xN(item) matrix, we will have U (MxK), Σ (KxK), V (KxN) as svd results. Then, we can retrieve X_reduce = $U[:M, \ :M/2]^T$ X $V[:N/2, \ :N]^T$.

```
from scipy.sparse.linalg import svds as svd
from scipy.sparse import csr_matrix
def buildMatrix(data, user_dict, movie_dict, num_user, num_movie):
      #matrix = csr_matrix((num_user, num_movie), dtype = np.float)
      #matrix = np.array([[0.0] * num_movie for _ in xrange(num_user)])
      ij = []
      rating_list = []
      for user, movie, rating, timestamp in data:
            rating_list.append(float(rating))
            ij.append((user_dict[user],movie_dict[movie]) )
      matrix = csr_matrix((rating_list, np.transpose(ij)), (num_user,
num_movie) )
      return matrix

def dim_reduction(X, w, l, u = None, v_T = None):
      if not u and not v_T:
            u, s, v_T = svd(X)
      X_reduce = ((u[:, :w].transpose()).dot(X)).dot(v_T[:l,
:].transpose())
      return X_reduce, u, v_T
```

2.2.2 K-Means clustering

In this method, we do K-means clustering on user space first where K=½ User, and then use the results to do K-means on item space, where K = ½ item. Assuming User-Item Matrix is M(User) x N(Item) matrix

```
def buildMatrix(data, user_dict, movie_dict, num_user, num_movie):

      matrix = np.array([[0.0] * num_movie for _ in xrange(num_user)])
      for user, movie, rating, timestamp in data:
            matrix[user_dict[user], movie_dict[movie]] = float(rating)

      return matrix
```

2.2.2.1 Reducing User Space

We use row as feature and fit a KMeans model where K = ½ M to reduce user space. Then we have new category representing a group of users with new rating (centroids value).

```
print "kmeans user"
train_matrix = buildMatrix(train, user_dict, movie_dict, num_user,
num_movie)
user_clf = kmeans(n_clusters = w, n_jobs = 4)
user_clf.fit(train_matrix)
new_rate = []
for centroid in user_clf.cluster_centers_:
      new_rate.append(centroid[1])
new_rate = np.array(new_rate)
```

2.2.2.2 Reducing Item Space

Using NewUser-Item matrix from 2.3.1, we transpose the matrix and then use row as feature to fit in KMeans model where K = ½ N to reduce item space. Then we map new item tags with its transformed value too. Now we have matrix with the dimension of ½ M x½ N.

```
print "kmeans item"
item_clf = kmeans(n_clusters = l, n_jobs = 4)
item_clf.fit(new_rate.transpose())
final_rate = []
```

```
for centroid in item_clf.cluster_centers_:
      final_rate.append(centroid[1])
train_reduce = np.array(final_rate).transpose()
print "save train"
with open(DATA_FOLDER + "train.dat", "wb") as f:
      for x in xrange(train_reduce.shape[0]):
            for y in xrange(train_reduce.shape[1]):
                  f.write(str(x) + "::" + str(y) + "::" +
str(train_reduce[x,y]) + "\n")
```

2.3 Model Evaluation

```
Root Mean Squared Error = 0.782490357984
```

This result use the same dataset as Fig.1.2, where the RMSE is 0.844. The recommendation error is improved by removing bias.

2.4 Recommendation System

```
Shadows of Forgotten Ancestors (1964)
Satan's Tango (Sátántangó) (1994)
More (1998)
Sun Shines Bright, The (1953)
Man Who Planted Trees, The (Homme qui plantait des arbres, L') (1987)
```

This one should be compared to Fig. 1.4. Although the order shuffled, 4 out of 5 movies are the same. Because the RMSE is lower, this one should have higher chance to give a better recommendation.

Section 3: ALS Recommendation on Million Song Dataset

The Million Song Dataset (MSD) was first downloaded. The structure of the train_triplets.txt was shown in Table 3.1.

Heading	User Song Play Count
Number of samples/rows	48,373,586

No. of Unique Users	1,019,318
No. of Unique Songs	384,546

Table 3.1 Summary Statistics of Dataset

The users and songs were hashed as string variables. Contrary to Section 1 and Section 2, Play Count represented a type of implicit rating.

3.1 Model Evaluation

The train_triplets.txt file was first parsed as an RDD using pyspark. The data sets were then converted into pyspark's Rating structure - Rating(user=int i, product=int j, rating=float k) - before being trained using pyspark's ALS.trainImplicit function. However, as the userID and songID were hashed string variables (instead of integers), dictionaries were built to convert each unique IDs into dummy integer values:

```
# Creating User Dictionary
userDict = rdd.map(lambda x: (x[0], x[1])).reduceByKey(lambda a,b :
1).collectAsMap()
userCount = 0
for key in userDict.keys():
    userDict[key] = userCount
    userCount+=1

# Creating Song Dictionary
songDict = rdd.map(lambda x: (x[1], x[0])).reduceByKey(lambda a,b :
1).collectAsMap()
songCount = 0
for key in songDict.keys():
    songDict[key] = songCount
    songCount+=1

# Preprocessing data to structure - Rating(user =int i, product=int j,
rating=float k)
train = rdd.zipWithIndex().filter(lambda x: x[-1] < 1000000*0.6)
train = train.map(lambda x: Rating(userDict[x[0][0]], songDict[x[0][1]],
x[0][2]))
```

Instead of using the full 48 million rows, we had used the first 1 million rows instead. This approach was employed as our local storage was insufficient to process all 48 million rows. This meant that the size of our partitions were: training set = 600,000, validation set = 200,000 and testing set = 200,000.

The model was trained using model.trainImplict(), which assigned ratings score based on a combination of binary preference and confidence values. In this case the play count directly reflected the confidence value in a user's rating towards a song. In order to choose the best parameters for the ALS model, different latent factors and regulation parameters were used to train the model, and their corresponding root mean square errors were recorded:

```
No. of latent factors: 10        Regulation Parameter: 0.01
RMSE = 1.66987305012


No. of latent factors: 10        Regulation Parameter: 0.1
RMSE = 1.66818674869


No. of latent factors: 10        Regulation Parameter: 1.0
RMSE = 1.67226755025


No. of latent factors: 10        Regulation Parameter: 10.0
RMSE = 1.67226890756


No. of latent factors: 20        Regulation Parameter: 0.01
RMSE = 1.66668902432


No. of latent factors: 20        Regulation Parameter: 0.1
RMSE = 1.65695627574


No. of latent factors: 20        Regulation Parameter: 1.0
RMSE = 1.67226599272


No. of latent factors: 20        Regulation Parameter: 10.0
RMSE = 1.67226890756
```

```
No. of latent factors: 30        Regulation Parameter: 0.01
RMSE = 1.64735651008


No. of latent factors: 30        Regulation Parameter: 0.1
RMSE = 1.62372194883


No. of latent factors: 30        Regulation Parameter: 1.0
RMSE = 1.67226278263


No. of latent factors: 30        Regulation Parameter: 10.0
RMSE = 1.67226890756


No. of latent factors: 40        Regulation Parameter: 0.01
RMSE = 1.63378871304


No. of latent factors: 40        Regulation Parameter: 0.1
RMSE = 1.61281897848


No. of latent factors: 40        Regulation Parameter: 1.0
RMSE = 1.67226284536


No. of latent factors: 40        Regulation Parameter: 10.0
RMSE = 1.67226890756

No. of latent factors: 50        Regulation Parameter: 0.01
RMSE = 1.61849200291


No. of latent factors: 50        Regulation Parameter: 0.1
RMSE = 1.61387232278


No. of latent factors: 50        Regulation Parameter: 1.0
RMSE = 1.67226067147


No. of latent factors: 50        Regulation Parameter: 10.0
RMSE = 1.67226890756
```

Figure 3.1 RMSE of ALS Model on MSD Validation Set with Different Latent Factors and Regulation Parameters

3.2 Recommendation System

Using the ALS Model generated in Section 3.1, a song recommendation system was built using pysparks's recommendProducts function. Similar to Section 1.1.2 and Sectopm 1.2.2, the recommended songs should not be ones that have already been played by a user. Hence, recommended songs were first filtered using a dictionary of songs played by each respective users. The results were shown below:

```
['SOYIJIL12A6701F1C1']
['SOMYECL12A6701D9C8']
['SOJSTYO12A8C13F200']
['SOSPXWA12AB0181875']
['SOERYLG12A6701F07F']
['SOAYTRA12A8C136D0E']
['SOXDQPZ12A8C13F4FC']
['SOJJKTR12A6701F083']
['SOUMOMJ12A6701DFDC']
['SOOABBO12A6701DFDA']
['SODBMRI12A8151AF45']
['SOCZTMT12AF72A078E']
['SOOGZYY12A6701D9CB']
['SOKLVUI12A6701BF1B']
['SOENRRU12A6701BF1A']
['SOGCDYR12AC961854A']
['SOKUTUM12A6701D9CD']
['SOYDOZE12A6701FC22']
['SODJKMC12A8C137EC0']
['SOHGBHN12A6701F082']
['SOOQPIK12A6701F1C5']
['SOBNTFK12A6701F1CF']
['SODRJZO12AC4684FF6']
['SONCOJJ12A6701FC24']
['SOQLVIT12A8C137EA2']
['SOVGLTY12AF72A39CD']
['SOSRERB12A8C139735']
['SOJHVSF12A6701F084']
['SOKHHXJ12AF72A5325']
['SOHTSKK12A6701F07C']
['SOKMXEQ12A6D4F6AA8']
['SOISXVJ12A6701F1CD']
['SOBZCUC12A58A7D9AD']
['SOIUITF12A58A7D86C']
['SOAJNYK12AF729F33B']
['SOLLBAK12A6D4F6AA7']
['SOQPQWL12A58A7B964']
['SOBLIPF12AF729F53E']
['SOPBTDA12A58A7B7C3']
['SOVPAJA12A58A77B15']
```

Figure 3.3 Top Recommended Songs

However, these hashed songs IDs were not converted to actual titles, as we were unable to secure the API Key for pyechonest.