BIA-678: Big Data Technology

Project on

"Building Recommendation Engines for Music"

By Team 02-

Urvashi Chatterjee, Rajiv Mahajan, Devanshi Mehta, Idriss Dem

Table of Contents

Introduction 3

Content-based vs Collaborative filtering 3

Implicit vs Explicit Data 6

Data Pipeline 7

Data Collection 7

Data Cleaning 8

Exploratory Data Analysis 9

Data Modeling 10

Model Results 14

Evaluating Model Results 16

Conclusion 19

Appendix 21

References 23

Introduction

Like the cosmos, the immense amount of information found on the internet and digital media may seem infinite and ever-expanding. Given the precious resource that is time, we need to navigate this ocean of data with the utmost efficiency and accuracy. Whether you're looking for that perfect outfit for a YouTube video, the advent of recommendation systems has significantly helped us find specific personalized, sought-out items from this ocean of information. You have probably seen recommendation systems before, especially on media sites like YouTube and Spotify, where the site uses predetermined algorithms to evaluate and predict the user's preferences and tastes to help them to expand on their playlist. In this report, we strive to elucidate the utility and architecture of music recommendation systems.

Broadly speaking, recommender systems are algorithms aimed at suggesting relevant items to users. You can find such recommendation systems all over the web: on websites like amazon or applications like Spotify and Netflix. Recommendation systems are incredibly popular critical, particularly in retail, showing customers news related to their purchase and search history, in hopes they will buy them- hence increasing revenue. This includes sections like "people who bought this also bought…" and "you may also like…" sections of Macy's.

Content-based vs Collaborative filtering

The operation of these systems is based on two methods: collaborative and content-based methods. Firstly, Collaborative methods are based solely on the past interactions recorded between users and items in order to produce new recommendations. The main idea here is that

these past user-item interactions are sufficient to detect similar users and/or similar items and make predictions based on these estimated proximities.

Collaborative filtering algorithms are divided into two types: memory-based and model-based approaches. Memory-based approaches directly work with values of recorded interactions, assuming no model, and are essentially based on the nearest neighbor's search. The main advantage of collaborative approaches is that they require no information about users or items and, so they can be used in many situations. Moreover, the more users interact with things, the more new recommendations become accurate.

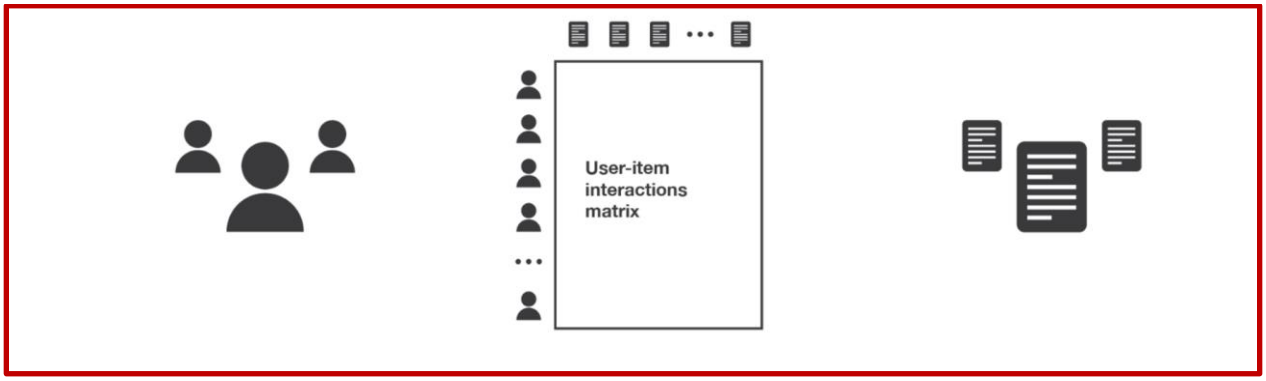


Figure 1: Collaborative filtering

On the other hand, content-based filtering algorithms determine its suggestions based on the user's personal information, such as age, gender, and location. For example, in the case of Netflix, if the program knew that the user was an elderly woman in California, a state with a relatively large Latino population, Netflix would likely recommend Spanish telenovelas and movies like "Yo soy Betty," "la fea" and "Tierra de Reyes." Here, the idea behind content-based methods is to try to build a model based on the available "features" that explain the observed user-item interactions. One advantage of content-based filtering is that content-based methods suffer far less from the cold start problem than collaborative approaches: new users or items can

be described by their characteristics (content), and so relevant suggestions can be made for these further use based solely on their background information. One disadvantage of content-based filtering is that it tends to be biased, assuming that the user follows the stereotypical preferences of his or her demographic. In fact, going back to the Netflix example, there are numerous Floridians who like Bollywood (Indian) movies!

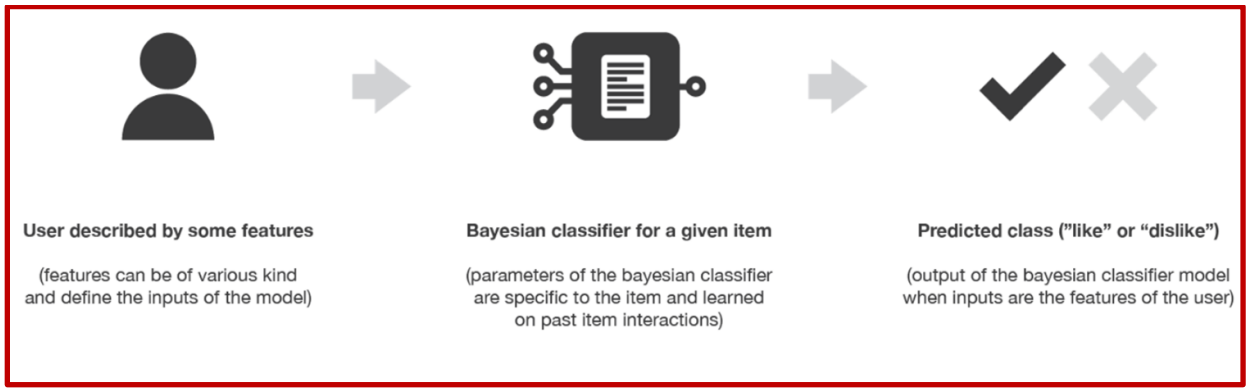


Figure 2: Item-Centered Bayesian Classifier

Technically speaking, the recommendation system can be broken down into three parts: Candidate generation, scoring, and re-ranking. In the first step, the system starts from a potentially colossal corpus and generates a much smaller subset of candidates. The model needs to evaluate queries quickly, given the enormous size of the corpus. A given model may provide multiple candidate generators, each nominating a different subset of candidates. Secondly, another model scores and ranks the candidates in order to select the set of items (on the order of 10) to display to the user. Since this model evaluates a relatively small subset of items, the system can use a more precise model relying on additional queries. Finally, the system must take

into account other constraints for the final ranking. For example, the system removes items that the user explicitly disliked or boosted the score of fresher content. Re-ranking can also help ensure diversity, freshness, and fairness. Going back to the Netflix scenario, this process can be illustrated by the following infographic:

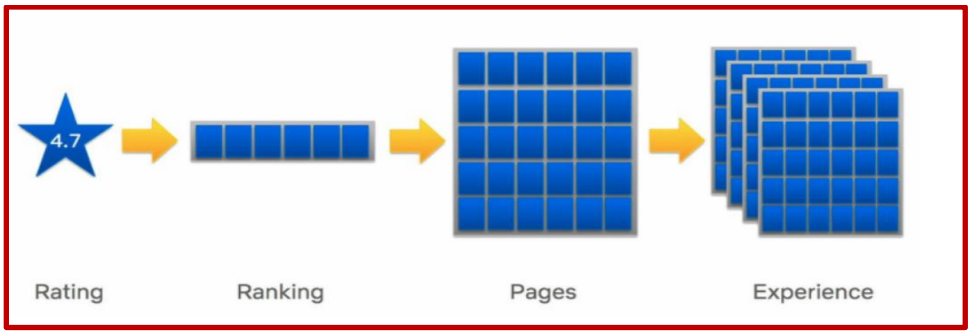


Figure 3: Ranking in Netflix example

Implicit vs Explicit Data

Explicit data is the information where we have user appraisals related with a song/film on a fixed scale. For example, a 1-5 rating on the Netflix dataset. From such a rating, we can decipher how much a user likes/despises a film. Yet, it is difficult to get such information on the grounds that by and large, users don't want to rate each film they see.

Implicit data is the sort of information we are utilizing to make a song recommendation. The information is accumulated from user conduct, with no express appraising related with it. It may very well be how frequently a user played a song or watched a film, how long they have spent pursuing a specific article and so on. The benefit here is we have a ton of such information however, it is normally extremely uproarious and untrustworthy.

At the point when a user rates a film 1 on 5, that implies he did not care for the film. Yet, with the play check of a melody, we ca not make any certain supposition that the user adored the tune or loathed the tune or some place in the middle. Likewise, on the off chance that they do not play a melody does not imply that they do not care for the tune. Thus, we center on what we think about the user's conduct and the certainty we have in whether they like any given thing. For example, we can have higher trust in a melody if the user played it multiple times against a tune that he played one time.

Data Pipeline

Data Collection

Before leaving for any information related excursion, you need to procure information to examine first. Before collecting data, following questions should be kept in mind:

- Which source should I gather the information from?
- What sort of information do I require for the examination that I am going to begin?
- What sort of assortment techniques or channels are accessible to me?

There are numerous sorts of information assortment: reviews, calls, records, structures, clinical investigations, and so forth. One could get it either from an essential source (the information your organization gathers through different methods) or an optional source (government informational indexes or online archives).

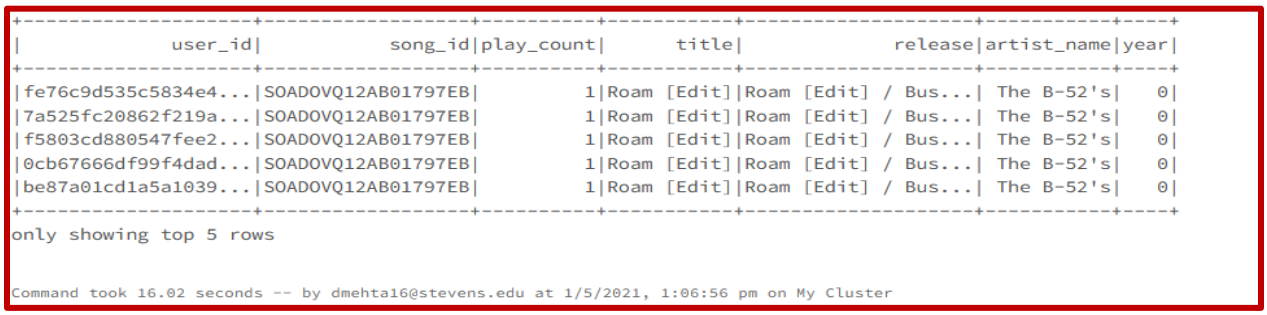
For our project, we used a dataset provided by Million Songs Dataset, which was taken from Kaggle, source https://www.kaggle.com/c/msdchallenge/data, and is also posted on Columbia

University. Source: https://labrosa.ee.columbia.edu/millionsong/. We used two files: song_data.csv and triplets_10000.txt.

Data Cleaning

This is the way towards recognizing and revising (or eliminating) bad or wrong records from a record set, table, or data set and alludes to distinguishing fragmented, mistaken, off base or immaterial pieces of the information and afterward supplanting, adjusting, or erasing the grimy or coarse information.

In this project, since the two datasets, which were used had different information, to make sense of it and for processing it further, we used left join to combine the two datasets. We used the merge function and the key on which join was performed was song_id. After performing the join, there were certain repetitive columns. We dropped those in order to reduce redundancy. The first five rows of dataset after cleaning and manipulating data is as shown below.



user_id	song_id	play_count	title	release	artist_name	year
fe76c9d535c5834e4...	SOADOVQ12AB01797EB	1	Roam [Edit]	Roam [Edit] / Bus...	The B-52's	0
7a525fc20862f219a...	SOADOVQ12AB01797EB	1	Roam [Edit]	Roam [Edit] / Bus...	The B-52's	0
f5803cd880547fee2...	SOADOVQ12AB01797EB	1	Roam [Edit]	Roam [Edit] / Bus...	The B-52's	0
0cb67666df99f4dad...	SOADOVQ12AB01797EB	1	Roam [Edit]	Roam [Edit] / Bus...	The B-52's	0
be87a01cd1a5a1039...	SOADOVQ12AB01797EB	1	Roam [Edit]	Roam [Edit] / Bus...	The B-52's	0

only showing top 5 rows

Command took 16.02 seconds -- by dmehta16@stevens.edu at 1/5/2021, 1:06:56 pm on My Cluster

Figure 4: Data cleaning and Manipulation

Exploratory Data Analysis

Exploratory Data Analysis is utilized by information researchers to break down and research informational indexes, and sum up their fundamental attributes, frequently utilizing information representation strategies. It decides how best to control information sources to find the solutions you need, making it simpler for information researchers to find designs, spot abnormalities, test a speculation, or check presumptions.

With our Million Song Dataset, we started off with basic exploration. Initially, we analyzed how big is the data, i.e. the number of rows in dataset, which was a little above 2 million. Later, we found the number of unique user and songs: 76353 and 10,000 respectively.

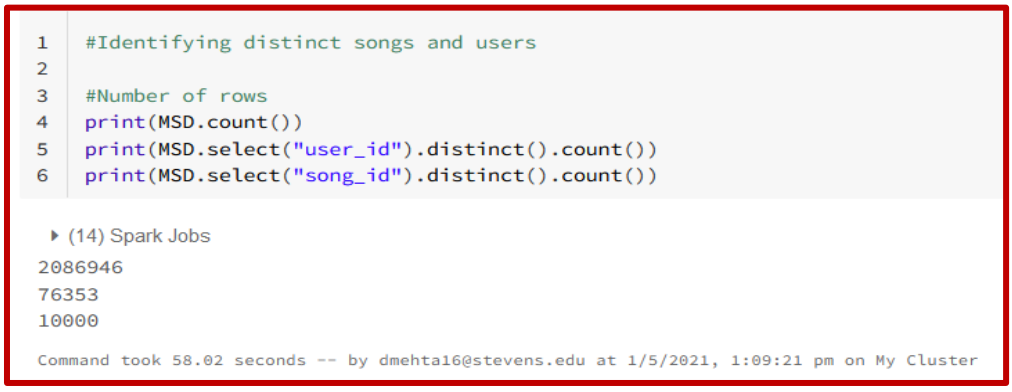


Figure 5: Initial Exploration

Further, we determined the “Top 10 most played songs” and “Top 20 most heard Artists” based on the play counts.

	artist_name	title	number_of_total_play
1	Dwight Yoakam	You're The One	54136
2	Björk	Undo	49253
3	Kings Of Leon	Revelry	41418
4	Barry Tuckwell/Academy of St Martin-in-the-Fields/Sir Neville Marriner	Horn Concerto No. 4 in E flat K495: II. Romance (Andante cantabile)	31153
5	Harmonia	Sehr kosmisch	31036
6	Florence + The Machine	Dog Days Are Over (Radio Edit)	26663
7	Kings Of Leon	Use Somebody	22140
8	OneRepublic	Secrets	22100
9	Five Iron Frenzy	Canada	21019
10	Tub Ring	Invalid	19645

Showing all 10 rows.

Figure 6: Top 10 most played songs

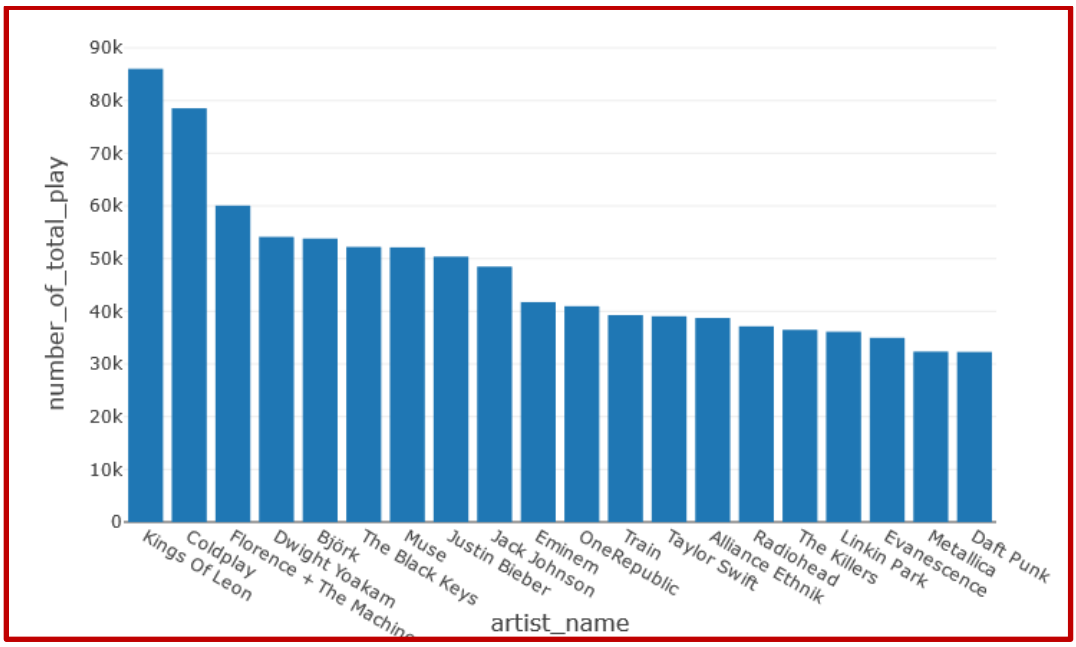


Figure 7: Top 20 most heard artists

Data Modeling

Data modeling is the way toward creating an unmistakable chart of connections between different kinds of data that are to be put away in a data set. One of the objectives of information

demonstration is to make the most proficient technique for putting away data while accommodating total access and revealing.

For building recommendation engine for music, we used the Alternating Least Squares modeling technique. Before starting modeling, we had to fit our data to model. In order to do so, we converted all the unique user ids. Currently, they were alphanumeric or string type. For model, we needed it to be an integer. We did the same thing for song id as well. Post this; we split the data into training and test data, the ratio for which was 70 to 30.

We decided to go forward with ALS because it is memory efficient and easy to implement matrix factorization model. It is very popular for movie recommendation systems and was the method of choice in the event of the Netflix Prize competition. Moreover, the model-based collaborative filtering that is ALS, overcomes shortcomings that other methods encounter such as scalability and data sparseness. Thus, it was very relevant to use this method to build our music recommendation system as it is adapted for large scale collaborative filtering matters.

The ALS is a matrix factorization algorithm that attempts at estimating the ratings matrix R (in our case it will the number of plays) as the product of two matrices, which are U (user matrix) and V (item matrix or "song matrix" for our study), such that R = Ut * V. Generally, these 2 matrices are called "factor" matrices. The ALS formula is as follows:

$$\arg\min_{U,V} \sum_{\{i,j|r_{i,j}\neq 0\}} \left(r_{i,j} - u_i^T v_j\right)^2 + \lambda \left(\sum_i n_{u_i} \|u_i\|^2 + \sum_j n_{v_j} \|v_j\|^2\right)$$

Figure 8: ALS formula

The ratings matrix R approximation is made by solving this quadratic equation similar to the Ordinary Least Squares (OLS). The distance between the actual values in R and the predicted values of R is to be minimized. In order to regularize the data and avoid overfitting, a weighted lambda regularization factor is also incorporated in the formula: it is a L2 regularization factor. In this formula, there are:

(R)i,j=ri,j as the ratings matrix R;

λ = the regularization factor;

nu = the number of items the user i has rated

nv = the number of times the item j has been rated.

The ALS follows an iterative process: during each iteration, one of the factor matrices is held constant while the other is solved via least squares. Then, the newly solved matrix is fixed while the other factor matrix is solved. By iteratively applying these steps, the matrix factorization can be iteratively improved and thus the error is minimized.

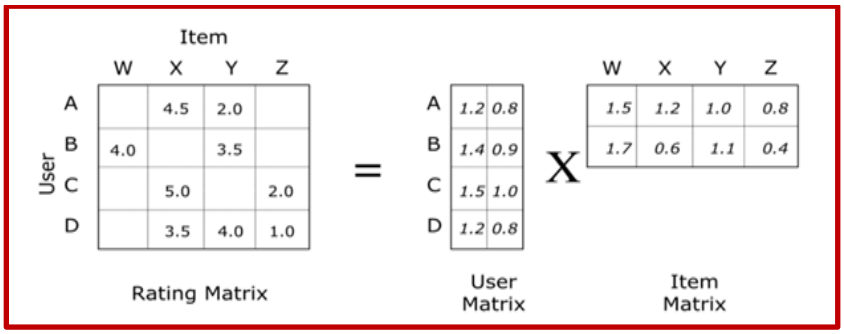


Figure 9: ALS Matrix Factorization

Matrix factorization approximation in the case of movie recommendation systems:

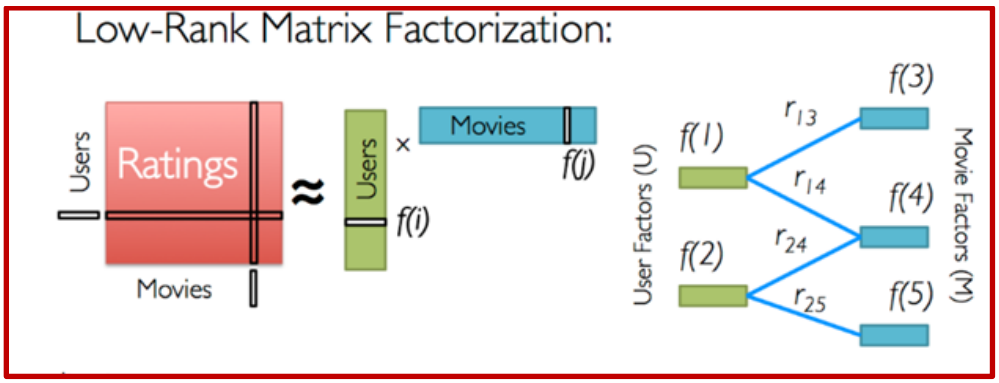


Figure 10: Movie Recommendation Matrix factorization

The training set goal is to help the algorithm train and learn with minimal error. On the other hand, the test set is to apply the trained algorithm on unseen data. At first, the ALS was executed without optimizing its hyper-parameters. Later on, to achieve better predictions, it was decided to tune the most important hyper-parameters of ALS ML: “rank”, “MaxIter”, “RegParam”. For this, a cross validation with 5 folds needed to be implemented: after building a parameter grid and regression evaluator, both were fed into the cross validator including the ALS model as well. The cross validated model was fit to the train set and its parameters were extracted to know which were the best hyper-parameters (“rank”:16, “MaxIter”:11, “RegParam”:0.16). Afterwards, the model was transformed. This was fit to the test data. With the aim to evaluate model performance, it was decided to use the ROEM (rank order error metric). Metrics such as RMSE does not work in our case because it is associated with explicit rating due to its relying on matching predictions back to a true measure of user preference. Instead, ROEM, which is

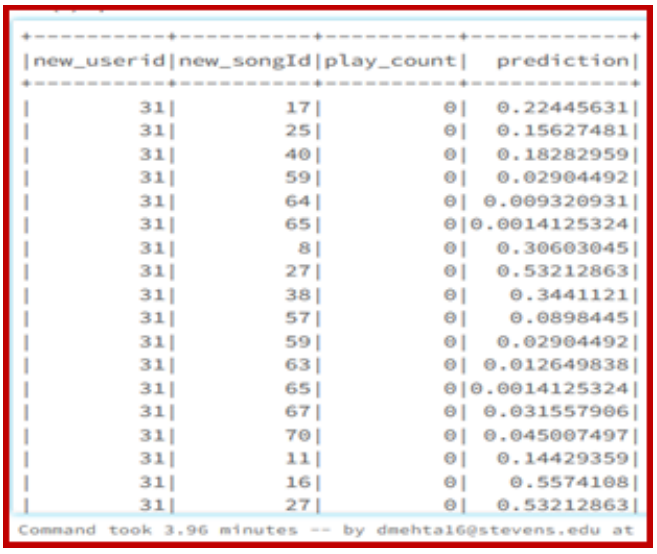
adapted to implicit rating, is utilized as it checks if songs with higher number of plays have higher numbers of predictions. ROEM formula is as shown below:

$$\mathrm{ROEM} = \frac{\sum_{u,i} r^t_{u,i} \mathrm{rank}_{u,i}}{\sum_{u,i} r^t_{u,i}}$$

Figure 11: Rank Order Error Metric Formula

Model Results

After running the model, we got results of predictions on the test data. The image below shows us the predictions for a user with id 31.



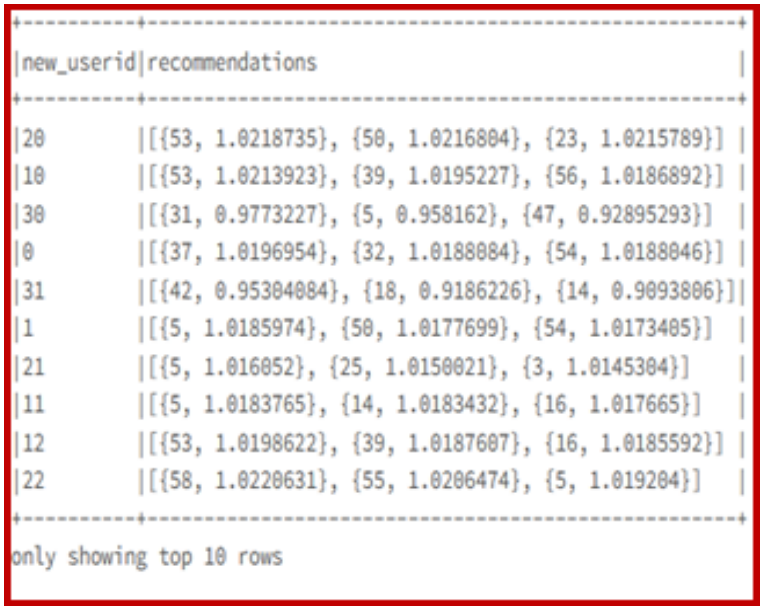
new_userid	new_songId	play_count	prediction
31	17	0	0.22445631
31	25	0	0.15627481
31	40	0	0.18282959
31	59	0	0.02904492
31	64	0	0.009320931
31	65	0	0.0014125324
31	8	0	0.30603045
31	27	0	0.53212863
31	38	0	0.3441121
31	57	0	0.0898445
31	59	0	0.02904492
31	63	0	0.012649838
31	65	0	0.0014125324
31	67	0	0.031557906
31	70	0	0.045007497
31	11	0	0.14429359
31	16	0	0.5574108
31	27	0	0.53212863

Command took 3.96 minutes -- by dmehta16@stevens.edu at

Figure 12: Predictions

This table shows the initial plays count for a song and its prediction. In the case the plays count is 0 and the prediction is close to 0 such as 0.22 for song number 17, it means that this song is predicted not be played. On the other hand, if a song has 2 plays and is predicted with 1.5, it can be asserted that the song is predicted to be played between 2 times. On the other hand, if the play count for a song is 0 and the predicted value is 0.53 (as song #27) or above, then the prediction is wrong as it tends to go towards 1 play instead of predicting 0 plays.

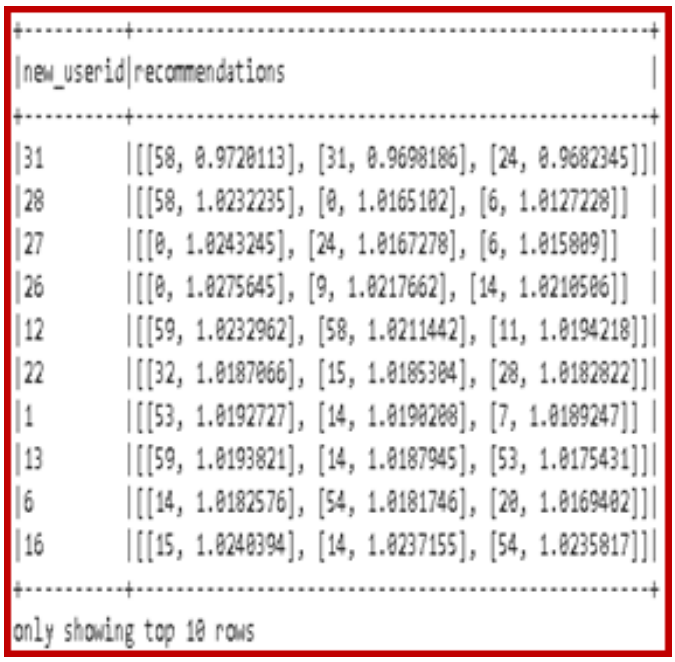
Because the model was run at first in Databricks and later on via Amazon Web Service(AWS), shown below are the top 10 recommendations made on both APIs.



new_userid	recommendations
20	[{53, 1.0218735}, {50, 1.0216804}, {23, 1.0215789}]
10	[{53, 1.0213923}, {39, 1.0195227}, {56, 1.0186892}]
30	[{31, 0.9773227}, {5, 0.958162}, {47, 0.92895293}]
0	[{37, 1.0196954}, {32, 1.0188084}, {54, 1.0188046}]
31	[{42, 0.95304084}, {18, 0.9186226}, {14, 0.9093806}]
1	[{5, 1.0185974}, {50, 1.0177699}, {54, 1.0173405}]
21	[{5, 1.016852}, {25, 1.0150021}, {3, 1.0145304}]
11	[{5, 1.0183765}, {14, 1.0183432}, {16, 1.017665}]
12	[{53, 1.0198622}, {39, 1.0187607}, {16, 1.0185592}]
22	[{58, 1.0220631}, {55, 1.0206474}, {5, 1.019204}]

only showing top 10 rows

Figure 13: Recommendations on Databricks



new_userid	recommendations
31	[[58, 0.9720113], [31, 0.9698186], [24, 0.9682345]]
28	[[58, 1.0232235], [0, 1.0165102], [6, 1.0127228]]
27	[[0, 1.0243245], [24, 1.0167278], [6, 1.015809]]
26	[[0, 1.0275645], [9, 1.0217662], [14, 1.0210586]]
12	[[59, 1.0232962], [58, 1.0211442], [11, 1.0194218]]
22	[[32, 1.0187066], [15, 1.0185304], [28, 1.0182822]]
1	[[53, 1.0192727], [14, 1.0190208], [7, 1.0189247]]
13	[[59, 1.0193821], [14, 1.0187945], [53, 1.0175431]]
6	[[14, 1.0182576], [54, 1.0181746], [20, 1.0169402]]
16	[[15, 1.0240394], [14, 1.0237155], [54, 1.0235817]]

only showing top 10 rows

Figure 14: Recommendations on AWS

For instance, new user #31 in AWS is recommended songs #58, #31 and #24.

Evaluating Model Results

Evaluating results is used to improve association and the executives, arranging, help dynamic, help strategy making, demonstrate where the activity is required, improve observing, show where specialized help and preparing are required, and so on. We used RMSE and Rank Ordered Error Metric as metrics to evaluate our model. The ROEM value came to be 0.5026, which means our model was 50.26% accurate. For every two songs recommended to a player, one song was relevant to the user. Further, we also ran our model on cloud (AWS) and compared the model performance on Databricks and AWS. The results were as follows:

Databricks Runtime:

Scalability	Runtime
40.0%	74
65.0%	73
70%	74

Table 1: Databricks Runtime

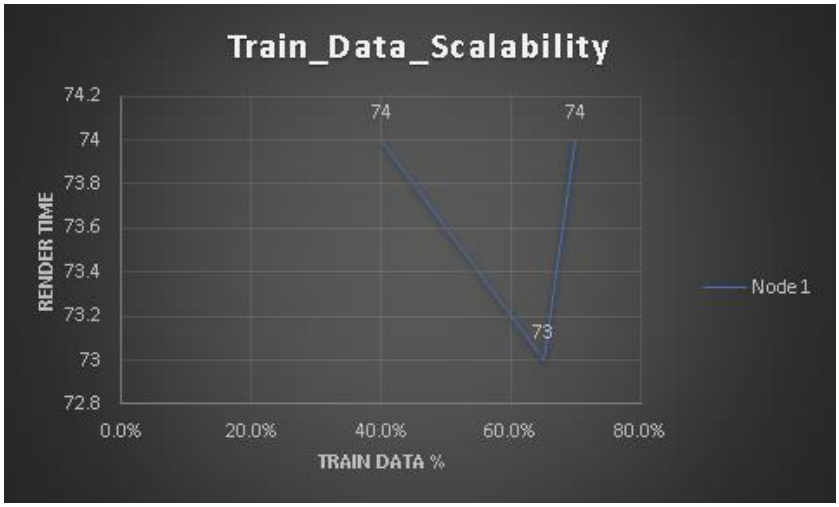


Figure 15: Scalability of training data

AWS Runtime:

Nodes	**40%**	**65%**	**70%**
3	1.33	1.49	2.16
5	1.1	1.39	1.45
7	0.45	1.21	1.35

Table 2: AWS Runtime

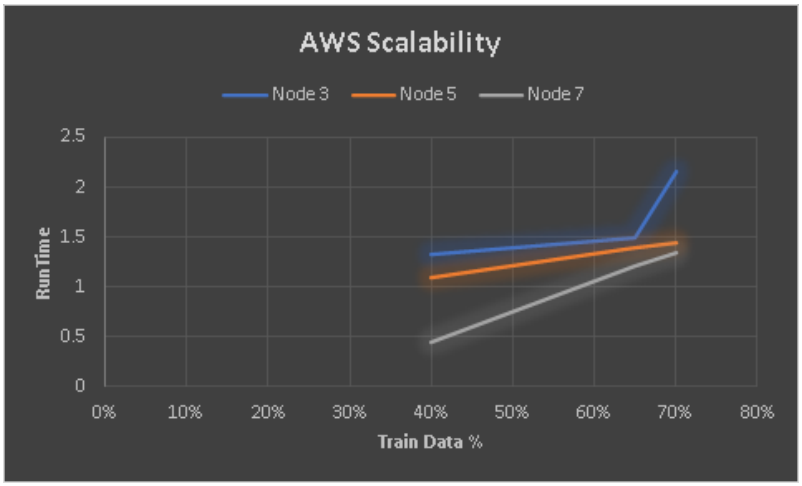


Figure 16: Scalability on different number nodes of AWS

Conclusion

This introductory project on building a music recommendation system went well as new algorithm never seen before was applied as well as a newly learned accuracy metric. Based on the ROEM (rank order error metric), the model achieved an accuracy of around 50.26%. This means that for two songs recommended to a user, 1 is relevant. Furthermore, for a model with a margin of improvement, these results are convincing and solidify the relevancy of the study.

Nonetheless, the RMSE (root mean squared error) was calculated as well, even though not significant enough in the music recommendation case, with a score of 0.98 when the model was run in Databricks and a score of 1.073 when the latter was run in AWS. Due to the closeness in results on the 2 platforms, the most important fact to differentiate is that AWS allocates multiple instances to clusters, to run the entire model, which allows to optimize running time. Indeed, the execution time difference between Databricks and AWS is fulgurant : on average, AWS is 25 times faster than Databricks. This leads to understanding that AWS is more adapted for large scale problems than Databricks, in terms of run time.

The algorithm was trained on approximately 4000 users while later tested on around 1714 users. By allocating 70% of the dataset to training and the remainder of 30% to testing, it was possible to achieve the results discussed before. Nevertheless, with the aim to improve the overall model accuracy (ROEM), several actions could be taken to optimize:

- Training the model on significantly more data (thus, find a larger dataset for instance)

- In terms of programming, establishing a new pipeline via grid search cross validation; removing some irrelevant features in the current model; or use early stopping (because accuracy follows a U shaped curve, stopping the model when it reaches its lowest error before it goes up again, could improve model's accuracy)
- Utilizing other techniques such as ensemble techniques, especially boosting

To conclude, the scope of the study could be oriented towards developing a sentiment analysis algorithm revealing the sentiment for a song or artist for various geographic locations.

Appendix

```
#Building ALS model

##Setting the ALS hyperparameters
from pyspark.ml.recommendation import ALS

model = ALS(userCol = "new_userid", itemCol="new_songId", ratingCol = "play_count", rank=10, maxIter=10, alpha=20,
regParam = 0.05, coldStartStrategy="drop", nonnegative = True, implicitPrefs = True)
```

Code-1

```
# Import the requisite packages
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
```

```
Command took 0.06 seconds -- by idem@stevens.edu at 5/6/2021, 12:46:27 PM on uyf
```

```
Cmd 20
```

```
# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
             .addGrid(model.rank, [10, 50, 100, 150]) \
             .addGrid(model.regParam, [.01, .05, .1, .15]) \
             .build()
```

```
Command took 0.03 seconds -- by idem@stevens.edu at 5/6/2021, 12:46:42 PM on uyf
```

```
Cmd 21
```

```
# Define evaluator as RMSE and print length of evaluator
evaluator = RegressionEvaluator(
            metricName="rmse",
            labelCol="play_count",
            predictionCol="prediction")
print ("Num models to be tested: ", len(param_grid))
```

Code-2

Cmd 26

```
#Fit cross validator to the 'train' dataset
fitted_model = cv.fit(train_data)
#Extract best model from the cv model above
best_model = model.bestModel
```

Code-3

.

Cmd 20

```
fitted_model = model.fit(train_data)
```

▸ (4) Spark Jobs

Command took 1.07 hours -- by idem@stevens.edu at 5/4/2021, 5:44:26 PM on uyfjhb

Cmd 21

```
predictions = fitted_model.transform(test_data)
```

▸ predictions: pyspark.sql.dataframe.DataFrame = [new_userid: integer, new_songId: integer .

Command took 0.14 seconds -- by idem@stevens.edu at 5/4/2021, 7:32:43 PM on uyfjhb

Cmd 22

```
validation_performance = ROEM(predictions)
```

Code-4

References

https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1

https://rpubs.com/sandipan/204499

https://www.kaggle.com/c/msdchallenge/data