Project definition

Project Overview

Recommending products is a key challenge in different kinds of business. E-commerce and retail companies are leveraging the power of data and boosting sales by implementing recommender systems on their websites. Nowadays, Recommender Systems are increasingly used in a lot of well-known websites and it's a great opportunity to explore this technique deeply.

Recommender systems can be also applied in multiple use cases: retail cases, e-commerce, bank products. Improving forecast quality is of great advantage to make accurate business decisions or provide to the client a new and personalized product or service for them.

How recommender system works

Recommender systems function with two kinds of information:

- Characteristic information. This kind of recommender system uses information about items (keywords, categories, etc.) and users (preferences, profiles, etc.).
- User-item interactions. Ratings, number of purchases, likes, etc. given by a user for the items

Based on this, I can distinguish between three algorithms used in recommender systems:

- Content-based systems, which use characteristic information.
- Collaborative filtering systems, which are based on user-item interactions.
- Hybrid systems, which combine both types of information with the aim of avoiding problems that are generated when working with just
- one kind.

Next, I will dig a little deeper into content-based and collaborative filtering systems and see how they are different.

Content-based recommender systems recommend items similar to those liked by a given user. These systems analyze a set of documents and/or descriptions of items previously rated by a user in order to build a profile of the user's interests.

Collaborative filtering methods produce user-specific recommendations of items based on ratings patterns (like the 1-5 stars on a Netflix movie) or usage (e.g., Amazon purchases) without the need for other information about the user or the item. Two different types are used: user-based that exploits correlations between users, or item-based that exploits correlations between items. Another option is to join both options in a matrix factorization algorithm, which will be further explained in the Algorithms and Techniques part.

Problem Statement

Nowadays, there are a lot of companies that recommend services and products on their websites. Products like movies and online ecommerce websites are offering products with a recommender system algorithm in the backend.

The problem here is to recommend books using the Goodreads database. In this website reviews and ratings are available there for the books in their database.

What I want to do is to recommend books to the usual users of the website, since new users should be treated using another algorithm. New users in a website in a recommender system problem.

To solve this problem I will make use of the state of art solution based on Matrix Factorization. The algorithm I am going to use to solve the problem is SVD in the scikit-surprise Python library. This algorithm needs user_id, item_id and rating to build a matrix, typically sparse. The algorithm divides this matrix in a product of two matrices as I will explain in Algorithms and Techniques section.

This algorithm proves to do very well for recommendation system as it is the base of the Netflix prize competition, that used ALS (Alternative Least Squared) which is based on Matrix Factorization in a parallel algorithm.

Metrics

What kind of problem is this? Well, Ihave to observe what the output is. Since the predicted value is the rating that a user is going to provide, this value must be between 1 and 5 and it follows an increasing order. Therefore, I model the output as a “regression” problem and metrics such as RMSE or MAE should be used.

I will use the RMSE for the predicted rating compared to the real rating using a 5 fold cross validation. Then I will be able to compare the algorithms proposed

$$\text{RMSE} = \sqrt{\sum((A'_{predicted} - A_{observed})^2)}$$

Only the observed elements of matrix A (matrix of users and items) the metric is computed

I prefer RMSE over MAE because RMSE reduces larger errors since its error function penalizes the difference between true and predicted value squaring it.

I will also observe the MAE metric. In case MAE is too high in the lowest RMSE, I will decide on another algorithm with different parameters to deploy in production.

Analysis

Data Exploration

Datasets and Inputs

As I mentioned in Capstone Proposal, I am going to use a Goodreads database from this Kaggle Database available in this link

This database contains information about the books and the interactions between users and those books in goodreads website.

Basically, Ihave to types of datasets: books data and interactions data:

- Every book data csv has the following features:
    - pagesNumber. Book number of pages
    - Authors. Author name
    - Publisher
    - Rating. Average rating in the whole dataset.
    - Language. 15% are informed.
    - RatingDistTotal
    - RatingDist5. Number of rating 5 given to the book
    - RatingDist3. Number of rating 3 given to the book
    - CountsOfReview.
    - PublishDay. Day of month of publication
    - ISBN
    - RatingDist4. Number of rating 4 given to the book
    - PublishMonth. Month of publication
    - Id. Book ID
    - PublishYear. Year of publication
    - RatingDist1. Number of rating 1 given to the book
    - RatingDist2. Number of rating 2 given to the book
    - Name. Book name
    - Description.
    - Count of text reviews. Number of reviews with a text.
    - PagesNumber. Book number of pages. Same meaning of the first variable, but when the first is informed, the second is not and vice versa.
- Every book user interaction data csv has the following features:
    - ID. User ID.
    - Name. Book name
    - Rating. These values are in a string format with increasing levels. "This user doesn't have any rating", "did not like it", "it was ok", "liked it", "really liked it" and "it was amazing"

Data Exploration

Basics counts

- Number of unique books in books dataset: 1636235
- Number of unique ID (books ID) in books dataset: 1850115

  This does not make any sense, so it seems that the book's name should be used.

- Rating values: These values are in a string format with increasing levels. "This user doesn't have any rating", "did not like it", "it was ok", "liked it", "really liked it" and "it was amazing"
- Rating transformation. I cannot use that rating as a value. I should change the values by a number. Then I have 0 to 5 ratings. But zeros will be filtered.
    - "This user doesn't have any rating": 0,
    - "did not like it”: 1
    - "it was ok”: 2
    - "liked it": 3
    - "really liked it": 4
    - "it was amazing": 5
- Number of users: 8919
- Number of books in interactions dataset: 103533
- Number of ratings: 362596
- Number of users after filtering zeros: **4154**
- Number of books in interactions dataset after filtering zeros: **103532**
- Number of ratings after filtering zeros: **357831**
- Number of authors in interactions after filtering: **23294**

4765 users have interactions with any book. These users are not going to go through the model since Collaborative Filtering works with historical data. Recommending items to these users is out of scope of this project. But in the future, it could be interesting to analyze what kind of books I can recommend to these users. However, I do not have any data of these users. Therefore, I cannot recommend any book using Machine Learning. I would recommend the most popular books. Then, we could build a profile for these users. But, as I previously mentioned, this is out of scope.

Exploratory Visualization

Rating values

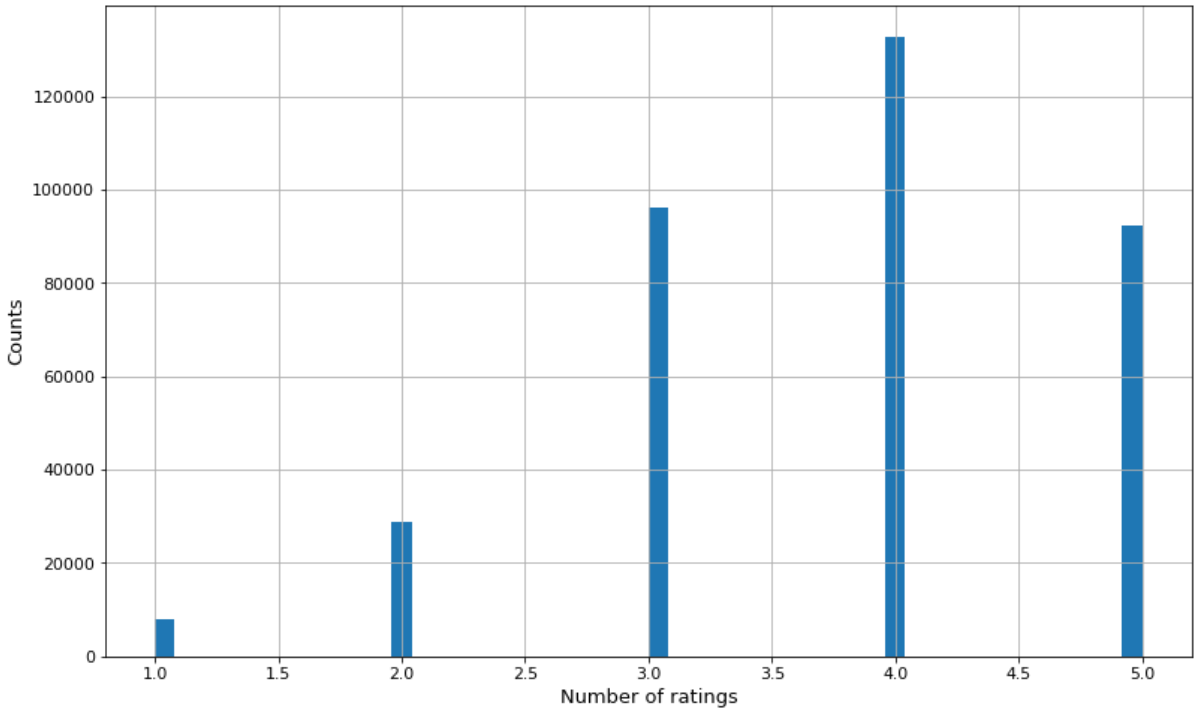


Figure 1. Histogram of rating values

Number of ratings done by book

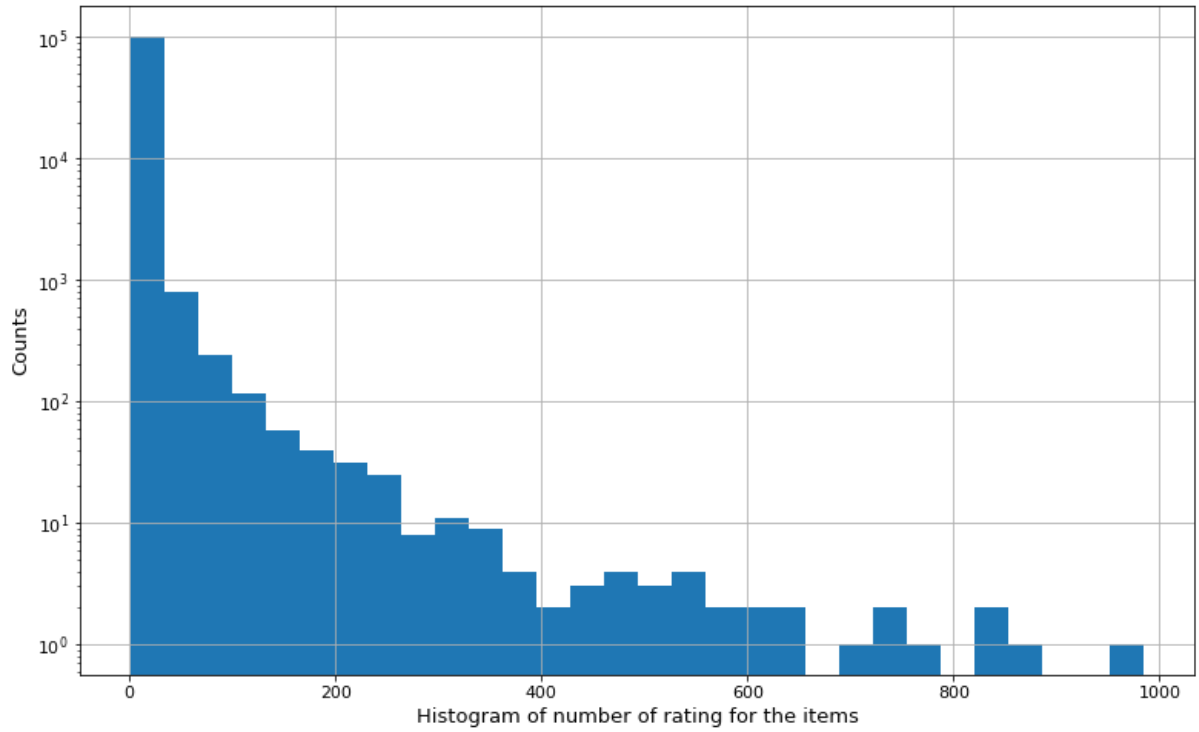


Figure 2. Number of ratings done by book. (Y-axis is in log-scale)

We can see that we have a lot of books with only one or two ratings. In normal circumstances, I would filter these books. However, the little amount of data prevents me from discarding this and I am going to keep this data due to the amount of data.

Avg rating distribution by book

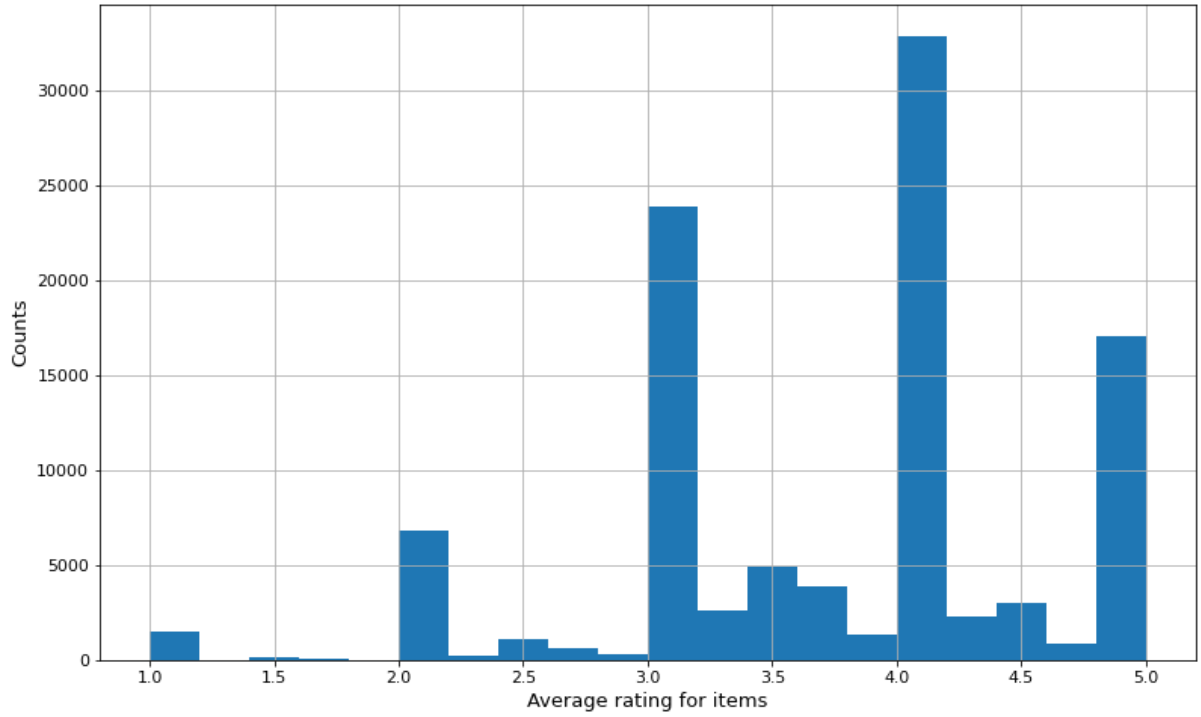


Figure 3. Average book rating distribution.

Most of the average are in 1, 2, 3, 4, 5 since we have a lot of books and not so many ratings.

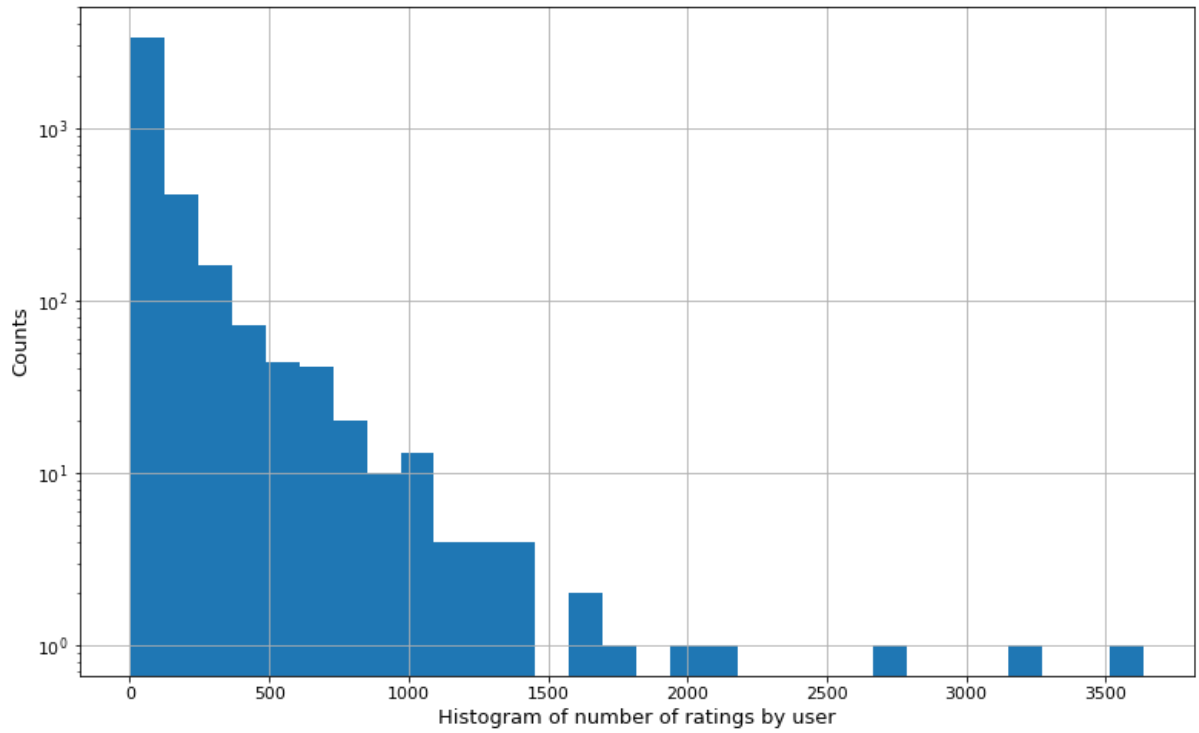


Figure 4. Number of ratings by user.

In figure 4, the effect exists again, as in figure 2. However, it seems not to be as sharp as the previous one, since we have only 4100 users and around 300K ratings.

Avg rating distribution by users.

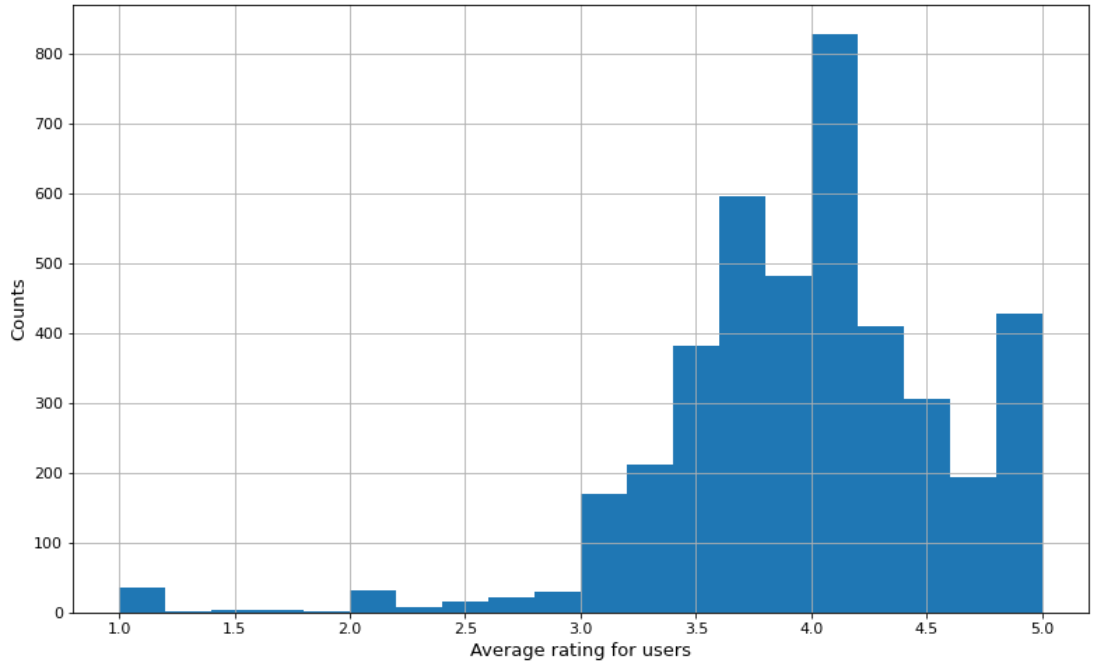


Figure 5. Average user rating distribution.

In figure 5, it seems that most of the users really like what they read and rate. This could be a bias if people do not rate books that they do not like, but we cannot know this for sure. Here, the effect that figure 3 has is not that sharp since we have a rating/user rate very high.

Algorithms and Techniques

Algorithm 1 - Matrix Factorization

A matrix factorization is a way of reducing a matrix into its constituent parts. It is an approach that can simplify more complex matrix operations that can be performed on the decomposed matrix rather than on the original matrix itself.

An analogy is the factoring of numbers as we studied in elementary school, such as the factoring of 30 into 2 x 3 x 5. However, there are many ways to decompose a matrix, hence there are a range of different matrix decomposition techniques.
One of the most used methods in Collaborating Filtering using Matrix Factorization technique is Singular value decomposition (SVD) algorithm. The aim is to provide users with books' recommendations from the latent features of item-user matrices.
Here the original matrix A has dimensions number of users (N_u) times number of items (N_i) (N_u x N_i). A is factored in two matrices; the first one, U, will have N_u x M dimensions and the second matrix, V, will have N_i x M being M a parameter of the algorithm.

The final matrices will be $A = U V^T$.

Finally, we can see U as a matrix with users' features and V a matrix with items' features. To build these two matrices we use the registers we have in A, since this matrix will usually be very sparse (a user only reads/buys/uses a few products) and U and V are build iteratively to minimize the error when these two matrices are multiplied.

In this first algorithm, we use A simply as the user-item interaction.

Algorithm 2 - Combined matrix with Matrix Factorization

For this algorithm, We want to introduce bias for A adding the author information. To do that I am going to compute this:

1. Let us call A to the matrix with user_id, item_id and ratings matrix. A (N_u x N_i)
2. And A_2, to the matrix with user_id, author_id and ratings matrix. This matrix will have the same dimension as A, there will be duplicates with user_id and author_id because we want to sum the same values. This is going to be computed with a Window function to maintain the same elements.
3. A' will be the weighted sum of A and A_2:

   A' = (1- k) A + k A_2 for k in [0,1].

   A' will be the weights used for training, but the error will be computed with A.

RMSE = $\sqrt{\sum((A'_{predicted} - A_{observed})^2)}$

After that, I apply the first algorithm for this matrix with the criteria I showed.

Benchmark

Searching solutions with the same input data in Kaggle dataset competition, I found a notebook exploring different algorithms in this link. That notebook uses RMSE as error metrics and algorithms such as ALS and Deep Learning.
The results were the following for that notebook.

Algorithm	**MSE**	**RMSE**
ALS	3.73	1.9319
Deep Learning	1.7362	1.317

Table 1. Error obtained in the notebook of the benchmark studied

My goal here is to have better results in my algorithms than that Deep Learning algorithm. However, I would like to be ambitious and my aim is to get an RMSE lower than 1, thus, the rating error is not higher than 1 step in the predictions. For example if a book really has a rating of 4, my prediction is not outside the range between 3 and 5 in the majority of the cases.

Methodology and results

Data Preprocessing

Where is the data?

As I mentioned in previously, we are going to use a Goodreads database from this Kaggle Database available in this link

This database contains information about the books and the interactions between users and those books in goodreads website.

The way I read in the project is based on this csvs. If it has changed contact me. Even so, if only the numbers have changed, you only have to change them in config.py. The csvs used here are:

Books data:

- *book1-100k.csv*
- *book100k-200k.csv*
- *book200k-300k.csv*
- *book300k-400k.csv*
- *book400k-500k.csv*
- *book500k-600k.csv*
- *book600k-700k.csv*
- *book700k-800k.csv*
- *book800k-900k.csv*
- *book900k-1000k.csv*
- *book1000k-1100k.csv*
- *book1100k-1200k.csv*
- *book1200k-1300k.csv*
- *book1300k-1400k.csv*
- *book1400k-1500k.csv*
- *book1500k-1600k.csv*
- *book1600k-1700k.csv*
- *book1700k-1800k.csv*
- *book1800k-1900k.csv*
- *book1900k-2000k.csv*
- *book2000k-3000k.csv*
- *book3000k-4000k.csv*
- *book4000k-5000k.csv*

Interactions data:

- *user_rating_1000_to_2000.csv*
- *user_rating_2000_to_3000.csv*

- *user_rating_3000_to_4000.csv*
- *user_rating_4000_to_5000.csv*
- *user_rating_5000_to_6000.csv*
- *user_rating_6000_to_11000.csv*

Modules

All the code is in the github link provided in the submission

config.py

Contains all the constants of the project execution.

transform_data/input_books.py

This script contains a class, BooksReader, which is in charge of building books dataset. This script reads all books csvs and concatenates them

transform_data/input_interactions.py

This script contains a class, InteractionsReader, which is in charge of building user-item interactions dataset. This script reads all books csvs and concatenates them

transform_data/books_authors.py

This class, BookAuthor, is developed to take the columns of books dataset that we need
BOOK_AUTHORS_COLS_OLD_NAMES: Columns that we need with original names
BOOK_AUTHORS_COLS: New columns names for the selected columns

transform_data/author_preparation.py

In this class, AuthorPreparation, we join books and authors. We compute here the biased rating based on alpha value

transform_data/rating.py

Here we build the Dataset to be ready to the surprise model

sklearn_estimator/train.py

Here we build the estimator to be ready for AWS Sagemaker

notebooks/Project Notebook.ipynb

I have the whole process in this notebook. Therefore, this process will be explained here.

1. The first step is to read the csvs. Remember that both book and rating dataset are splitted in several csvs as I already explained in Dataset and Inputs section.
    a. For the interactions dataset, I use transform_data/input_interactions.py script. The class constructor maps the ratings to numbers as is detailed in Basic Counts section. After that I call the InteractionsReader method, read_data. This method has an input path, defined in config.py, and a list of values that fills the string path in all the iterations. In every iteration, a csv is read with the pandas library and appended to the whole dataset. However, the last value of the list is not read since the path is built with the iteration value and the next one.
    b. I did the same with the book's dataset, using the BooksReader class in transform_data/input_books.py python script. However, there is one change here in the way of reading (that's why I did not use a parent class for both data readers), because in the first csv I did not have a character *k* that is present in the others.
2. The next step is filter the ratings with 0, since the meaning of those ratings is that a user has not rated any book of the dataset
3. After that, I made the Exploratory Data Analysis with basic counts and plot as I previously explained in Data Exploration and Exploratory Visualization sections
4. The next section is the models
    a. Making a first approach in order to test the algorithms previously presented. For both Matrix Factorization algorithms using scikit-surprise library I had to build a Dataset surprise object in order to use it in SVD algorithms. To do that I use a class, Rating (in transform_data/rating.py script), which prepares the data (Dataset surprise needs a Reader with rating scale and a dataset with only user, item and ratings) so that it is the algorithm input.
    b. For algorithm 2, I built a script with a class, AuthorPreparation, which transform the interactions dataset adding author bias as explained in Algorithms 2 section. Another difference here is the way of training this, because the real rating is not the algorithm input value. To do that I wrote this little piece of code.

```
algo = SVD(n_factors=n_factors, n_epochs=grid['n_epochs'][0], random_state=1)
kf = KFold(n_splits=5, random_state=1)
rmse_list = []
for uad, uar in zip(kf.split(user_authors_dat), kf.split(user_authors_real)):
    trainset, _ = uad
    _, testset = uar
    algo.fit(trainset)
    predictions = algo.test(testset)
    rmse_list.append(accuracy.rmse(predictions))
```

5. The next process is the Grid Search for both algorithms and check the test errors in cross validation process for all the Grid Search values tested as I will show in Refinement section

Refinement

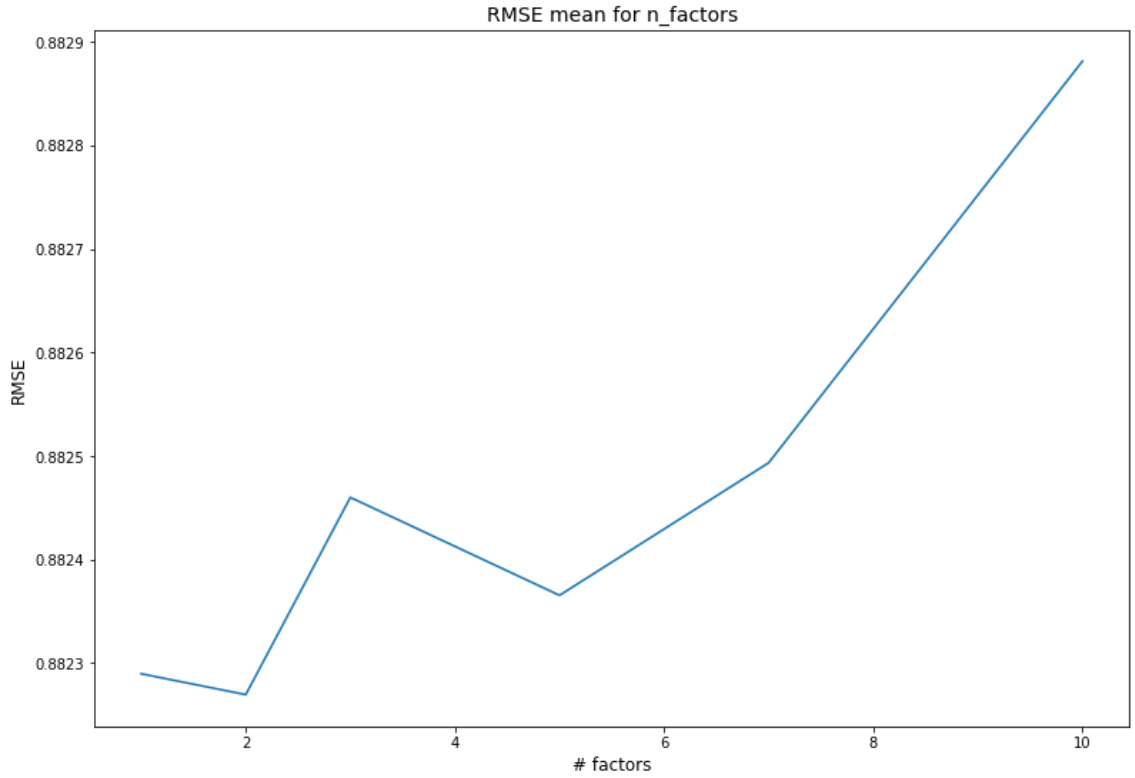
Algorithm 1.

After explaining algorithm 1 with the Matrix Factorization in Algorithms and Techniques section, the unique parameter to tune in this algorithm is M. M is the maximum rank of the SVD matrices. Therefore, the greater the value of M, the more hidden features for items and users are computed; however, the more complex the algorithm is and it is going to take more time to train and to predict.

In state of art, M is usually in the order of fourth root of the number of items. Nevertheless, this is an empirical result that may not be the best one. So I am going to test several values for M (the fourth root of the number of the items is around 18), from 5 to 30. The optimal value will be the one with the lower RMSE. And if there are some RMSE values in the same range, I will pick the lowest M, since it is the lowest energy solution.

I made a GridSearch with cross validation with 5 folds, using several values for n_factors (the unique parameter). The values used are 1, 2, 3, 5, 7 and 10

The results are the following:



The best value is for 2 factors with 0.8822 of RMSE error. It seems that if I had more data the factors could be higher, but it is a way to automatize the selection.

Algorithm 2

For this algorithm, the M parameter is the same as the first algorithm as I explained in the Algorithm 2 - Combined matrix with Matrix Factorization section. The second parameter, k, is the percentage of bias introduced by the author. This value must be between 0 and 1. Here, I am going to test a grid search with M from 5 to 30 and k from 0.25 to 1. Again, the optimal value will be the one with the lower RMSE, taking into account the lowest energy solution from M parameter.

I did the same here adding the parameter k. The best option is for # factors is 1 and alpha equal to 0.25 with a RMSE of 0.88031. However, as 1 could not get some features for users and items I prefer to use 3 factors since the RMSE is a little bit higher: 0.88037, but almost negligible.

	Alpha				
n_factors	**0**	**0,25**	**0,5**	**0,75**	**1**
1	0,88229	**0,88031**	0,88301	0,89055	0,90282
2	0,88227	0,88043	0,88309	0,89060	0,90287
3	0,88246	**0,88038**	0,88308	0,89062	0,90290
5	0,88237	0,88047	0,88316	0,89069	0,90296
7	0,88249	0,88053	0,88319	0,89070	0,90297

Table 2. Grid Search RMSE errors for Algorithm 2

Implementation

Taking into account all the results, the model selected is the second algorithm with 3 factors and k (alpha) equals to 0.25.

This is the model which I feel more comfortable to deploy in terms of RMSE and maintenance.

Justification

Taking into account my results and the metrics shown in the Benchmark section and the goal I set.

Algorithm	**MSE**	**RMSE**
ALS	3.73	1.9319
Deep Learning	1.7362	1.317
SVD (author bias)	**0.77**	**0,8803**

I can say that the goal has been reached and my ambition of getting a RMSE lower than 1 has also been reached with success.