University of Information Technology & Sciences

Maddha Naya Nagar, Vatara, Baridhara, Dhaka-1212

Department of Computer Science and Engineering
Course Title: Data Mining and Warehouse Lab
Course Code: CSE 426

Submitted by

Md Arbin Zaman
ID: 2125051006
Batch: 50
Section: 8A

Submitted To

Mr. Mrinmoy Biswas Akash
Lecturer & Course Coordinator
University of Information Technology and Sciences

Project 1: Movie Recommendation System using Cosine Similarity

Introduction

The objective of this project is to develop a personalized movie recommendation system using collaborative filtering techniques. In the era of digital entertainment, providing intelligent content recommendations has become a major feature for platforms like Netflix, YouTube, and Amazon Prime. Collaborative filtering identifies patterns in user behavior to suggest content they are likely to enjoy.

In this project, a content-based recommendation model was built using user-movie ratings and cosine similarity. The similarity score between movies was calculated based on the rating vectors provided by users, and this similarity was used to suggest movies to users that they have not rated yet but are similar to their top-rated ones.

Objectives

- To load and preprocess a movie rating dataset.
- To create a sparse user-movie rating matrix using `csr_matrix`.
- To compute pairwise similarity between movies using cosine similarity.
- To generate personalized recommendations based on the highest-rated movies of a particular user.

Dataset Description

The dataset used includes two files:

- `ratings.csv`: Contains `userId`, `movieId`, and `rating`.
- `movies.xlsx`: Contains `movieId`, `title`, and `genres`.

These two datasets were merged on the `movieId` column to combine ratings and movie metadata.

Methodology

Data Preparation and Merging

The first step involved reading the CSV and Excel files using Pandas and merging them to form a single dataset with both rating and movie information.

Sparse Matrix Construction

Since the number of users and movies is large and most users have rated only a few movies, the dataset was converted to a sparse matrix using Scipy's csr_matrix. Each row represented a movie, and each column represented a user, with cell values corresponding to ratings.

Cosine Similarity

Scikit-learn's cosine_similarity() was applied to the sparse matrix to compute similarity scores between each pair of movies. This metric measures the cosine of the angle between two non-zero vectors, offering a normalized similarity value between 0 and 1.

Recommendation Generation

For a given user (example: userId = 1), the top 3 rated movies were identified. For each of these movies, the most similar movies (based on cosine similarity) were selected. Already rated movies were filtered out, and the remaining top 5 suggestions were returned as recommendations.

Code

```python
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix
```

```python
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
# Load datasets
ratings_path = "/content/drive/MyDrive/UITS/UITS /Eight Semester/Data Mining/Assignments/Assignment 1/ratings.csv"
movies_path = "/content/drive/MyDrive/UITS/UITS /Eight Semester/Data Mining/Assignments/Assignment 1/movies.xlsx"

ratings_df = pd.read_csv(ratings_path)
movies_df = pd.read_excel(movies_path)
```

```python
# Merge datasets
df = pd.merge(ratings_df, movies_df, on='movieId')

# Create sparse matrix
row = df['movieId'].astype('category').cat.codes
col = df['userId'].astype('category').cat.codes
data = df['rating'].values
sparse_matrix = csr_matrix((data, (row, col)))
```

```python
# Compute cosine similarity (sparse)
movie_similarity = cosine_similarity(sparse_matrix, dense_output=False)
```

```python
# Convert similarity matrix to DataFrame
movie_ids = df['movieId'].astype('category').cat.categories
movie_sim_df = pd.DataFrame(movie_similarity.toarray(), index=movie_ids, columns=movie_ids)
```

```python
# Generate recommendations for a user
user_id_example = 1
user_ratings = df[df['userId'] == user_id_example].sort_values(by='rating', ascending=False)
top_movies = user_ratings['movieId'].head(3).values
```

```python
similar_movies_list = set()
for movie in top_movies:
    similar_movies_list.update(movie_sim_df[movie].nlargest(6).iloc[1:].index)

unrated_movies = list(similar_movies_list - set(user_ratings['movieId']))
recommended_movies = movies_df[movies_df['movieId'].isin(unrated_movies)].head(5)
```

```python
print("\nRecommended movies for the user:")
print(recommended_movies)
```

Output

```
...
    Recommended movies for the user:
         movieId                          title                        genres
    43        47  Seven (a.k.a. Se7en) (1995)              Mystery|Thriller
    46        50   Usual Suspects, The (1995)        Crime|Mystery|Thriller
    76        85    Angels and Insects (1995)                 Drama|Romance
    257      296          Pulp Fiction (1994)  Comedy|Crime|Drama|Thriller
    284      326      To Live (Huozhe) (1994)                         Drama
```

Conclusion

This recommendation system demonstrates the power of collaborative filtering and cosine similarity in delivering personalized experiences to users. The system successfully recommended high-quality movies similar to those the user has already rated highly. This approach can be extended with hybrid models by incorporating metadata such as genres or using advanced techniques like matrix factorization.

Project 2: Discovering Heart Disease Patterns using Association Rule Mining

Introduction

This project explores the application of Association Rule Mining in a healthcare dataset. The primary goal is to discover patterns and relationships between different medical attributes that are associated with heart disease. By identifying combinations of symptoms and risk factors that frequently occur together, we can better understand the progression of the disease and support medical diagnosis.

The project uses the Apriori algorithm to extract frequent itemsets from a preprocessed version of a heart disease dataset and then generates association rules that meet specified thresholds for support and confidence.

Objectives

- To transform a medical dataset into a format suitable for rule mining.
- To apply one-hot encoding on categorical data for compatibility with the Apriori algorithm.
- To discover frequent itemsets with a minimum support of 0.3.
- To generate strong association rules with confidence greater than or equal to 0.7.
- To interpret the most significant rules in a medical context.

Dataset Description

The dataset used is `heart_disease.csv`, which includes 14 features such as:

- Age group (low, medium, high)
- Sex (0, 1)
- Chest pain type (`cp`)
- Blood pressure (`trestbps`)
- Cholesterol level (`chol`)

- Fasting blood sugar (fbs)
- Thalassemia type (thal)
- Heart disease diagnosis (target)

All features were originally categorical or converted into categories for mining.

Methodology

Data Encoding

Using pandas.get_dummies(), each categorical attribute was transformed into a binary (one-hot encoded) column, resulting in a DataFrame with 42 columns. This format is compatible with the Apriori algorithm.

Frequent Itemset Generation

Using mlxtend.frequent_patterns.apriori(), itemsets that occurred in at least 30% of the data were extracted. A total of 93 frequent itemsets were discovered.

Association Rule Generation

From these frequent itemsets, the association_rules() function generated rules with a minimum confidence threshold of 0.7. These rules were then sorted by both confidence and lift values to identify the most interesting ones.

Code

Importing libraries and dataset

Generate | Code | Markdown

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.frequent_patterns import apriori, association_rules
```

[15] Python

```python
from google.colab import drive
drive.mount('/content/drive')
```

[16] Python

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Connecting Dataset

```python
df = pd.read_csv('/content/drive/MyDrive/UITS/UITS /Eight Semester/Data Mining/Projects/Project 2/Dataset/heart_disease.csv')
df
```

[17] Python

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	high	1	3	high	medium	1	0	medium	0	high	0	0	1	1
1	low	1	2	medium	medium	0	1	high	0	high	0	0	2	1
2	low	0	1	medium	low	0	0	high	0	high	2	0	2	1
3	medium	1	1	low	medium	0	1	high	0	medium	2	0	2	1
4	medium	0	0	low	high	0	1	high	1	medium	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	medium	0	0	high	medium	0	1	low	1	medium	1	0	3	0
299	low	1	3	low	high	0	1	low	0	medium	1	0	3	0
300	high	1	0	high	low	1	1	low	0	high	1	2	3	0
301	medium	1	0	medium	low	0	1	low	1	medium	1	1	3	0
302	medium	0	1	medium	medium	0	0	high	0	low	1	1	2	0

303 rows × 14 columns

Encoding Dataset

```python
import pandas as pd

df_encoded = pd.get_dummies(df, columns=df.columns)
df_encoded
```

[18] Python

	age_high	age_low	age_medium	sex_0	sex_1	cp_0	cp_1	cp_2	cp_3	trestbps_high	...	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	thal_2	thal_3	target_0	target_1
0	True	False	False	False	True	False	False	False	True	True	...	False	False	False	False	False	True	False	False	False	True
1	False	True	False	False	True	False	False	True	False	False	...	False	False	False	False	False	False	True	False	False	True
2	False	True	False	True	False	False	True	False	False	False	...	False	False	False	False	False	False	True	False	False	True
3	False	False	True	False	True	False	True	False	False	False	...	False	False	False	False	False	False	True	False	False	True
4	False	False	True	True	False	True	False	False	False	False	...	False	False	False	False	False	False	True	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	False	False	True	True	False	True	False	False	False	True	...	False	False	False	False	False	False	False	True	True	False
299	False	True	False	False	True	False	False	False	True	False	...	False	False	False	False	False	False	False	True	True	False
300	True	False	False	False	True	True	False	False	False	True	...	False	True	False	False	False	False	False	True	True	False
301	False	False	True	False	True	True	False	False	False	False	...	True	False	False	False	False	False	False	True	True	False
302	False	False	True	True	False	False	True	False	False	False	...	True	False	False	False	False	False	True	False	True	False

303 rows × 42 columns

Frequent itemset generation

Generate + Code + Markdown

```python
frequent_itemsets = apriori(df_encoded, min_support=0.3, use_colnames=True)
print(frequent_itemsets)
```

[19] Python

```
     support                         itemsets
0   0.343234                      (age_high)
1   0.313531                       (age_low)
2   0.343234                    (age_medium)
3   0.316832                         (sex_0)
4   0.683168                         (sex_1)
..       ...                              ...
88  0.376238         (exang_0, thal_2, target_1)
89  0.336634            (thal_2, ca_0, target_1)
90  0.323432     (fbs_0, exang_0, ca_0, target_1)
91  0.330033  (thal_2, fbs_0, exang_0, target_1)
92  0.303630      (fbs_0, thal_2, ca_0, target_1)

[93 rows x 2 columns]
```

Generate + Code + Markdown

```python
for item in frequent_itemsets.iloc[90]:
    print(item)
```

[20] Python

```
0.3234323432343234
frozenset({'fbs_0', 'exang_0', 'ca_0', 'target_1'})
```

Association rule generation

```python
num_itemsets = len(frequent_itemsets)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7, num_itemsets = num_itemsets)
rules
```

[21]

Snipping Tool

Screenshot copied to clipboard

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski
0	(cp_0)	(sex_1)	0.471947	0.683168	0.343234	0.727273	1.064559	1.0	0.020815	1.161716	0.114844	0.422764	0.139205	0.614844
1	(sex_1)	(fbs_0)	0.683168	0.851485	0.574257	0.840580	0.987192	1.0	-0.007450	0.931593	-0.039337	0.597938	-0.073430	0.757499
2	(restecg_0)	(sex_1)	0.485149	0.683168	0.339934	0.700680	1.025633	1.0	0.008496	1.058506	0.048544	0.410359	0.055272	0.599132
3	(thal_3)	(sex_1)	0.386139	0.683168	0.336634	0.871795	1.276106	1.0	0.072836	2.471287	0.352467	0.459459	0.595353	0.682274
4	(target_0)	(sex_1)	0.455446	0.683168	0.376238	0.826087	1.209200	1.0	0.065092	1.821782	0.317703	0.493506	0.451087	0.688406
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
94	(fbs_0, ca_0, target_1)	(thal_2)	0.379538	0.547855	0.303630	0.800000	1.460241	1.0	0.095699	2.260726	0.507979	0.486772	0.557664	0.677108
95	(thal_2, ca_0, target_1)	(fbs_0)	0.336634	0.851485	0.303630	0.901961	1.059280	1.0	0.016992	1.514851	0.084361	0.343284	0.339869	0.629275
96	(thal_2, ca_0)	(fbs_0, target_1)	0.376238	0.468647	0.303630	0.807018	1.722016	1.0	0.127308	2.753375	0.672188	0.560976	0.636809	0.727452
97	(thal_2, target_1)	(fbs_0, ca_0)	0.429043	0.511551	0.303630	0.707692	1.383424	1.0	0.084153	1.671009	0.485423	0.476684	0.401559	0.650620
98	(ca_0, target_1)	(fbs_0, thal_2)	0.429043	0.481848	0.303630	0.707692	1.468704	1.0	0.096897	1.772625	0.558934	0.500000	0.435865	0.668915

99 rows × 14 columns

Generate the top 10 association rules based on confidence and lift

Generate   + Code   + Markdown

```python
rules = rules.sort_values(by=['confidence', 'lift'], ascending=False)
rules.head(10)
```

Python

...

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski
8	(trestbps_low)	(fbs_0)	0.333333	0.851485	0.313531	0.940594	1.104651	1.0	0.029703	2.500000	0.142105	0.359848	0.600000	0.654406
55	(thal_2, ca_0)	(fbs_0)	0.376238	0.851485	0.343234	0.912281	1.071399	1.0	0.022874	1.693069	0.106838	0.388060	0.409357	0.657691
95	(thal_2, ca_0, target_1)	(fbs_0)	0.336634	0.851485	0.303630	0.901961	1.059280	1.0	0.016992	1.514851	0.084361	0.343284	0.339869	0.629275
75	(thal_2, ca_0)	(target_1)	0.376238	0.544554	0.336634	0.894737	1.643062	1.0	0.131752	4.326733	0.627451	0.576271	0.768879	0.756459
45	(exang_0, ca_0)	(fbs_0)	0.432343	0.851485	0.386139	0.893130	1.048908	1.0	0.018005	1.389675	0.082141	0.430147	0.280407	0.673309
62	(slope_2, thal_2)	(exang_0)	0.336634	0.673267	0.300330	0.892157	1.325115	1.0	0.073686	3.029703	0.369854	0.423256	0.669935	0.669118
15	(ca_0)	(fbs_0)	0.577558	0.851485	0.511551	0.885714	1.040199	1.0	0.019769	1.299505	0.091482	0.557554	0.230476	0.743245
58	(ca_0, target_1)	(fbs_0)	0.429043	0.851485	0.379538	0.884615	1.038909	1.0	0.014214	1.287129	0.065594	0.421245	0.223077	0.665176
92	(fbs_0, thal_2, ca_0)	(target_1)	0.343234	0.544554	0.303630	0.884615	1.624476	1.0	0.116721	3.947195	0.585318	0.519774	0.746656	0.721096
16	(thal_2)	(fbs_0)	0.547855	0.851485	0.481848	0.879518	1.032922	1.0	0.015358	1.232673	0.070493	0.525180	0.188755	0.722705

```python
# Step 3: Sort rules by lift in descending order and select top 10
top_10_rules = rules.sort_values(by=["confidence", "lift"], ascending=False).head(10)

# Step 4: Display the top 10 rules
print(top_10_rules[['confidence', 'lift']])
```

[23]

Output

```
...
    confidence      lift
8     0.940594  1.104651
55    0.912281  1.071399
95    0.901961  1.059280
75    0.894737  1.643062
45    0.893130  1.048908
62    0.892157  1.325115
15    0.885714  1.040199
58    0.884615  1.038909
92    0.884615  1.624476
16    0.879518  1.032922
```

Interpretation

One significant insight from the rules is that patients with low resting blood pressure (trestbps_low) have a 94% probability of also having a normal fasting blood sugar level (fbs_0). This could help healthcare professionals in anticipating blood sugar conditions based on blood pressure readings. Similarly, the presence of specific thalassemia and calcium levels (thal_2, ca_0) was also predictive of healthy fasting glucose levels.

Conclusion

Association Rule Mining proved highly effective in identifying clinically meaningful relationships between medical attributes. These rules can serve as early indicators for heart disease screening and patient stratification. This method provides a strong foundation for further clinical decision support systems. Enhancements like numerical binning or hybrid algorithms like FP-Growth may be applied for improved performance.

Project 3: Domain-Specific Search Engine with Crawling and Link Analysis

Introduction

The third project simulates the design and operation of a simple, domain-specific search engine. Modern search engines rely heavily on web crawling, text indexing, and link analysis to rank results. This project focuses on crawling websites within a given domain (e.g., weather platforms), building an inverted index, constructing a link graph, and using PageRank to rank pages based on authority.

The goal is to understand and implement the fundamental building blocks of search engine architecture.

Objectives

- Crawl a set of webpages from weather-related domains.
- Extract and clean the text data for indexing.
- Build an inverted index to map keywords to URLs.
- Create a hyperlink graph representing the structure of crawled pages.
- Apply PageRank algorithm to determine page importance.
- Build a query system to search and return ranked results.

Methodology

Web Crawling

Python's requests and BeautifulSoup were used to perform web scraping. Starting from a list of seed URLs (e.g., weather.com, accuweather.com, etc.), the crawler explored internal links and captured raw text along with hyperlink relationships.

Text Cleaning and Tokenization

Each page's text content was cleaned by removing punctuation, converting to lowercase, and filtering out stopwords using NLTK. Tokens were used to populate an inverted index.

Inverted Index Construction

A dictionary was built where each keyword was mapped to the list of URLs where it appeared. This allowed efficient retrieval of relevant pages based on search queries.

Hyperlink Graph and PageRank

A directed graph (networkx.DiGraph) was created from the collected link structure. The PageRank algorithm was applied to assign a weight to each page based on its incoming links.

Search Interface

Given a user query, matching URLs were retrieved from the inverted index. These URLs were then ranked by their PageRank scores, and the highest-ranked ones were returned.

Code

```python
import requests
from bs4 import BeautifulSoup
```

Python

Stopwords are used when building the inverted index. The inverted index will ignore stopwords.

```python
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

STOPWORDS = stopwords.words('english')
print(STOPWORDS)
```

Python

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'and', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be', 'because', 'been', 'before', 'bei
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Add custom stopwords if you deem it necessary

```python
custom_STOPWORDS = [] # Add your own stopwords here
STOPWORDS.extend(custom_STOPWORDS)
```

Python

```python
from collections import defaultdict

# Inverted index: word -> set of URLs
inverted_index = defaultdict(set)
url_list = set()
```

Python

```python
# This dictionary will be used to build the connection between links
web_connection = {'source':[], 'target':[]}
```

Python

```python
import re

# This function will clean the content of web page in order to build the inverted index.
def clean_and_tokenize(text):
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text.lower())  # Remove punctuation and lowercase
    tokens = text.split()
    return [t for t in tokens if t not in STOPWORDS and len(t) > 1]
```

Python

```python
from urllib.parse import urljoin, urlparse

# The crawl function has 5 parameters
# url = The url to crawl
# base_domain = the base domain of the url. During crawling, the crawler will ignore links from other domains

def crawl(url, base_domain, visited, visit_limit, limit):
    if limit==0 or len(visited)==visit_limit:
        return

    try:
        response = requests.get(url, timeout=5)
        if response.status_code != 200:
            return
    except requests.RequestException:
        return

    visited.add(url)
    print("-"*(10-limit), end=" ")
    print(f"Crawled: {url}")

    soup = BeautifulSoup(response.text, 'html.parser')
```

```python
    soup = BeautifulSoup(response.text, 'html.parser')
    text = soup.get_text(separator=' ', strip=True)
    words = clean_and_tokenize(text)

    for word in words:
        inverted_index[word].add(url)
        url_list.add(url)

    # Recursively follow links
    for tag in soup.find_all('a', href=True):
        link = urljoin(url, tag['href'])
        parsed = urlparse(link)

        # Store external links as connection
        web_connection['source'].append(url)
        web_connection['target'].append(link)

        if parsed.netloc == base_domain and link not in visited:
```

Python

```python
def crawl_roots(root_urls, max_per_root=2, visit_limit=75):
    for root in root_urls:
        print(f"\nStarting crawl from: {root}")
        domain = urlparse(root).netloc
        visited = set()
        crawl(root, domain, visited, visit_limit, max_per_root)
```

Python

```python
seed_urls =  [
    'https://www.espn.com/soccer/',
    'https://www.goal.com',
    'https://www.skysports.com/football',
    'https://www.bbc.com/sport/football',
    'https://www.football365.com',
```

```python
seed_urls =  [
    'https://www.espn.com/soccer/',
    'https://www.goal.com',
    'https://www.skysports.com/football',
    'https://www.bbc.com/sport/football',
    'https://www.football365.com',
    'https://www.fourfourtwo.com',
    'https://www.theguardian.com/football',
    'https://www.cbssports.com/soccer/',
    'https://www.90min.com',
    'https://www.squawka.com',
    'https://www.fifa.com/news',
    'https://www.uefa.com/news/',
    'https://www.premierleague.com/news',
    'https://www.laliga.com/en-GB/news',
    'https://www.bundesliga.com/en/news',
    'https://www.seriea.com/en/news',
    'https://www.mlssoccer.com/news/',
    'https://www.afc.com/news',
    'https://onefootball.com/en/news',
    'https://www.football-italia.net'
]

crawl_roots(seed_urls, max_per_root=10)
```

Python

```
Starting crawl from: https://www.espn.com/soccer/

Starting crawl from: https://www.goal.com
 Crawled: https://www.goal.com
- Crawled: https://www.goal.com/en-us
-- Crawled: https://www.goal.com/en-us/live-scores
--- Crawled: https://www.goal.com/en-us/news
---- Crawled: https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks
----- Crawled: https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db
------ Crawled: https://www.goal.com/en-us/category/analysis/1/blt0e4843c7e245b533
------- Crawled: https://www.goal.com/en-us/category/power-rankings/1/blt262ce0e5159ea8fe
-------- Crawled: https://www.goal.com/en-us/category/player-ratings/1/blt9e3963966f918671
```

```
Starting crawl from: https://www.espn.com/soccer/

Starting crawl from: https://www.goal.com
 Crawled: https://www.goal.com
- Crawled: https://www.goal.com/en-us
-- Crawled: https://www.goal.com/en-us/live-scores
--- Crawled: https://www.goal.com/en-us/news
---- Crawled: https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks
----- Crawled: https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db
------ Crawled: https://www.goal.com/en-us/category/analysis/1/blt0e4843c7e245b533
------- Crawled: https://www.goal.com/en-us/category/power-rankings/1/blt262ce0e5159ea8fe
-------- Crawled: https://www.goal.com/en-us/category/player-ratings/1/blt9e3963966f918671
--------- Crawled: https://www.goal.com/en-us/category/winners-and-losers/1/blt05e54ed95ba7b0f8
--------- Crawled: https://www.goal.com/goalchampions
--------- Crawled: https://www.goal.com/goaleditions/2/index.html
--------- Crawled: https://www.goal.com/en-us/category/culture/1/edzsmxzmp4y81pp4ozi9lrdeh
--------- Crawled: https://www.goal.com/en-us/category/kits/1/bltcdcab888b4154243
--------- Crawled: https://www.goal.com/en-us/category/boots/1/blt6917022f6674df56
--------- Crawled: https://www.goal.com/en-us/category/shopping/1/jay1lq18ea7j1bks90cjvzor9
--------- Crawled: https://www.goal.com/en-us/category/Gaming/1/1sv3aw5srmki010flis5dpyze1
--------- Crawled: https://www.goal.com/en-us/major-league-soccer/287tckirbfj9nb8ar2k9r60vn
--------- Crawled: https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5
--------- Crawled: https://www.goal.com/en-us/laliga/34pl8szyvrbwcmfkuocjm3r6t
--------- Crawled: https://www.goal.com/en-us/serie-a/1r097lpxe0xn03ihb7wi98kao
...
------- Crawled: https://www.football-italia.net/category/news/#content
-- Crawled: https://www.football-italia.net#tan-main-banner-latest-trending-popular-recent
--- Crawled: https://www.football-italia.net#tan-main-banner-latest-trending-popular-popular
---- Crawled: https://www.football-italia.net#tan-main-banner-latest-trending-popular-categorised
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
# print inverted index
print("\nSample inverted index (first 20 words):")
for word in list(inverted_index.keys())[:20]:
    print(f"{word}: {list(inverted_index[word])}")
```

Python


```python
# print inverted index
print("\nSample inverted index (first 20 words):")
for word in list(inverted_index.keys())[:20]:
    print(f"{word}: {list(inverted_index[word])}")
```

Python

...

```
Sample inverted index (first 20 words):
soccer: ['https://www.theguardian.com/science', 'https://www.theguardian.com/podcasts', 'https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://w
news: ['https://www.theguardian.com/science', 'https://www.theguardian.com/podcasts', 'https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://www
live: ['https://www.theguardian.com/science', 'https://www.theguardian.com/podcasts', 'https://www.bundesliga.com/en/bundesliga/news/tobias-rau-interview-bayern-munich-g
scores: ['https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://www.cbssports.com/nfl/schedule/', 'https://www.laliga.com/en-GB/futbol-femenino/
results: ['https://www.football-italia.net/#tan-main-banner-latest-trending-popular-recent', 'https://www.bundesliga.com/en/bundesliga/news/tobias-rau-interview-bayern-m
transfers: ['https://www.theguardian.com/podcasts', 'https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://www.football-italia.net/#tan-main-bar
goalcom: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.goal.com/en-us/player/l-yamal/abr79wsl0folgkyvl821ggs2c', 'https://www.goal
us: ['https://www.theguardian.com/science', 'https://www.theguardian.com/podcasts', 'https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://www.p
latest: ['https://www.theguardian.com/science', 'https://www.theguardian.com/podcasts', 'https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://w
features: ['https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://www.bundesliga.com/en/bundesliga/news/tobias-rau-interview-bayern-munich-germa
opinion: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.theguardian.com/science', 'https://www.theguardian.com/podcasts', 'https:/
analysis: ['https://www.theguardian.com/science', 'https://www.theguardian.com/podcasts', 'https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https:/
power: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.football365.com/brighton', 'https://www.premierleague.com/en/footballandcommu
rankings: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.mlssoccer.com/news/topics/voices-sam-jones/', 'https://www.cbssports.com/b
player: ['https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://www.bundesliga.com/en/bundesliga/news/tobias-rau-interview-bayern-munich-germany
ratings: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe', 'https://www.goal
winners: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.football365.com/brighton', 'https://www.theguardian.com/podcasts', 'https:/
losers: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.football365.com/brighton', 'https://www.theguardian.com/podcasts', 'https:/
goal: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.fourfourtwo.com/news/manchester-united-to-receive-huge-transfer-windfall-follo
usa: ['https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5', 'https://www.football365.com/brighton', 'https://www.premierleague.com/en/events/summerserie
```

```python
# Print first 20 connections

for source, target in list(zip(web_connection['source'], web_connection['target']))[:20]:
    print(f"{source} -> {target}")
```

Python

```
https://www.goal.com -> https://www.goal.com/en-us
https://www.goal.com/en-us -> https://www.goal.com/en-us
https://www.goal.com/en-us -> https://www.goal.com/en-us/live-scores
https://www.goal.com/en-us/live-scores -> https://www.goal.com/en-us
https://www.goal.com/en-us/live-scores -> https://www.goal.com/en-us/live-scores
https://www.goal.com/en-us/live-scores -> https://www.goal.com/en-us/news
https://www.goal.com/en-us/news -> https://www.goal.com/en-us
https://www.goal.com/en-us/news -> https://www.goal.com/en-us/live-scores
https://www.goal.com/en-us/news -> https://www.goal.com/en-us/news
https://www.goal.com/en-us/news -> https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks
https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks -> https://www.goal.com/en-us
https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks -> https://www.goal.com/en-us/live-scores
https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks -> https://www.goal.com/en-us/news
https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks -> https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks
https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks -> https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db
https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db -> https://www.goal.com/en-us
https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db -> https://www.goal.com/en-us/live-scores
https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db -> https://www.goal.com/en-us/news
https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db -> https://www.goal.com/en-us/category/transfers/1/k94w8e1yy9ch14mllpf4srnks
https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db -> https://www.goal.com/en-us/category/opinion/1/bltda2eefda7fac61db
```

```python
import networkx as nx

web_graph = nx.DiGraph()
for source, target in zip(web_connection['source'], web_connection['target']):
    web_graph.add_edge(source, target) # Add edges for individual source-target pairs
```

Python

```python
len(web_graph.nodes)
```

Python

```
20971
```

```python
pagerank_scores = nx.pagerank(web_graph, alpha=0.85, max_iter=100, tol=1e-6)
print("\nPageRank Scores:", pagerank_scores)
```

Python

```python
def search_engine(query, index, scores):
    query_terms = query.lower().split()
    results = set()
    for term in query_terms:
        if term in index:
            if not results:
                results = set(index[term])
            else:
                results = results.intersection(index[term])  # Find common websites

    # Sort results based on score
    ranked_results = []
    for website in results:
        if website in scores:
          ranked_results.append((website, scores[website]))
    ranked_results.sort(key=lambda x: x[1], reverse=True)

    return ranked_results
```

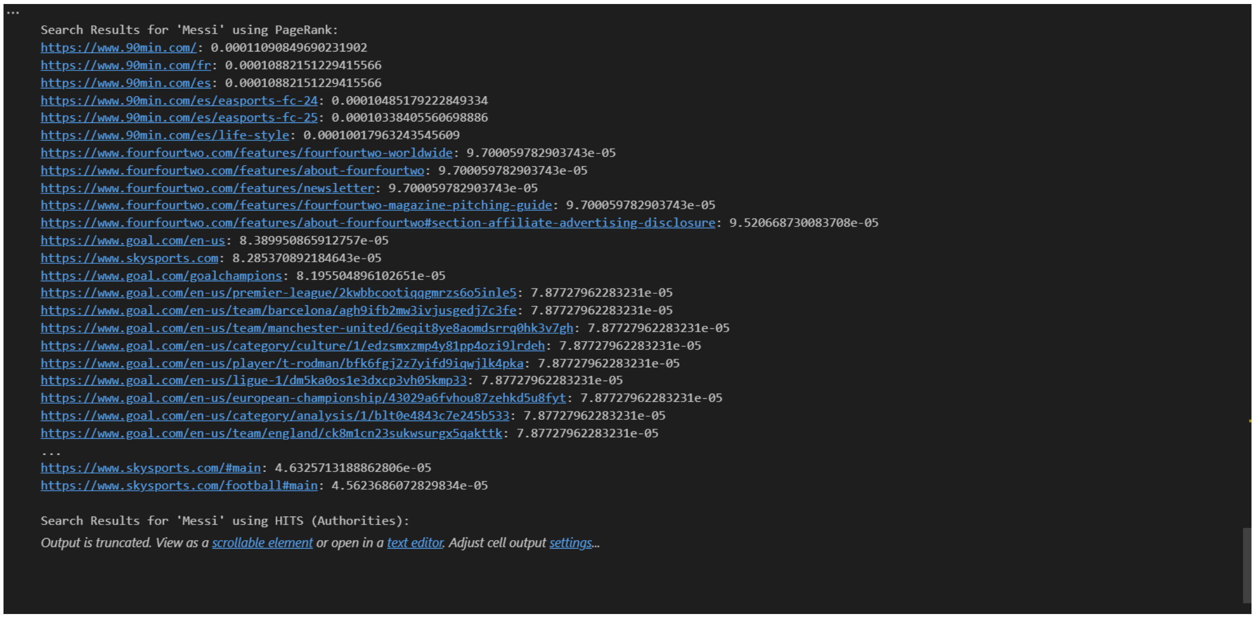
Python

```python
# Query and display results
query = "Messi"
print(f"\nSearch Results for '{query}' using PageRank:")
results = search_engine(query, inverted_index, pagerank_scores) # Changed 'index' to 'inverted_index'

for page, score in results:
    print(f"{page}: {score}")  # Removed web_content[page] as web_content is not defined


print(f"\nSearch Results for '{query}' using HITS (Authorities):")
# Calculate HITS scores if needed
# authorities = nx.hits(web_graph)[1]  # Uncomment if you have HITS scores calculated
# results = search_engine(query, inverted_index, authorities)  # Uncomment if you have HITS scores calculated

# Placeholder for HITS results
# for page, score in results:
#     print(f"{page}: {score}") # Removed web_content[page] as web_content is not defined
```

[139]

Python

Output



```
...
Search Results for 'Messi' using PageRank:
https://www.90min.com/: 0.00011090849690231902
https://www.90min.com/fr: 0.00010882151229415566
https://www.90min.com/es: 0.00010882151229415566
https://www.90min.com/es/easports-fc-24: 0.00010485179222849334
https://www.90min.com/es/easports-fc-25: 0.00010338405560698886
https://www.90min.com/es/life-style: 0.00010017963243545609
https://www.fourfourtwo.com/features/fourfourtwo-worldwide: 9.700059782903743e-05
https://www.fourfourtwo.com/features/about-fourfourtwo: 9.700059782903743e-05
https://www.fourfourtwo.com/features/newsletter: 9.700059782903743e-05
https://www.fourfourtwo.com/features/fourfourtwo-magazine-pitching-guide: 9.700059782903743e-05
https://www.fourfourtwo.com/features/about-fourfourtwo#section-affiliate-advertising-disclosure: 9.520668730083708e-05
https://www.goal.com/en-us: 8.389950865912757e-05
https://www.skysports.com: 8.285370892184643e-05
https://www.goal.com/goalchampions: 8.195504896102651e-05
https://www.goal.com/en-us/premier-league/2kwbbcootiqqgmrzs6o5inle5: 7.87727962283231e-05
https://www.goal.com/en-us/team/barcelona/agh9ifb2mw3ivjusgedj7c3fe: 7.87727962283231e-05
https://www.goal.com/en-us/team/manchester-united/6eqit8ye8aomdsrrq0hk3v7gh: 7.87727962283231e-05
https://www.goal.com/en-us/category/culture/1/edzsmxzmp4y81pp4ozi9lrdeh: 7.87727962283231e-05
https://www.goal.com/en-us/player/t-rodman/bfk6fgj2z7yifd9iqwjlk4pka: 7.87727962283231e-05
https://www.goal.com/en-us/ligue-1/dm5ka0os1e3dxcp3vh05kmp33: 7.87727962283231e-05
https://www.goal.com/en-us/european-championship/43029a6fvhou87zehkd5u8fyt: 7.87727962283231e-05
https://www.goal.com/en-us/category/analysis/1/blt0e4843c7e245b533: 7.87727962283231e-05
https://www.goal.com/en-us/team/england/ck8m1cn23sukwsurgx5qakttk: 7.87727962283231e-05
...
https://www.skysports.com/#main: 4.6325713188862806e-05
https://www.skysports.com/football#main: 4.5623686072829834e-05

Search Results for 'Messi' using HITS (Authorities):
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Conclusion

This project successfully replicated a miniature version of a search engine for a specific domain. It integrated crawling, indexing, and link analysis to return ranked search results. The use of PageRank allowed the engine to prioritize authoritative sources, while keyword indexing ensured relevance. Future expansions may include TF-IDF scoring, multi-word phrase matching, and HITS algorithm support.

GitHub Links

- **Project 1:** https://github.com/arbinzaman/Data-Mining/blob/main/Projects/Project%201/Project_1_Arbin_2125051006.ipynb
- **Project 2:** https://github.com/arbinzaman/Data-Mining/blob/main/Projects/Project%202/project_2_Generate_top_10_Reasons_For_Heart_Disease_using_Association_Rule_Mining.ipynb
- **Project 3:** https://github.com/arbinzaman/Data-Mining/blob/main/Projects/Project%203/project_3_Domain_Specific_Search_Engine_with_Crawling_and_Link_Analysis.ipynb