Recommender System using Collaborative Filtering

1. **Contributors:** Prashant Garmella (pg1910@nyu.edu)
Twishikana Bhattacharjee (tb2517@nyu.edu)

2. Overview:

Recommender Systems have become an indelible part of the digital world today. Starting from Netflix, Spotify to Amazon, every website uses a recommender system to help make itself more in-tune and personalized to individual users. With the digitizing of the world and a whole set of services going online, recommender systems create an opportunity to suggest items to users that align with their choices. In the course of the past few decades, recommender systems haves become entwined with the internet in such a way that it is practically impossible to surf through the internet without stumbling upon one.
The Recommender Systems in today's world can broadly be classified into two major categories:

- Content based filtering systems that examine properties of the items recommended.
- Collaborative filtering systems that recommend items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users.

The challenges that every recommender system encounters is how to use implicit feedback to extract substantial information and how to handle the huge incoming data. Recommender systems are critical in industries where the efficiency of the site depends on the recommendation accuracy. Their accuracy alone can help them stand out from their competitors and also bring in more users on-board. Content providing platforms like Netflix, YouTube and Spotify have flourished in their domains because of their accurate recommendations and on-point recommender systems. For example: the accuracy of recommendation on Netflix is way higher than that on Amazon Prime Video and so as user, we find Netflix more in-tune with our choices. It has often happened that we have watched a movie which we would not have watched otherwise because it popped on the Netflix recommendations and ended up liking it. In our project we deal with implicit feedbacks and huge data sets to provide book recommendations.

3. Data Processing:

We worked on the Goodreads dataset which had the following csv files:

- User ID mapping (hdfs:/user/bm106/pub/goodreads/user_id_map.csv)
- Book ID mapping (hdfs:/user/bm106/pub/goodreads/book_id_map.csv)
- User interactions (hdfs:/user/bm106/pub/goodreads/goodreads_interactions.csv)

As per the instructions, we were to split the goodreads_interactions.csv dataset which contains 228,648,342 interactions between various user_ids and book_ids in 60%, 20% and 20% for training, validation and testing respectively. Before splitting the data, we filtered the dataset to include user ids that had more than 10 interactions atleast. This was done to base the recommender system on user ids that have significant interactions with the system. It helped us to remove the user ids that have probably just read 1 or 2 books and have not communicated with the system enough. We also removed interactions for the rating was 0. This sliced the dataset down significantly and we now had a smaller dataset to work with (around 104551549).

Data Downsampling:
Even though removing users with less interactions helped bring down the size of the dataset but it was a significantly huge data size to work with (around 104551549 interactions). So, we tested our model on

- 1 percent of the data (around 1045043)

- 10 percent of the data (around 10409421)

We converted the queried dataframes into parquet files as for large sized data, operations perform better on parquet files. In the testsplit file, we record the 1% data from the dataset. In the 10p_datasplit file we record 10% data from the dataset. We tried testing our model for the entire dataset but due to resource shortage could not successfully do that. We encountered a broken pipeline after 6 hours into the running of the code.

Data splitting:
We implemented our 60, 20, 20 random split on every downsampled and filtered dataset. The 20% that had been assigned to testing and validation respectively, was then sorted in ascending order and 50% of this data was then added back to training dataset. We added a new column(row_number) using the Window() to help index our data so that we can make sure that we have atleast half the interaction data of every user in training dataset. Before moving ahead, we drop the newly added row_number column so that the dataset goes back to what we started with and to avoid column mismatch with the already existing training data. We added the even rows from both the test and validation dataset to the training dataset. That left the odd numbered rows in the testing and validation datasets. This was done to ensure that the training dataset has every user id. So effectively, we added back 10% data from testing and validation to training dataset. This meant the training set now had 80% of the data but also had 100% of the user ids. The final training, testing and validation dataset was now 80%, 10% and 10% respectively. As we split our 1% downsampled data into training, test and validation, we saved the training in
hdfs:/user/pg1910/pub/goodreads/training_sample_1p.parquet, test in
hdfs:/user/pg1910/pub/goodreads/testing_sample_1p.parquet and validation in
hdfs:/user/pg1910/pub/goodreads/validation_sample_1p.parquet.
Further, when we split our 10% downsampled data into training, test and validation, we saved the training in hdfs:/user/tb2517/pub/goodreads/training_sample_10p.parquet, test in
hdfs:/user/tb2517/pub/goodreads/testing_sample_10p.parquet and validation in
hdfs:/user/tb2517/pub/goodreads/validation_sample_10p.parquet.

4. Model and Experiments:

In this project, we have tried to build a Book Recommender System and implemented it on the Goodreads dataset

Method used: Collaborative Filtering

Algorithm used: ALS or Alternaing Least Squares

ALS attempts to estimate the ratings matrix R as the product of two lower-rank matrices, X and Y, i.e. $X*Yt=R$. Typically these approximations are called 'factor' matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix hence the name alternating least squares.

Our Implemention:

Objective of the project: *Your recommendation model should use Spark's alternating least squares (ALS) method to learn latent factor representations for users and items.*

We implemented the ALS algorithm using the pyspark.ml.recommendation module.

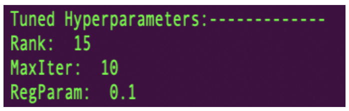
This model has some hyper-parameters that help optimize performance. We have used

- the rank (dimension) of the latent factors} using .getRank() from the pyspark.ml.recommendation module
- the regularization parameter lambda} using .getRegParam() from the pyspark.ml.recommendation module
- the iteration using .getMaxIter() from the pyspark.ml.recommendation module

The **Baseline** for this project was modelled using the ALS in pyspark.ml.recommendation module. As we called the ALS model, we used the following to set up the model

- userCol="user_id", i.e. we set the user column for the model as user_id attribute from the data
- itemCol="book_id", i.e. we set the item column for the model as book_id attribute from the data
- ratingCol="rating" sets the rating attribute from data to the rating column.
- coldStartStrategy="drop". We set coldStartStrategy to drop as we do not want any NaN values in our evaluation metrics
- nonnegative=True helps us to make sure we filter out any non-negative rating predictions by the model

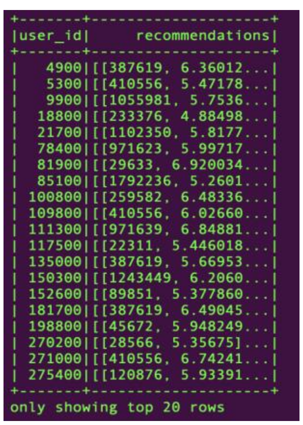
We imported ParamGridBuilder from pyspark.ml.tuning. Using the ParamGridBuilder(), we built a parameter grid (param_grid) such that we could add three options per hyperparameter. We used .addGrid() and passed als.rank and [15,25,35], the 15, 25 and 35 being the ranks options for the ALS model. Similarly, we pass 5, 8 and 10 as options for als.maxIter and 0.08, 0.09 and 0.10 as options for als.regParam. We then call .build() to build the parameter grid. We tried various such triplets of combinations ranging from 5 to 35 on the rank hyper-parameter. We varied the lambda triplet between 0.06 and 0.10. We kept the max iterations triplet as [5,8,10]. After multiple such settings based on various combinations we narrowed down to the following setting:



```
Tuned Hyperparameters:------------
Rank:  15
MaxIter:  10
RegParam:  0.1
```

Figure 1: Tuned Hyper-parameters on our training dataset

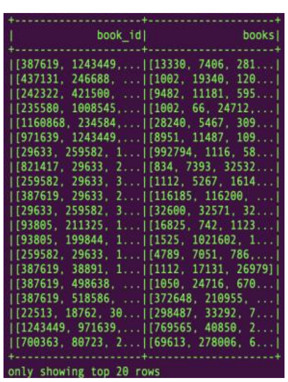
param_grid=ParamGridBuilder().addGrid(als.rank,[15,25,35]).addGrid(als.maxIter,[5,8,10]).addGrid(als.regParam, [0.08,0.09,0.10]).build()}



```
+-------+--------------------+
|user_id|     recommendations|
+-------+--------------------+
|   4900|[[387619, 6.36012...|
|   5300|[[410556, 5.47178...|
|   9900|[[1055981, 5.7536...|
|  18800|[[233376, 4.88498...|
|  21700|[[1102350, 5.8177...|
|  78400|[[971623, 5.99717...|
|  81900|[[29633, 6.920034...|
|  85100|[[1792236, 5.2601...|
| 100800|[[259582, 6.48336...|
| 109800|[[410556, 6.02660...|
| 111300|[[971639, 6.84881...|
| 117500|[[22311, 5.446018...|
| 135000|[[387619, 5.66953...|
| 150300|[[1243449, 6.2060...|
| 152600|[[89851, 5.377860...|
| 181700|[[387619, 6.49045...|
| 198800|[[45672, 5.948249...|
| 270200|[[28566, 5.35675]...|
| 271000|[[410556, 6.74241...|
| 275400|[[120876, 5.93391...|
+-------+--------------------+
only showing top 20 rows
```

Figure 2: Extrapolating the top 500 recommendations for every user_id

We then used the .fit() on the training dataset to fit the data to the model and stored it in a variable *model*. Then, we called a property .*bestModel* to get the best hyper-parameters for the model on the training dataset. In our case, we observed the turned hyper-parameters as Rank = 15, MaxIter = 10 and RegParam = 0.1. This is also specified in Figure 1. Once we found our best model, we then extrapolated the top 500 recommendations for every user (as represented in Figure 2) so that we could provide each user with a customized recommendation list. We then went on to test our model on the validation and test dataset. Along with testing the model on validation and testing data, we use evaluation metrics to assess the performance of the model on testing and validation data. To implement RankingMetrics, we had to collect all the books read by a particular user to help create a 'ground truth' by using *agg(expr("collect_set(book_id) as books"))*. Now, from the recommendation that we obtained for each user, we pick up the 500 recommendations using a select query to pick up *user_id* and *recommendations.book_id*. This would form our 'recommendation' list. We further join the 'ground truth' and 'recommendation' on the *user_id* attribute and generate a new RDD which is represented in Figure 3.



```
+--------------------+--------------------+
|             book_id|               books|
+--------------------+--------------------+
|[387619, 1243449,...|[13330, 7406, 281...|
|[437131, 246688, ...|[1002, 19340, 120...|
|[242322, 421500, ...|[9482, 11181, 595...|
|[235580, 1008545,...|[1002, 66, 24712,...|
|[1160868, 234584,...|[28240, 5467, 309...|
|[971639, 1243449,...|[8951, 11487, 109...|
|[29633, 259582, 1...|[992794, 1116, 58...|
|[821417, 29633, 2...|[834, 7393, 32532...|
|[259582, 29633, 3...|[1112, 5267, 1614...|
|[387619, 29633, 2...|[116185, 116200, ...|
|[29633, 259582, 3...|[32600, 32571, 32...|
|[93805, 211325, 1...|[16825, 742, 1123...|
|[93805, 199844, 1...|[1525, 1021602, 1...|
|[259582, 29633, 1...|[4789, 7051, 786,...|
|[387619, 38891, 1...|[1112, 17131, 26979]|
|[387619, 498638, ...|[1050, 24716, 670...|
|[387619, 518586, ...|[372648, 210955, ...|
|[22513, 18762, 30...|[298487, 33292, 7...|
|[1243449, 971639,...|[769565, 40850, 2...|
|[700363, 80723, 2...|[69613, 278006, 6...|
+--------------------+--------------------+
only showing top 20 rows
```

Figure 3: RDD with the 'ground truth' represented by column books and 'recommendation'

	Test data (1%)	**Test data (10)%**	**Validation data (1%)**	**Validation data (10%)**
RMSE	0.767666191	0.703687541	0.751559592	0.692310754
MAP	0.002136002	0.001954720	0.002723102	0.002572148

We used Regression Metrics and Ranking Metrics to further evaluate our model and analyze its performance. We used the RMSE from Regression Metrics and MAP from Ranking Metrics to evaluate the performance of our model. We observe that the RMSE on both the data is less than 1 which tells us that the squared loss between the predicted and the actual is not very high. This means that our model is actually performing pretty well on the test and validation data. When we observe the MAP, we see around 2% precision match between the ground truth and prediction on both 1% and 10% dataset. This makes some sense intuitively as the prediction dataset would have a very low chance of overlapping with the list of books a user has already read as there is a possible subset of books that the user would have read from the entire set of books and hence getting a reasonably high MAP seems less likely. The ‘on a scale of 1 to 5’ rating structure makes the MAP a little less efficient as a metric as it operates on binary relevancies. So in this case, we would have to forcefully threshold the fine rating to a binary version that would skew the relevancy. It would end up treating 1 and 3 as same (if threshold is set at 3) which would affect the recommendation and therefore affect precision in the process.

5. Extension:

We worked on the single machine implementation extension using LightFM. In LightFM, like in a collaborative filtering model, users and items are represented as latent vectors. But these users and items are entirely defined by functions of latent vectors of the content features that describe each product or user as is observed in Content Based recommenders. For example, if the book ‘Harry Potter and the Order of the Phoenix’ is described by the following features: ‘fantasy’, ‘J K Rowling’, and ‘Harry Potter’, then its latent representation will be given by the sum of these features’ latent representations. In doing so, LightFM unites the advantages of content based and collaborative recommenders. We remodeled our user interactions to fit the LightFM input dataset definition. We converted the Spark dataframes to Pandas dataframe. We then defined a function ‘dataformatting’ which read the Pandas dataframe and converted it to a coo matrix for the training and testing set plus the raw training dataframe for later evaluation of the model.The coo matrix is a suitable input for LightFM. As we implemented the LightFM model, we set no_components=110 and learning_rate=0.027. We used 'warp' loss function. We tried implementing the extension on 1% downsampled data. The model took way less time to fit the training data in comparison to ALS model fitting. We got an AUC score of 0.979900419 on training data and as a high AUC score is equivalent to low rank-inversion probability, this means that the chances of the recommender mistakenly ranking an unattractive item higher than an attractive item is extremely low. On using LightFM, the model seems to fit better to the training dataset and also performs better than our baseline model. When we tried to check the performance on the test and validation data set, we encountered a ValueError('Incorrect number of features in item_features') error. We tried debugging it, but could not effectively get an AUC score on the test and validation data.

6. Contributions:

Prashant: ALS model training; hyper-parameter turning; evaluating performance; LightFM extension.
Twishikana: Data processing; Baseline model training; parameter tuning; evaluating performance; LightFM extension.

7. References

[1] Y. Hu, Y. Koren and C. Volinsky, *Collaborative Filtering for Implicit Feedback Datasets* 2008 Eighth IEEE International Conference on Data Mining, Pisa, 2008, pp. 263-272, doi: 10.1109/ICDM.2008.22.

[2]Baptiste Rocca. Introduction to recommender systems In Towards Data Science. Article published June 2, 2019 {https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada}

[3] Elena Cuoco. Alternating Least Squares (ALS) Spark ML. Article published on December 22, 2016 {https://www.elenacuoco.com/2016/12/22/alternating-least-squares-als-spark-ml/?cn-reloaded=1}

[4] Victor Kohler. ALS Implicit Collaborative Filtering. Article published on August 23, 2017 {https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe}

[5] Maciej Kula. *Metadata Embeddings for User and Item Cold-start Recommendations*. Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th {ACM} Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16 – 20, 2015 CEUR Workshop Proceeding

8. Appendix

Predictions for test dataset:

user_id	book_id	is_read	rating	is_reviewed	prediction
388300	833	1	4	0	3.5499504
335200	833	1	3	0	3.5047996
7800	833	1	5	0	3.4052932
306000	833	1	4	0	3.787898
56900	833	1	3	0	3.5185552
187400	833	1	4	0	3.2379704
351100	833	1	3	0	3.9899566
298200	833	1	4	0	3.871078
39500	833	1	4	0	4.311697
100500	833	1	5	0	4.817074
383200	833	1	5	0	4.6948113
550800	833	1	1	0	1.1561894
272400	833	1	5	0	4.9035773
326100	833	1	5	0	4.412736
50000	833	1	4	0	3.724393
68300	833	1	2	0	2.7946465
249600	833	1	3	0	3.8226752
232100	833	1	3	0	4.1451707
4500	833	1	3	0	4.238667
23400	833	1	3	0	3.4682145

only showing top 20 rows

Predictions for validation dataset:

user_id	book_id	is_read	rating	is_reviewed	prediction
419800	148	1	5	0	4.4174027
159900	148	1	4	1	4.371121
303300	148	1	5	0	3.7146497
275900	148	1	4	0	3.9534147
232200	833	1	4	0	3.8103554
1500	833	1	3	0	3.1449232
713400	833	1	5	0	4.480324
81300	833	1	4	0	4.098479
144000	833	1	3	0	4.136464
609600	833	1	1	0	2.0396054
52500	833	1	5	0	3.9171903
768500	833	1	4	0	3.8429735
394400	833	1	5	0	3.4630318
222000	833	1	1	0	3.4960454
222500	833	1	5	0	4.9831734
302800	833	1	3	0	3.2730904
140600	833	1	4	0	3.7466953
575600	833	1	5	0	3.9776993
230500	833	1	4	0	3.471947
382100	833	1	5	0	3.9052968

only showing top 20 rows