January 5, 2025

1 Books Recommendation System

Elif İdil Ayata - 200201039
Ömer Erdem Dilek - 200204036
Muhammed Yasinhan Yaşar - 200201010

1.0.1 Neccesary Imports

[116]:
```
import pandas as pd
import numpy as np
import ast
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import pairwise_distances
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
↪precision_score, recall_score, f1_score
```

[117]:
```
data = pd.read_csv("data/goodreads_data.csv")
data.head()
```

[117]:

	Unnamed: 0	Book	Author	Description	Genres	Avg_Rating	Num_Ratings
0	0	To Kill a Mockingbird	Harper Lee	The unforgettable novel of a childhood in a sl…	['Classics', 'Fiction', 'Historical Fiction', …	4.27	5,691,311
1	1	Harry Potter and the Philosopher's Stone (Harr…	J.K. Rowling	Harry Potter thinks he is an ordinary boy - un…	['Fantasy', 'Fiction', 'Young Adult', 'Magic',…	4.47	9,278,135
2	2	Pride and Prejudice	Jane Austen	Since its immediate success in 1813, Pride and…			
3	3	The Diary of a Young Girl	Anne Frank	Discovered in the attic in which she spent the…			
4	4	Animal Farm	George Orwell	Librarian's note: There is an Alternate Cover …			

```
2  ['Classics', 'Fiction', 'Romance', 'Historical…        4.28   3,944,155
3  ['Classics', 'Nonfiction', 'History', 'Biograp…        4.18   3,488,438
4  ['Classics', 'Fiction', 'Dystopia', 'Fantasy',…        3.98   3,575,172

                                                  URL
0  https://www.goodreads.com/book/show/2657.To_Ki…
1  https://www.goodreads.com/book/show/72193.Harr…
2  https://www.goodreads.com/book/show/1885.Pride…
3  https://www.goodreads.com/book/show/48855.The_…
4  https://www.goodreads.com/book/show/170448.Ani…
```

1.1 1) Data Cleaning Step

We drop here the URL column, it will never be used

[118]:
```
df = pd.DataFrame(data)
df = df.drop(columns=["URL"])
df.head()
```

[118]:
```
   Unnamed: 0                                               Book  \
0           0                            To Kill a Mockingbird
1           1  Harry Potter and the Philosopher's Stone (Harr…
2           2                              Pride and Prejudice
3           3                        The Diary of a Young Girl
4           4                                     Animal Farm

          Author                                        Description  \
0     Harper Lee  The unforgettable novel of a childhood in a sl…
1   J.K. Rowling  Harry Potter thinks he is an ordinary boy - un…
2    Jane Austen  Since its immediate success in 1813, Pride and…
3     Anne Frank  Discovered in the attic in which she spent the…
4  George Orwell  Librarian's note: There is an Alternate Cover …

                                              Genres  Avg_Rating Num_Ratings
0  ['Classics', 'Fiction', 'Historical Fiction', …        4.27   5,691,311
1  ['Fantasy', 'Fiction', 'Young Adult', 'Magic',…        4.47   9,278,135
2  ['Classics', 'Fiction', 'Romance', 'Historical…        4.28   3,944,155
3  ['Classics', 'Nonfiction', 'History', 'Biograp…        4.18   3,488,438
4  ['Classics', 'Fiction', 'Dystopia', 'Fantasy',…        3.98   3,575,172
```

Checked for the empt values of rows

[119]:
```
data.isnull().sum()
```

[119]:
```
Unnamed: 0      0
Book            0
Author          0
Description    77
```

```
Genres          0
Avg_Rating      0
Num_Ratings     0
URL             0
dtype: int64
```

Drop that rows that has empty Descriptions

[120]:
```
data_cleaned = df.dropna(subset=['Description'])
data_cleaned.isnull().sum()
```

[120]:
```
Unnamed: 0     0
Book           0
Author         0
Description    0
Genres         0
Avg_Rating     0
Num_Ratings    0
dtype: int64
```

Make sure that Avg_Rating has the range of `0 >= Avg_Rating <= 5`

[121]:
```
data_cleaned.loc[:, 'Avg_Rating'] = data_cleaned['Avg_Rating'].clip(lower=0,␣
↪upper=5)
```

'Num_Ratings' written in a format that you put a ',' comma for every three digit (Americans!) We fixed it

[122]:
```
data_cleaned.loc[:, 'Num_Ratings'] = data_cleaned['Num_Ratings'].astype(str).
↪apply(
    lambda x: int(x.replace(',', '')) if x.replace(',', '').isdigit() else 0
)
data_cleaned.head()
```

[122]:

	Unnamed: 0	Book	Author	Description
0	0	To Kill a Mockingbird	Harper Lee	The unforgettable novel of a childhood in a sl…
1	1	Harry Potter and the Philosopher's Stone (Harr…	J.K. Rowling	Harry Potter thinks he is an ordinary boy - un…
2	2	Pride and Prejudice	Jane Austen	Since its immediate success in 1813, Pride and…
3	3	The Diary of a Young Girl	Anne Frank	Discovered in the attic in which she spent the…
4	4	Animal Farm	George Orwell	Librarian's note: There is an Alternate Cover …

	Genres	Avg_Rating	Num_Ratings
0	['Classics', 'Fiction', 'Historical Fiction', …	4.27	5691311
1	['Fantasy', 'Fiction', 'Young Adult', 'Magic',…	4.47	9278135
2	['Classics', 'Fiction', 'Romance', 'Historical…	4.28	3944155
3	['Classics', 'Nonfiction', 'History', 'Biograp…	4.18	3488438
4	['Classics', 'Fiction', 'Dystopia', 'Fantasy',…	3.98	3575172

as you can se here Genres has a format like `['Fantasy', 'Fiction', 'Young Adult', 'Magic',...]`
it is really similar to python's built-in arrays, so we just used the built-in ast(Abstract Syntax Tree) module of python to directly parse them.

[123]:
```
def ensure_list(value):
    if isinstance(value, str):
        try:
            return ast.literal_eval(value)
        except (ValueError, SyntaxError):
            return ['Unknown']
    elif isinstance(value, list):
        return value
    return ['Unknown']
```

If there was an empty Genres list like `[]` just append 'Unknown' to it

[124]:
```
data_cleaned = data_cleaned.copy()  # Create a copy

# Convert all Genres to lists
data_cleaned.loc[:, 'Genres'] = data_cleaned['Genres'].apply(ensure_list)

# Replace empty lists with 'Unknown'
data_cleaned.loc[:, 'Genres'] = data_cleaned['Genres'].apply(lambda x:␣
 ↪['Unknown'] if len(x) == 0 else x);
```

1.2 2) Similarty Measurement

Making that array of Genres space seperated strings to vectorize them word by word

[125]:
```
# Convert genres list to space-separated string
data_cleaned['Genres'] = data_cleaned['Genres'].apply(lambda x: ' '.join(x))

# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(
    stop_words='english',  # Remove common stop words
    max_features=5000,     # Limit features to top 5000 terms
    ngram_range=(1, 1)     # Use unigrams only for genres
)
```

[146]:
```
data_cleaned['Genres'].head()
```

[146]:
```
0    Classics Fiction Historical Fiction School Lit…
1    Fantasy Fiction Young Adult Magic Childrens Mi…
2    Classics Fiction Romance Historical Fiction Li…
3    Classics Nonfiction History Biography Memoir H…
4    Classics Fiction Dystopia Fantasy Politics Sch…
Name: Genres, dtype: object
```

Calculating cosine similarties of word vectors and creating `cosine_similarty_matrix`

[147]:
```
# Transform the Genres column
tfidf_matrix = tfidf_vectorizer.fit_transform(data_cleaned['Genres'])
# Compute cosine similarity
cosine_similarity_matrix = cosine_similarity(tfidf_matrix)
```

convert cos matrix to a data frame

[148]:
```
# Create a DataFrame for the similarity matrix
cosine_similarity_df = pd.DataFrame(
    cosine_similarity_matrix,
    index=data_cleaned['Book'],  # Use book titles as index
    columns=data_cleaned['Book']  # Use book titles as columns
)

# Display the similarity DataFrame (first 3x3 part)
cosine_similarity_df.iloc[:3, :3]
```

[148]:
```
Book                                                To To Kill a Mockingbird  \
Book
To Kill a Mockingbird                                                1.000000
Harry Potter and the Philosopher's Stone (Harry…                     0.271020
Pride and Prejudice                                                  0.678404

Book                                                Harry Potter and the
Philosopher's Stone (Harry Potter, #1)  \
Book
To Kill a Mockingbird
0.27102
Harry Potter and the Philosopher's Stone (Harry…
1.00000
Pride and Prejudice
0.13171

Book                                                Pride and Prejudice
Book
To Kill a Mockingbird                                          0.678404
Harry Potter and the Philosopher's Stone (Harry…               0.131710
Pride and Prejudice                                            1.000000
```

Finding jaccard similarity matrix
We used here a `MultiLabelBinarizer()` not to iterate through nested for loops (it was very slow)

[129]:
```
# Initialize MultiLabelBinarizer
mlb = MultiLabelBinarizer()

# Transform the genres into a one-hot encoded matrix
genre_matrix = mlb.fit_transform(data_cleaned['Genres'])

# Compute Jaccard similarity using pairwise distances
jaccard_matrix = 1 - pairwise_distances(genre_matrix, metric='jaccard')
```

```
c:\Users\Lenovo\Desktop\data-mining\data-mining-project\data-mining\Lib\site-
packages\sklearn\metrics\pairwise.py:2466: DataConversionWarning: Data was
converted to boolean for metric jaccard
  warnings.warn(msg, DataConversionWarning)
```

convert jaccard matrix to a data frame

[149]:
```
# Convert to a DataFrame for readability
jaccard_df = pd.DataFrame(
    jaccard_matrix,
    index=data_cleaned['Book'],
    columns=data_cleaned['Book']
)

# Display the Jaccard similarity DataFrame (first 3x3 part)
jaccard_df.iloc[:3, :3]
```

[149]:
```
Book                                                To To Kill a Mockingbird  \
Book
To Kill a Mockingbird                                                1.000000
Harry Potter and the Philosopher's Stone (Harry…                     0.760000
Pride and Prejudice                                                  0.692308

Book                                                Harry Potter and the
Philosopher's Stone (Harry Potter, #1)  \
Book
To Kill a Mockingbird
0.760000
Harry Potter and the Philosopher's Stone (Harry…
1.000000
Pride and Prejudice
0.571429

Book                                                Pride and Prejudice
Book
To Kill a Mockingbird                                          0.692308
```

```
Harry Potter and the Philosopher's Stone (Harry…    0.571429
Pride and Prejudice                                 1.000000
```

1.3 3) Recommendation Generation

Generating recomendation with genres with given similarity matrix

[131]:
```
def generate_recommendations(user_interacted_books, similarty_df, top_n=5):
    # Get the similarity scores for the books the user interacted with
    similar_books_scores = pd.Series(dtype=float)

    for book in user_interacted_books:
        similar_books_scores = similar_books_scores.add(similarty_df[book] /␣
 ↪len(user_interacted_books), fill_value=0)
        # similar_books_scores = similar_books_scores.add(similarty_df[book],␣
 ↪fill_value=0)

    # Rank books by their similarity score
    recommendations = similar_books_scores.sort_values(ascending=False)

    # Remove books the user has already interacted with
    recommendations = recommendations[~recommendations.index.
 ↪isin(user_interacted_books)]

    # Return top N recommendations
    return recommendations.head(top_n)
```

[132]:
```
user_interacted_books = ['1984']
top_recommendations_jaccard = generate_recommendations(user_interacted_books,␣
 ↪jaccard_df, top_n=3)
top_recommendations_cosine = generate_recommendations(user_interacted_books,␣
 ↪cosine_similarity_df, top_n=3)


top_recommendations_jaccard
top_recommendations_cosine
```

[132]:
```
Book
Animal Farm / 1984                                0.899684
Brave New World / Brave New World Revisited       0.839162
Island                                            0.800050
dtype: float64
```

Applying freshenss to our `generate_recommendations` function to have variaty of different results each time

[150]:
```
def generate_recommendations_with_freshness(user_interacted_books,␣
↪similarity_df, previous_recommendations=None, freshness_weight=0.5, top_n=3):
    # Get the similarity scores for the books the user interacted with
    similar_books_scores = pd.Series(dtype=float)

    for book in user_interacted_books:
        similar_books_scores = similar_books_scores.add(similarity_df[book] /␣
↪len(user_interacted_books), fill_value=0)

    # Remove books the user has already interacted with
    similar_books_scores = similar_books_scores[~similar_books_scores.index.
↪isin(user_interacted_books)]

    # If previous recommendations are provided, apply churn
    if previous_recommendations:
        freshness_scores = pd.Series(index=similar_books_scores.index,␣
↪dtype=float).fillna(1.0)
        for book in similar_books_scores.index:
            if book in previous_recommendations:
                freshness_scores[book] *= (1 - freshness_weight)  # Penalize␣
↪previously recommended items
        similar_books_scores *= freshness_scores  # Combine similarity with␣
↪freshness

    # Rank books by their adjusted similarity score
    recommendations = similar_books_scores.sort_values(ascending=False)

    # Return top N recommendations
    return recommendations.head(top_n)
```

We first used `generate_recommendations` for generating recomendations without freshness and than used that value as the `previous_recommendations` of `generate_recommendations_with_freshness`

[ ]:
```
user_interacted_books = ['1984']
previous_recommendations = set(top_recommendations_jaccard.index)

# Assuming similarity_df is already defined
top_recommendations_with_churn = generate_recommendations_with_freshness(
    user_interacted_books,
    similarity_df=cosine_similarity_df,
    previous_recommendations=previous_recommendations,
    freshness_weight=0.3,
    top_n=5
)

top_recommendations_with_churn
```

[ ]:
```
Book
Animal Farm / 1984    0.899684
Cat's Cradle          0.789169
Atlas Shrugged        0.787444
A Clockwork Orange    0.784548
Brave New World       0.758446
dtype: float64
```

1.4 4) Model Evaluation Metrics:

Precision, Recall, F1-Score

[135]:
```
# 1. Calculate Precision, Recall, F1-Score
def calculate_classification_metrics(recommended_books, relevant_books):
    # Binary relevance labels
    y_true = [1 if book in relevant_books else 0 for book in recommended_books.
 ↪index]
    y_pred = [1] * len(recommended_books)  # All recommended books are treated␣
 ↪as "positive"

    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    return precision, recall, f1
```

[136]:
```
relevant_books = ['Animal Farm / 1984', 'Brave New World / Brave New World␣
 ↪Revisited', 'Blindness']

jaccard_precision, jaccard_recall, jaccard_f1 =␣
 ↪calculate_classification_metrics(top_recommendations_jaccard, relevant_books)
cosine_precision, cosine_recall, cosine_f1 =␣
 ↪calculate_classification_metrics(top_recommendations_cosine, relevant_books)

# Print metrics
print("\nMetrics for Jaccard Recommendations:")
print(f"Precision: {jaccard_precision}, Recall: {jaccard_recall}, F1-Score:␣
 ↪{jaccard_f1}")

print("\nMetrics for Cosine Recommendations:")
print(f"Precision: {cosine_precision}, Recall: {cosine_recall}, F1-Score:␣
 ↪{cosine_f1}")
```

```
Metrics for Jaccard Recommendations:
Precision: 0.6666666666666666, Recall: 1.0, F1-Score: 0.8
```

```
Metrics for Cosine Recommendations:
Precision: 0.6666666666666666, Recall: 1.0, F1-Score: 0.8
```

MAE and RMSE

[137]:
```
# 2. Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE)
def calculate_regression_metrics(recommended_books, relevant_books):
    # Generate ratings (1 if relevant, 0 otherwise)
    actual_ratings = [1 if book in relevant_books else 0 for book in␣
 ↪recommended_books.index]
    predicted_ratings = recommended_books.values  # Use similarity scores as␣
 ↪predicted ratings

    mae = mean_absolute_error(actual_ratings, predicted_ratings)
    rmse = np.sqrt(mean_squared_error(actual_ratings, predicted_ratings))
    return mae, rmse
```

[138]:
```
relevant_books = ['Animal Farm / 1984', 'Brave New World / Brave New World␣
 ↪Revisited', 'Blindness']

jaccard_mae, jaccard_rmse =␣
 ↪calculate_regression_metrics(top_recommendations_jaccard, relevant_books)
cosine_mae, cosine_rmse =␣
 ↪calculate_regression_metrics(top_recommendations_cosine, relevant_books)

# Print regression metrics
print("\nRegression Metrics for Jaccard Recommendations:")
print(f"MAE: {jaccard_mae}, RMSE: {jaccard_rmse}")

print("\nRegression Metrics for Cosine Recommendations:")
print(f"MAE: {cosine_mae}, RMSE: {cosine_rmse}")
```

```
Regression Metrics for Jaccard Recommendations:
MAE: 0.34848484848484845, RMSE: 0.5534408466734372

Regression Metrics for Cosine Recommendations:
MAE: 0.3537347513976199, RMSE: 0.4746969511600921
```

nDCG

[139]:
```
def calculate_ndcg(relevant_scores, recommended_scores, k=5):
    relevance = [1 if score in relevant_scores else 0 for score in␣
 ↪recommended_scores[:k]]
    dcg = sum(rel / np.log2(idx + 2) for idx, rel in enumerate(relevance))
    idcg = sum(1 / np.log2(idx + 2) for idx in range(min(len(relevant_scores),␣
 ↪k)))
    return dcg / idcg if idcg > 0 else 0
```

```
ndcg_cos = calculate_ndcg(relevant_books, top_recommendations_cosine.index.
 ↪tolist(), k=5)
ndcg_jac = calculate_ndcg(relevant_books, top_recommendations_jaccard.index.
 ↪tolist(), k=5)
print(f"nDCG for cos: {ndcg_cos}")
print(f"nDCG for jac: {ndcg_jac}")
```

```
nDCG for cos: 0.7653606369886217
nDCG for jac: 0.5307212739772434
```

ARHR

[140]:
```
def calculate_arhr(relevant_books, recommended_books):
    hits = [1 / (idx + 1) if book in relevant_books else 0 for idx, book in␣
 ↪enumerate(recommended_books)]
    return sum(hits) / len(relevant_books)

arhr_cos = calculate_arhr(relevant_books, top_recommendations_cosine.index.
 ↪tolist())
arhr_jac = calculate_arhr(relevant_books, top_recommendations_jaccard.index.
 ↪tolist())
print(f"ARHR for cos: {arhr_cos}")
print(f"ARHR for jac: {arhr_jac}")
```

```
ARHR for cos: 0.5
ARHR for jac: 0.27777777777777773
```

CHR

[141]:
```
# General Accuracy
def calculate_chr(relevant_books, recommended_books):
    hits = [1 for book in recommended_books if book in relevant_books]
    return len(hits) / len(recommended_books)

chr_cos = calculate_chr(relevant_books, top_recommendations_cosine.index.
 ↪tolist())
chr_jac = calculate_chr(relevant_books, top_recommendations_jaccard.index.
 ↪tolist())
print(f"CHR for cosine: {chr_cos}")
print(f"CHR for jaccard: {chr_jac}")
```

```
CHR for cosine: 0.6666666666666666
CHR for jaccard: 0.6666666666666666
```

Serendipity

[142]:
```
def calculate_serendipity(recommended_books, relevant_books,␣
 ↪user_interacted_books):
```

```
    unexpected_books = [book for book in recommended_books if book not in␣
 ↪user_interacted_books]
    serendipitous_hits = [book for book in unexpected_books if book in␣
 ↪relevant_books]
    return len(serendipitous_hits) / len(recommended_books) if␣
 ↪recommended_books else 0

serendipity_cos = calculate_serendipity(top_recommendations_cosine.index.
 ↪tolist(), relevant_books, user_interacted_books)
serendipity_jac = calculate_serendipity(top_recommendations_jaccard.index.
 ↪tolist(), relevant_books, user_interacted_books)
print(f"Serendipity for cosine: {serendipity_cos}")
print(f"Serendipity for jaccard: {serendipity_jac}")
```

```
Serendipity for cosine: 0.6666666666666666
Serendipity for jaccard: 0.6666666666666666
```

Diversity

[143]:
```
def calculate_diversity(recommended_books, similarity_df):
    similarities = []
    for i, book_i in enumerate(recommended_books):
        for j, book_j in enumerate(recommended_books):
            if i < j:
                similarities.append(similarity_df.loc[book_i, book_j])
    return 1 - np.mean(similarities) if similarities else 0

diversity_cos = calculate_diversity(top_recommendations_cosine.index.tolist(),␣
 ↪cosine_similarity_df)
diversity_jac = calculate_diversity(top_recommendations_jaccard.index.tolist(),␣
 ↪jaccard_df)
print(f"Diversity: {diversity_cos}")
print(f"Diversity: {diversity_jac}")
```

```
Diversity: 0.21028068784270326
Diversity: 0.08574879227053145
```

Coverage

[144]:
```
def calculate_coverage(recommended_books, catalog):
    return len(set(recommended_books)) / len(catalog)

catalog = pd.DataFrame({
    "Book": ["1984", "Animal Farm", "Blindness", "Brave New World"],
    "Num_Ratings": [4201429, 3575172, 265298, 159408] # like popularity
})
```

```
coverage_cos = calculate_coverage(top_recommendations_cosine.index.tolist(),␣
 ↪catalog)
coverage_jac = calculate_coverage(top_recommendations_jaccard.index.tolist(),␣
 ↪catalog)
print(f"Coverage for cosine: {coverage_cos}")
print(f"Coverage for jaccard: {coverage_jac}")
```

```
Coverage for cosine: 0.75
Coverage for jaccard: 0.75
```

[145]:
```
# You can see freshness usage with churn above with␣
 ↪'generate_recommendations_with_freshness' function implementation
```

Elif İdil Ayata - 200201039
Ömer Erdem Dilek - 200204036
Muhammed Yasinhan Yaşar - 200201010