Machine Learning Engineer Nanodegree

Capstone Project

Mariusz Kowalewski
September 14, 2019

I. Definition

Project Overview

Goodreads is a social cataloging website that allows individuals to search its database of books, quotes and reviews. Users can sign up and register books to generate library catalogs and reading lists. They can also create their own groups of book suggestions, surveys, polls, blogs, and discussions.

Goodreads is the world's largest site for book readers. Being a bookish myself, I think it would be very interesting to explore differences in people's tastes and find recommendations for individual users.

Problem Statement

According to some statistics people read only 12 books per year on average and this figure is probably inflated. Taking into consideration the number of books available in shops and ebooks in the Internet, it is very hard to find one that will be consistent with reader's taste. Moreover, lists of bestsellers are not neceserilly useful for everybody. Having a big database of books, reviews and ratings, it's possible to build an engine able to recommend readers similar books adjusted to their preferences.

The solution to the problem will require the following steps:

1. Data collection - main datasets in CSV format will be obtained from Kaggle website. Additionally I'll be using external Google Books API to get more details about books.
2. Data exploration and preprocessing:
    a. analyze the achieved datasets
    b. select the most relevant features for recommendation system
    c. normalize features in necessary
    d. make datasets consistent
    e. choose the good subset of data to visualize
3. Choose a clustering algorithm and test it on smaller subsets of data.
4. Evaluate chosen technique with defined metrics.

5. Implement the chosen algorithm using preprocessed dataset.
6. Refine the model if necessary, for example by adjusting some parameters.
7. Present the results - the expected solution will be the list of 15 recommended book for a specific user.

Metrics

In order to evaluate K-means clustering and find the right number of clusters I'll be using *Silhouette score*[1] and *Elbow method*[2].

The **silhouette score** is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

The Silhouette score ***s*** for a single sample is then given as:

$$s = \frac{b - a}{max(a, b)}$$

where:

- **a**: The mean distance between a sample and all other points in the same class
- **b**: The mean distance between a sample and all other points in the next nearest cluster

The **elbow method** works by plotting the ascending values of *k* versus the total error calculated using that *k*. One way to calculate the error is *squared error*. The idea of the elbow criterion method is to choose the *k* (number of clusters) at which the SSE decreases abruptly. The SSE is defined as the sum of the squared distance between each member of the cluster and its centroid. The formula is following:

$$SSE = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Say we are calculating the error for *k* = 2. We'd have two clusters each having one "centroid" point. For each point in our dataset, we'd subtract its coordinates from the centroid of the cluster it belongs to. We then square the result of that subtraction to get rid of the negative values, and

[1] https://en.wikipedia.org/wiki/Silhouette_(clustering)
[2] https://en.wikipedia.org/wiki/Elbow_method_(clustering)

sum the values. This would leave us with an error value for each point. If we sum these error values, we'd get the total error for all points when $k = 2$.

II. Analysis

Data Exploration

I'll be using two datasets available on Kaggle website, **books.csv**[3] and **ratings.csv**[4]. Both were obtained using Goodreads API[5] and provide details about books and ratings with corresponding book's identification number. The files have the following structures:

books.csv:

- book_id - a unique identification number for each book
- title - the name under which the book was published
- authors - names of the authors of the book, multiple authors are delimited with ;
- average_rating - the average rating of the book received in total
- isbn - another unique number to identify the book, the International Standard Book Number
- isbn13 - a 13-digit ISBN to identify the book, instead of the standard 11-digit ISBN
- language_code - helps understand what is the primary language of the book, for instance, eng is standard for English
- # num_pages - number of pages the book contains
- ratings_count - total number of ratings the book received
- text_reviews_count - total number of written text reviews the book received

ratings.csv:

- user_id - a unique identification number for each user
- book_id - a unique Identification number for each book
- rating - a rating of a book with range 1-5 (1 is the very bad and 5 is very good)

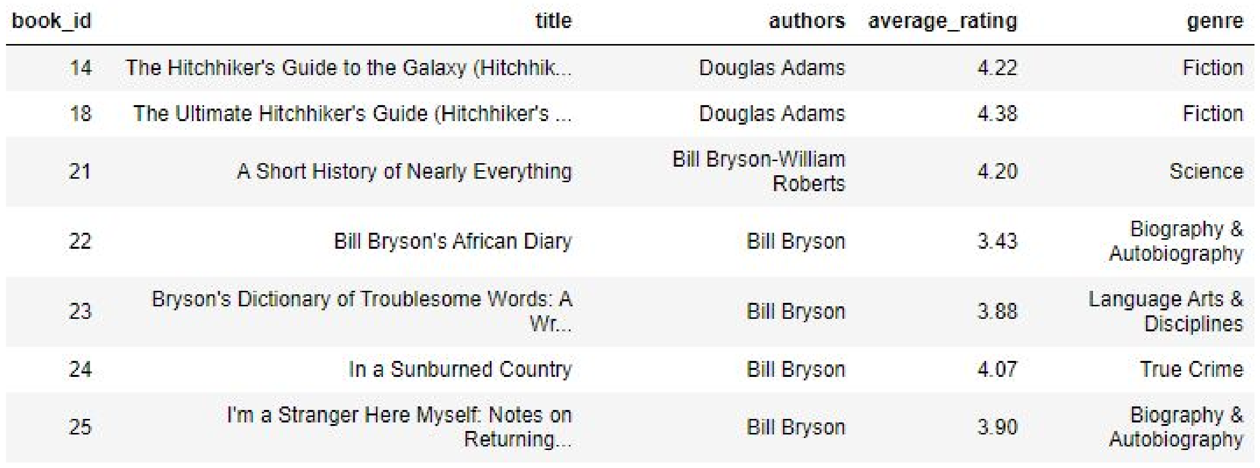
In order to categorize books by genres I wanted to use Goodreads API to create additional dataset, however it turned out that it's not directly available there. According to Goodreads' forum[6], books don't have assigned genres. Instead each user is able to add a book to specific shelves or tags created by themselves. In result there are tens of thousands shelves with similar names which don't categorize books very well.

[3] Source: https://www.kaggle.com/jealousleopard/goodreadsbooks
[4] Source: https://www.kaggle.com/zygmunt/goodbooks-10k
[5] https://www.goodreads.com/api
[6] https://goodreads.com/topic/show/19317219-getting-the-genre-of-a-book

After getting familiar with Goodreads API and exploring available data I decided to use Google Books API[7] to obtain genres using books' ISBN numbers. In turn, I managed to add *genre* column to **books.csv** file categorizing each book. In order to have access to the original file I created a new one **books_genres.csv** with added column.

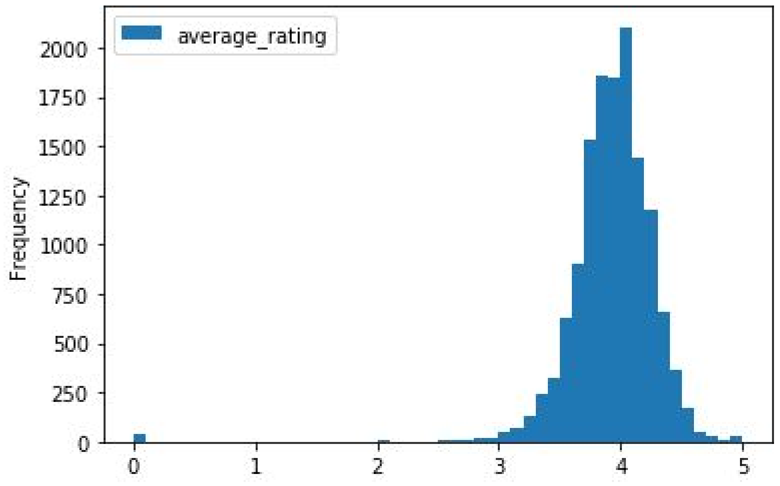
Exploratory Visualization

The **books_genres.csv** dataset contains details about 13,719 books, however for not all of them Google Books API was able to find a genre. Hence, I decided to use 10,007 records with assigned genres. To further analysis I'll be using only the below columns:



book_id	title	authors	average_rating	genre
14	The Hitchhiker's Guide to the Galaxy (Hitchhik...	Douglas Adams	4.22	Fiction
18	The Ultimate Hitchhiker's Guide (Hitchhiker's ...	Douglas Adams	4.38	Fiction
21	A Short History of Nearly Everything	Bill Bryson-William Roberts	4.20	Science
22	Bill Bryson's African Diary	Bill Bryson	3.43	Biography & Autobiography
23	Bryson's Dictionary of Troublesome Words: A Wr...	Bill Bryson	3.88	Language Arts & Disciplines
24	In a Sunburned Country	Bill Bryson	4.07	True Crime
25	I'm a Stranger Here Myself: Notes on Returning...	Bill Bryson	3.90	Biography & Autobiography

Every book has an average ratings withing 1-5 range. The significant part of all ratings are above 3.5 and much less are below 3:



To be exact, the dataset cointains:

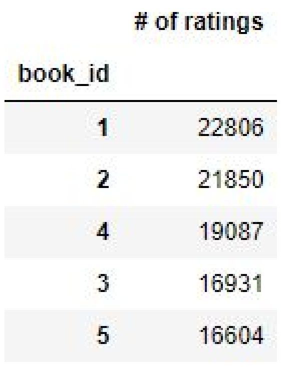
- 32 books with ratings higher than 4.7

[7] https://developers.google.com/books/

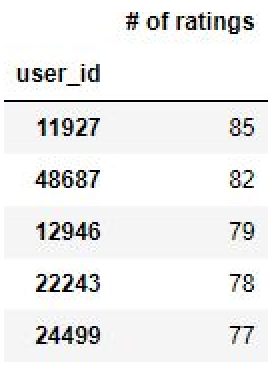
- 4,264 books with ratings higher than 4
- 9,307 books with ratings higher than 3.5
- 63 books with ratings lower than 3

Most of the books have on average quite good ratings around 4. The reason for that might be the fact that the dataset contains most popular books. Also, people usually base on friends' recommendations and that's why they leave rather positive feedback.

The **ratings.csv** dataset contains 5,976,479 ratings given by all users, however only 1,911,018 refer to books with assigned genres. Ratings per book range from 66 to 22806 and ratings per user range from 2 to 85. Here are the most frequently rated books and users with the highest number of ratings:

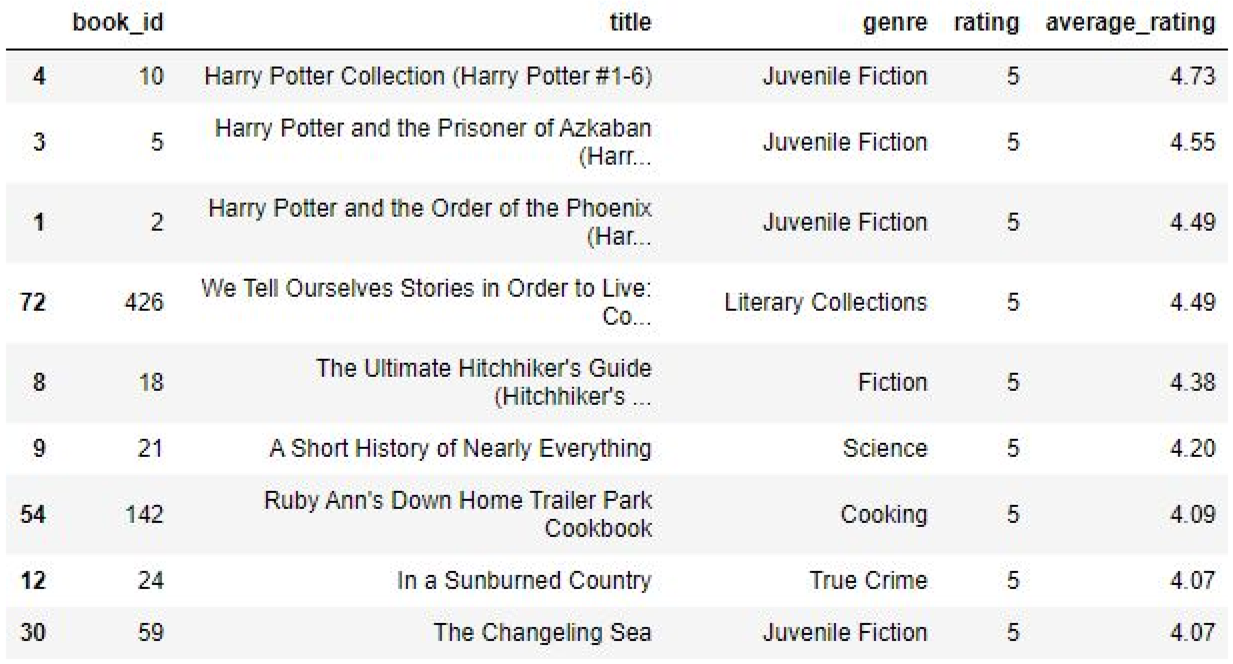


	# of ratings
book_id	
1	22806
2	21850
4	19087
3	16931
5	16604



	# of ratings
user_id	
11927	85
48687	82
12946	79
22243	78
24499	77

Finally, let's take look at books rated by user with most number of ratings (11927):



	book_id	title	genre	rating	average_rating
4	10	Harry Potter Collection (Harry Potter #1-6)	Juvenile Fiction	5	4.73
3	5	Harry Potter and the Prisoner of Azkaban (Harr...	Juvenile Fiction	5	4.55
1	2	Harry Potter and the Order of the Phoenix (Har...	Juvenile Fiction	5	4.49
72	426	We Tell Ourselves Stories in Order to Live: Co...	Literary Collections	5	4.49
8	18	The Ultimate Hitchhiker's Guide (Hitchhiker's ...	Fiction	5	4.38
9	21	A Short History of Nearly Everything	Science	5	4.20
54	142	Ruby Ann's Down Home Trailer Park Cookbook	Cooking	5	4.09
12	24	In a Sunburned Country	True Crime	5	4.07
30	59	The Changeling Sea	Juvenile Fiction	5	4.07

Algorithms and Techniques

There are two main techniques[8] used for building recommendation systems: content-based and collaborative filtering.

The main idea of **content-based** approach is to recommend products which are similar to the ones that a user has liked or bought in the past. Here the features of users and products are important. For book recommendations we use features like authors, number of pages, genre, reviews, etc. in order to create a profile of each user and each book. For example, if a person liked the book "Harry Potter and the Half-Blood Prince", then this algorithm will recommend books that are in the same genre.

The major drawback of this algorithm is that it is limited to recommending items in the same type. So if a user read or liked only fiction books in the past, the system will never recommend books from other genres.

Collaborative filtering is based on assumption that if user 1 and user 2 liked the same book, user 1 is more likely to enjoy a book that user 2 liked than a book rated by a random user. The algorithm uses "user behaviour" for recommending items. This is one of the most commonly used algorithms in the field because it doesn't depend on any additional information.

There are different types of collaborating filtering techniques. One of them is **User-User collaborative filtering[9]**. This algorithm first finds the similarity score between users. Based on this score, it selects the most similar users and recommends books which these similar users have liked or rated previosly.

In terms of recommendation system for books, the algorithm finds the similarity between each user based on the ratings they have previously given to different books. The prediction of an item for a user *u* is calculated by computing the weighted sum of the user ratings given by other users to an item .

The prediction $P_{u,i}$ is given by:

$$P_{u,i} = \frac{\Sigma_v (r_{v,i} * s_{u,v})}{\Sigma_v \, s_{u,v}}$$

where:

- $P_{u,i}$ is the prediction of an item

[8] https://towardsdatascience.com/recommendation-systems-models-and-evaluation-84944a84fb8e
[9] https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/

- $r_{v,i}$ is the rating given by a user *v* to a book *i*
- $s_{u,v}$ is the similarity between users

Now, we have the ratings for users in profile vector and based on that we have to predict the ratings for other users. Following steps are followed to do so:

1. For predictions we need the similarity between the user *u* and *v*.
2. First we find the items rated by both the users and based on the ratings, correlation between the users is calculated.
3. The predictions can be calculated using the similarity values. This algorithm, first of all calculates the similarity between each user and then based on each similarity calculates the predictions. Users having higher correlation will tend to be similar.
4. Based on these prediction values, recommendations are made.

This the technique that will be used in the project.

Benchmark

For a benchmark model I'll be using the *Kneedle algorithm* proposed by Ville Satopaa, Jeannie Albrecht, David Irwin and Barath Raghavan described in *Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior*[10] report, for determining the optimal number of clusters which provides a satisfactory trade-off between selected number of clusters and classification accuracy.

Using this algorithm I'll be able to compare the results of two other metrics, *Silhouette score* and *Elbow method*.

III. Methodology

Data Preprocessing

In order to use K-means clustering algorithm I needed to assign genres for each book in **books.csv** dataset. I found that Goodreads API doesn't allow to get such data as books are assigned to shelves which are kind of tags defined by users. I've done some research looking for other APIs enabling to get genre data and I came across Google Books API. Having ISBN numbers in the dataset I was able to obtain details about books including genres. As a result I managed to assign new feature to 10,007 out of 13,719 books in the dataset. Dataset with new feature has been saved as **books_genres.csv** file.

[10] https://raghavan.usc.edu//papers/kneedle-simplex11.pdf

Because books_genres.csv and ratings.csv datasets come from different sources I needed to make them consistent. The **ratings.csv** dataset contains 5,976,479 ratings given by all users, however only 1,911,018 refer to books with assigned genres. I decided to remove these that do not match.

The **books_genres.csv** contains the following features: *book_id*, *title*, *authors*, *average_rating*, *isbn*, *isbn13*, *language_code*, *#num_pages*, *ratings_count*, *text_reviews_count, genre*. Not all of them were needed in analysis and some of them were dropped. *Isbn* and *isbn13* have been already used to obtain *genre* feature and all I needed to group users according to their preferences were the following features: *book_id*, *title*, *average_rating*, *genre*. The finally selected features didn't require transformation.

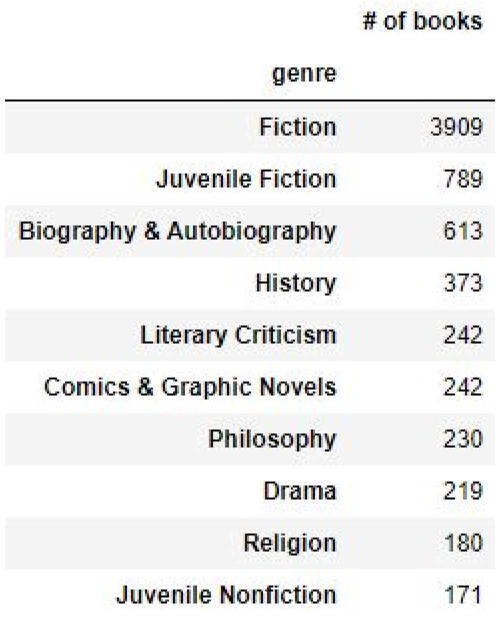
Implementation

One of possible solutions is to use a K-means algorithm to group readers according to their ratings which in result will return clusters of readers with similar ratings and generally similar preferences in books. Basing on this, we can calculate the ratings for certain books that weren't read by averaging ratings of other readers in the same cluster.

At the beginning of implementation of recommendation system let's check if K-means algorithm would work fine with our dataset. In order to do it I will try to compare K-means clustering on chosen subset of data.

Test K-means algorithm

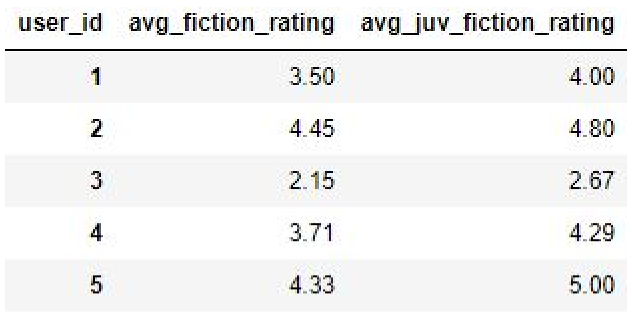
Let's explore genres in **books_genres.csv** dataset. Here are the most popular genres with number of assigned books:



genre	# of books
Fiction	3909
Juvenile Fiction	789
Biography & Autobiography	613
History	373
Literary Criticism	242
Comics & Graphic Novels	242
Philosophy	230
Drama	219
Religion	180
Juvenile Nonfiction	171

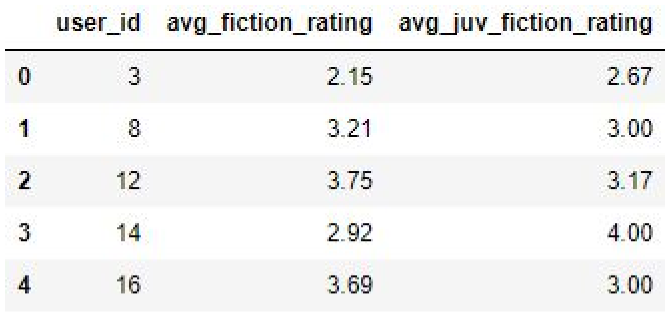
The three most popular genres are: *Fiction*, *Juvenile Fiction* and *Biography & Autobiography*.

Let’s take subset of users interested in *Fiction* and *Juvenile Fiction* genres and see what their preffered genres are.



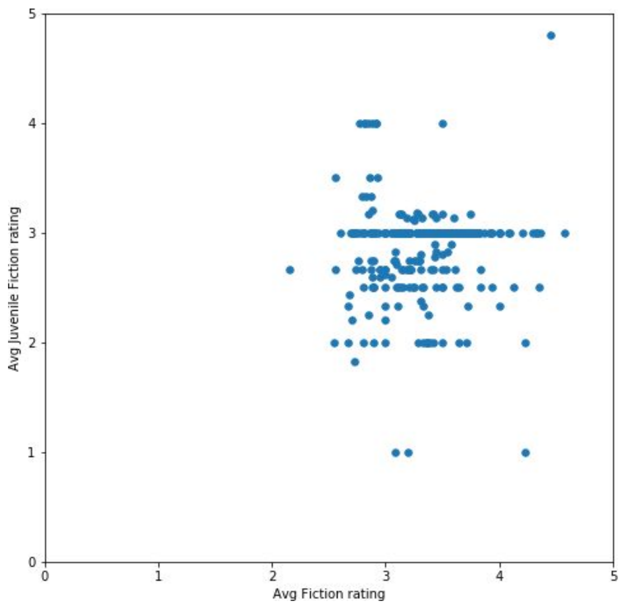
user_id	avg_fiction_rating	avg_juv_fiction_rating
1	3.50	4.00
2	4.45	4.80
3	2.15	2.67
4	3.71	4.29
5	4.33	5.00

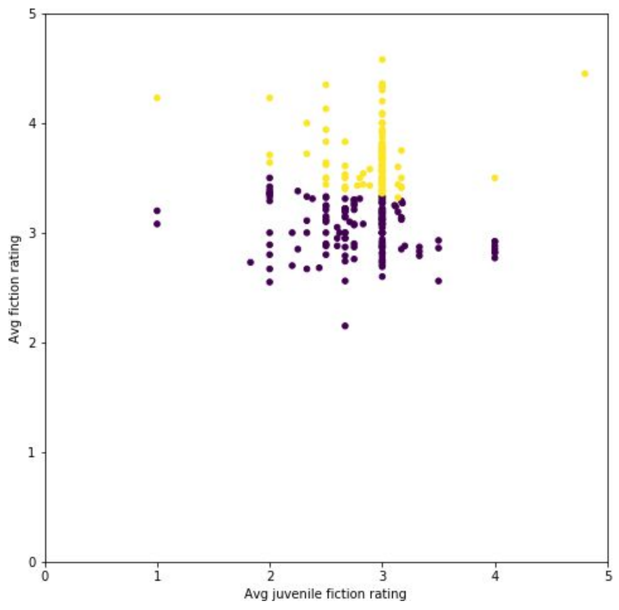
Now let’s bias our dataset by removing readers who like both *Fiction* and *Juvenile Fiction* so clusters will define users as liking one genre more than the other.



	user_id	avg_fiction_rating	avg_juv_fiction_rating
0	3	2.15	2.67
1	8	3.21	3.00
2	12	3.75	3.17
3	14	2.92	4.00
4	16	3.69	3.00

In result we have 302 users with their average ratings of *Fiction* and *Juvenile Fiction* books they’ve read. We can see some bias in the left plot:

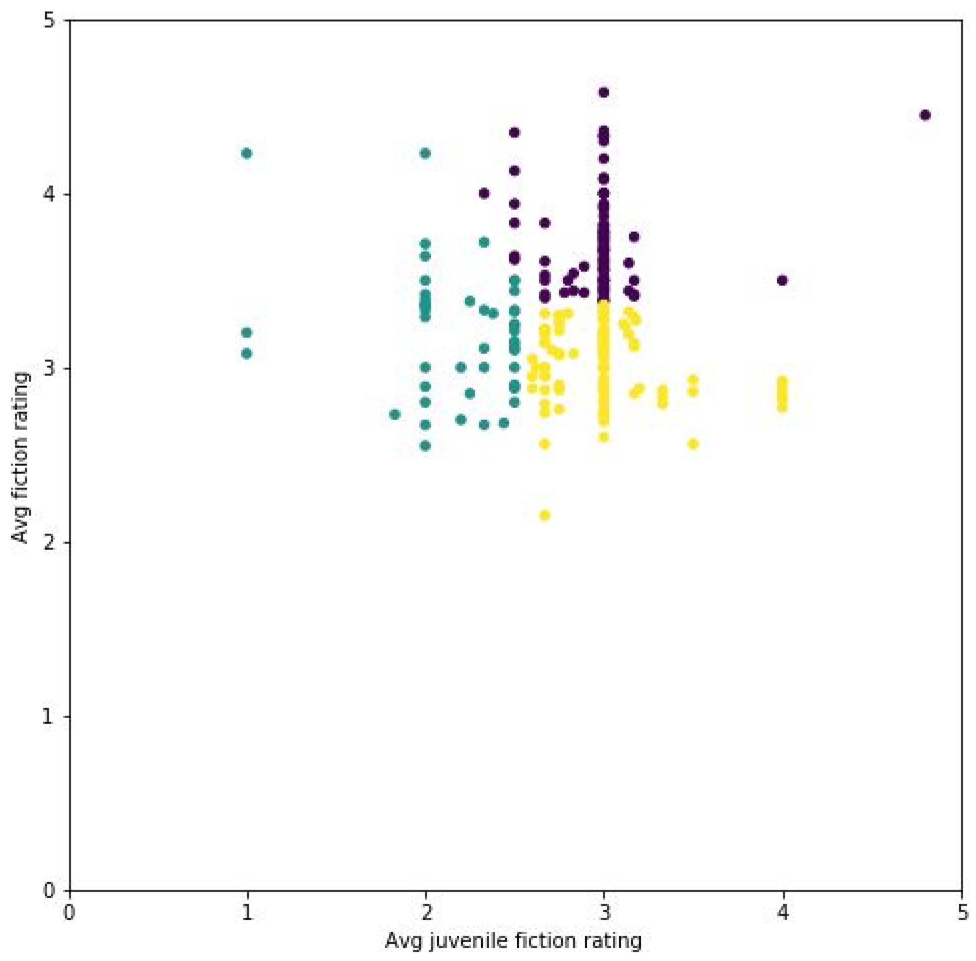




If we split the biased data into two groups using K-means we can see on the right plot that groups are mostly based on how each user rated *Fiction* books. It their average rating of *Fiction* books is over 3.3, they belong to one group. Otherwise, they belong to the other group.

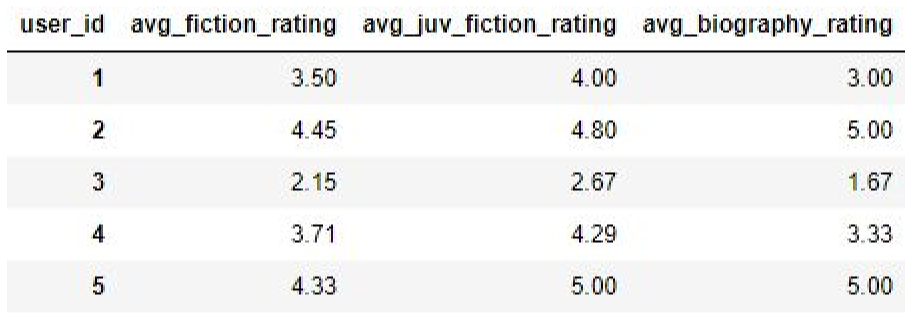
If we break the sample into 3 clusters, we can recognize the following groups:

- Users who like *Fiction* but not *Fiction Juvenile*
- Users who like *Juvenile Fiction* but not *Fiction*
- Users who like both *Fiction* and *Fiction Juvenile*

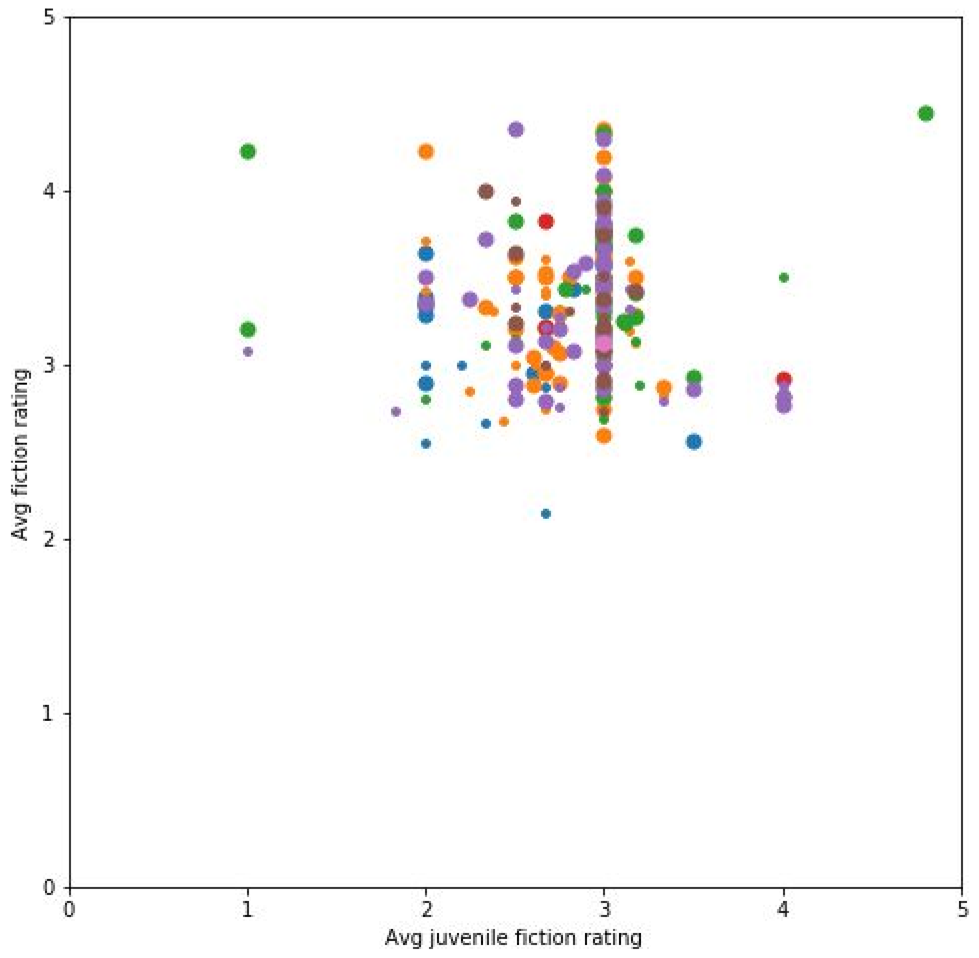


We can draw a conclusion that the more clusters we split into our dataset, the more similar are tastes of each cluster to each other.

Now, let’s look how subset of dataset with three genres (*Fiction*, *Juvenile Fiction* and *Biography & Autobiography*) look like:



user_id	avg_fiction_rating	avg_juv_fiction_rating	avg_biography_rating
1	3.50	4.00	3.00
2	4.45	4.80	5.00
3	2.15	2.67	1.67
4	3.71	4.29	3.33
5	4.33	5.00	5.00

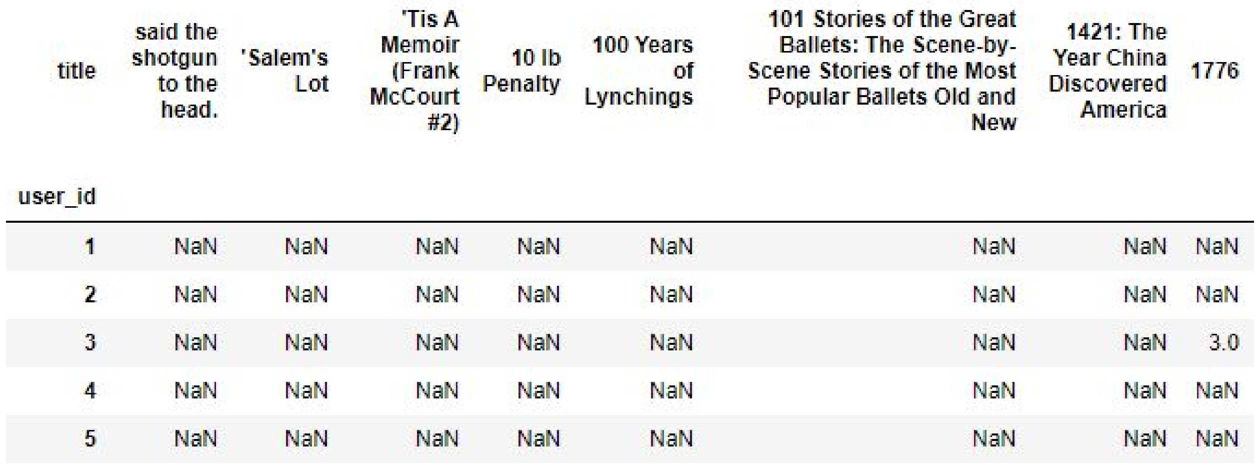


X and Y axes show *Fiction* and *Juvenile Fiction* genres respectively and size of the dot depicts the *Biography & Autobiography* rating (large dots means average ratings over 3 and small otherwise).

We can see that the added genre is changing how users are clusterred. The more data we use in K-means, the more similar the tastes of the users in each group are.

Book-level clustering

Now that we’ve earned some trust in how K-means groups users by their preferences in genres, let’s take a look at how users rated individual books. To do that we’ll shape the dataset in the form of *user_id* vs *user rating* for each book. Here, is the sample subset of dataset:

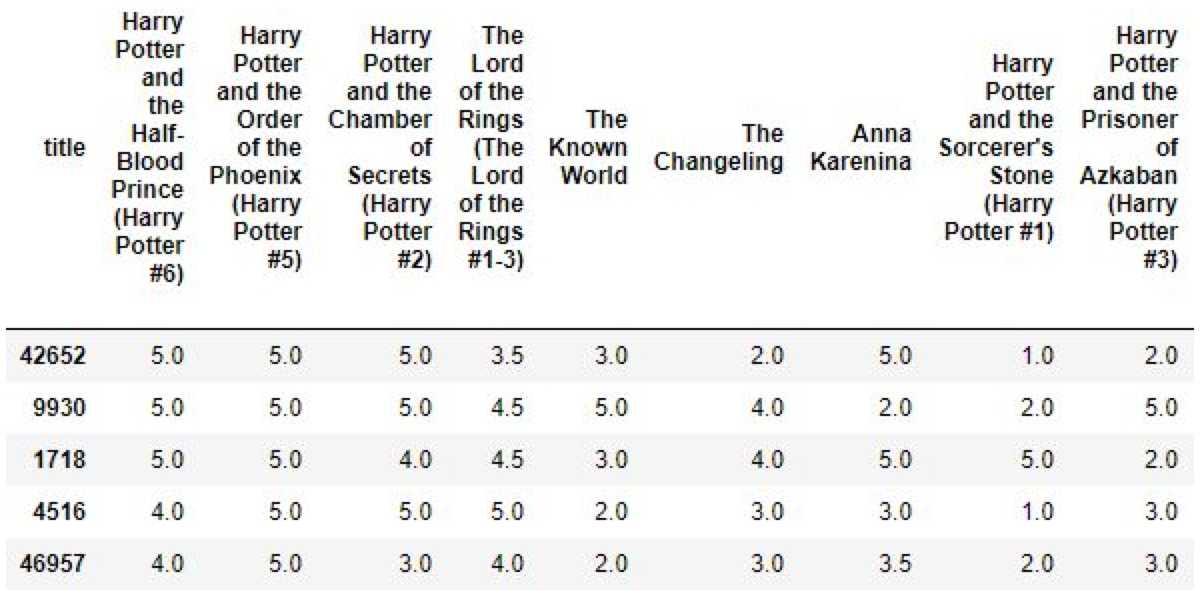


title	said the shotgun to the head.	'Salem's Lot	'Tis A Memoir (Frank McCourt #2)	10 lb Penalty	100 Years of Lynchings	101 Stories of the Great Ballets: The Scene-by-Scene Stories of the Most Popular Ballets Old and New	1421: The Year China Discovered America	1776
user_id								
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

The supremacy of *NaN* values shows the first issue. Most users didn't rate and read most books. Datasets like this are called *sparse* because only a small number of cells have values.

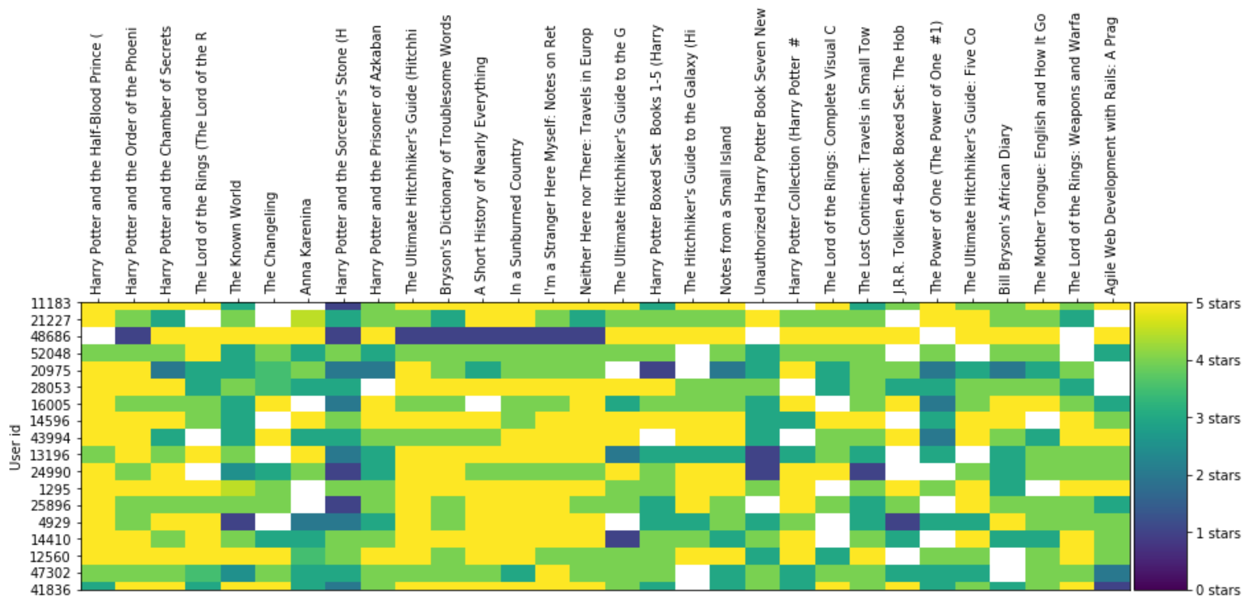
To deal with this, let's sort dataset by the most rated books and the users who have rated the most number of books. That will present a more *dense* region when we peak at the top of the dataset.

If we want to choose the most-rated books vs users with most ratings, it would look like this:



title	Harry Potter and the Half-Blood Prince (Harry Potter #6)	Harry Potter and the Order of the Phoenix (Harry Potter #5)	Harry Potter and the Chamber of Secrets (Harry Potter #2)	The Lord of the Rings (The Lord of the Rings #1-3)	The Known World	The Changeling	Anna Karenina	Harry Potter and the Sorcerer's Stone (Harry Potter #1)	Harry Potter and the Prisoner of Azkaban (Harry Potter #3)
42652	5.0	5.0	5.0	3.5	3.0	2.0	5.0	1.0	2.0
9930	5.0	5.0	5.0	4.5	5.0	4.0	2.0	2.0	5.0
1718	5.0	5.0	4.0	4.5	3.0	4.0	5.0	5.0	2.0
4516	4.0	5.0	5.0	5.0	2.0	3.0	3.0	1.0	3.0
46957	4.0	5.0	3.0	4.0	2.0	3.0	3.5	2.0	3.0

Let's visualize these ratings using colors instead of numbers so we can attempt to visually recognize the ratings when we look at bigger subsets.



Each column represents a book and each row represents a user. The color of the cell is how the user rated that book based on the scale on the right of the graph. White cells mean that particular user did not rate a book. This is an issue because K-means generally does not like missing values. To address this problem, we'll take advantage of SciPy library[11].

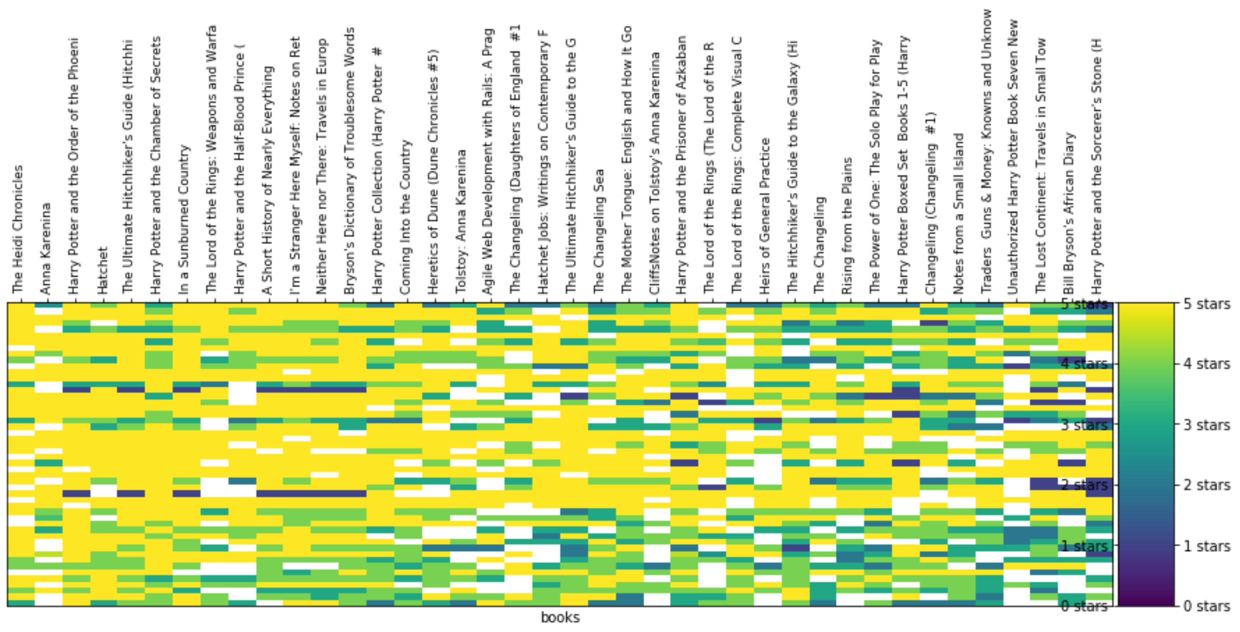
Use K-means

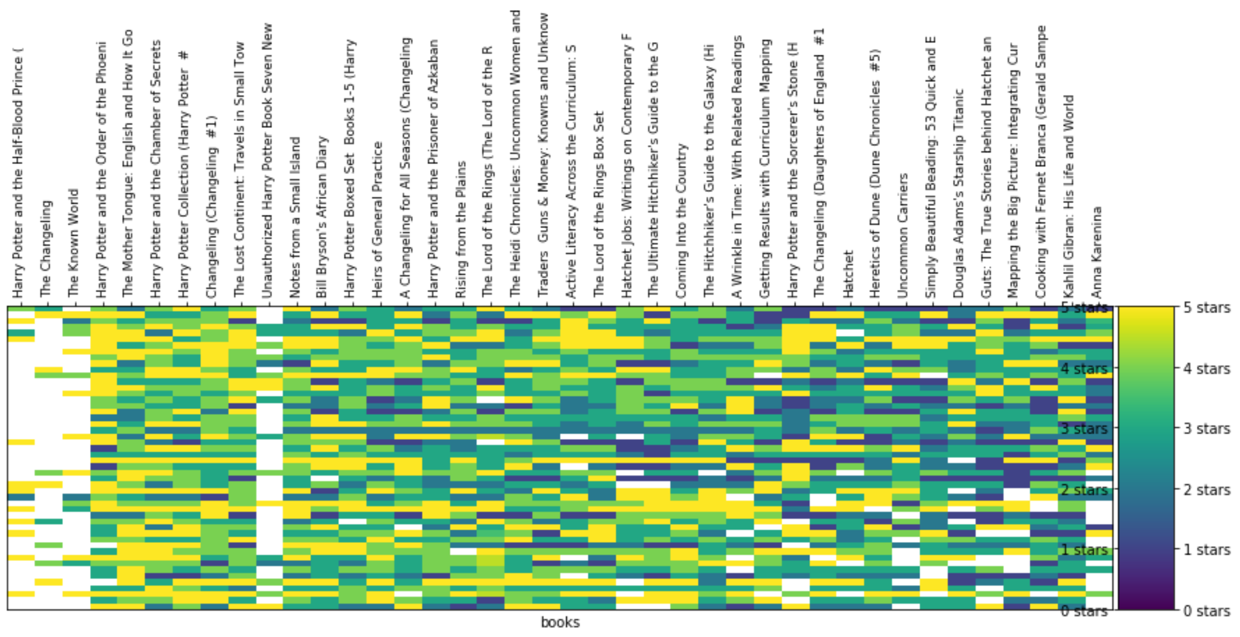
With K-means, we have to specify *k*, the number of clusters. To find the right number of clusters for this dataset I used the *elbow method*[12] and *Silhouette score* for each *k* between 2 and the number of elements in our dataset.

After evaluation of two methods and substantiating my choice with *Kneedle* algorithm I used *k* = 37.

To visualize a sample clusters, we can plot them as a heat map:

[11] https://docs.scipy.org/doc/scipy/reference/
[12] https://en.wikipedia.org/wiki/Elbow_method_(clustering)



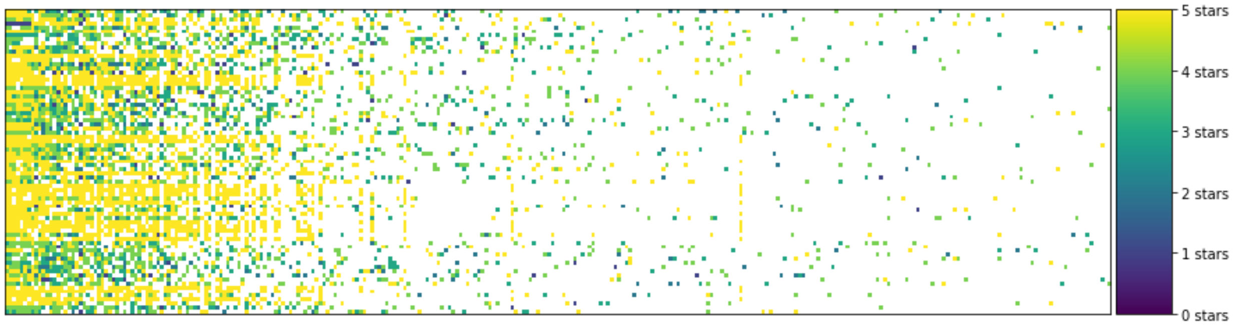


Here are several observations we can notice from above graphs:

- The more similiar ratings in a cluster, the more vertical lines in similar colors we can see
- Horizontal lines with similar colors show users without a lot of variety in their ratings
- Some clusters are more sparse than others, including users who probably read and rated less books than in other clusters
- Some clusters are mostly yellow and contain users who really like a particular group of books; other clusters are mostly green or blue bringing together users who rate a certain set of books very low

Prediction

Let's take the first from above clusters (cluster 22). In the following graph we can see the region of dataset with the most number of values:



The actual ratings in the cluster look like below:

	The Ultimate Hitchhiker's Guide (Hitchhiker's Guide to the Galaxy #1-5)	In a Sunburned Country	Neither Here nor There: Travels in Europe	I'm a Stranger Here Myself: Notes on Returning to America After Twenty Years Away	Harry Potter and the Order of the Phoenix (Harry Potter #5)	A Short History of Nearly Everything	Bryson's Dictionary of Troublesome Words: A Writer's Guide to Getting It Right	Harry Potter and the Chamber of Secrets (Harry Potter #2)
358	5.0	5	5	5	5	5	3	
649	5.0	5	5	5	5	5	5	5
386	4.0	4	4	5	5	4	4	5
805	1.0	1	1	1	1	1	1	5
382	4.0	4			4	4	4	5
670	5.0	5	5	5	5	5	5	5
234	3.0	3	3	3	3	3	3	3

Blank cells in the table mean that a user didn't rate particular book. Since the user is in a cluster of users that seem to have similar taste, we can take the average of the votes for a book in the cluster and that would be a reasonable prediction that a user would enjoy the book.

The average rating for *In a Sunburned Country* book is 4.54 and this is our prediction for how the user would rate the book.

Recommendation

I have used K-means to cluster users according to their ratings. This leads us to clusters of users with similar ratings and similar taste in books. Based on this, when one user did not have a rating for a certain book we averaged the ratings of all the other users in the cluster and that was our guess to how this one user would like the book.

Using this logic, if we calculate the average score in this cluster for every book, we'd have an understanding for how the cluster feels about each book in the dataset. Here are the average ratings of 10 books rated by the users in the cluster:

The Ultimate Hitchhiker's Guide (Hitchhiker's Guide to the Galaxy #1-5)	4.533333
In a Sunburned Country	4.540541
Neither Here nor There: Travels in Europe	4.462687
I'm a Stranger Here Myself: Notes on Returning to America After Twenty Years Away	4.578125
Harry Potter and the Order of the Phoenix (Harry Potter #5)	4.567164
A Short History of Nearly Everything	4.542857
Bryson's Dictionary of Troublesome Words: A Writer's Guide to Getting It Right	4.348485
Harry Potter and the Chamber of Secrets (Harry Potter #2)	4.727273
The Mother Tongue: English and How It Got That Way	4.095238
Harry Potter Collection (Harry Potter #1-6)	4.523077

This becomes very useful because we can now use it as a recommendation system that enables our users to discover books they're likely to enjoy. The formula for these recommendations is to select the cluster's highest-rated books that a user didn’t rate yet.

Let’s peak a user_id = 358. These are top 15 recommendations to this user:

Little Women	5.000000
Phaedrus	5.000000
Lincoln at Gettysburg: The Words That Remade America	5.000000
Corelli's Mandolin	5.000000
Into the Wild	5.000000
Trump: The Art of the Deal	5.000000

Gates of Fire: An Epic Novel of the Battle of Thermopylae	5.000000
Metamorphoses	5.000000
The Kite Runner	5.000000
A Book of Common Prayer	4.818182
Love Letters	4.750000
Harry Potter and the Chamber of Secrets (Harry Potter #2)	4.727273
Mason & Dixon	4.678571
Lysistrata	4.666667
The Sea	4.666667

Refinement

In order to improve the model I've made two adjustments in K-means algorithm:

1. For performance reasons, I only used ratings for 1000 books out of the 10000+ available in the dataset. Even this subset of data required about 30 minutes for K-means clustering.

2. Because the dataset is sparse and K-means clustering is sensitive to missing values, I casted dataset to the *sparse_csr_matrix*[13] type defined in the *SciPi* library. To convert from a pandas dataframe to a sparse matrix, I had to convert to *SparseDataFrame* and then use pandas' *to_coo()* method for the conversion.

3. Because of sparse dataset I was filtering, sorting and slicing data to make the observations more visible.

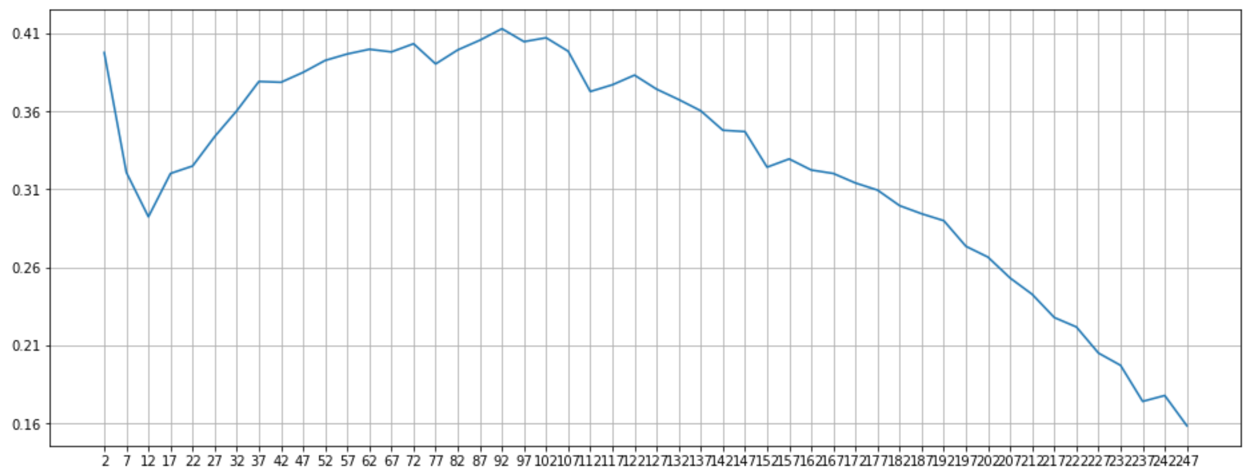
IV. Results

Model Evaluation and Validation

For our evaluation I decided to use the *Silhouette score* and for K-means clustering specifically, the *Elbow method*.

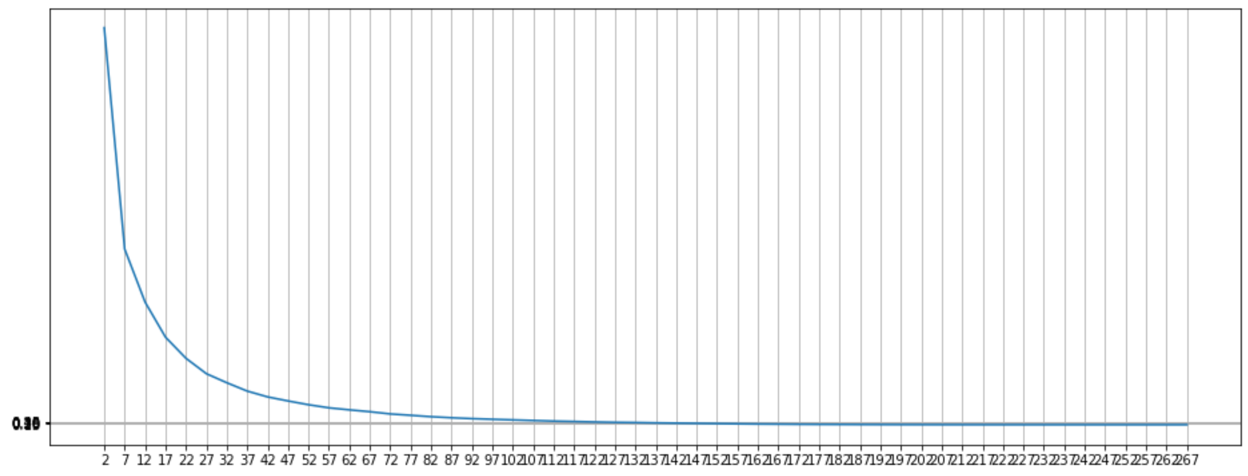
The silhoutte scores are plotted below for *k* between 2 and number of elements in our dataset:

[13] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html



Looking at the graph above presenting results of *silhoutte* values, good choices for *k* include 37, 72, 92, amongst other values. Increasing the number of clusters beyond that range starts to result in worse clusters according to silhouette score.

Let's look now at the plot presenting *Elbow method* results:

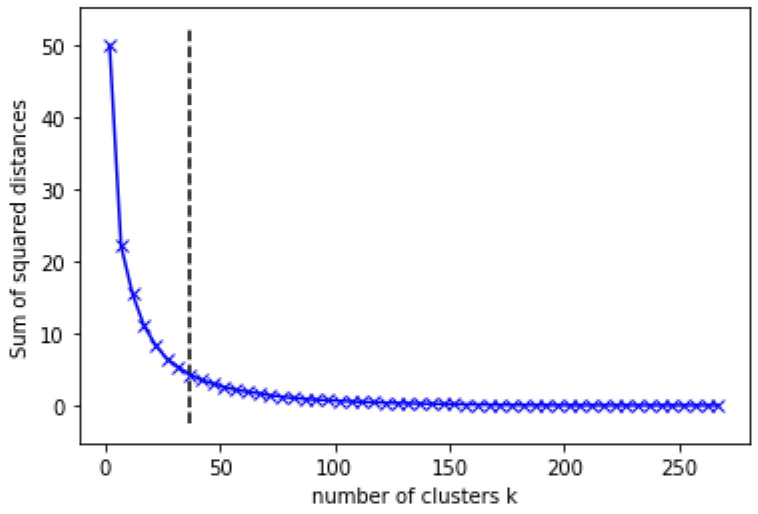


We can notice that at point *k* = 37 the SSE (sum of squared error) starts to decrease abruptly. This *elbow* point should determine the optimal number of clusters.

Comparing results of *Silhouette score* and *Elbow method*, my choice for number of clusters used in K-means algorithm is 37.

Justification

Taking advantage of implementation[14] of *Kneedle algorithm*, I managed to find an elbow point determining *k*:



From this and two other metrics, *Silhouette score* and *Elbow method*, we can conclude that 37 is the optimal number of clusters for K-means clustering on our dataset.

V. Conclusion

Free-Form Visualization

The visualizations of datasets and implementation process has been provided and described thoroughly in Analysis and Methodology sections.

Reflection

The recommendation system that have been implemented is very simple and shows the most basic idea of collaborative filtering. It is a method of making predictions about the interests of user by analysing tastes of similar users.

One of the most time-consuming aspects of the project was finding and using external API to assign genres to books dataset. Yet, it wasn't a perfect solution. Google Books API turned out

[14] https://github.com/arvkevi/kneed

to have limit of 1000 queries per day what wasn't enough for more dataset consisted of more than 10,000 books.

Another interesting aspect regards the role of bias. In general, some readers are more important compared to others. Similarly, some books can be considered universally good and will be rated high by most of the readers. This information is captured by bias.

Improvement

There are many heuristics and methods that may improve the recommendation system that has been built. We can take the inspiration from *BellKor's Pragmatic Chaos* team that won the Netflix Prize[15] for implementing a recommendation engine for Netflix.

Another area of improvement is in datasets. There are much more books in the world than 10,000. Books can be also categorized better. The assigned genres taken from Google Books API turned out to be better than shelves available in Goodreads API, however some of them had many variations of similar names which in fact should be the same. Collected genres would be much more useful if they're normalized in more general groups. Additionally, Goodreads API contains information about books that users would like to read. Taking advantage of this information, the recommendation engine would probably work even better.

I could also try to implement other clustering algorithm and compare the results with K-means.

Reference

- Udacity Project: k-means Clustering of Movie Ratings
- Kaggle, https://www.kaggle.com/jealousleopard/goodreadsbooks
- Kaggle, https://www.kaggle.com/zygmunt/goodbooks-10k
- Goodreads API, https://www.goodreads.com/api
- Article: *Recommendation Systems — Models and Evaluation*, https://towardsdatascience.com/recommendation-systems-models-and-evaluation-84944a84fb8e
- Article: *Comprehensive Guide to build a Recommendation Engine from scratch (in Python)*, https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/
- Kneedle algorithm implementation, https://github.com/arvkevi/kneed

[15] https://en.wikipedia.org/wiki/Netflix_Prize