Data Analytics

Konstantinos Pytharoulios
kpytharoulios@gmail.com

Exploring and describing the data

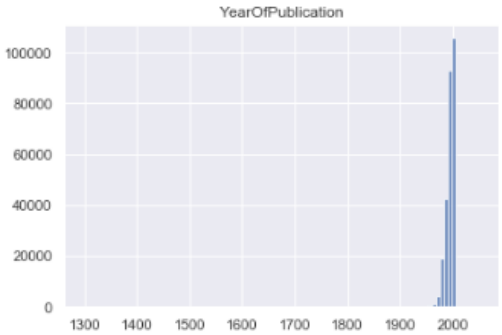
The main idea is to take firstly each one of the csv files (*BX-Book-Ratings-Test.csv, BX-Book-Ratings-Train.csv,BX-Books.csv,BX-Users.csv)* **and check their content individually. In order to do so, we are creating a Panda Dataframe for each one of them.**

1. Books Dataframe

Here we observe that BX-Books.csv contains 8 different columns of information for each book. The key here is ISBN and information about the title of the book, its author, the year of its publication, the publisher as well as three links for photos of the book are given.
Urls will not really help us anywhere in our calculations, so we can drop them off.

Next we check about the NaN values in the Dataframe. It is very helpful that only one Book Author is missing and only two Publishers. Not a problem for a corpus of 271360 books.

Next step is to check the unique values of Years of Publication. Here we can find some unreal values like years in the future (after our current 2020th year) and years like 0 when obviously no books were existed. In order to fix this, because there are many years like this, we are going to use either "mean" if the distribution of years is normal or "median" if the distribution of years is skewed.



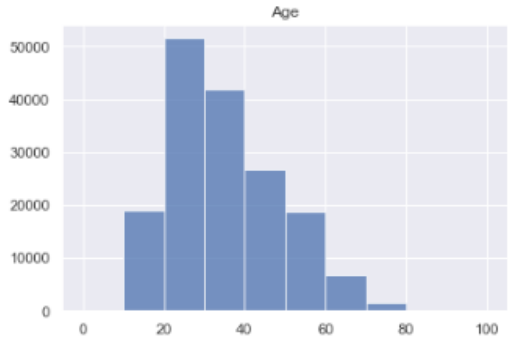
The distribution seems like negative skewed thus we are going to use "median" to fill the unreal values.
It is interesting but also logic that the plot after the substitution of unreal years with the median seems almost the same. That is because only a small portion (4632/271/360) of the years of publication was unreal. (the updated plot is only included in the jupyter in order to save some space here for the other information I want to show.)
We are also checking the authors with the most books, the publisher with the most publications, and checking about repeated titles..
As it seems, the same book from the same author may have plenty of unique ISBNs. This happens because the same book could either be published by different publishers or in different years. This is something that we should later consider because our goal is to recommend books. So maybe it would be a good idea if we could give the same ISBN to the same books. But we will figure out this later when developing the recommendation system.

2. Users Dataframe

In Users' Dataframe we observe that in the column "Location" three different infos are given so it would be better to split them in three columns in order to handle them better later. So we split Location to <City, Region, Country>.
Clearly we can also see that **age is not a good metric** because almost the half of the ages is either missing or unreal, **so we will not use it for our predictions later**. However, by the following

plot (img 2) which is based on the ages we have, it is interesting to say that the most users are between 20 and 40 years old.

img 2



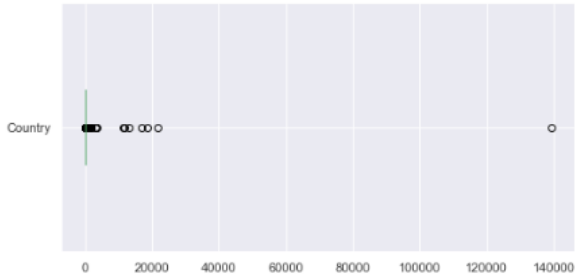
img 3

```
[49]: 1 booksRatings_train.head()
```

[49]:

	User_ID	ISBN	Book_Rating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

The box plot on the right shows us the distribution of users' countries. Except from an outlier all the others seem too close. So we are not going to use neither of these information: Country/Region/City.



3. Books Ranking (Training) Dataframe

We observe that BX-Book-Ratings-Train contains a list of Book-Ratings (1-10 for explicit ratings or 0 for implicit ratings) that specific users gave to specific books. Each user may has given ratings for more than one book in the list as we can easily see from the 3rd and 4th row of the panda in img 3.

```
dtype: object
```

```
1 explicit_booksRatings_train["Book_Rating"].describe()
```

```
count    379020.000000
mean          7.599786
std           1.838548
min           1.000000
25%           7.000000
50%           8.000000
75%           9.000000
max          10.000000
Name: Book_Rating, dtype: float64
```

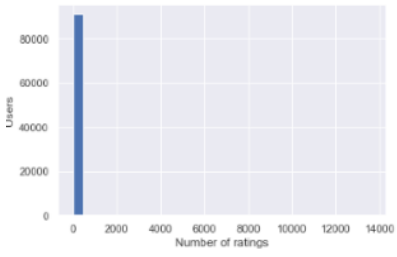
Interesting to say is that the mean value of explicit rating is around 7,5 and that only 379.020/999.999 ratings are not 0, which is the actual portion of ratings from users which we will later use for training.

Also the number of unique users who have rated at least one book is 91407/999999.

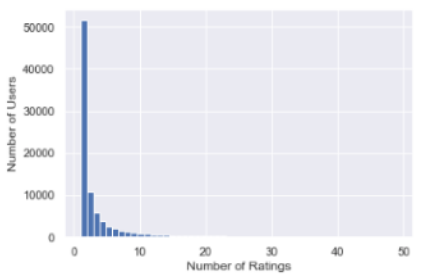
It is interesting that if we take a look at the top 10 users with the most ratings (img 4) we will see that even the 10th has given 3067 ratings! But…soon after this, based on the plots on img 5 and img 6 we will see that these users are outliers, because the most users have 1-2 ratings.

```
Top 10 users with the most ratings are:
User_ID
11676     13602
198711     7550
153662     6109
98391      5891
35859      5850
212898     4785
278418     4533
76352      3367
110973     3100
235105     3067
Name: ISBN, dtype: int64
```

img 4

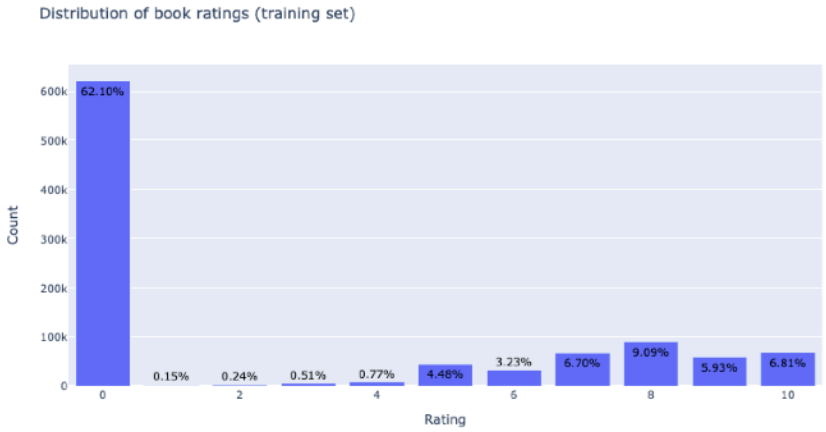


img 5



img 6

As about the distribution of the ratings now, if we don't take implicit rating (rating 0) into consideration which by far the most common, we can see that 8 is the most common rating and generally the most ratings are between 7 and 10, which means for us that we have to be prepared to wait for pretty good ratings predicted later.



4. Books Ranking (Testing) Dataframe

After doing the same check as with training set, which you can find detailed in Jupyter notebook, the important things to keep from here are that only **13877/149781** ratings are not 0, which is the actual portion of **explicit ratings** from users and that there are **13877/149781 unique users.**

Please have in mind that you can observe even more findings inside the Jupyter Notebook.

Recommendation Systems

First of all, we are going to join the **Books Dataframe** with **Books Ratings Dataframe**, in order to get a single Dataframe which will contain for each book, each rating from each user who have rated it. The columns "User_ID", "ISBN" and "Book_Rating" will be used.
Then we remove rows with unknown book titles because if we do not have a book's title, we can not recommend it properly.

After this we are making a check about the Top 10 highly rated books and the Top 10 worst books. In this calculation only books with more than 30 ratings are taking part because we want a more objective mean rating. Interesting part here is that the best rating is 9.33 and the worst 4,46!

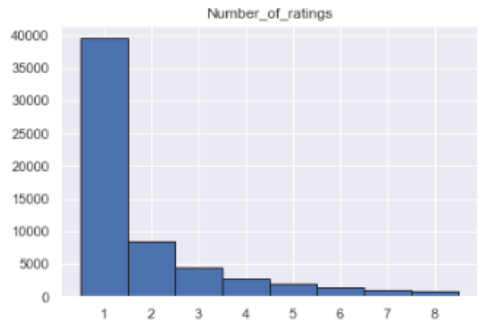
Best

```
BookTitle
The Two Towers (The Lord of the Rings, Part 2)              9.328125
The Return of the King (The Lord of the Rings, Part 3)      9.271739
The Hobbit                                                  9.204545
Love You Forever                                            9.187500
Charlotte's Web (Trophy Newbery)                            9.155172
The Stand (The Complete and Uncut Edition)                  9.121212
Lonesome Dove                                               9.121212
Harry Potter and the Goblet of Fire (Book 4)                9.117647
Harry Potter and the Prisoner of Azkaban (Book 3)           9.108225
84 Charing Cross Road                                       9.075472
Name: Book_Rating, dtype: float64
```

Worst

```
BookTitle
Dr. Atkins' New Diet Revolution    6.500000
My Gal Sunday                      6.468085
The Sky Is Falling                 6.378378
The Little Friend                  6.325000
Sex &amp;amp; the City             6.222222
Violin                             5.921053
4 Blondes                          5.634146
Four Blondes                       5.533333
Isle of Dogs                       5.356322
Wild Animus                        4.461538
Name: Book_Rating, dtype: float64
```

Surprise Method



So at the moment we are beginning with Surprise method. We have already created a Panda Dataframe with these three columns: User_ID, ISBN, Book_Rating. Our next step is to keep books with more ratings than 5 and users who have given more ratings than 5 because the dataset is really huge and we want to reduce it a lot in order for the next steps to take logical amount of time. By doing this we reduce the rows from 335206 to 106525. The graph on the left shows us that a great amount of users have rated less than 5 books, so if we remove them we will reduce our dataset a lot.

Next step is to collect **benchmarks** for the different algorithms which can be used with **Surprise** in order to find which has the best metrics.

Algorithm	test_rmse	fit_time	test_time
SVD	1.585825	6.781608	0.440792
BaselineOnly	1.592879	0.690099	0.584124
SVDpp	1.595431	195.651053	6.044965
CoClustering	1.760242	6.407754	0.617867
KNNBaseline	1.785684	7.734443	3.341514
KNNWithZScore	1.789177	6.636177	2.873625
KNNWithMeans	1.798982	6.569046	2.508903
SlopeOne	1.915254	9.277485	3.994931
KNNBasic	1.948529	6.553240	2.438572
NormalPredictor	2.404311	0.225338	0.556426

SVD appears to be the optimal solution but **BaselineOnly** has really close metrics with SVD. So we are checking both of them with Alternating Least Squares (**ALS**).
After training and testing, we get the predictions. We firstly check the 10 best predictions and then we check the worst 10 predictions for SVD.

These are the worst 10 predicted ratings:

	uid	iid	rui	est	details	Iu	Ui	err
46549	274766	0385486804	1.0	8.124591	{'was_impossible': False}	0	64	7.124591
46852	275111	034530988X	1.0	8.156555	{'was_impossible': False}	0	11	7.156555
5410	243312	0765342987	1.0	8.187607	{'was_impossible': False}	0	19	7.187607
29938	261829	0316776963	1.0	8.199465	{'was_impossible': False}	0	98	7.199465
45098	273898	0451191153	1.0	8.235076	{'was_impossible': False}	0	19	7.235076
41367	270605	1558506462	1.0	8.242471	{'was_impossible': False}	0	11	7.242471
6147	243930	0349101779	1.0	8.381651	{'was_impossible': False}	0	19	7.381651
40931	270154	0316168815	1.0	8.399246	{'was_impossible': False}	0	26	7.399246
17624	251541	0316168815	1.0	8.399246	{'was_impossible': False}	0	26	7.399246
17876	251754	0385504209	1.0	8.557915	{'was_impossible': False}	0	272	7.557915

img 7

img 8

As we can see in img 7 the worst prediction here is 8,55 which is really far from the real rating of book with ISBN: 0971880107 which is 1,0. The reason for this wrong prediction is obvious. As we can understand from the plot in img 8, rating with 1 this specific book is an outlier. In fact it is the least common rating for this book. The most users have rated it with 8, 9 or 10! So that is why the prediction is around 8,5.

Finally it is interesting to say that all the estimations of RMSE, MSE, MAE and FCP look really good.

```
RMSE: 1.8585
MSE: 3.4540
MAE:  1.4755
FCP:  0.3118
```

After this, we do exactly the same exploration of **BaselineOnly** algorithm from which we have exactly the same notices. For this reason we are skipping the details about this here but of course you can find them all inside the Jupyter Notebook.

LightFM

I decided just for curiosity to inspect one more method of implementing a Recommendation System, LightFM. Again, we are limiting our ratings to only explicit ones (from 1 to 10) and books and users to those with more than 5 ratings taken and given respectively.

Next step is to prepare the training and testing set which consists of the same columns as in Surprise method (User_ID, ISBN, Book_Rating), so as to have the appropriate form in order for LightFM to be trained and then predict.

We checked two different methods for LightFM: **WARP (Weighted Approximate-Rank Pairwise loss)** and **BPR (Bayesian Personalised Ranking pairwise loss).**

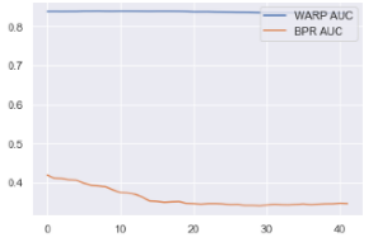
In order to decide which one of them better fits our needs, we calculated the AUC scores and Precessions of both training and testing sets.

Have in mind that Precision scores are focused on the quality of ranking of the top of the list. So, the important thing is that the top K items are mostly positive.

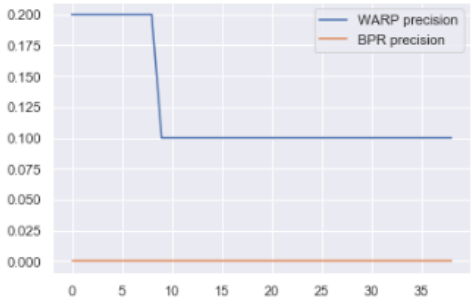
While on the other hand...

AUC scores are focused on the quality of all rankings in general." It can be interpreted as the probability that a randomly chosen positive item is ranked higher than a randomly chosen negative item" and also "a high AUC score will then give you confidence that your ranking is of high quality throughout" as mentioned here: "https://stackoverflow.com/questions/45451161/evaluating-the-lightfm-recommendation-model" .

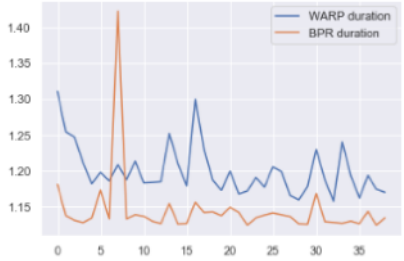
But before doing this, we used skopt which is a hyper parameter optimisation library. With this library we were able to know an optimal set of parameters for LightFM.



And of course, it is pretty much obvious from the plot on the left, that WARP produces better results than BPR. As it also seems, after the 20th epoch, WARP starts to decrease a little bit which means that it would be better to stop somewhere there when fitting.

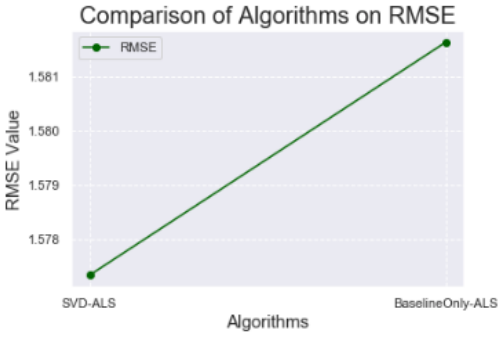


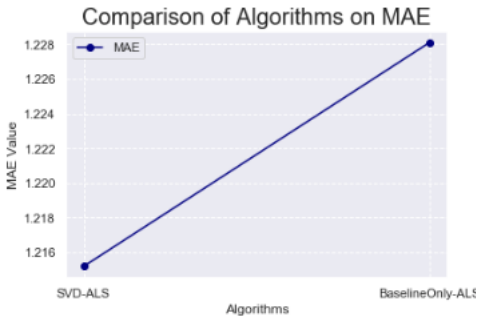
Based on the plot on the right, WARP appears to have a better precision at all for k=10.

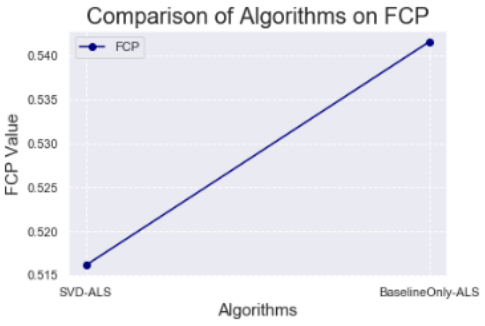


It is interesting that WARP is slower than BPR for all epochs except from around 7th epoch when suddenly BPR increases its duration a lot and then it decreases again.

Comparison of SVD and BaselineOnly







All the plots above show us that SVD-ALS has smaller RMSE and MAE than BaselineOnly. But BaselineOnly has a greater FCP value. Of course we must have in mind that RMSE and MAE the smaller values they have the better it is for the algorithm whereas for FCP is the opposite. The greater it is the better for the algorithm. But looking at the values closely, we have to admit that the differences between them are tiny.

The only difference we could say that it is obvious is that **SVD is slower** because it takes almost 9 seconds to predict the test set while **BaselineOnly** takes approximately 3 seconds to predict the same test set

Comparison of Surprise and LightFM

The truth is that it is not really easy to compare these two methods because the metrics which are used for them (RMSE/MSE/MAE/FCP for Surprise and AUC/Precision for LightFM) are totally different. I spent almost two days trying to find a way to calculate for example AUC of SVD(Surprise) or RMSE of LightFM but unfortunately I was not able to find a way. The tutorial about Recommendation Systems didn't have any information about this either. The only thing I managed to find is that RMSE is not a good metric for SVD neither is AUC for LightFM…

However as about the time, **Surprise** is by far **faster method** than LightFM since in our tests it took only **9,17** seconds to train and predict (fit & test) whereas **LightFM (WARP)** took **219.80** seconds and LightFM (BPR) **220.20** seconds.

Although SVD-Surprise is faster, I have to admit from my experience with those two methods, that LightFM gives better predictions at all.