Books Recommender System

Abstract

In today's world a customer is flooded with choices when it comes to buying or renting books under any genre. Customers waste a lot of time browsing through an ocean of books to find the ones that suit his/her taste and if even after browsing he could not find one, he/she might look for recommendations from others. Sometimes even after consulting others the customer loses interests and gives up.
This is where we come in, we will be building a recommender system that will help the customers choose their choice of books based on previous choices of the customer. Including recommendations in systems is an attractive bet. For the customers, it increases their experience and creates engagement. For the business, it generates more revenue.

Introduction:

On the Internet, where the choice of decisions is overwhelming, there is a need to channel, organize and effectively convey important data to mitigate the issue of data over-burden, which has made a possible issue to numerous Internet users.

Data recovery frameworks, for example, Google, DevilFinder and Altavista have incompletely tackled this issue however prioritization and personalization (where a framework maps accessible substance to client's inclinations and inclinations) of data were missing. This has expanded the interest in recommender frameworks like never previously.

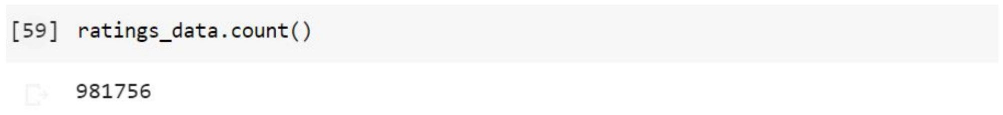
This is hence not surprising that during the most recent couple of years, with the ascent of YouTube, Amazon, Netflix and numerous other such web services, recommender frameworks have assumed increasingly more positions in our lives. From internet business (propose to purchasers' articles that could intrigue them) to online commercials (propose to clients the correct substance, coordinating their inclinations), recommender frameworks are today unavoidable in our everyday online excursions.

In this project we tried to tackle a similar problem of overburden of choices when selecting books and try to help users by designing some recommendation systems that would suggest books based on the book's popularity, book's content and based upon the user's previous choice and likes and dislikes. We used recommender systems that are too naive to very sophisticated personalized one and try to gauge each system to check their possible merits and demerits.

Dataset Description:

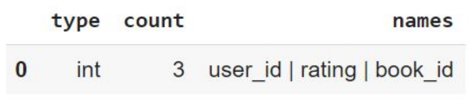
We used Amazon Dataset which contains 1 million ratings across 10,000 unique books. In most cases, there are at least 10 books rated by each user and the rating lies between 0 to 5.

The ratings data set provides a list of ratings that users have given to books. It includes 981756 records and 3 fields: *user_id, rating, book_id*.



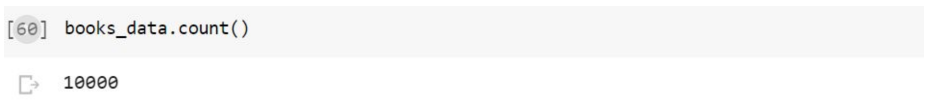
```
[59] ratings_data.count()

981756
```



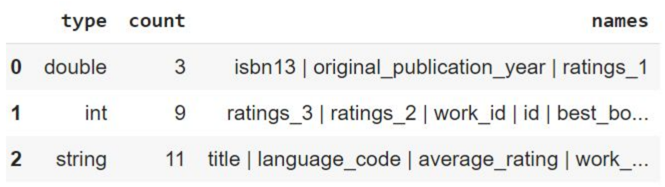
	type	count	names
0	int	3	user_id | rating | book_id

The books data set provides a list of books and their related details. It includes 10,000 records and columns: *'id', 'book_id', 'best_book_id', 'work_id', 'books_count', 'isbn', 'isbn13', 'authors', 'original_publication_year', 'original_title', 'title', 'language_code', 'average_rating', 'ratings_count', 'work_ratings_count', 'work_text_reviews_count', 'ratings_1', 'ratings_2', 'ratings_3', 'ratings_4', 'ratings_5'*.



```
[60] books_data.count()

10000
```

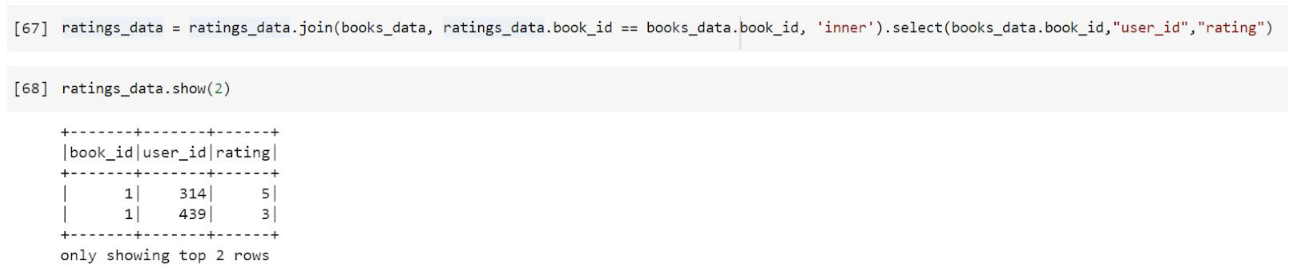


	type	count	names
0	double	3	isbn13 | original_publication_year | ratings_1
1	int	9	ratings_3 | ratings_2 | work_id | id | best_bo...
2	string	11	title | language_code | average_rating | work_...

Some especially useful columns we used for our analysis were rating, *user_id*, *book_id*, rating, *original_title*, *title*, and *author*. The column *user_id* is the identification number of various active users. *book_id* is the book's identification number. *rating* is the rating each user provided to each book. *original_title* is the title of the book and the author is the author of the book.

Data Preprocessing and Feature Extraction:

First before starting the analysis we wanted to make sure the book_id present in the ratings_data also exists in books_data. Otherwise, the collaborative filtering gives recommendations whose book title cannot be found.



```
[67] ratings_data = ratings_data.join(books_data, ratings_data.book_id == books_data.book_id, 'inner').select(books_data.book_id,"user_id","rating")

[68] ratings_data.show(2)

+-------+-------+------+
|book_id|user_id|rating|
+-------+-------+------+
|      1|    314|     5|
|      1|    439|     3|
+-------+-------+------+
only showing top 2 rows
```

We even processed the text in the books dataset in a manner by removing stop words such as the, and of and punctuations such as "," and "." to help us while tokenizing and extracting features. We used the regexTokenizer present in pyspark.ml.feature library to help us to perform this operation.

```
[69] concat_udf = F.udf(lambda cols: " ".join([x if x is not None else "*" for x in cols]), StringType())
     books_datacont = books_data.withColumn("desc", concat_udf(F.array("authors","title")))
     books_datacont = books_datacont.withColumn("desc", F.regexp_replace("desc", "[/(,)]", " "))
```

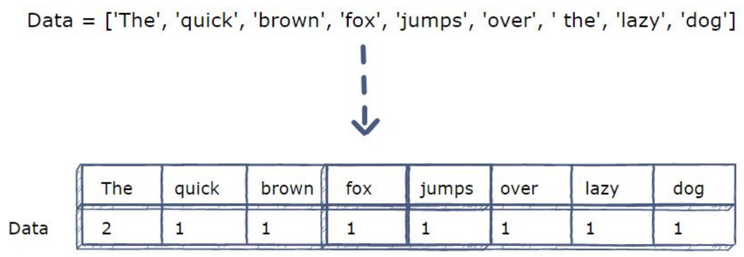
```
regexTokenizer = RegexTokenizer(inputCol="desc", outputCol="tokens")
regexTokenized = regexTokenizer.transform(books_datacont)
```

To effectively make recommendations we wanted to use a better feature extractor and thus to address this, we used scikit-learn library as it provides utilities for the most common ways to extract features from text content. Below are the utilities incorporated:

Countvectorizer:

CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It likewise empowers the pre-processing of text data before producing the vector representation.

This functionality makes it a highly flexible feature representation module for text.


Data = ['The', 'quick', 'brown', 'fox', 'jumps', 'over', ' the', 'lazy', 'dog']
The
quick
brown
fox
jumps
over
lazy
dog
Data
2
1
1
1
1
1
1
1

```
[ ] """
    Tried to find the content based on countvectorizer
    """
    from sklearn.feature_extraction.text import CountVectorizer
    cv = CountVectorizer()
    count_matrix = cv.fit_transform(books_datacont['desc'])
```

Tf–idf term weighting:

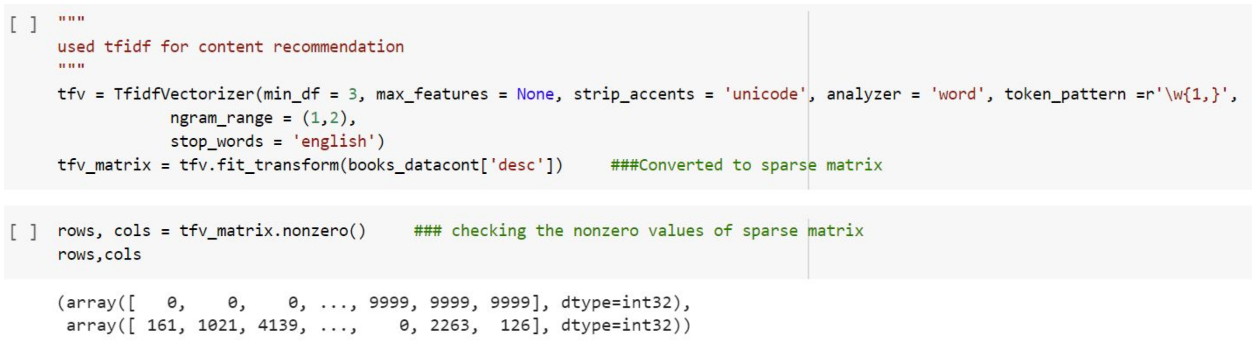
Due to the presence of functional words or perhaps other common word tokens, the information carried by them becomes minimal in regards to actual information. Such frequent terms will thus overshadow the rarer yet focused terms while making recommendations by a recommender system.

The Term-Frequency Inverse Document frequency - which is intended to reflect how important a word is to a document in a collection or corpus - is computed as:

$$tf(t) * idf(t) = tf(t) * (\log log \left(\frac{1+n}{1+df(t)}\right) + 1)$$

Where “n” is the total number of documents in the document set, and df(t) is the number of documents in the document set that contain term t. Note, we also incorporated bigram and trigram tf-idf so that we capture syntactic dependencies too in our feature vector.

The below screenshot shows the implementation of tfidf in our code using tfidfVectorizer present in the sklearn.feature_extraction.text.



```
[ ] """
    used tfidf for content recommendation
    """
    tfv = TfidfVectorizer(min_df = 3, max_features = None, strip_accents = 'unicode', analyzer = 'word', token_pattern =r'\w{1,}',
                ngram_range = (1,2),
                stop_words = 'english')
    tfv_matrix = tfv.fit_transform(books_datacont['desc'])     ###Converted to sparse matrix
```

```
[ ] rows, cols = tfv_matrix.nonzero()     ### checking the nonzero values of sparse matrix
    rows,cols

    (array([   0,    0,    0, ..., 9999, 9999, 9999], dtype=int32),
     array([ 161, 1021, 4139, ...,    0, 2263,  126], dtype=int32))
```

Recommender systems techniques incorporated:

1. Popularity based recommendation:

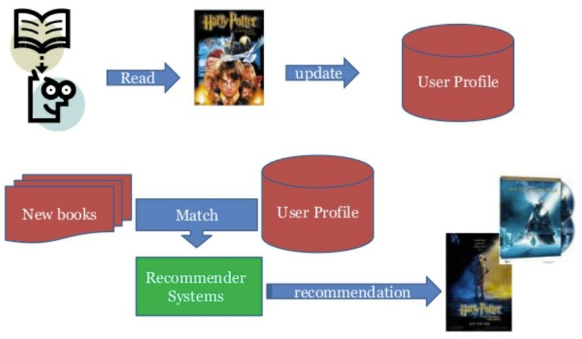
- Works on the principle of popularity or latest trend. These systems check about the product or book which are in trend or are most popular among the users and directly recommend those.
  For example, if any product which is usually read by every user and is rated high then there are chances that it may suggest that item to the user who just signed up.

- This is the most naïve method of recommendation.

Merit: It is especially useful if a new user arrives and wants some recommendation on the books.

Demerit: The problems with popularity-based recommendation systems is that the personalization is not available with this method i.e., even though you know the behavior of the user you cannot recommend items accordingly.

2. Content Based recommendation:



Content-based filtering methods basically are based on a description of the item and a profile of the user's preferences.
Content-based filtering uses item features to recommend on similar books based on similar attributes like genre, author, publication etc. So, the idea in content-based filtering is to tag products using certain keywords, understand what the user likes, look up those keywords in the database and recommend different products with the same attributes.
These methods are suited in situations where there is known data on an item (name, location, description, etc.), but not on the user.

In the model-building stage, the system first finds the similarity between all pairs of items, then it uses the most similar items to a user's already-rated items to generate a list of recommendations in the recommendation stage.
For example, if someone watches Harry Potter and the Philosopher's stone, the system may recommend Harry Potter and the Prisoner of Azkaban based on similarity.

Usually, the features for finding the similarity between the items is extracted using the description of the item. Here we used the **countVectorizer** and **tf-idf** for this task about which we have discussed in the data preprocessing and feature extraction section.

The similarity is thus measured using techniques such as **cosine similarity**.
Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

The values here range from -1 to 1, where -1 is perfectly dissimilar and 1 is perfectly similar.

Merit:
- User independence: Content-based method only have to analyze the items and user profile for recommendation
- No cold start: new items can be suggested before being rated by a substantial number of users.

Demerit:
- Limited content analysis: if the content does not contain enough information to discriminate the items precisely, the recommendation will be not precisely at the end
- The recommendations provided are not that personalized. This filtering helps to get recommendations based on the content of the item but doesn't take into account the user's likes or dislikes.

3. Collaborative based filtering:

Collaborative filtering methods are based on collecting and analyzing a large amount of information on user behaviors, activities or preferences and predicting what users will like based on their similarity to other users.

The fundamental assumption in these kinds of filtering techniques is that similar user preferences over the items could be exploited to recommend those items to a user who has not seen or used it before.

Since sparsity and scalability are the two biggest challenges for standard Collaborative Filtering method, there comes a more advanced method that decomposes the original sparse matrix to low-dimensional matrices with latent factors/features and less sparsity. That is **Matrix Factorization**.

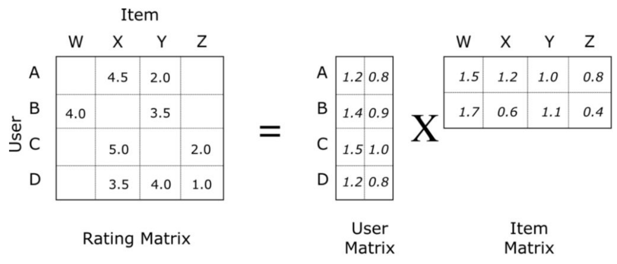


Figure 1: User-item matrix data matrix factorization

$$r_{ui} \approx q_i^T p_u$$

Where a rating r_{ui} can be estimated by dot product of user vector p_u and item vector q_i.

In collaborative filtering, **matrix factorization** algorithms work by decomposing the user-item interaction matrix into the product of **two lower dimensionality rectangular matrices**.

- User-matrix where rows are represented by users and columns are latent factors.
- Item-matrix where rows are represented by latent factors and columns are items.

Latent factors are the features in the lower dimension latent space projected from the user-item interaction matrix. Matrix factorization is to use latent factors to represent user preferences or movie topics in a much lower dimension space.

Alternating Least Square (ALS) for collaborative filtering:
Alternating Least Square (ALS) is also a matrix factorization algorithm. ALS is implemented in Apache Spark ML and built for a larger-scale collaborative filtering problem. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

Loss function: $\min_{p,q,b_u,b_i} \sum (r_{ui} - (p_u^T q_i + \mu + b_u + b_i))^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$

Regularization: ALS uses L2 regularization

Training routine: ALS minimizes two loss functions alternatively; It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix.

Most important hyper-params in Alternating Least Square (ALS):

- maxIter: the maximum number of iterations to run (defaults to 10)
- rank: the number of latent factors in the model (defaults to 10)
- regParam: the regularization parameter in ALS (defaults to 1.0)

Merit:

- Not required to understand item content: The content of the items does not necessarily tell the whole story, such as movie type/genre, and so on.
- More personalized recommendations based on user likes and dislikes.

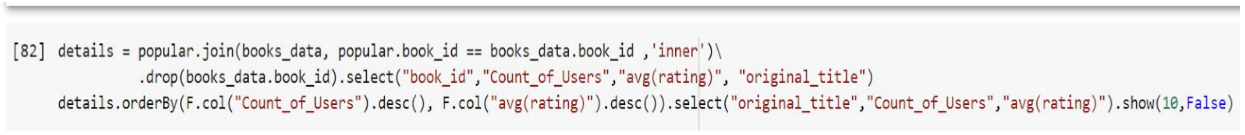
- Captures the change in user interests over time: Focusing solely on content does not provide any flexibility on the user's perspective and their preferences.

Demerit:
The cold start problem. The item cold-start problem refers to when items added to the catalogue have either none or very little interactions. This constitutes a problem mainly for collaborative filtering algorithms because they rely on the item's interactions to make recommendations.

Results:

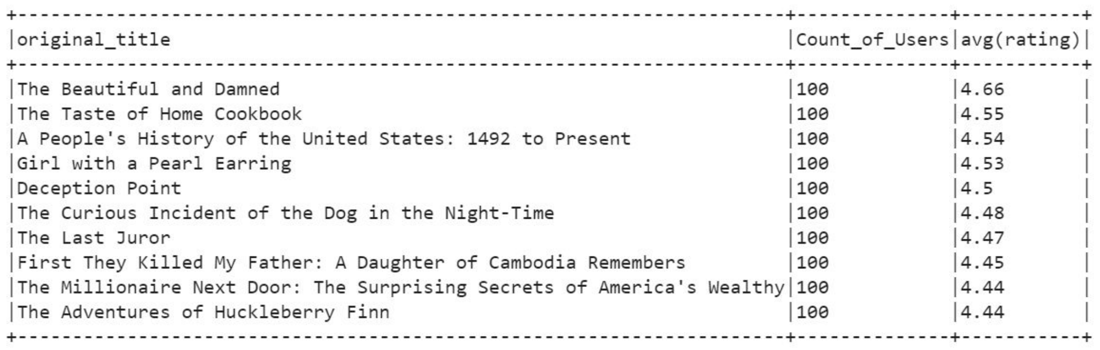
1. Popularity based recommendation system:

For each book, we found out the number of users who read the book and the average rating they gave, and we sorted them in descending order.



```
[82] details = popular.join(books_data, popular.book_id == books_data.book_id ,'inner')\
              .drop(books_data.book_id).select("book_id","Count_of_Users","avg(rating)", "original_title")
     details.orderBy(F.col("Count_of_Users").desc(), F.col("avg(rating)").desc()).select("original_title","Count_of_Users","avg(rating)").show(10,False)
```

We join the books dataset to the results obtained to get the details of the books.



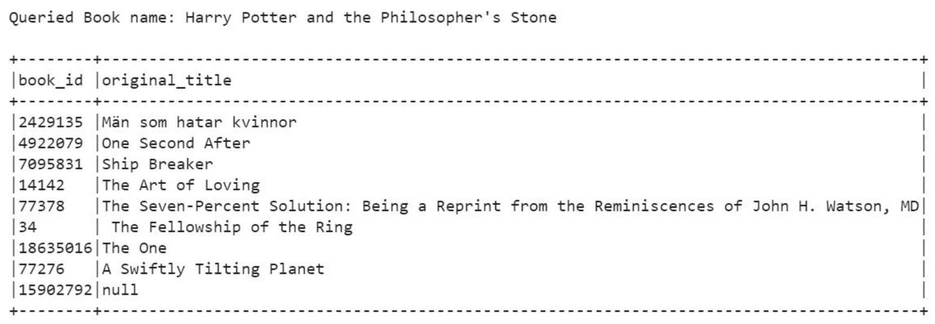
```
+--------------------------------------------------------------------+--------------+-----------+
|original_title                                                      |Count_of_Users|avg(rating)|
+--------------------------------------------------------------------+--------------+-----------+
|The Beautiful and Damned                                            |100           |4.66       |
|The Taste of Home Cookbook                                          |100           |4.55       |
|A People's History of the United States: 1492 to Present           |100           |4.54       |
|Girl with a Pearl Earring                                           |100           |4.53       |
|Deception Point                                                      |100           |4.5        |
|The Curious Incident of the Dog in the Night-Time                   |100           |4.48       |
|The Last Juror                                                      |100           |4.47       |
|First They Killed My Father: A Daughter of Cambodia Remembers       |100           |4.45       |
|The Millionaire Next Door: The Surprising Secrets of America's Wealthy|100         |4.44       |
|The Adventures of Huckleberry Finn                                  |100           |4.44       |
+--------------------------------------------------------------------+--------------+-----------+
```

So, it returns the list of the popular books for the user but since it is a popularity-based recommendation system the recommendation for the users will not be affected.

2. Content based recommender system:

We used two methods to find the content-based recommendations.

- tf-idf feature extraction
- cosine similarity for item similarities measurement.

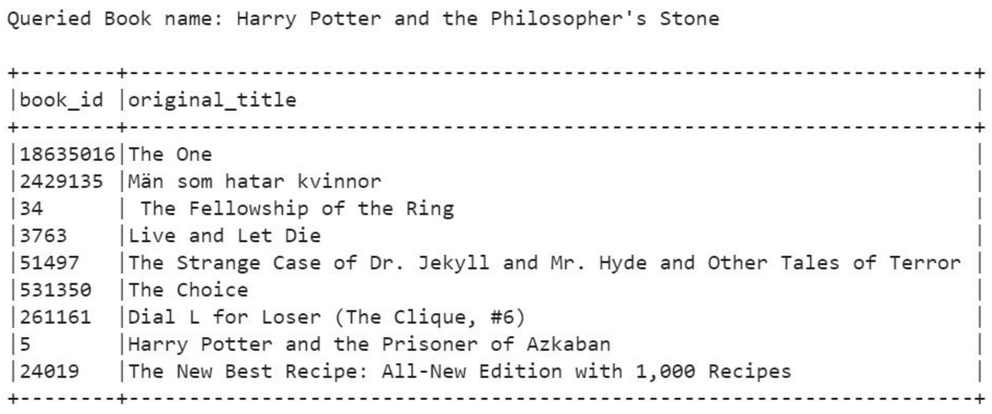


```
Queried Book name: Harry Potter and the Philosopher's Stone

+--------+-------------------------------------------------------------------------------+
|book_id |original_title                                                                 |
+--------+-------------------------------------------------------------------------------+
|2429135 |Män som hatar kvinnor                                                          |
|4922079 |One Second After                                                               |
|7095831 |Ship Breaker                                                                   |
|14142   |The Art of Loving                                                              |
|77378   |The Seven-Percent Solution: Being a Reprint from the Reminiscences of John H. Watson, MD|
|34      | The Fellowship of the Ring                                                    |
|18635016|The One                                                                        |
|77276   |A Swiftly Tilting Planet                                                       |
|15902792|null                                                                           |
+--------+-------------------------------------------------------------------------------+
```

Figure 2: The queried book name and the corresponding recommendations

- Countvectorizer for feature extraction
- cosine similarity for item similarities measurement.



```
Queried Book name: Harry Potter and the Philosopher's Stone

+--------+--------------------------------------------------------------------+
|book_id |original_title                                                      |
+--------+--------------------------------------------------------------------+
|18635016|The One                                                             |
|2429135 |Män som hatar kvinnor                                               |
|34      | The Fellowship of the Ring                                         |
|3763    |Live and Let Die                                                    |
|51497   |The Strange Case of Dr. Jekyll and Mr. Hyde and Other Tales of Terror |
|531350  |The Choice                                                          |
|261161  |Dial L for Loser (The Clique, #6)                                   |
|5       |Harry Potter and the Prisoner of Azkaban                            |
|24019   |The New Best Recipe: All-New Edition with 1,000 Recipes             |
+--------+--------------------------------------------------------------------+
```

Figure 3: The queried book name and the corresponding recommendations

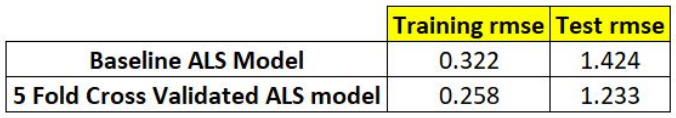
3. Collaborative filtering using ALS Model:

Baseline Model parameters:

- maxIter = 5
- regParam = 0.1
- rank = 10
- coldStartStrategy = “drop”

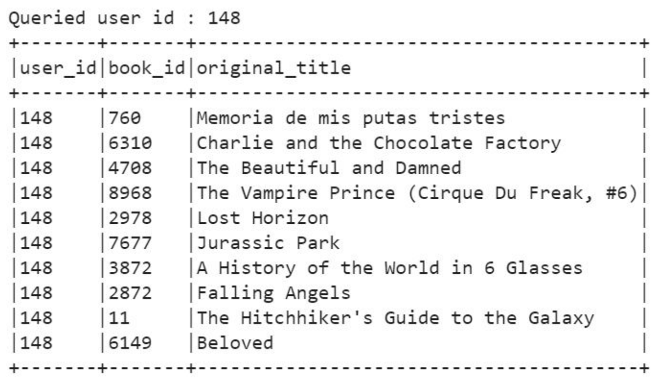
5-Fold Cross Validated Model parameters:

- maxIter = 5
- regParam = [0.1, 1, 10]
- rank = [10,12]
- coldStartStrategy = “drop”
- numFolds = 5



	Training rmse	Test rmse
Baseline ALS Model	0.322	1.424
5 Fold Cross Validated ALS model	0.258	1.233

Since 5-Fold Cross Validated ALS Model has better test rmse we shared its results:



```
Queried user id : 148
+-------+-------+--------------------------------------+
|user_id|book_id|original_title                        |
+-------+-------+--------------------------------------+
|148    |760    |Memoria de mis putas tristes          |
|148    |6310   |Charlie and the Chocolate Factory     |
|148    |4708   |The Beautiful and Damned              |
|148    |8968   |The Vampire Prince (Cirque Du Freak, #6)|
|148    |2978   |Lost Horizon                          |
|148    |7677   |Jurassic Park                         |
|148    |3872   |A History of the World in 6 Glasses   |
|148    |2872   |Falling Angels                        |
|148    |11     |The Hitchhiker's Guide to the Galaxy  |
|148    |6149   |Beloved                               |
+-------+-------+--------------------------------------+
```

Figure 4: Queried userid and the corresponding book_id and title recommendations

Future scope:

- A **hybrid system using both content-based and collaborative filtering**. This would mean incorporating information about the individual items, including features like subject matter, page size, color vs. black and white, creator name, text to image ratio, cost, etc.

- **Multi-armed bandit algorithms:**
  They can recommend products with the highest expected value while still exploring other products. They do not suffer from the cold-start problem and therefore do not require customer preferences or information about products.

  In addition, multi-armed bandits consider the limitations of how much data you have as well as the cost of gathering data (the opportunity cost of sub-optimal recommendations).
  The name multi-armed bandit comes from the one-armed bandit, which is a slot machine. In the multi-armed bandit experiment, there are multiple slot machines with different probabilities of payout and with potentially different amounts.

Team contribution:

The team members have contributed equally to every task namely, brainstorming, data gathering and preparation, building the recommender systems along with the documentation of the process and presentation of the process.

References:

1. https://www.educative.io/edpresso/countvectorizer-in-python
2. https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/
3. https://hackernoon.com/popularity-based-song-recommendation-system-without-any-library-in-python-12a4fbfd825e
4. https://en.wikipedia.org/wiki/Recommender_system#Content-based_filtering
5. https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1
6. https://www.offerzen.com/blog/how-to-build-a-product-recommender-using-multi-armed-bandit-algorithms