Collaborative Filtering: Book Search

Kevin Price
Nic Carlson
Reshma Patil
Ross Cole

01 Problem statement & assumptions

Problem statement

Finding the right book

Members that subscribe to Library of Congress have access to a digitized catalog of books where they can search for books and store their book ratings.

Users can search to find books on specific topics but have difficulty finding books that they actually enjoy reading. The library needs help to develop a way to integrate ratings into search results so that members can find books that match their reading preferences.

02 Proposed solution

Proposed solution

Integrate collective filtering into search

Collaborative filtering is a method of making predictive recommendations to users based on their past preferences combined with peers that have similar preferences.

We propose doing this by suggesting:

- Books similar to user's taste
- Books that similar users liked

The user-facing output will be recommended book lists:

- 'You liked *xyz* book. You might also like:'
- 'Users like you also liked:'
- 'Checkout these new *xyz genre* books:'

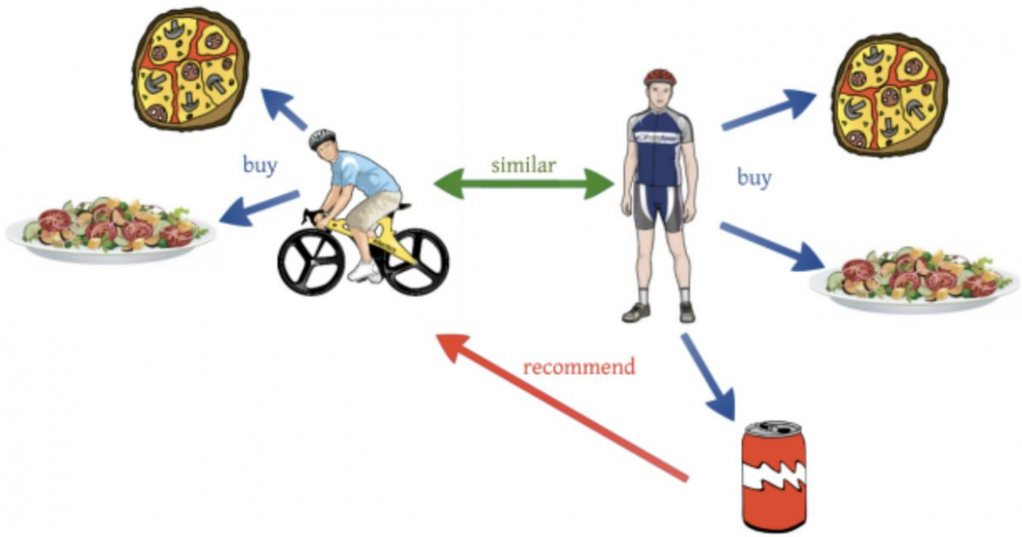
Implementation:

- Integrate rating feature into library app
- Provide users option to use API to connect their library account with Goodreads account

Summary of Research

Collaborative Filtering is among the most basic, yet most common recommender systems used by broad-based user sets from Amazon, Netflix, Spotify and iTunes

Collaborative filtering models are based on the assumption that people like things similar to other things they like, and things that are liked by other people with similar taste



Source: towardsdatascience.com

Summary of Research

There are multiple collaborative filtering approaches to consider, broadly defined as memory based and model based:

	Memory Based	Model Based
Description	*Makes use of user rating information to calculate the likeness between the users or items*	*Models are created by using data mining, and the system learns algorithms to look for habits according to training data*
Advantages	Easier to explain Easier to implement on lower-scale datasets	Better performance under missing or sparse data Can predict unrated items
Disadvantages	Difficult to scale (e.g., sparse datasets become computationally slow and expensive) Low rating activity can disqualify items	More difficult to explain inference due to hidden / latent factors driven by trained model
Other Relevant Notes	Item and user-based algorithms can be deployed and compared for efficacy	Requires more sophisticated machine-learning techniques such as PCA, SVD, neural nets

Source: towardsdatascience.com
Source: University of Minnesota DCSE

Proposed solution: Linear algebra concepts used

	Memory-based		**Model-based**
Linear Algebra Concept	User-based	Item-based	Truncated SVD
Vectors/Matrices	x	x	x
Matrix diagonalization	x		x
Matrix characteristics	Under-determined	Over-determined	Combination
Dot products	x	x	x
Cosine similarities		x	x
SVD			x
Vector Length	x		
...			

The data

To build our recommendation engine for the Library of Congress members, which currently has no rating data, we used Goodreads data as model, merging book and user ratings

Goodreads is the world's largest site for readers and book recommendations

Merged on

Key Goodreads extracts:

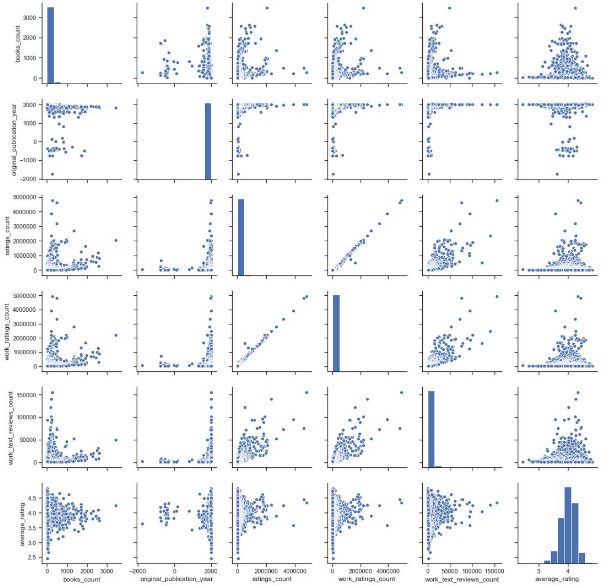
- **Book ID (10k books)**
- Author
- Title
- Avg Rating
- Rating Count

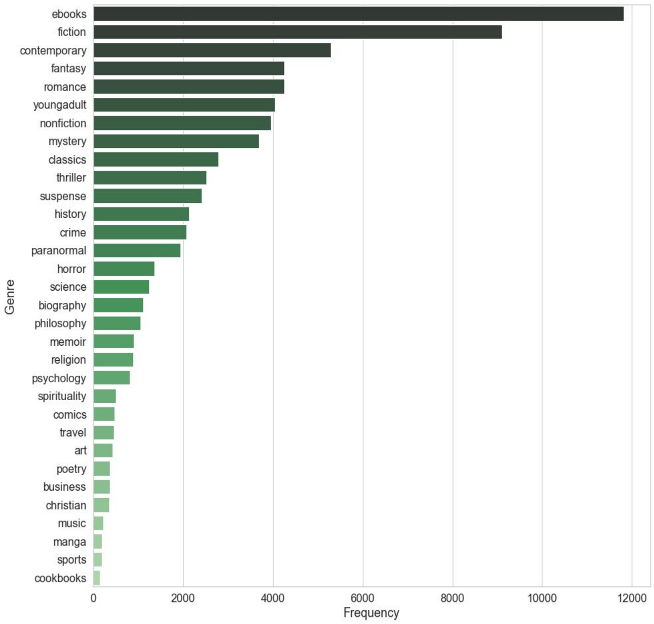
Library extracts:

- User ID (~$53k users)
- **Book ID**
- Book title
- Ratings (~1M ratings)

Assigned tags:

- Genre





Assumptions

People using library of congress are interested in finding similar books to what they've already read (i.e. there are casual readers)

The Library has a pre-existing interface where users either already engage or are willing to engage

For item-based recommendations: Users know of a book title within the category they are looking.

For collaborative filtering: Users that share a similar preferences in the past will share similar preferences in the future.

Applications: Memory Based Filtering

User-based filtering

Purchases	Cat halloween costume Catnip Band-aids	Cat costume Catnip *You might also like:* Band-aids
Views	Funny cat videos Cat sneezing You'll enjoy, 'Cooking for one'	Funny cat videos Cat sneezing Cooking for one
Ratings	5 stars: Cat in the Hat 3 stars: All dogs Go to Heaven You'll enjoy, 'Cat in Paris'	5 stars: Cat in the Hat 3 stars: All dogs Go to Heaven 5 stars: Cat in Paris

Process

Cleaning

- Removed books with no ratings
- Removed users with less than three ratings
- Removed book titles with non-English names using language-code and TextBlob package
- Normalize rating for users subtracting mean of rating given by user

Selected first base similar users

- Created book ID and user ID matrices and ratings as values
- Select user (we'll call them user 'A')
- Bucket users that have rated at least one book from user A's book list

Filtered similar user by two methods

- Method 1 – By rating
  - Built a correlation matrices for user A and first base similar users.
  - Select top 15 percentile of similar users.
- Method 2 – By Genre – Likeliness index
  - Created array count of tags (genre) associated with books.
  - Calculated a dot product between first base similar users and user A.
  - Select top 15 percentile of similar users.

Normalizing scores and predict recommendation list of books

- Take out books read by similar users and calculate ($i_1 ... i_k$), mean and number of rating
- Select top 10 percentile of items rated by users (Filter 1)
- Select top 10 highest mean rated books as a recommendation list of books.

Script and output

Method 1 – Correlation matrices of Rating

```python
def get_similar_user_by_rating(current_user, cutoff,  book_matrix_norm, selected_user):
    corr_mat = dict()
    user1 = book_matrix_norm[current_user].reset_index()
    for n_user in selected_user:
        if n_user != current_user:
            user2 = book_matrix_norm[n_user].reset_index()
            temp   = pd.merge(user1, user2, on='book_id')
            temp.fillna(0, inplace=True)
            temp.drop('book_id', axis=1, inplace=True)
            corr_mat[n_user] = temp.corr(method ='pearson').loc[current_user, n_user]
    corr_df = pd.DataFrame(corr_mat.items(), columns=['users', 'correlation'])
    corr_df = corr_df.sort_values(by=['correlation'], ascending=False)

    cut_off                 = corr_df.quantile(cutoff, numeric_only=True)
    similar_users           = corr_df[corr_df['correlation']>=float(cut_off)]['users'].values

    return similar_users
```

Method 2 – Dot product /Vector Length of Genre

```python
def get_similar_user_by_genre(current_user , cutoff, sel_genre, book_matrix_norm, selected_user):
    sel_genre_df = pd.DataFrame(pd.Series(sel_genre), columns=['Genre'])
    for user in selected_user:
        books_read      = list(book_matrix_norm[pd.notnull(book_matrix_norm[user])].index)
        us_tags_dict    = get_tags_user(books_read, user)
        sel_genre_df[user] = sel_genre_df['Genre'].apply(lambda x :us_tags_dict[x])

    sel_genre_df.set_index('Genre', inplace=True)
    dot_product_user                = pd.DataFrame(selected_user, columns=['users'])
    dot_product_user['dot_product'] = dot_product_user['users'].apply(lambda x :
                                          get_norm_dot_product(x, current_user, sel_genre_df))
    dot_product_user.set_index('users', inplace=True)
    dot_product_user.drop(current_user, inplace=True)

    cut_off                 = float(dot_product_user.quantile(cutoff, numeric_only=True))
    similar_users           = list(dot_product_user[dot_product_user['dot_product']>=cut_off].index)

    return similar_users
```

List of books User read earlier

```
get_list_of_books_user("17329", book_matrix_norm)
```

```
7                        The Catcher in the Rye
23          Harry Potter and the Goblet of Fire
144                             Deception Point
267                              Never Let Me Go
645                          Tales of Caunterbury
911                           Two for the Dough
2816                        Chapterhouse: Dune
5044                    La ciudad de las bestias
5314                          Play It as It Lays
6843                                 Moon Palace
9669                  Trump: The Art of the Deal
```

Result 1 – Recommendation by Rating

```
book_recommendation("17329", 'RATING', book_matrix_norm)
```

```
70    Surely You're Joking, Mr. Feynman! Adventures ...
0                                          Dune Messiah
84                           Animal Farm: A Fairy Story
32                                     Bleach—ブリーチ— 15
73                                  The Woman in White
72                                   A Bend in the River
39                              Job: A Comedy of Justice
44                                   The Quiet American
69                                          The Summons
```

Result 2 – Recommendation by Genre

```
book_recommendation("17329", 'GENRE', book_matrix_norm)
```

```
57               Harry Potter and the Philosopher's Stone
133                            The Plot Against America
72                             Job: A Comedy of Justice
0                 Harry Potter and the Half-Blood Prince
74          The Lord of the Rings: Weapons and Warfare
150                                      The Egypt Game
105   Raise High the Roof Beam, Carpenters / Seymour...
5                                       Children of Dune
152                                                Heidi
148                                      Of Mice and Men
```

Applications: Memory Based Filtering

Purchases

Kitten scratching post

Kitten nail clipper

Band-aids

Views

Cat sneezing

Cat laughing

Funny cat videos

Ratings

5 stars: Cat in the Hat

3 stars: All Dogs Go to Heaven

4(?) stars: Cats vs. Dogs

Item based filtering

Process: Item-Item filtering

Overview

- Predict preferences for **new** users based on rating patterns between items.
- If two books tend to have the same users like and dislike them, then they are similar, and users are expected to have similar preferences for similar books.

Cleaning

- Drop users with less than two ratings
- Drop books with less than 1,500 ratings

Create an *m*x*n* user-book matrix

- Each row corresponds to a user (i.e., there are m users)
- Each column corresponds to a particular book (i.e., there are *n* books)
  - Note: If the book matrix is over-determined, item similarities can be pre-computed, leading to performance gains.

Generate book recommendations for new users

- New users select a book they liked from a dropdown list
- Compute correlation between the ratings of the “liked” book and all other books in the list
- Provide top-10 recommendations
  - This list consists of books with the highest correlations

Script and output

Recommendation process

Let $\mathbf{R}$ be an mxn matrix of book ratings (on a 1-5 scale) with m users and n books.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix}$$

We can compute how similar two books i and j are by calculating the correlation between them. To do this, we look at the set of users who rated both books i and j. Call this set of users $\mathbf{U}$.

Then, the correlation between them is given by

$$Corr(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}}$$

where $R_{u,i}$ denotes the rating of user u on book i, and $\bar{R}_i$ is the mean rating of book i. Similarly, $R_{u,j}$ denotes the rating of user u on book j, and $\bar{R}_j$ is the mean rating of book j.

If we first standardize the columns of $\mathbf{R}$ to have a mean of zero and standard deviation of 1, we can use Cosine Similarity to arrive at the same result:

$$Corr(i,j) = cos(\vec{i},\vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\| \times \|\vec{j}\|}$$

In [71]:

```
# Dropdown menu containing the list of unique book titles
users = widgets.Dropdown(
    options=['Select title']+sorted(df['title'].unique().tolist()),
    description='Title:',
    layout=Layout(width='80%',height='30px'),
    disabled=False)

# Run code after new user selects a book they liked
buttonuser_info = widgets.Button(
    description='Get recommendations',
    layout=Layout(width='40%',height='30px'))

outuser_info = widgets.Output()

def on_buttonuser_info_clicked(b):
    with outuser_info:
        clear_output()

        # Flag the book selected by new user
        selection = df[df['title']==users.value]
        user_liked = book_matrix[users.value]

        # Correlation between ratings of selected book and all others in the matrix
        similar_books = book_matrix.corrwith(user_liked)

        # Sort in descending order according to correlations. Then the books that are most
        # similar will be listed first
        similar_books.sort_values(inplace=True, ascending=False)
        correlations = pd.Series(similar_books)

        # Loop through and print the top 10 recommandations
        for i in range(11):
            if i==0:
                print('\n\nRecommendations based on: {: <46}{: >3}\n{:_<90}'.format(correlations.ind
            else:
                print('[{:<}] {: <60} \t {:.2f}'.format(i, correlations.index[i], correlations[i]))
            if i==11:
                print('{:_<90}'.format(''))

# Links button: buttonuser_info to its output function
buttonuser_info.on_click(on_buttonuser_info_clicked)

# Display
display(widgets.HBox([users,buttonuser_info]))
display(widgets.VBox([outuser_info]))
```

Script and output continued

Title: Harry Potter and the Sorcerer's Stone (Harry Potter, #1) | Get recommendations

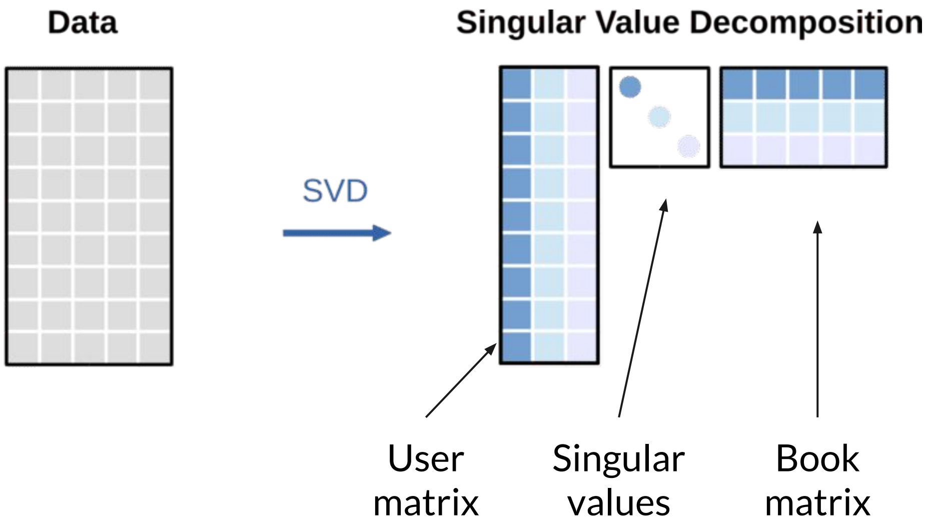
Recommendations based on: Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	Correlation
[1] Harry Potter and the Chamber of Secrets (Harry Potter, #2)	0.74
[2] Harry Potter and the Prisoner of Azkaban (Harry Potter, #3)	0.67
[3] Harry Potter and the Goblet of Fire (Harry Potter, #4)	0.63
[4] Harry Potter and the Half-Blood Prince (Harry Potter, #6)	0.57
[5] Harry Potter and the Order of the Phoenix (Harry Potter, #5)	0.57
[6] Harry Potter and the Deathly Hallows (Harry Potter, #7)	0.53
[7] The Green Mile	0.32
[8] The Da Vinci Code (Robert Langdon, #2)	0.31
[9] The Hunger Games (The Hunger Games, #1)	0.29
[10] Angels & Demons (Robert Langdon, #1)	0.29

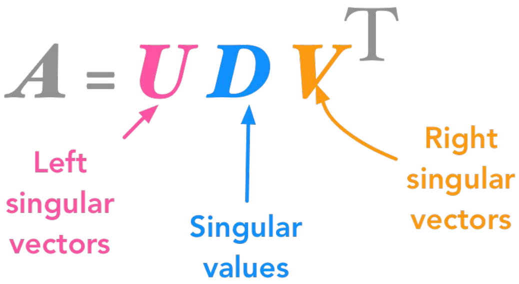
Title: A Light in the Attic | Get recommendations

Recommendations based on: A Light in the Attic	Correlation
[1] Where the Sidewalk Ends	0.75
[2] Matilda	0.46
[3] The Cat in the Hat	0.45
[4] Oh, The Places You'll Go!	0.45
[5] How the Grinch Stole Christmas!	0.44
[6] The BFG	0.43
[7] Green Eggs and Ham	0.43
[8] The Lorax	0.43
[9] The Very Hungry Caterpillar Board Book	0.42
[10] Charlie and the Chocolate Factory (Charlie Bucket, #1)	0.42

Applications: Model-based - truncated SVD

Very similar to PCA, except that this compares importance of features in the data itself versus in the covariance matrix





Returns most significant data

https://www.researchgate.net/figure/Conceptual-architecture-of-the-randomized-singular-value-decomposition-The-data-are_fig3_305995021

Process

Cleaning

- Removed books with no ratings → no need since SVD will rule them out as important
- Removed users with less than three ratings → future consideration
- Removed book titles with non-English names using language tag (imperfect)

Matrix Factorization

- Create matrix with books as variables, user as rows, and ratings as values
- Run truncated SVD on the matrix to isolate books with biggest influence on ratings
- Return truncated matrix
- Create a correlation matrix from the truncated matrix

Normalizing scores and selecting 'alike' users

- For user($i_1 ... i_k$), take out user mean rating from each rating
- Keep 15th percentile of users with highest correlation
- This 15th percentile book ratings can now be applied to user A

Run function to calculate correlation and return books, which is essentially a centered version of cosine similarity

Efficiency gains from using truncated SVD are impressive

99.96%

matrix dimensionality reduction...

```
(53424, 8112) : Original matrix size prior to Truncated Singular Value Decomposition (SVD)
(8112, 20) : New matrix size after applying Truncated SVD model
```

...without hurting rating integrity

```
Book Recommendations For:  A Walk to Remember
B
```

```
Book Recommendations For:  The Lord of the Rings
```

	Book title	Correlation
484	The Hitchhiker's Guide to the Galaxy	0.99
203	Harry Potter Collection (Harry Potter, #1-6)	0.98
510	The Lord of the Rings: Weapons and Warfare	0.98
575	The Ultimate Hitchhiker's Guide: Five Complete...	0.97
209	Hatchet	0.97
207	Harry Potter and the Philosopher's Stone	0.96
208	Harry Potter and the Prisoner of Azkaban	0.95
325	Notes from a Small Island	0.92
206	Harry Potter and the Order of the Phoenix	0.92

```
5 7442                True Believer     0.93
  4414                  Safe Haven      0.92
```

Given more time, we would...

04 Future work

- ➔ Clean-up book tags (genre and language) with automated solution
- ➔ Embed the item-based algorithm into pre-existing content search functionality (i.e. less restrictive)
- ➔ Explore Goodreads APIs to see if we can leverage user's existing accounts paired with our book database
- ➔ For users unwilling to link Goodreads accounts and/or with few ratings, develop an 'onboarding' process for library users with less than *n* ratings so they are asked to select books they've read in order to 'prime' the algorithm for their first search
- ➔ Compare the three methods using accuracy (SMSE) / efficiency (speed/storage) testing to determine best method
- ➔ Modify the algorithm to mix in random related suggestions to protect against the 'rich get richer' effect

Questions?