MovieLens Recommender System

Mohammed Minhaas B S †
Computer Engineering
New York University
New York City, NY, USA
mb7979@nyu.edu

Prithvi S Naidu
Computer Engineering
New York University
New York City, NY, USA
psn5377@nyu.edu

Prajwal Suresh Naidu
Computer Engineering
New York University
New York City, NY, USA
pn2140@nyu.edu

INTRODUCTION

With the advent in technology and the rise in web services such as Netflix, Amazon, Youtube, Disney etc recommendation systems have begun to play an important role in our lives. Recommender systems aim at suggesting similar items to users from their past search interests and ratings. There are many ways in which a recommender system can be developed such as collaborative filtering, content based filtering, demographic based , knowledge based and hybrid recommender systems. We have implemented a recommender system based on collaborative filtering using the movieLens dataset. The ALS Model is used to create the RANK matrix.

DATASET

For the recommender system we use the MovieLens dataset which consists of unique users and unique movies. The dataset has four csv files namely links.csv, movies.csv, ratings.csv and tags.csv which contain numeric and alphanumeric data. Rows in the dataset show the interaction between the user and the movie. Using this data we build a popularity model and a collaborative model.

The spliting of the data was performed by means of stratified sampling by using the pyspark sampleBy() function. This function takes three parameters namely, i) a column that defines a strata ii) fracions – a sampling fraction for each stratum iii) a seed value which is random. This function returns a stratified sample without replacement based on the fraction given to each stratum.

```
ratings = ratings.join(movies, "movieId", "inner").select("userId", "movieId", "rating", "genres")
fractions_init = ratings.select("userId").distinct().withColumn("fraction", lit(0.8)).rdd.collectAsMap()
seed = 1231
train_init = ratings.stat.sampleBy("userId", fractions_init, seed)
test = ratings.subtract(train_init)
fractions_fin = train_init.select("userId").distinct().withColumn("fraction", lit(0.8)).rdd.collectAsMap()
train = train_init.stat.sampleBy("userId", fractions_fin, seed)
val = train_init.subtract(train)
```

To implement the UMAP extension movies.csv and ratings.csv were joined on movieId.

DATA PREPROCESSING

The HDFS cluster was used for efficient processing of the large dataset. After splitting of the csv files they were converted to parquet format and sorted based on userId before being processed on the cluster.

MODEL IMPLEMENTATION

Our model was implemented using the spark framework which has several predefined recommender functions. Collaborative filtering technique in recommender systems uses a small set of latent factors to predict missing entries in the utility matrix. The ALS or Alternating Least Squares algorithm was used to fit our model. The ALS algorithm factorizes the utility matrix into two matrices 1)user matrix and 2)item matrix. The latent factors are uncovered by the algorithm which explains user to item ratings and the algorithm tries to minimize the least squares between predicted and actual ratings.

The baseline model implementation was based on the popularity model. The popularity model works on the principle of popularity. The system recommends a movie which is popular or in trend based on the user rating. However a key observation from the above implementation is that the recommended movie is not personalized for a particular user.

PARAMETERS

Collaborative filtering using spark.ml aims to predict missing values by describing a set of latent factors based on the user and items association matrix. Implementing ALS in spark.ml requires the following parameters :

1. rank - the number of independent rows and columns of user and item matrices, or the number of latent factors in the model (defaults to 10). We did choose a higher rank value as it was time computationally heavy to run. However we understood that higher the rank value better is the accuracy.

2. numBlocks defines the number of blocks the users and items will be partitioned into in order to parallelize computation (defaults to 10).

3. maxIter - the maximum number of iterations to run (defaults to 10). We have set the maxIter value to 5.

4. regParam- the regularization parameter in ALS (defaults to 1.0). We have set the regParam value to 0.1.

5. implicitPrefs- TRUE for implicit feedback Collaborative filtering (our case)

6. alpha- applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations (defaults to 1.0)

RANKING METRICS

The ranking metric we used is MAP (Mean Average Precision). This metric gives us the mean of average precision scores and tells us how a single sorted prediction compares with our ground truth. The mAP score is calculated by taking the mean AP over all classes and/or overall IoU thresholds,

HYPERPARAMETER TUNING

We have run our base model on the small and large dataset. To increase the performance of the ALS model we decided to tune some of the hyperparameters. We experimented with various ranges and values of rank, maxIter and regularization parameters. We ran several models having different values of these hyperparameters on our subsamples,loosely basing our code on the code pprovided here and the parameters that gave us the max MAP score. The metric scores were always greater for higher rank models, but with increase in rank, the runtimes also increased. We got our best results for the subsample datasets with rank= 250, regParam= 0.1, maxIter= 10. So, we ran our model on the entire movielens dataset using these values for our hyperparameters. All results for our larger datasets are available in the results section .

EXTENSION 1 – Lenskit

LensKit is an open-source set of tools for building recommender systems. We used the biased matrix factorization trained with alternating least squares. This is a prediction-oriented algorithm suitable for explicit feedback data, using the ALS approach to compute P and Q to minimize the regularized squared reconstruction error of the ratings matrix. We also used the item-item nearest-neighbor collaborative filtering with ratings. This implementation is based on the description of item-based Collaborative Filtering.

For evalution purposes the NDCG values were computed for each of the above mentioned algorithms. In our lenskit implementation we have used KNN algorithm to compute the ItemItem similarity and compared it to the ALS results computed from the biased matrix factorization. The evaluation results are available in the results section

EXTENSION 2 – UMAP

UMAP is an algorithm for dimension reduction based on manifold learning techniques and ideas from topological data analysis. It provides a very general framework for approaching manifold learning and dimension reduction, but can also provide specific concrete realizations. For implementation purposes of UMAP the standard scalar library from sklearn.preprocessing was used on a dataframe which had id, genre and itemFactor from the ALS model. PyArrow optimiation was used for conversion of spark RDD to Pandas dataframe, such as the above standard scalr method from sklearn could be implemented. The result generated was passed to np.vstack to vertically stack the features of the dataframe and this was passed to the UMAP.fit function to generate the data for the UMAP visualization. The UMAP plots generated on itemfactor and userfactor is available in the results section.

RESULTS

i) Popularity model on large_validation data output

```
+------+--------------------+
|userId|collect_list(movieId)|
+------+--------------------+
|     1| [592, 3793, 780, ...|
|     2| [46970, 333, 1228...|
|     3| [2018, 2105, 1263...|
|     4| [759, 2683, 4226,...|
|     5| [36, 589, 344, 34...|
|     6| [455, 1049, 708, ...|
|     7| [3793, 589, 5989,...|
|     8| [539, 318, 590, 4...|
|     9| [5378, 2012, 2023...|
|    10| [54286, 64969, 19...|
|    11| [1608, 1704, 356,...|
|    12| [1876, 2581, 2694...|
|    13| [3793, 1198, 3753...|
|    14| [590, 593, 351, 1...|
|    15| [79132, 260, 4995...|
|    16| [79132, 1214, 119...|
|    17| [2959, 2194, 593,...|
|    18| [36, 79132, 318, ...|
|    19| [380, 19, 455, 53...|
|    20| [6333, 3793, 4226...|
+------+--------------------+
only showing top 20 rows
```

ii) Popularity model on large_training data output

avg(rating)	movieId	count
4.545454545454546	1235	11
4.5	1245	10
4.433333333333334	720	15
4.409090909090909	162	11
4.402597402597403	1221	77
4.4	7156	10
4.394594594594595	318	185
4.388888888888889	922	18
4.38	1204	25
4.375	306	12
4.366666666666666	246	15
4.363636363636363	3836	11
4.35	176	10
4.35	1217	10
4.346153846153846	3468	13
4.346153846153846	951	13
4.333333333333333	898	15
4.318181818181818	1209	11
4.3161764705882355	912	68
4.307017543859649	904	57

only showing top 20 rows

iii) Popularity model for small data set

```
The evaluation metrics are :
MAP : 0.0022187831743455187
MAP@100 : 0.0022717636549604344
NDCG@10 : 0.013184839275035077
MAPRecall@1 : 0.0007791340844148057
MAPRecall@5 : 0.00344770396828737
MAPRecall@100 : 0.018735340498202326
```

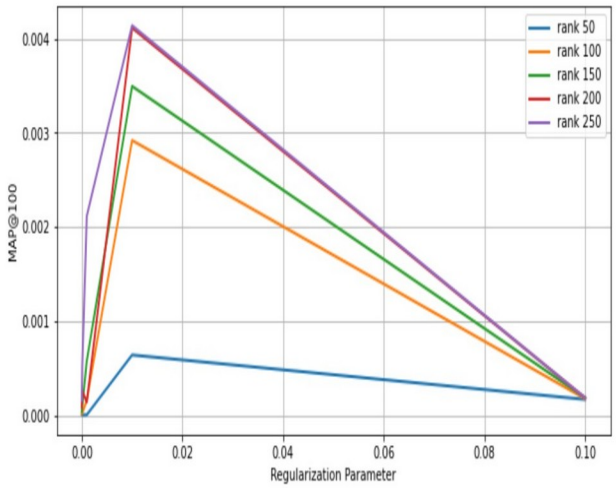
iv) Predictions of ALS for large data set

userId	movieId	rating	timestamp	prediction
53271	148	4.0	828462479	2.961468
175332	148	4.0	1027003224	2.9144454
115125	148	2.0	839178196	2.4205523
115912	148	3.0	837602927	2.8614
107574	148	1.0	1013115063	2.9103389
216371	148	2.0	843990753	2.282261
67319	148	2.5	1120729036	2.5252223
204845	148	3.0	859547410	3.1017025
12020	148	2.0	835601960	2.4031582
67910	148	3.5	1151344374	2.837632
74196	148	2.0	833173771	2.9567418
10059	148	2.0	838308516	3.0243287
8697	148	2.0	835094487	2.6787274
232608	148	3.0	833165538	3.4541924
45531	148	3.0	946918370	2.8021808
8350	148	4.0	1062945923	2.692477
182536	148	2.0	834483114	2.8176963
91374	148	3.0	837773116	2.3697858
8264	148	1.0	837520195	1.9592471
82425	148	3.0	860111242	3.0085444

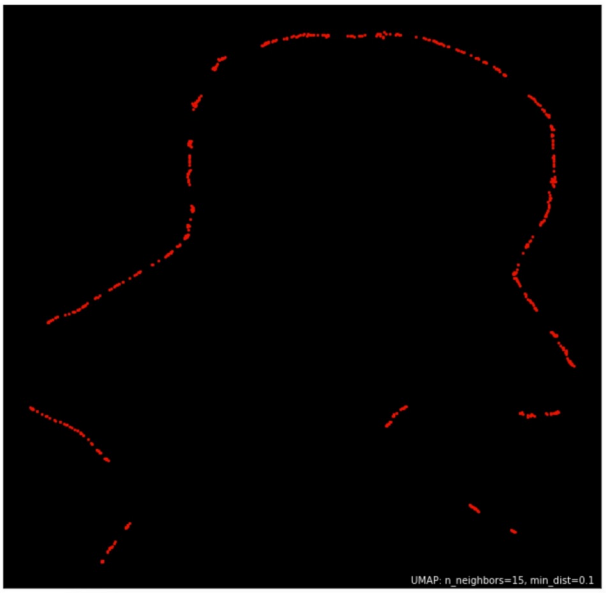
v) Parameters after Hyper Parameter Tuning

```
the MAP@100 is 0.0001887275696532505
the MAP is 0.00018866923713971927
regularization parameter 0.1
For maxIter 10, rank 250,
```

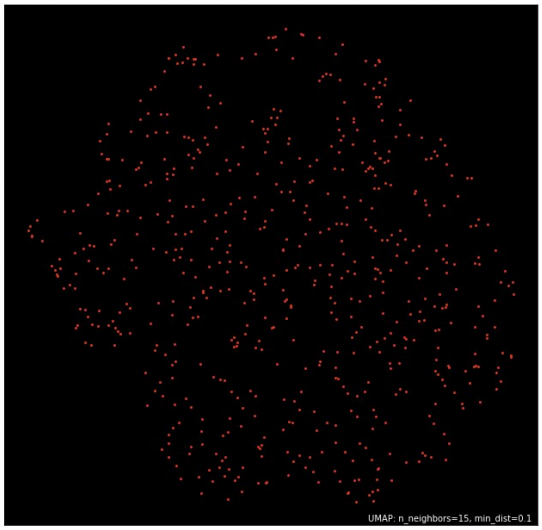
vi) Hyper Parameter tuning plot



vii) userFactors plot for UMAP



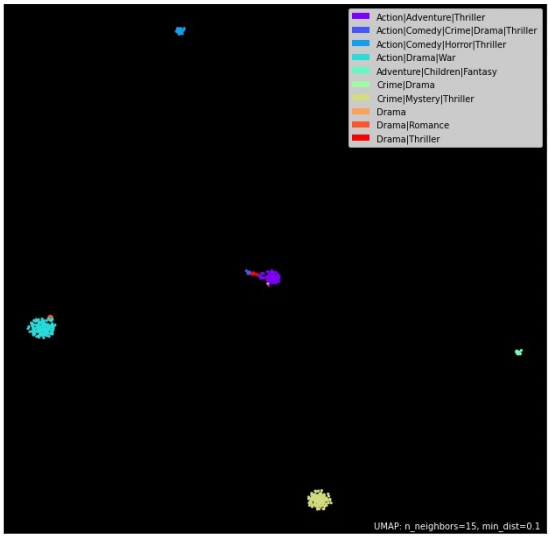
ix) UMAP for small-dataset (latent_users) 610 data points



viii) UMAP for Small dataset(latent_items) 400 data points

```
umap.plot.points(mapper, labels=df.genres[samples], theme='fire')
#plt.savefig("figures/items.png")
```

```
<AxesSubplot:>
```



x) Extension 1 – Lenskit

		nrecs	ndcg
Algorithm	**user**		
ItemItem	**2**	500	0.043685
	12	500	0.000000
	14	500	0.028729
	23	500	0.025802
	24	500	0.037843
	28	500	0.083357
	29	500	0.054352
	31	500	0.000000
	33	500	0.055952
	45	500	0.050593

Figure - NDCG values ItemItem score for each user for first 500 nrecs

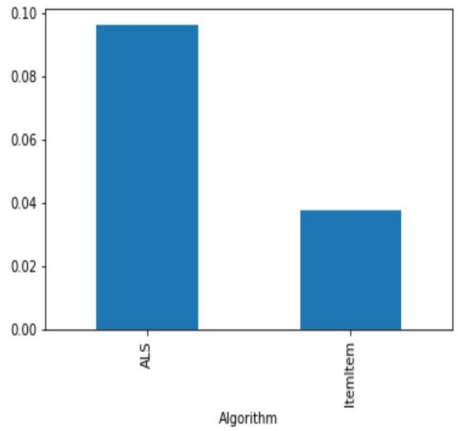
xi) NDCG mean value for ALS and ItemItem

```
Algorithm
ALS          0.096304
ItemItem     0.037733
Name: ndcg, dtype: float64
```

xii) Evaluation of lenskit for ndcg mean value - ALS and ItemItem

```
results.groupby('Algorithm').ndcg.mean().plot.bar()
```

```
<AxesSubplot:xlabel='Algorithm'>
```



CONTRIBUTIONS

ALS Model - Minhaas, Prithvi, Prajwal

Popularity Model – Prithvi, Prajwal
Extension1 - Minhaas, Prajwal

Extension2 - Prithvi , Minhaas

Report - Prajwal, Prithvi , Minhaas

GITHUB REPOSITORY

https://github.com/nyu-big-data/final-project-group_69

REFERENCES

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872

[2] https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html

[3]https://queirozf.com/entries/evaluation-metrics-for-ranking-problems-introduction-and-examplesmap-mean-average-precision

[4]https://analyticsindiamag.com/how-to-build-recommender systems-using-lenskit/

[5] https://spark.apache.org/docs/latest/api/python/