E09 - Recommender System

Çoban, Ömer Furkan

Task 1: Similarity Analysis

In this task, we aim to compare scientific papers based on their abstracts. We will fetch the data directly from the web, clean it, and then apply similarity metrics.

1. Scrape Abstracts from arXiv

We use the `requests` library to fetch the HTML content of the provided arXiv URLs. Then, we use `BeautifulSoup` to parse the HTML and extract the **Title** (inside `h1` tag with class `title`) and the **Abstract** (inside `blockquote` tag with class `abstract`).

In [1]:
```
import pandas as pd
import numpy as np
import re
import requests
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.stem import PorterStemmer
from IPython.display import display

urls = [
    "https://arxiv.org/abs/2007.08978",
    "https://arxiv.org/abs/2002.03389",
    "https://arxiv.org/abs/2006.16964",
    "https://arxiv.org/abs/2007.04095",
    "https://arxiv.org/abs/2008.00315",
    "https://arxiv.org/abs/1901.08151",
    "https://arxiv.org/abs/1901.10555",
    "https://arxiv.org/abs/1511.03085",
    "https://arxiv.org/abs/1905.03061",
    "https://arxiv.org/abs/2002.01759"
]

def scrape_arxiv(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')

        # Extract Title
        title_tag = soup.find('h1', class_='title')
        title = title_tag.text.replace('Title:', '').strip() if title_tag else "No Title"

        # Extract Abstract
        abstract_tag = soup.find('blockquote', class_='abstract')
        abstract = abstract_tag.text.replace('Abstract:', '').strip() if abstract_tag else "No Abstract"

        return {"url": url, "title": title, "abstract": abstract}
    except Exception as e:
        print(f"Error scraping {url}: {e}")
        return {"url": url, "title": "Error", "abstract": ""}

print("Scraping abstracts from arXiv...")
papers_data = [scrape_arxiv(url) for url in urls]
df_papers = pd.DataFrame(papers_data)
print("Scraping completed.")
df_papers[['title']]
```

```
Scraping abstracts from arXiv...
Scraping completed.
```

Out[1]:

	title
0	A large-scale comparative analysis of Coding S...
1	Trust in Data Science: Collaboration, Translat...
2	Data Science: Nature and Pitfalls
3	Data science and the art of modelling
4	A fresh look at introductory data science
5	Cloud BI: Future of Business Intelligence in t...
6	The Effects of Using Business Intelligence Sys...
7	Big Data and Business Intelligence: Debunking ...
8	Generalized formal model of big data
9	Quality Assurance Technologies of Big Data App...

2. Preprocessing

To ensure accurate similarity calculations, we clean the text data:

1. **Lowercasing**: Convert all text to lower case to treat "Data" and "data" as the same.

2. **Removing Noise**: Remove punctuation, numbers, and special characters.
3. **Stop Word Removal**: Remove common English words (e.g., "the", "is", "at") that do not carry significant meaning.
4. **Stemming**: Reduce words to their root form (e.g., "learning" -> "learn") using the Porter Stemmer.

In [2]:
```python
# Load Stop Words
try:
    with open('all_stop_words.txt', 'r') as f:
        stop_words = set(f.read().split())
except FileNotFoundError:
    print("Warning: 'all_stop_words.txt' not found. Creating empty set.")
    stop_words = set()

stemmer = PorterStemmer()

def preprocess_text(text):
    # Lowercase
    text = text.lower()
    # Remove special chars and numbers (keep only letters and spaces)
    text = re.sub(r'[^a-z\s]', '', text)
    # Tokenize
    words = text.split()
    # Remove stop words and stem
    words = [stemmer.stem(w) for w in words if w not in stop_words]
    return ' '.join(words)

df_papers['processed_abstract'] = df_papers['abstract'].apply(preprocess_text)
df_papers[['title', 'processed_abstract']].head()
```

Out[2]:

	title	processed_abstract
0	A large-scale comparative analysis of Coding S...	background meet grow industri demand data scie...
1	Trust in Data Science: Collaboration, Translat...	trustworthi data scienc system appli realworld...
2	Data Science: Nature and Pitfalls	data scienc creat excit trend controversi crit...
3	Data science and the art of modelling	datacentr enthusiasm grow strong varieti domai...
4	A fresh look at introductory data science	prolifer vast quantiti dataset larg complex na...

3. Similarity Calculations

We use two different metrics to calculate similarity between papers:

- **Jaccard Similarity**: Measures the overlap between the sets of unique words in two documents.
- **Cosine Similarity**: Measures the cosine of the angle between two TF-IDF vectors.

In [3]:
```python
# Jaccard Similarity
def get_jaccard_similarity(str1, str2):
    a = set(str1.split())
    b = set(str2.split())
    c = a.intersection(b)
    if len(a) + len(b) - len(c) == 0:
        return 0.0
    return float(len(c)) / (len(a) + len(b) - len(c))

# Compute Jaccard Matrix
n = len(df_papers)
jaccard_matrix = np.zeros((n, n))

for i in range(n):
    for j in range(n):
        jaccard_matrix[i, j] = get_jaccard_similarity(df_papers.iloc[i]['processed_abstract'], df_papers.iloc[j]['processed_abstra

df_jaccard = pd.DataFrame(jaccard_matrix, index=df_papers['title'], columns=df_papers['title'])

# Cosine Similarity using TF-IDF
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df_papers['processed_abstract'])
cosine_matrix = cosine_similarity(tfidf_matrix)

df_cosine = pd.DataFrame(cosine_matrix, index=df_papers['title'], columns=df_papers['title'])

print("Similarity matrices calculated.")
```

```
Similarity matrices calculated.
```

4. Results: Most Similar Papers

We present the results in a readable table format, identifying the most similar paper for each entry.

In [4]:
```python
def get_most_similar_df(sim_df):
    # Mask diagonal explicitly to avoid self-matching
    np.fill_diagonal(sim_df.values, -np.inf)

    results = []
    for title in sim_df.index:
        most_sim_title = sim_df.loc[title].idxmax()
        score = sim_df.loc[title].max()
        results.append({
            "Paper": title,
            "Most Similar Paper": most_sim_title,
            "Score": score
        })
```

```
    return pd.DataFrame(results).sort_values("Score", ascending=False)

print("--- Jaccard Similarity Results ---")
display(get_most_similar_df(df_jaccard.copy()))

print("\n--- Cosine Similarity Results ---")
display(get_most_similar_df(df_cosine.copy()))
```

```
--- Jaccard Similarity Results ---
```

	Paper	Most Similar Paper	Score
4	A fresh look at introductory data science	Quality Assurance Technologies of Big Data App...	0.089552
9	Quality Assurance Technologies of Big Data App...	A fresh look at introductory data science	0.089552
2	Data Science: Nature and Pitfalls	Big Data and Business Intelligence: Debunking ...	0.075758
7	Big Data and Business Intelligence: Debunking ...	Data Science: Nature and Pitfalls	0.075758
6	The Effects of Using Business Intelligence Sys...	Generalized formal model of big data	0.073684
8	Generalized formal model of big data	The Effects of Using Business Intelligence Sys...	0.073684
1	Trust in Data Science: Collaboration, Translat...	A fresh look at introductory data science	0.072000
3	Data science and the art of modelling	Data Science: Nature and Pitfalls	0.060000
5	Cloud BI: Future of Business Intelligence in t...	The Effects of Using Business Intelligence Sys...	0.058333
0	A large-scale comparative analysis of Coding S...	A fresh look at introductory data science	0.055556

```
--- Cosine Similarity Results ---
```

	Paper	Most Similar Paper	Score
7	Big Data and Business Intelligence: Debunking ...	Generalized formal model of big data	0.362810
8	Generalized formal model of big data	Big Data and Business Intelligence: Debunking ...	0.362810
9	Quality Assurance Technologies of Big Data App...	Generalized formal model of big data	0.341554
1	Trust in Data Science: Collaboration, Translat...	Data Science: Nature and Pitfalls	0.290747
2	Data Science: Nature and Pitfalls	Trust in Data Science: Collaboration, Translat...	0.290747
4	A fresh look at introductory data science	Data Science: Nature and Pitfalls	0.236506
0	A large-scale comparative analysis of Coding S...	Trust in Data Science: Collaboration, Translat...	0.221797
5	Cloud BI: Future of Business Intelligence in t...	The Effects of Using Business Intelligence Sys...	0.188607
6	The Effects of Using Business Intelligence Sys...	Cloud BI: Future of Business Intelligence in t...	0.188607
3	Data science and the art of modelling	Trust in Data Science: Collaboration, Translat...	0.162726

Task 2: Movie Recommendation

In this task, we implement a recommender system using User-based Collaborative Filtering. We start with a SQL database dump, parse it into a usable format, and then build the recommendation engine to predict missing ratings.

1. Parse SQL & Generate CSVs

The data is provided in a PostgreSQL dump file (`ratingdb_postgresql.sql`). We use Regular Expressions (`re`) to parse the `INSERT` statements and extract data for:

- **Movies**: ID, Title, Release Year, Director.
- **Reviewers**: ID, Name.
- **Ratings**: Reviewer ID, Movie ID, Stars, Date.

We then save these structured datasets as CSV files for easier loading.

In [5]:
```
# Read SQL file
sql_file = 'ratingdb_postgresql.sql'
try:
    with open(sql_file, 'r') as f:
        sql_content = f.read()

    # Extract Movies
    movies = []
    movie_matches = re.findall(r"INSERT INTO movie \(m_id,title,release_year,director\) VALUES \((\d+),'([^']*)',(\d+),([^)]*)\);"
    for match in movie_matches:
        m_id, title, year, director = match
        director = director.strip("'") if director != 'NULL' else None
        movies.append({'m_id': int(m_id), 'title': title, 'year': int(year), 'director': director})

    df_movies_extracted = pd.DataFrame(movies)
    df_movies_extracted.to_csv('movies.csv', index=False)
    print("Created movies.csv")

    # Extract Reviewers
    reviewers = []
    reviewer_matches = re.findall(r"INSERT INTO reviewer \(r_id,reviewer_name\) VALUES \((\d+),'([^']*)'\);", sql_content)
    for match in reviewer_matches:
        r_id, name = match
        reviewers.append({'r_id': int(r_id), 'reviewer_name': name})

    df_reviewers_extracted = pd.DataFrame(reviewers)
```

```
    df_reviewers_extracted.to_csv('reviewers.csv', index=False)
    print("Created reviewers.csv")

    # Extract Ratings
    ratings = []
    rating_matches = re.findall(r"INSERT INTO rating \(r_id,m_id,stars,rating_date\) VALUES \((\d+),(\d+),(\d+),([^)]*)\);", sql_c
    for match in rating_matches:
        r_id, m_id, stars, date = match
        date = date.strip("'") if date != 'NULL' else None
        ratings.append({'r_id': int(r_id), 'm_id': int(m_id), 'stars': int(stars), 'date': date})

    df_ratings_extracted = pd.DataFrame(ratings)
    df_ratings_extracted.to_csv('ratings.csv', index=False)
    print("Created ratings.csv")

except FileNotFoundError:
    print(f"Error: {sql_file} not found. Utilizing existing CSVs if available.")
```

```
Created movies.csv
Created reviewers.csv
Created ratings.csv
```

2. Prepare Rating Matrix (Pivot Table)

We merge the extracted CSVs into a single DataFrame and then create a Pivot Table.

- **Rows**: Reviewers
- **Columns**: Movies
- **Values**: Ratings (Stars)

This matrix is the foundation for Collaborative Filtering.

In [6]:
```
df_movies = pd.read_csv('movies.csv')
df_ratings = pd.read_csv('ratings.csv')
df_reviewers = pd.read_csv('reviewers.csv')

# Merge
df = pd.merge(df_ratings, df_movies, on='m_id')
df = pd.merge(df, df_reviewers, on='r_id')

# Pivot Table (Users x Movies)
df_pivot = df.pivot_table(index='reviewer_name', columns='title', values='stars', aggfunc='mean')

# Ensure all movies are columns
all_movie_titles = df_movies['title'].unique()
df_pivot = df_pivot.reindex(columns=all_movie_titles)

print("Rating Matrix Shape:", df_pivot.shape)
df_pivot
```

```
Rating Matrix Shape: (8, 8)
```

Out[6]:

title	Gone with the Wind	Star Wars	The Sound of Music	E.T.	Titanic	Snow White	Avatar	Raiders of the Lost Ark
reviewer_name								
Ashley White	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN
Brittany Harris	NaN	NaN	2.0	NaN	NaN	NaN	NaN	3.0
Chris Jackson	NaN	NaN	3.0	2.0	NaN	NaN	NaN	4.0
Daniel Lewis	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN
Elizabeth Thomas	NaN	NaN	NaN	NaN	NaN	5.0	3.0	NaN
James Cameron	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN
Mike Anderson	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sarah Martinez	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3. Collaborative Filtering (User-Based)

We employ **User-based Collaborative Filtering**. The core idea is that users who agreed in the past (gave similar ratings) will agree in the future.

1. **Centering**: Different users have different baseline ratings (some are generous, some critical). We normalize this by subtracting each user's mean rating from their scores.
2. **Cosine Similarity**: We calculate the similarity between all pairs of users using the centered ratings.

In [7]:
```
# Center the ratings (subtract user mean)
df_centered = df_pivot.apply(lambda x: x - x.mean(), axis=1).fillna(0)

# User Similarity
user_sim = cosine_similarity(df_centered)
user_sim_df = pd.DataFrame(user_sim, index=df_pivot.index, columns=df_pivot.index)

# Item Similarity (Transposed)
item_sim = cosine_similarity(df_centered.T)
item_sim_df = pd.DataFrame(item_sim, index=df_pivot.columns, columns=df_pivot.columns)

print("Similarities calculated.")
```

```
Similarities calculated.
```

Questions

1. Which algorithm did you select and why? I selected **User-based Collaborative Filtering with Centered Cosine Similarity**. This method is effective because it:

- **Handles Bias**: Centering removes individual rating biases (optimism/pessimism).
- **Leverages Community**: It uses the collective wisdom of similar users to predict unknown preferences.

4. Identify Most Similar Movies & Users

Using the calculated similarity matrices, we can now answer:

- Which two movies are most similar based on user ratings?
- Which two users have the most similar taste?

In [8]:
```python
def get_most_similar_pair(sim_df):
    # Mask indices to avoid self-comparison
    # We use a copy to avoid checking diagonal
    sim_masked = sim_df.copy()
    np.fill_diagonal(sim_masked.values, -np.inf)

    sim_stack = sim_masked.stack()
    return sim_stack.idxmax(), sim_stack.max()

# Similar Movies
movie_pair, movie_score = get_most_similar_pair(item_sim_df)
print(f"Most similar movies: '{movie_pair[0]}' and '{movie_pair[1]}' (Score: {movie_score:.4f})")

# Similar Users
user_pair, user_score = get_most_similar_pair(user_sim_df)
print(f"Most similar users: '{user_pair[0]}' and '{user_pair[1]}' (Score: {user_score:.4f})")
```

```
Most similar movies: 'Gone with the Wind' and 'Star Wars' (Score: 0.0000)
Most similar users: 'Brittany Harris' and 'Chris Jackson' (Score: 0.5000)
```

Most Similar Users (Score: 0.5000)

- **Brittany Harris** and **Chris Jackson** have a positive correlation because they both rated 'Raiders of the Lost Ark' above their personal averages.

Most Similar Movies (Score: 0.0000)

- The score of **0.0** is correct for this dataset.
- There are **no movie pairs** with a positive correlation in this small dataset.
- 'Star Wars' has no ratings, resulting in a 0.0 similarity with everything.
- Since all other distinct pairs have negative correlations, 0.0 becomes the maximum score (and thus 'Gone with the Wind' and 'Star Wars' are returned alphabetically/by index as the best match).

5. Personal Recommendations

Finally, we put the recommender to the test.

1. **Create Profile**: Add a "My Profile" user with specific ratings (e.g., liking *Star Wars*).
2. **Re-calculate Similarities**: Update the user similarity matrix to include this new profile.
3. **Predict Ratings**: Estimate ratings for movies the profile hasn't seen yet (e.g., *E.T.*, *Titanic*) by taking a weighted average of ratings from the most similar users.

In [9]:
```python
# Create My Profile
my_ratings = pd.Series(name='My Profile', dtype=float)
my_ratings['Star Wars'] = 5.0
my_ratings['Raiders of the Lost Ark'] = 5.0
my_ratings['Avatar'] = 4.0

# Add to Pivot
df_pivot_with_me = pd.concat([df_pivot, my_ratings.to_frame().T])

# Re-calculate similarities
df_centered_with_me = df_pivot_with_me.apply(lambda x: x - x.mean(), axis=1).fillna(0)
user_sim_with_me = cosine_similarity(df_centered_with_me)
user_sim_with_me_df = pd.DataFrame(user_sim_with_me, index=df_pivot_with_me.index, columns=df_pivot_with_me.index)

def predict_rating(user, item, pivot_df, sim_df):
    # Get similar users
    user_similarities = sim_df[user].sort_values(ascending=False).drop(user)
    # Get ratings for the item
    item_ratings = pivot_df[item].drop(user).dropna()

    # Intersection
    relevant_users = item_ratings.index.intersection(user_similarities.index)

    if len(relevant_users) == 0:
        return pivot_df.loc[user].mean()

    weights = user_similarities[relevant_users]
    if weights.sum() == 0:
        return pivot_df.loc[user].mean()

    ratings = item_ratings[relevant_users]
```

```
    prediction = np.dot(weights, ratings) / weights.sum()
    return prediction

# Predict for unrated movies
target_movies = ['E.T.', 'Titanic']
print("--- Predictions for My Profile ---")
for movie in target_movies:
    pred = predict_rating('My Profile', movie, df_pivot_with_me, user_sim_with_me_df)
    print(f"Predicted rating for '{movie}': {pred:.2f}")
```

```
--- Predictions for My Profile ---
Predicted rating for 'E.T.': 2.00
Predicted rating for 'Titanic': 4.67
```