

University of Maryland, College Park
Robert H. Smith School of Business

785B Pyspark Project
Book Recommendation System

Instructor: Kunpeng Zhang

Group Members:

Xin Gu

Zixuan Cao

Guanyu Chen

Jia Guo

Ying Wu

Year 2018 May

Table of Contents

1. SUMMARY	2
2. DATA PROCESSING: DATA CLEANING.....	3
3. MODELS	5
3.1 BASELINE PREDICTION MODEL.....	5
3.2 ALS MODEL	7
3.3 RANDOM FOREST MODEL:	13
3.4 GRADIENT BOOST MODEL:	13
4. MODEL SELECTION AND SYSTEM IMPLEMENTATION ..	14
5. FRONT END	16
6. IMPROVEMENT AND LEARNING.....	18
APPENDIX.....	19
REFERENCES.....	24

1. Summary

Our project is going to solve a book recommendation prediction problem, which will build a system to recommend books to registered users or unregistered users in USA. This information will assist book shops or online book retailers to know which book they should recommend to their existed users or future users. In this book recommender system, we have two different ways for users with different status (either registered or unregistered), one way is based on existed users' historical data (ALS, Baseline, RandomForest, Gradient Boost), and another way is based on unregistered users' features, such as age and location, (RandomForest, Gradient Boost). Having cleaned data, we tuned models and chose the best model and parameters to get lowest RMSE on test data set.

In the model selection process, we have three steps to be done:

- 1) Analyze each column of three tables in data set, then figure out how should we clean each.
- 2) Split the data set carefully to make sure the test data set used to calculate RMSE is same for every model.
- 3) Try different models, adjust parameters, compare the RMSE of each model on testing data set, then choose the best model for each sub system (for registered user and unregistered user).

After finding the best models, we can recommend books via Pyspark at the back end and use MySQL link to PHP to exhibit the results at the front end. We use our well-trained model to predict ratings of each book which was not read by the user, then we recommend this user five books with the highest predicted ratings. So, if you are an unregistered user or a newly registered user whose reading history we don't have, we will use your basic information (such as age, location) to recommend books to you based on our database, and if you are an existed user, once you sign in system, we will recommend you five books based on your reading history.

Now the RMSE is still not ideal, if we have more time, we will try more models and tune our model better, because due to the limitation of pc computation power, we didn't try as many combinations of parameters as possible. Furthermore, we can do some text mining based on the title of the book which may help us improve our recommendation system.

2. Data Processing: data cleaning

We obtained our data from Book-Crossing Dataset, issued by Department of Computer Science, University of Freiburg. The original data set was BX-Books.csv, BX-Users.csv, BX-Book-Ratings.csv. In BX-Books table, there was unique 271379 books, and contains information like: ISBN, Book-Title, Book-Author, Year-Of-Publication, Publisher, Image-URL-S, Image-URL-M, Image-URL-L. In BX-Users table, there was 276271 unique users and contains information like: User-ID, Location, Age. In BX-Book-Ratings, there was 1048575 ratings from users on different books. For convenience, in the following paragraphs, “book” table refers to the dataframe loaded from BX-Books.csv, “user” table for BX-Users.csv, and “rating” table for BX-Book-Ratings.csv.

After we examined our data, we processed the data in the following steps:

1. Every element is in double quotation marks, and separated by semicolon. Thus, we loaded the data into data frame with delimiter ‘;’. But we failed to load it correctly because there are semicolons also in book titles or other fields in our dataset. Finally, we combined "" and; to separate the data (i.e. sep = ““;””)
2. After successfully loading the file, there were half quotation mark left with the first and last column in every dataframe. We formatted the column names and data by stripping the left or right quotation mark.
3. Then, we checked null values in each table and found that “Age” in “user” table have 110763 missing values. Besides, there are one or two missing values in “Location”, “Publisher”, “Book-Author”. We dropped all records with missing values.
4. Having a glance of the rating table, we noticed that there are many zeros in the “rating” column. To compute baseline and other models more conveniently, we dropped all records in “rating” table whose rating is 0.
5. By printing out the number of unique users in “user” and “rating” tables and the number of unique books in “book” and “rating” tables, it seems that some books in the “rating” table are not in “book” table. Same problem exists in the users in both “rating” and “user” tables, which will encumber our prediction models. Besides, the data size is too large for efficient operation of our demo system. So we decided to drop records in “rating” table if the book is not in “book” table. Then we sampled

100,000 unique books from resultant “rating” table to further reduce the amount of data. We also dropped some records in “user” table if they didn’t appear in “rating” table for the user problem.

6. Location column in “user” table can be separated into city, state and country, which will be more efficient to our analysis. So, we decided to create three new columns by separate Location column using comma. We are focused on USA users only, and many elements in “city” column are not readable or valid, such as empty string, combined punctuation and letters, street address, etc. Hence, we dropped all records whose city does not start with a letter, either lowercase or upper case.

Finally, we got three tables named after original dataset: BX-Books.csv (43579 records), BX-Users.csv (22165 records), BX-Book-Ratings.csv (139296 records). To have a quick view of cleaned data, please see Appendix.

3. Models

Model selection: we split data into 80% and 20% with same seed as training and testing data, and then we use 80% data to train models. Afterwards we use testing data to compare the RMSE of our models and choose the best.

3.1 Baseline Prediction Model

We want to know whether the prediction of our model is good or not, so we should use a baseline prediction algorithm which provides a set of predictions that we can help us evaluate our models. After we established it, we can know how much better our model is as compared to the naïve baseline. For our dataset, we choose random prediction algorithm as our baseline algorithm. Here's how implement it:

Step 1): algorithm understanding

Training Data								
	B1	B2	B3	B4	B5	B6	B7	B8
U1	●	●		●		●	●	
U2			●		●			●
U3		●				●		
U4	●		●	●			●	
AVG	●	●	●	●	●	●	●	●

Testing Data								
	B1	B2	B3	B4	B5	B6	B7	B8
U5	▲			▲			▲	▲
U6		▲		▲	▲		▲	
U7			▲			▲		▲
U8		▲		▲		▲		
Eros	●	●	●	●	●	●	●	●

For example, we split our dataset into training dataset and testing dataset, and we can get a matrix like two plots above. Then we calculate the average rating for each book in our training dataset as our future predicted rating for this book (sum all blue circles in same column and calculate the average value, green circle) no matter who is the future user. Thus, in this process we are training our baseline prediction model, and our predictions are same for each book, item-based baseline prediction.

Step 2): get the baseline of RMSE

After getting the predictions (average rating, green circles), then we can use these results to predict ratings for users in testing data set. For each column in testing data, we use corresponding average rating from training data minus actual value in the testing data, take a square ((B1 green circle – B1 blue triangle) **2), add all these errors together and take an average to get our baseline RMSE of testing dataset (add all red circles together then take average).

Step 3): calculate our dataset baseline of RMSE

We split our data set to 20% testing and 80% training, use groupby function then calculate the mean of a book. Afterwards, using merge on 'ISBN', we joined the average rating dataframe with testing dataframe, calculate the difference between average rating and actual rating, take a square and add then all together calculating the mean to get our baseline RMSE on our testing data set.

								mean	
	ISBN	avg(rating)		user	ISBN	rating	avg(rating)	ISBN	
0	92660981	7.416667	0	276964	49359484	6	7.000000	25163	1.361111
1	75984640	3.000000	1	276964	94036543	10	7.222222	25205	0.000000
2	87493215	7.000000	2	36163	94036543	5	7.222222	27765	4.000000
3	32434142	7.000000	3	93363	94036543	4	7.222222	33858	2.826446
4	89539798	7.500000	4	276986	75690523	10	7.761905	57440	0.062500

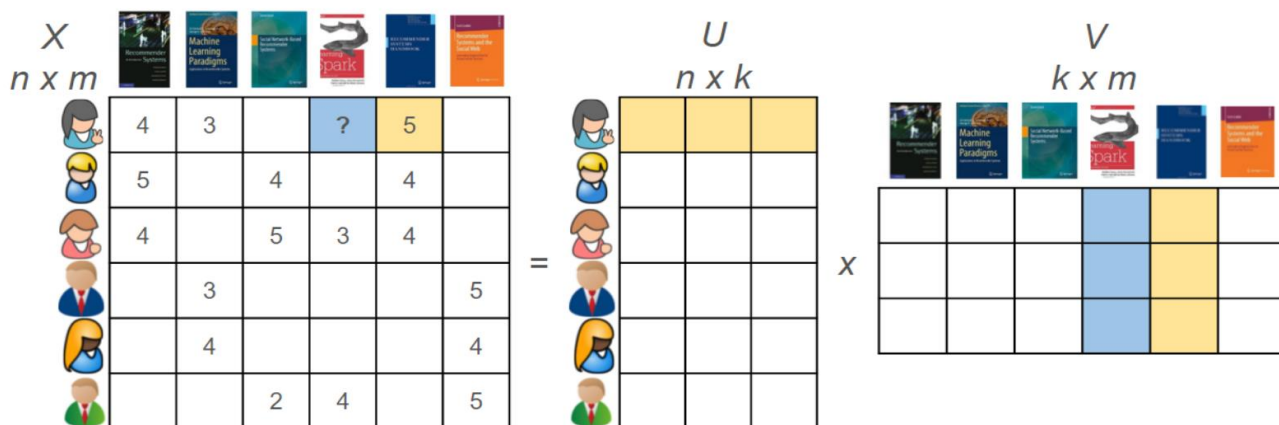
Our final RMSE baseline of testing data set is 4.06, so we will evaluate every model based on this score.

3.2 ALS Model

3.2.1 introduction

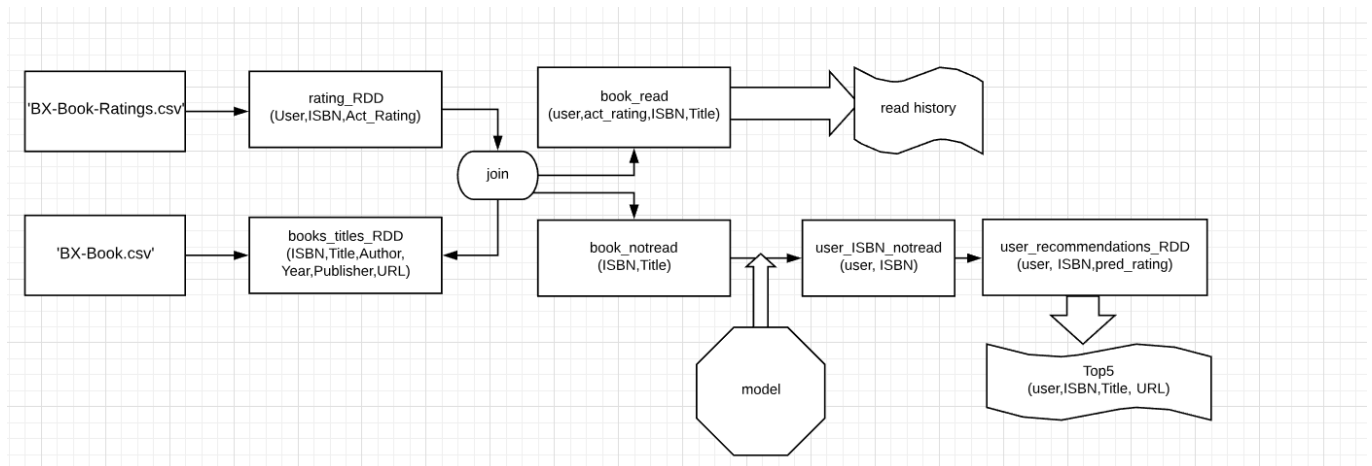
We consider ALS as our first choice model to do recommendation system because the ALS algorithm is a very popular technique used in Recommender System problems, especially when we have implicit datasets (for example clicks, likes etc). Here, we have a very large dataset with three tables with more than one million data. ALS can handle large volumes of data reasonably well we can find many good implementations in various Machine Learning frameworks. And more, spark has nested ALS algorithm in the MLlib component, which makes it convenient for us to build a simple recommendation system.

ALS in Spark MLlib supports the model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. However, unlike user-based and item-based filtering algorithm, which calculate the similarity to predict score and recommend, ALS uses Matrix Factorization to predict users' ratings of books as follow:



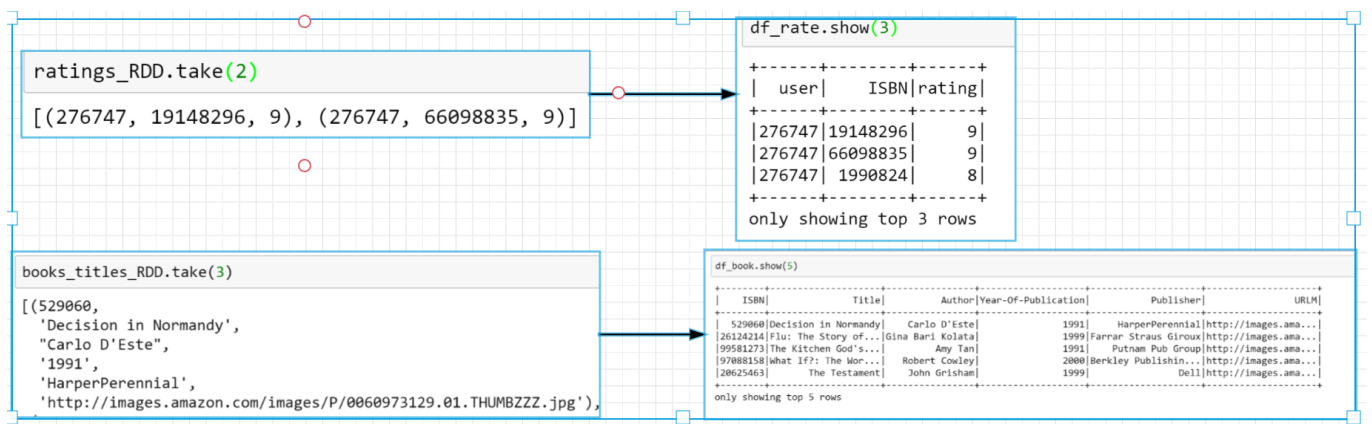
3.2.2 training process:

The brief overview of our training process as shown below :



We use two tables: BX-Books.csv and BX-Book-Ratings.csv.

- 1) firstly, we imported two tables into pyspark as RDDs. However, since the default form in ALS for productid (i.e. ISBN) is integer but here ISBN is string so we need to hash ISBN and take the first 8 bits as new ISBN. We also converted them into two dataframes separately for later use. The result is shown as below:



- 2) build recommendation system based on ALS

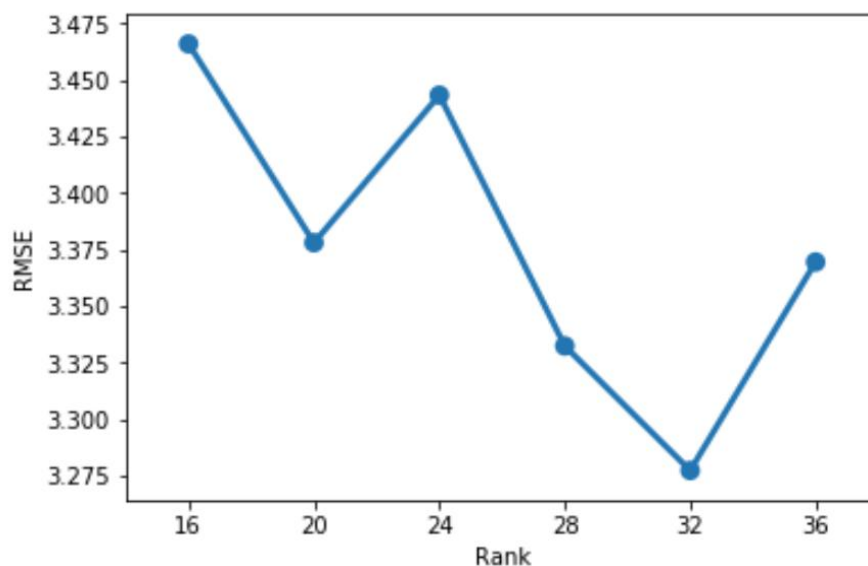
We did several following steps:

First, we split data into testing and training subset at the proportion of 2/8 and put train dataset into `ALS.train()` model and then used model to predict ratings for each (user, ISBN) in the whole

testing dataset (20%). At the meantime, we also wrote a for-loop function nested ALS model to tune our first parameter-rank in a reasonable range. And then we got the lowest RMSE with 3.277 when rank is equal to 32, which has already been lower than the 4.27 RMSE of the baseline with under the circumstance where both two models now calculate RMSE on whole test dataset (20% original dataset). Therefore, we can regard ALS as a good model to more things.

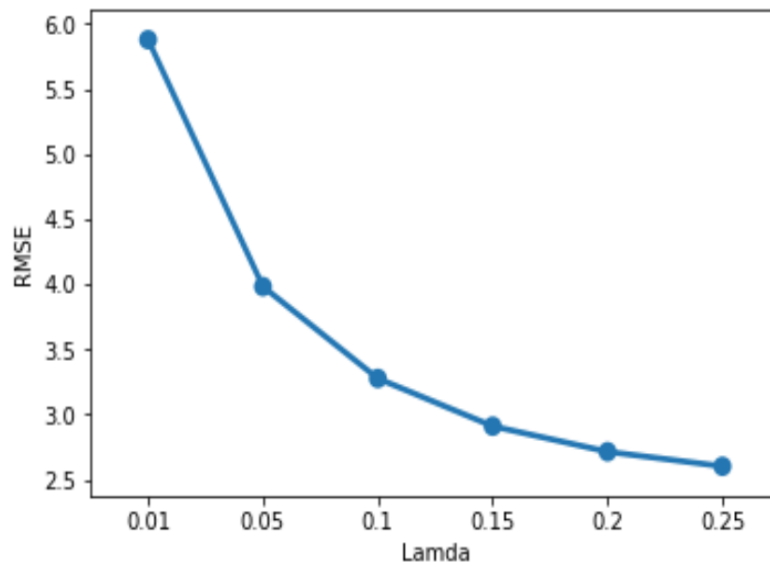
Second, we start to tune main parameters in ALS.train model. There are three main parameters:

1. **rank**: Number of features to use. This is the required parameter with no default. Therefore, that's why we tuned it together with training our original ALS model in the first step. We took [16,20,24,28,32,36] as multiple values, kept iterations = 10 and regularization_parameter = 0.1 and then plot our result as follow:



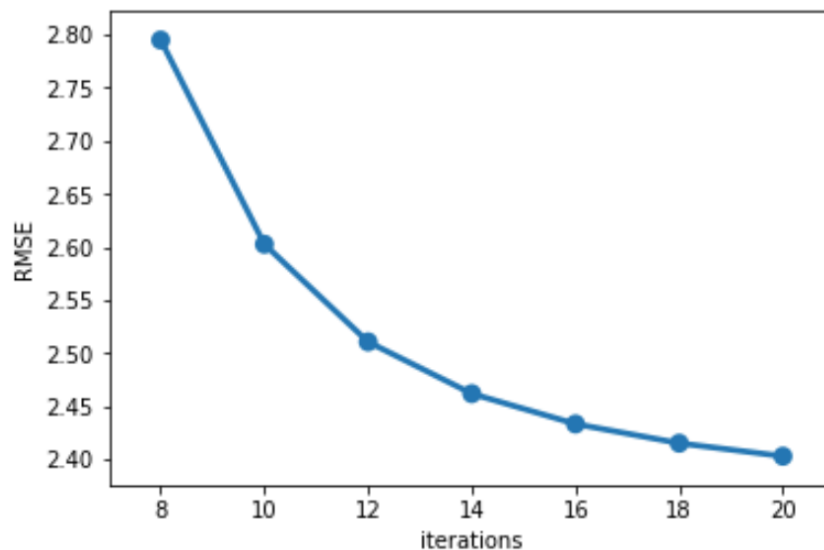
From the plot, we can tell in the range of [16,36], the RMSE fluctuated dramatically as rank increases and we got the lowest RMSE when rank=32, with the value of 3.277. Therefore, we fixed rank to be 32.

2. **lambda**: regularization_parameter. we tried setting [0.01,0.05,0.10,0.15,0.20,0.25] as values for lambda, numbers often selected to train model. By keeping rank=32, we plot out the result:



from this plot, we can tell there is a distinct decreasing trend as lambda increase. Especially, it goes down seriously from 0.01 to 0.05, and then begins to slow down the descending trend until the slope is almost to zero from 0.2 to 0.25. Here, we set the best $\lambda=0.25$.

3. **iterations:** Number of iterations of ALS, default is 5. From our domain knowledge, higher iterations, better the final test result but longer time to run model. Here, under the consideration of limited computational power, we set a set of iterations: [8,10,12,14,16,18,20] and plot it out.



Not surprisingly, we got the lowest RMSE when iteration reached the highest number of iterations, which is 20. Hence, tune parameter part is ended up with the rank=32, seed=5, iterations=20; lambda_=0.25, nonnegative=True.

3) evaluate model (based on comparison among four models on RMSE)

At the end, after training the best model, we recalculated a new RMSE again on the subset (20% of 20% test dataset) of our original testing dataset, same idea like randomly selecting several data points from our test matrix). The reason why we recalculated it on subset of our test dataset is to make sure we can compare RMSE at the same level among these models, who use 20%*20%* original dataset size as their test part due to their own limitations. At this time, by slicing down our test subset, baseline got 4.06 RMSE, and ALS returns 2.41 RMSE, which indicates ALS performs better than baseline. However, the RMSE of ALS is a little lower than that of random forest and gradient boost.

4) Extract read history

In order to determine what books user has already read and rated so that we can make a list of historical records, we extract a dataframe by inner join tables: rating dataframe and book dataframe. One instance is shown as below:

```
+-----+-----+-----+-----+
| user|rating|  ISBN|      Title|
+-----+-----+-----+-----+
| 36163|    7|13343384| "Whirlwind (Tyler|
| 36163|    6|28583879|Let's Plays Impos...|
| 36163|    5|94036543|  The Terminal Man|
|276964|   10|49887662|"The Fires of Hea...|
|276964|    7|13668921|  Tailchaser's Song|
|276964|    9|34556700|"The Dragon Token...|
|276964|    8|47316683|"Stone of Farewel...|
|276964|    7|31898998|"Crown of Shadows...|
|276964|    9|52714474|The First Deadly Sin|
|276964|    6|49359484|To The Last Breat...|
|276964|    8|12888169|The Girl Who Love...|
|276964|    5|28159654|  The Stars Compel|
|276964|    6|51810340|  Darkling I Listen|
|276964|   10|61418089|"Skybowl (Dragon ...|
|276964|   10|94036543|  The Terminal Man|
|276964|    8|34031594|      In the Dark|
|276964|   10|74753193|"The Power That P...|
|276964|    9|47371576|      The Chamber|
|276964|   10|58572765|      Crying Wolf|
|276964|    8|56671962|"Stronghold (Drag...|
+-----+-----+-----+-----+
only showing top 20 rows
```

5) Predict ratings for not reading books

After filtering books which one user has read, we extracted the structure (user, ISBN) pair RDDs for not-reading books dataframe and then input them into our model using `als.predictAll` (`user_ISBN_notread_RDD`). when we got result, we sorted them from high predicted ratings to low and selected top 5 recommended books. And we use sql command to extract related information we want (ISBN, Title, URL) other dataframe as the final output. One example is shown as below:

user	ISBN	Title	URL
276746	25216874	Call of Duty (Bat...	http://images.ama...
276746	83121098	Mending Your Hear...	http://images.ama...
276746	45021606	The Skeptical Env...	http://images.ama...
276746	12853142	200 Small House P...	http://images.ama...
276746	52915918	Reflecting His Im...	http://images.ama...
276746	25216874	Call of Duty (Bat...	http://images.ama...
276746	83121098	Mending Your Hear...	http://images.ama...
276746	45021606	The Skeptical Env...	http://images.ama...
276746	12853142	200 Small House P...	http://images.ama...
276746	52915918	Reflecting His Im...	http://images.ama...



3.3 Random Forest Model:

After we loaded data from cleaned csv files, we got three data frames: book_info, user_info and book_rating. We joined book_rating with book_info on key 'ISBN', then joined this table with user_info on 'User-ID' and dropped missing values to form our final dataset called final_dataset. Since we need to use OneHotEncoder to map label index into binary vector, we transformed string columns, such as city, state, author, into columns of label indices using StringIndexer. Then we used VectorAssembler to combine those single columns into one vector column, and output a column named "features", which will be fed as the default parameter "features" in RandomForestRegressor. Similarly, we output "label" by using StringIndexer to transform rating, which is our response variable.

We created a pipeline to put all vectors we created before and the random forest model we set ahead, then fit training data to this pipeline. We tried three different random forest regressor with 100, 200 and 500 trees. The 200-tree forest didn't decrease the RMSE by even 0.0001 compared with 100-tree forest, but took more than twice as much time as 100-tree model did. The 500-tree forest even threw a "OutOfMemory" error after running for almost one hour. Finally, we decided to select 100 as the best number of trees and got a RMSE of 1.754.

3.4 Gradient Boost Model:

Similar to the pre-model process as Random forest, we firstly join the three datasets in to one final dataset, and transfer the string type of "Age" and "year" in to Integer type. Then we encode the categorical features into numerical vectors, including book information (author, publisher, title) and user information (city, state, country), and combine the vectored features into an assembler to go through the pipeline which we build GBT regressor model on later. In addition, we use StringIndexer to transform the response variable rating to output "label".

Then we fit our training data with the pipeline and go through the gradient boost trees regressor model. We also tried some values for the model parameter maxIter, and get the best result of maxIter=10 in terms of RMSE, 1.752.

4. Model Selection and System Implementation

After tuning our three models, (ALS, Random Forest and Gradient Boost), we got the lowest RMSE for each model: ALS: 2.41; Random Forest: 1.754; Gradient Boost: 1.752, and the baseline prediction model gives us the RMSE 4.06, which shows all three models perform better than random guess or intuitive guess (baseline). How to choose the best model for our recommendation system is pretty hard, not only because of the size of our dataset, but also because of the computation speed of each model.

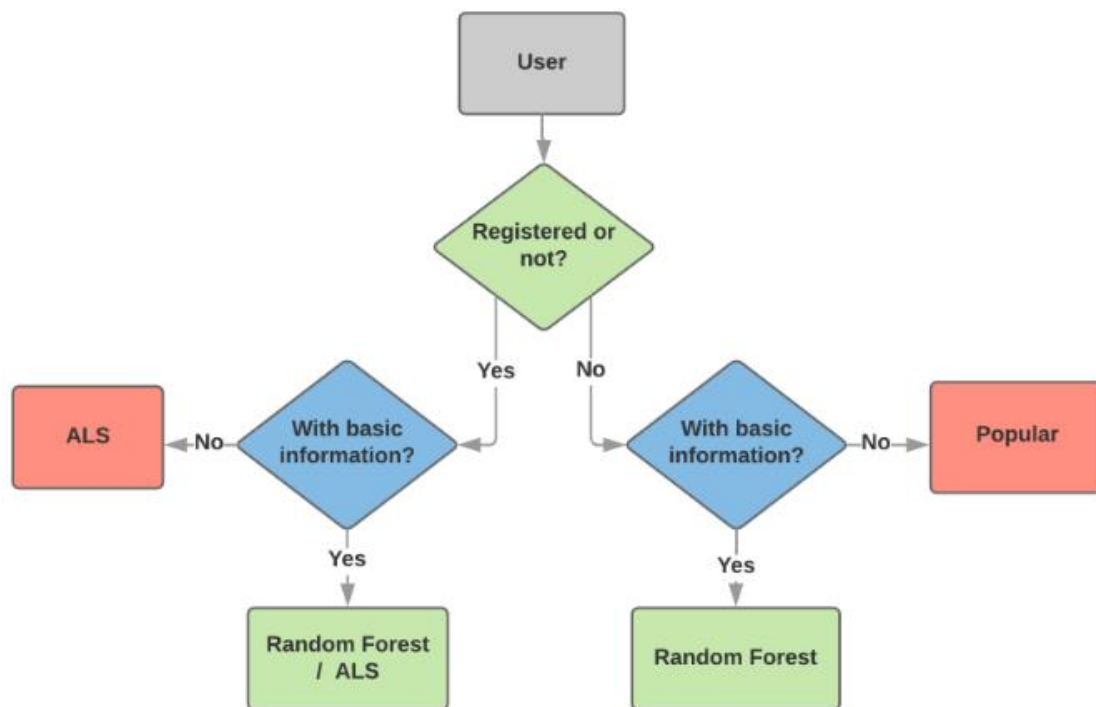
For the problem about the size of data, both tree based models, Random Forest and Gradient Boost, are taking pretty long time to train the model with a relatively small dataset, which means in the same time, these two models only can be trained by far less data comparing to ALS (for example, in 10 mins, ALS can be trained by 100,000 pieces of data, but other two models can only be trained by 1000 pieces of data.). This will make these two models less general if they only can be trained by few data.

Furthermore, not like ALS, we only need UserID and BookID to train and predict, another two models have higher requirements on input data, they need features of both books and users (age, location, publish year, publisher, book title). For the book features, it would be easy to get, but for the user basic information, it's not easy for us to collect due to user's preference of privacy. For example, if you want to get a book information, you only need an ISBN, then you can get every basic information of this book, but if you want to get a user's information, this will highly depend on user's preference, they would not give you their basic personal information due to privacy concern.

Therefore, our strategy is that we use the strength of each model to make a prediction as much precise as possible, here's how it works:

First, we will check whether this user is registered or not, if he registered, which means he's an existed user, then we will check if we have his basic information then. If we have, then we will use Random Forest to predict user's ratings of each book based on his basic information and the information of books, but if we don't have this registered user basic information and only have

UserID and reading history, we will use ALS to predict his ratings of each books then recommend first five to him. However, if the user is not registered, which means he is a totally new user, we will ask him whether he would like to give us some his basic information (age and location), if he's willing to give us, then we will use Random Forest to give him recommendation based on the users with similar features in our database, but if he doesn't want to, then we will recommend him five most popular books in our dataset. The whole decision process is in the flowchart below.

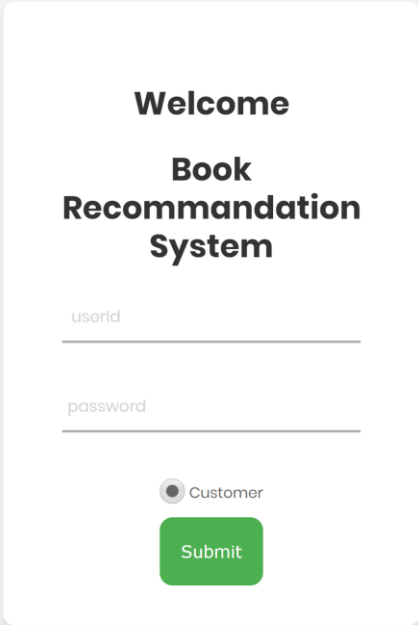


In conclusion, Random Forest and Gradient Boost works well according to their relative lower RMSE, but they higher requirement of input data and sometimes unable to deal with large dataset in a short time. So, if we only can choose one model in our recommender system, we will choose ALS because of its more general applicability, while if we can choose two models work together, we will choose Random Forest and ALS to make our prediction as precise as possible according to different user status (whether registered and basic information available or not.)

5. Front End

For front end, we used PHP to connect with MySQL and HTML to display our result. In MySQL, we create database named book, and created three tables, which are Brec, history and User, in to our book database. In Brec table, we have 'BookName' that we predicted and 'Url' that links to the cover of book. In history table, we have 'BookName' that this user read before and 'Rate' that he or she gave to this book. In User table, we just input all 'userId' that we wanted to predict, and create same password, 12345, for all user. And we wrote two PHP file, one is login.php, the other one is customer_suggestion.php.

Since we are making predictions based on user reading history, we want to make sure the history and recommendation can only be seen by user. Thus, we made a login page to protect our user's information privacy. User needs to input 'userId' and 'password' to login in to customer_suggestion page. In our login.php file, data is connected through MySQL. We also start a PHP session in our php file, so we can get the userid from login page and use that in customer_suggestion page. When a 'userId' and 'password' are input and submit button was clicked, a SQL query will be triggered to select 1 row of data that contains this userId and password combination in user table, if there is one record, we will link this page to customer_suggestion, if not, we will return an error saying, 'Incorrect Password'.



Welcome

Book Recommendation System

userid

password

☒ Customer

Submit

Once userId and password passed inspection, it will directly link to customer_suggesting.php. There are two main table in customer_suggestion, one is reading history, the other one is recommended books. We used the userId we got from login page, then two SQL queries will be executed. The first one is select 'BookName' and 'Rate' from history table based on userId. The second one is select 'BookName' and 'Url' from Brec table based on same userId. And we wrote a html img tag to display the Url as a picture. Due to change of time, some of the picture are no longer exists, so in Preview column it might return empty. After viewing recommendation information, customer can click logout button, which is locate at left upper corner, once you click the button, it will direct you to login page. We think this will makes more sense to give other customers the choice to login into this system.

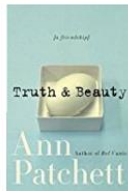



Log out

Your reading history

BookName	Rating
Child of the Hunt (Buffy the Vampire Slayer)	10
Visitors (Buffy the Vampire Slayer)	10
Charlotte's Web	5
Sons of Entropy (Buffy the Vampire Slayer: The Gatekeeper Trilogy?)	10

You might want to look at this

BookName	Preview
Truth & Beauty : A Friendship	
The Wind-Up Bird Chronicle	

6. Improvement and Learning

1. Make tree-based model more robust

As mentioned in section 4, both Random Forest and Gradient Boost Tree are heavily relying on features fed into the model. But in practical, we are not likely to have full set of features we need for every user. Moreover, it's important to improve tree-based models to handle larger data size. We believe that reducing features can greatly save computation power. We may realize less feature input either by adding default value to every feature or by allowing flexible number of features input. In either way, we need to figure out will they still have lower RMSE than ALS with less features? What is the minimum number of features for the model to have a relatively low RMSE, as well as handle larger data size?

2. More parameter tuning for each model

We tried three algorithms so far and got a far lower RMSE than baseline for each model. But we noticed that ALS can greatly decrease RMSE by changing rank, lambda and iteration. With more combinations of optimal parameters, we can expect a better RMSE. For the other two models, we only tried to change the number of trees but failed to get a better result. Based on our knowledge on tree models, it makes sense to tune “maxDepth” and “mainInfoGain” to set an early stop to avoid overfitting, so that we can make the model more efficient and more “accurate”.

3. Combine other machine learning methods

We also consider to further dig our data with other methods. For example, mining topic similarity between books by TF-IDF, which may help us to target users who like to read books with similar topics. We can derive more features from current data so that the model can learn our data in more details. It helps to improve the performance even if we cannot feed in huge amount of data.

Appendix

Figure 2.1 Overview of BX-Book-Ratings.csv

```
In [5]: rating.head(10)
```

Out[5]:

	ISBN	Book-Rating
User-ID		
276747	0060517794	9
276747	0671537458	9
276747	0679776818	8
276804	0440498058	8
276808	0395547032	10
276811	0440414121	10
276896	0440241537	10
276928	059030271X	5
276928	0671021354	10
276928	0671026283	10

Figure 2.2 Overview of BX-Books.csv

```
In [8]: book.head(10)
```

Out[8]:

	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M
ISBN						
0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.0...	http://images.æ
0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.0...	http://images.æ
0399135782	The Kitchen God's Wife	Amy Tan	1991	Putnam Pub Group	http://images.amazon.com/images/P/0399135782.0...	http://images.æ
0425176428	What If?: The World's Foremost Military Histor...	Robert Cowley	2000	Berkley Publishing Group	http://images.amazon.com/images/P/0425176428.0...	http://images.æ
0440234743	The Testament	John Grisham	1999	Dell	http://images.amazon.com/images/P/0440234743.0...	http://images.æ
0452264464	Beloved (Plume Contemporary Fiction)	Toni Morrison	1994	Plume	http://images.amazon.com/images/P/0452264464.0...	http://images.æ

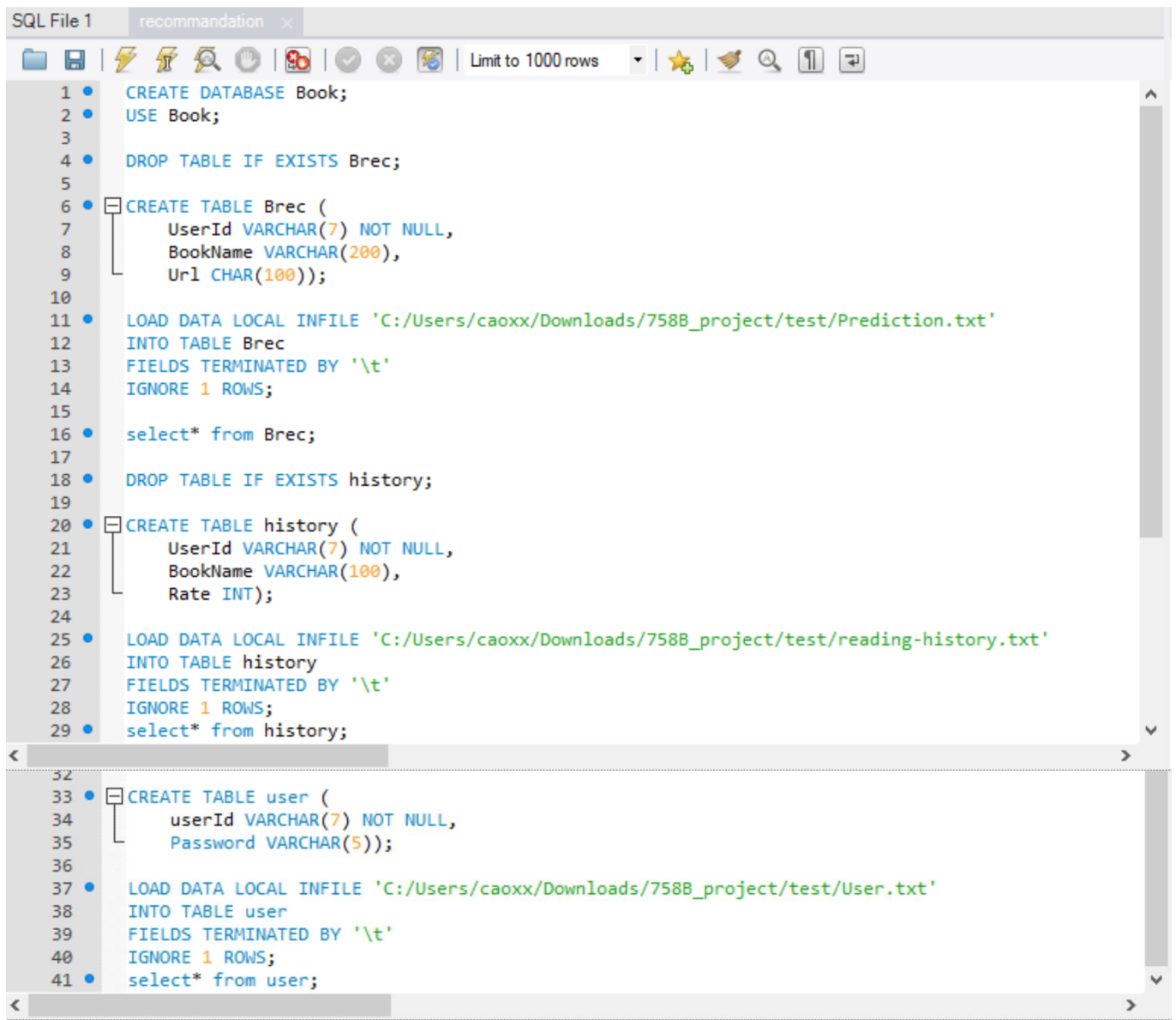
Figure 2.3 Overview of BX-Users.csv

```
In [10]: user.head(10)
```

Out[10]:

	city	state	Age
User-ID			
42	appleton	wisconsin	17
44	black mountain	north carolina	51
51	renton	washington	34
56	cheyenne	wyoming	24
67	framingham	massachusetts	43
70	rochester	new york	44
75	long beach	california	37
78	oakland	california	18
99	franktown	colorado	42
109	muncie	indiana	20

Figure 5.1 Create Book Database and Tables in MYSQL to Show in Front End



The screenshot shows a MySQL SQL File editor window titled "SQL File 1" with a tab labeled "recommandation". The editor contains SQL code for creating a database and tables, loading data from local files, and querying the data. The code is as follows:

```
1 CREATE DATABASE Book;
2 USE Book;
3
4 DROP TABLE IF EXISTS Brec;
5
6 CREATE TABLE Brec (
7     UserId VARCHAR(7) NOT NULL,
8     BookName VARCHAR(200),
9     Url CHAR(100));
10
11 LOAD DATA LOCAL INFILE 'C:/Users/caox/Downloads/758B_project/test/Prediction.txt'
12 INTO TABLE Brec
13 FIELDS TERMINATED BY '\t'
14 IGNORE 1 ROWS;
15
16 select* from Brec;
17
18 DROP TABLE IF EXISTS history;
19
20 CREATE TABLE history (
21     UserId VARCHAR(7) NOT NULL,
22     BookName VARCHAR(100),
23     Rate INT);
24
25 LOAD DATA LOCAL INFILE 'C:/Users/caox/Downloads/758B_project/test/reading-history.txt'
26 INTO TABLE history
27 FIELDS TERMINATED BY '\t'
28 IGNORE 1 ROWS;
29 select* from history;
30
31
32
33 CREATE TABLE user (
34     userId VARCHAR(7) NOT NULL,
35     Password VARCHAR(5));
36
37 LOAD DATA LOCAL INFILE 'C:/Users/caox/Downloads/758B_project/test/User.txt'
38 INTO TABLE user
39 FIELDS TERMINATED BY '\t'
40 IGNORE 1 ROWS;
41 select* from user;
```

Figure 3.2.2, 2) Example of tuning one parameter in ALS model

2.1 train ALS model and tune parameter based on the RMSE using the whole test dataset

```
: # tune parameter:rank [16,20,24,28,32,36]
seed = 5
iterations = 10
regularization_parameter = 0.1
rank_list=[]
rmse_list=[]
ranks = [16,20,24,28,32,36]
errors = [0]*len(ranks)
err = 0
tolerance = 0.02

min_error = float('inf')
best_rank = -1
best_iteration = -1
for rank in ranks:
    model = ALS.train(train, rank, seed=seed, iterations=iterations,
                      lambda_=regularization_parameter)
    predictions = model.predictAll(test1).map(lambda r: ((r[0], r[1]), r[2]))
    rates_and_preds = test.map(lambda r: ((int(r[0]), int(r[1]), float(r[2])))).join(predictions)
    error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())
    errors[err] = error
    err += 1
    rank_list.append(rank)
    rmse_list.append(error)
    print('For rank %s the RMSE is %s' % (rank, error))
    if error < min_error:
        min_error = error
        best_rank = rank

print('The best model was trained with rank %s' % best_rank)

For rank 16 the RMSE is 3.46602583561054
For rank 20 the RMSE is 3.3780337290558395
For rank 24 the RMSE is 3.4433434747312863
For rank 28 the RMSE is 3.3323389053232546
For rank 32 the RMSE is 3.2772340975673506
For rank 36 the RMSE is 3.3695030543670783
The best model was trained with rank 32
```

Figure 3.2.2, 5) Prediction Function

4. Predict with our recommendation system

```
[3]: li=[276747,276804,276811,276896,276928]

[4]: x_list=[]
    a=0
    for user in li:
        try:
            ### Determining what books user has already read and rated so that we can make a list of historical records.
            book_read = df_rate.filter(df_rate.user == user).alias('a').\
            join(df_book.alias('b'),col('a.ISBN') == col('b.ISBN'),'inner').select('a.user','a.rating','b.ISBN','b.Title')
            ### Determining what books user has not already read and rated so that we can make new e recommendations
            book_notread = df_rate.filter(df_rate.user == user).alias('a').\
            join(df_book.alias('b'),col('a.ISBN') == col('b.ISBN'),'right').filter('a.user is null').select('b.ISBN','b.Title')
            #make a list of not reading books for user
            notread_RDD=book_notread.rdd
            # get structure (user, ISBN) pairs for not-reading books
            user_ISBN_notread_RDD=notread_RDD.map(lambda x: (user, x[0]))
            user_recommendations_RDD = als.predictAll(user_ISBN_notread_RDD)
            #get predicting List as a desceding order
            Top= spark.createDataFrame(user_recommendations_RDD, ('user', 'ISBN','pred_rating')).sort('pred_rating',ascending=False)
            # get top5 recommended books
            Top5=sc.parallelize(Top.take(5)).toDF()
            # create SQL to get the final result
            df_book.registerTempTable('book')
            Top5.registerTempTable('Top5')
            if a==0:
                x=sqlContext.sql('Select Top5.user,book.ISBN, Title, URLM from book, Top5 where book.ISBN=Top5.ISBN')
                r=book_read
            else:
                y=sqlContext.sql('Select Top5.user,book.ISBN, Title, URLM from book, Top5 where book.ISBN=Top5.ISBN')
                x=x.union(y)
                b=book_read
                r=r.union(b)
            a+=1
            print(str(user)+' done')
        except:
            print(str(user)+' error')

276747 done
276804 done
276811 done
276896 done
276928 done
```


References

1. Image-URL-S, Image-URL-M, Image-URL-L columns are redundant, which is image URL with different size. We decided to drop two of them and only include Image-URL-M.
2. <https://github.com/XuefengHuang/RecommendationSystem>