Content-Based Filtering in News Aggregation

Submitted in partial fulfillment of the requirements degree

BACHELOR OF ENGINEERING IN COMPUTER ENGINEERING

By

Name	Roll No
Adnan Shaikh	211251
Rahil Shaikh	211253
Farhan Haider Rizvi	211408
Khushi Tripathi	312058

Guided By

Prof. Ahlam Ansari

Department of Computer Engineering
M. H. Saboo Siddik College of Engineering, Mumbai
University of Mumbai

(AY 2024-2025)

ACKNOWLEDGMENT

We would like to express our gratitude and appreciation to our parents for motivating and encouraging us throughout the career. We wish to express our sincere thanks to our Director Dr. Mohiuddin Ahmed and our Principal Dr. Ganesh Kame, M.H. Saboo Siddik College of Engineering for providing us all the facilities, support, and wonderful environment to meet our project requirements. We would also take the opportunity to express our humble gratitude to our Head of the Department of Computer Engineering Dr.Ahmed Shaikh for supporting us in all aspects and for encouraging us with her valuable suggestions to make our project successful. We are highly thankful to our internal course incharge Prof. Ahlam Ansari whose valuable guidance helped us understand the project better, her constant guidance and willingness to share her vast knowledge made us understand this project and its manifestations in great depth and helped us to complete the project successfully.

We would also like to acknowledge with much appreciation the role of the Computer Department staff, especially the Laboratory staff, who permitted us to use the labs when needed and the necessary material to complete the project. We would like to express our gratitude and appreciate the guidance given by other supervisors and project guides, their comments and tips helped us in improving our presentation skills.

TABLE OF CONTENTS

LIST OF FIGURES

I. Introduction

Content-Based Filtering is a recommendation technique that suggests items (such as news articles, products, or movies) based on the characteristics or attributes of the items and the user's previous preferences or interactions. In content-based filtering, the system analyzes the content of items (such as text, metadata, tags, or keywords) and compares it to the user's profile to make personalized recommendations.

In the digital age, the overwhelming volume of available news articles poses a significant challenge for users seeking relevant and personalized content. This project presents a content-based news recommendation system that leverages user preferences and article characteristics to provide tailored news suggestions .The system utilizes Natural Language Processing NLP techniques like stemming, tokenization, and stopword removal to preprocess news articles. The core of the recommendation mechanism relies on a Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer and cosine similarity. These enable the system to capture the content of news articles and recommend similar articles to users based on the textual features of the news they have selected. The application is built with an interactive user interface using Streamlit, providing a user-friendly experience where users can select a news article and receive recommendations of similar articles. This system aims to improve user engagement by delivering relevant news content tailored to their interests.

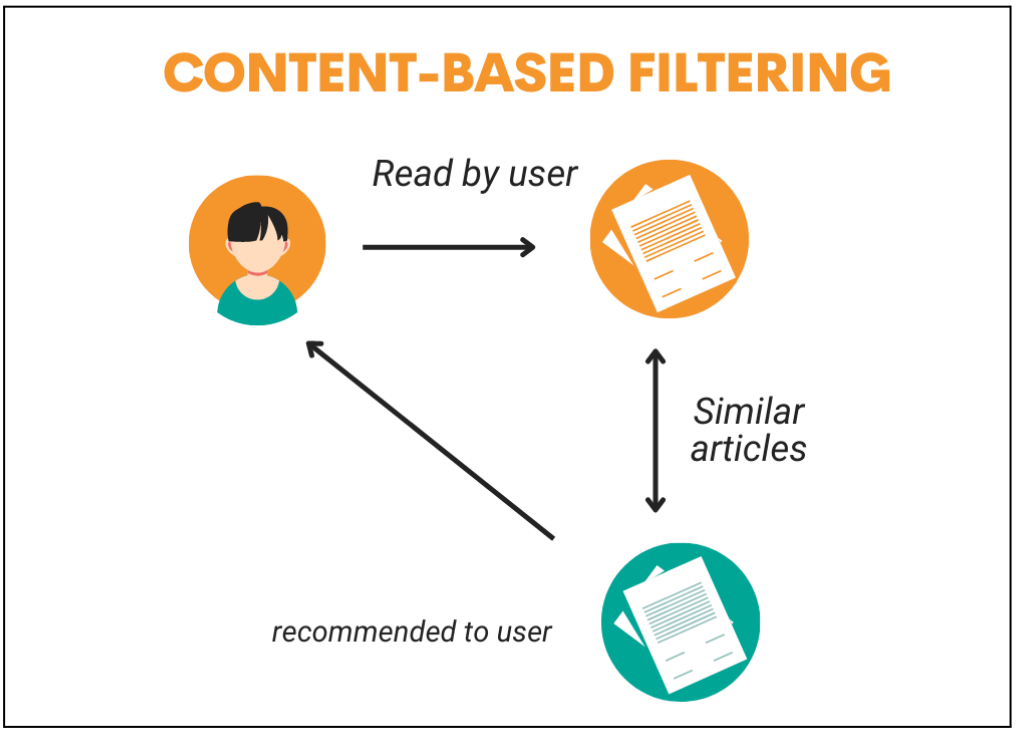

Fig 1. Content-Based Filtering

II. Implementation

This project focuses on developing a news recommendation system using content-based filtering. The system allows users to select a news article, and based on its content, it recommends similar articles to the user. The system is implemented using Python, Streamlit for the front-end interface, and a variety of machine learning and natural language processing (NLP) techniques.

Dataset

The dataset used for this project contains news articles, each consisting of a title, description, category, date, and URL. The description field is particularly important, as it forms the basis for the content-based recommendation system. We load this data using the pandas library, which allows for efficient handling and manipulation of structured data.

df = pd.read_csv("0_news_articles.csv")

In this step, we load the CSV file into a pandas DataFrame.

Text Preprocessing

To ensure accurate recommendations, the text of each news article (specifically, the Description field) is preprocessed. Text preprocessing involves several steps: converting all text to lowercase, removing stopwords (common words like "the", "is", etc., which do not provide meaningful information), stemming words (reducing them to their root form), and tokenizing the text.

We use the Natural Language Toolkit (NLTK) for these operations. This ensures that the text is cleaned and standardized, making it easier to compute meaningful similarities between articles.

```
def preprocess_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)
    text = [ps.stem(word) for word in text if word.isalnum() and word not in stop_words]
    return " ".join(text)
```

Here, the preprocess_text function performs the following:

1. Lowercasing: It converts all characters to lowercase to avoid treating the same word differently due to case.
2. Tokenization: It splits the text into individual words using NLTK's word_tokenize.
3. Stopword removal: It removes non-informative words (stopwords) using the NLTK stopword list.
4. Stemming: It reduces each word to its base form using the Porter Stemmer.

This processed text is stored in a new column processed_content for each article.

df['processed_content'] = df['Description'].apply(preprocess_text)

Feature Extraction Using TF-IDF

Once the text is preprocessed, we need to convert the text data into a numerical format that can be used to compute similarities. This is done using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer. TF-IDF assigns a weight to each word in a document based on how important that word is relative to the entire corpus.

tfidf = TfidfVectorizer(max_features=5000)
tfidf_matrix = tfidf.fit_transform(df['processed_content'])

In this step:

- The TfidfVectorizer is limited to the top 5000 most important words (max_features=5000).
- The matrix produced by this vectorizer is sparse and represents the frequency of each word across all documents.

This matrix is crucial for comparing articles, as it allows us to compute the similarity between any two articles based on their content.

Cosine Similarity

With the TF-IDF matrix, we can now compute the similarity between articles using cosine similarity. Cosine similarity measures the angle between two vectors (in this case, the TF-IDF representations of two articles) to determine how similar they are. A smaller angle indicates a higher degree of similarity.

similarity_matrix = cosine_similarity(tfidf_matrix)

This step produces a similarity matrix where each element at position (i, j) represents the similarity between article i and article j. The diagonal elements (where i == j) will have a similarity of 1 since an article is always identical to itself.

Recommendation Function

With the similarity matrix in hand, we define a function to generate recommendations. The function takes an article index as input, retrieves its similarity scores from the matrix, and sorts them to identify the most similar articles.

```
def recommend_news(article_index, num_recommendations=5):
    similarity_scores = list(enumerate(similarity_matrix[article_index]))
    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)
    similar_articles = similarity_scores[1:num_recommendations+1]
    recommended_indices = [i[0] for i in similar_articles]
    return df.iloc[recommended_indices][['Title', 'Description', 'Date', 'Category', 'URL']]
```

This function:

1. Retrieves the similarity scores for the chosen article.
2. Sorts the articles based on their similarity score, in descending order.
3. Returns a specified number (num_recommendations) of the most similar articles, excluding the selected article itself.

Streamlit Interface

The user interface is created using Streamlit, a Python framework that makes it easy to build interactive web applications. The layout is designed with two columns: one for article selection and another for displaying recommended articles.

col1, col2 = st.columns(2)

In the first column (col1), users select an article from a dropdown menu. The system then displays the article's title, date, category, and content.

selected_title = st.selectbox("Choose a news article to get recommendations:", article_titles)
article_index = df[df['Title'] == selected_title].index[0]
st.write(f"{df.iloc[article_index]['Title']}")

In the second column (col2), the system displays a list of recommended articles, showing the title, date, category, and a brief description for each recommendation.

recommended_articles = recommend_news(article_index)
for index, row in recommended_articles.iterrows():
    st.write(f"Title: {row['Title']}")

Deploying the System

For external access during development, we use ngrok, which creates a secure tunnel to the locally running Streamlit app.

public_url = ngrok.connect(8501, "http")

III. Modules Used

Streamlit :

- **Description**: Streamlit is a Python library used to build interactive web applications with ease. It allows developers to create user interfaces, visualize data, and handle user inputs with minimal code.
- **Usage in code**: Used to create the web interface, display the selected article, and show recommended articles in the UI.

Fig 2. Streamlit

Pandas :

- **Description**: Pandas is a powerful library for data manipulation and analysis in Python. It provides data structures like DataFrame, making it easy to manage and analyze structured data.
- **Usage in code**: Used to load the dataset (`news_articles.csv`) and handle data processing such as displaying the article information.

Fig 3. Pandas

Natural Language Toolkit (`nltk`):

- **Description**: NLTK is a library used for working with human language data (text). It provides tools for tasks like tokenization, stemming, and stopword removal.
-
- **Usage in code**: Used to preprocess text data (stemming, tokenization, and stopword removal) to make the content ready for similarity-based recommendations.

Fig 4. NLTK

Scikit-learn (`sklearn`):

- **Description**: Scikit-learn is a machine learning library that provides simple and efficient tools for data mining and data analysis. It includes utilities for vectorization and similarity measurement.
- **Usage in code**:
  - **TfidfVectorizer**: Converts text into a matrix of TF-IDF features for calculating the importance of words.
  - **cosine_similarity**: Computes the cosine similarity between vectors to measure similarity between articles.

Fig 5. Scikit learn

Pyngrok (`pyngrok`):

- **Description**: Pyngrok is a library that provides access to the ngrok service, which can create secure tunnels to localhost. It is often used to expose local development environments to the internet.
- **Usage in code**: Used to expose the local Streamlit app to the internet through a public URL for easy access.

Fig 6. pyngrok

IV. Applications

1. Personalized News Recommendation

- **News Platforms**: Your system can be integrated into news platforms (e.g., Google News, Flipboard) to provide personalized news suggestions to users based on their reading history and preferences. This helps users discover relevant content and improves user engagement.

2. E-Commerce

- **Product Recommendation Content:** E-commerce platforms can leverage the system to recommend product-related articles, blogs, or reviews that align with a user's interests or previous purchases.

3. Educational Platforms

- **Academic Research Tools**: The recommendation engine can be used in academic platforms like Google Scholar to recommend research papers, articles, and journals based on a researcher’s previous readings or interests.

4. Social Media

- **Content Feeds:** Social media platforms like Facebook or Twitter can use this to suggest articles or posts in a user’s feed based on their interactions with similar content.

5. Media and Entertainment

- **Article or Blog Recommendations**: Media companies can offer personalized recommendations on blogs, opinion pieces, or features that align with user preferences.

V. Code

```
import streamlit as st
import pandas as pd
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from pyngrok import ngrok
# Load and preprocess the dataset
df = pd.read_csv("0_news_articles.csv")
nltk.download('punkt_tab')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
stop_words = set(stopwords.words('english'))
# Text preprocessing function
def preprocess_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)
    text = [ps.stem(word) for word in text if word.isalnum() and word not in stop_words]
    return " ".join(text)
df['processed_content'] = df['Description'].apply(preprocess_text)
# Vectorize the content
tfidf = TfidfVectorizer(max_features=5000)
tfidf_matrix = tfidf.fit_transform(df['processed_content'])
# Compute similarity matrix
similarity_matrix = cosine_similarity(tfidf_matrix)
# Define recommendation function
def recommend_news(article_index, num_recommendations=5):
    similarity_scores = list(enumerate(similarity_matrix[article_index]))
    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)
    similar_articles = similarity_scores[1:num_recommendations+1]
    recommended_indices = [i[0] for i in similar_articles]

    # Return the title, content, date, category, and URL
    return df.iloc[recommended_indices][['Title', 'Description', 'Date', 'Category', 'URL']]
# Enhanced Streamlit UI Design
st.set_page_config(page_title="News Aggregator", layout="wide")
st.markdown("<h1 style='text-align: center; color: #4CAF50;'>📰 News Aggregation
System</h1>", unsafe_allow_html=True)
st.markdown("<h2 style='text-align: center;'>Content-Based Filtering for News
Recommendations</h2>", unsafe_allow_html=True)
# Create a layout with two columns: one for article selection, one for displaying results
```

```
col1, col2 = st.columns(2)
# News article selection in the first column
with col1:
    st.subheader("Select an Article")
    article_titles = df['Title'].tolist()
    selected_title = st.selectbox("Choose a news article to get recommendations:",
article_titles)

    # Display the selected article's content
    article_index = df[df['Title'] == selected_title].index[0]
    st.write("### Selected Article")
    st.write(f"{df.iloc[article_index]['Title']}")
    st.write(f"Date: {df.iloc[article_index]['Date']}")
    st.write(f"Category: {df.iloc[article_index]['Category']}")
    st.write(f"URL: {df.iloc[article_index]['URL']}")
    st.write(df.iloc[article_index]['Description'])
# Recommendations in the second column
with col2:
    st.subheader("Recommended Articles")
    recommended_articles = recommend_news(article_index)

    for index, row in recommended_articles.iterrows():
        st.write(f"Title: {row['Title']}")
        st.write(f"Date: {row['Date']}")
        st.write(f"Category: {row['Category']}")
        st.write(f"URL: {row['URL']}")
        st.write(row['Description'])
        st.write("---")
# Footer
st.markdown(
    """
    <style>
        footer {visibility: hidden;}
        .footer-text {
            position: fixed;
            left: 0;
            bottom: 0;
            width: 100%;
            background: linear-gradient(to right, #00c6ff, #0072ff);
            color: white;
            text-align: center;
            padding: 20px;
            font-family: 'Arial', sans-serif;
            font-size: 16px;
```

```
            box-shadow: 0px -3px 5px rgba(0,0,0,0.2);
        }
        .footer-text p {
            margin: 0;
            padding: 0;
        }
        .footer-text a {
            color: #ffcc00;
            text-decoration: none;
            font-weight: bold;
        }
        .footer-text a:hover {
            color: #fff;
            text-decoration: underline;
        }
    </style>
    <div class="footer-text">
        <p>Powered by <a href="https://streamlit.io" target="_blank">Streamlit</a> |
Content-Based Filtering System</p>
    </div>
    """, unsafe_allow_html=True
)
# Start ngrok tunnel on port 8501
public_url = ngrok.connect(8501, "http")
print("Streamlit App URL:", public_url)
```

VI. Output

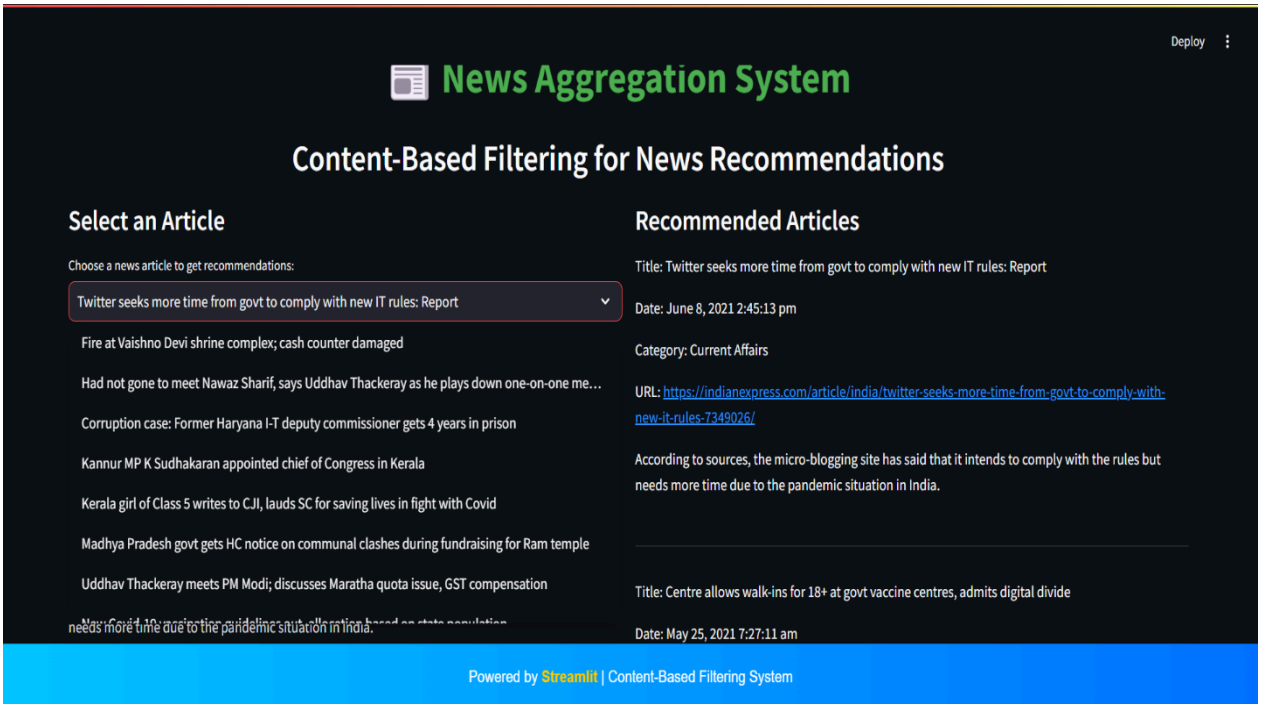

Fig 7. Output SS-1

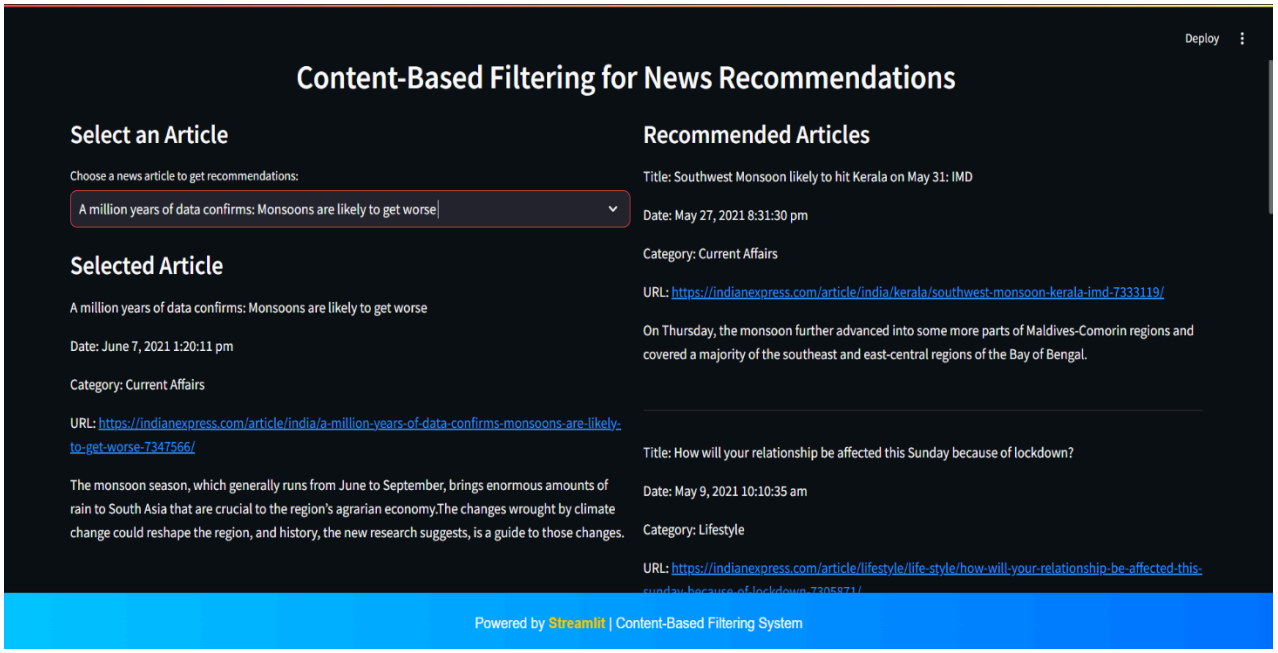

Fig 8. Output SS-2

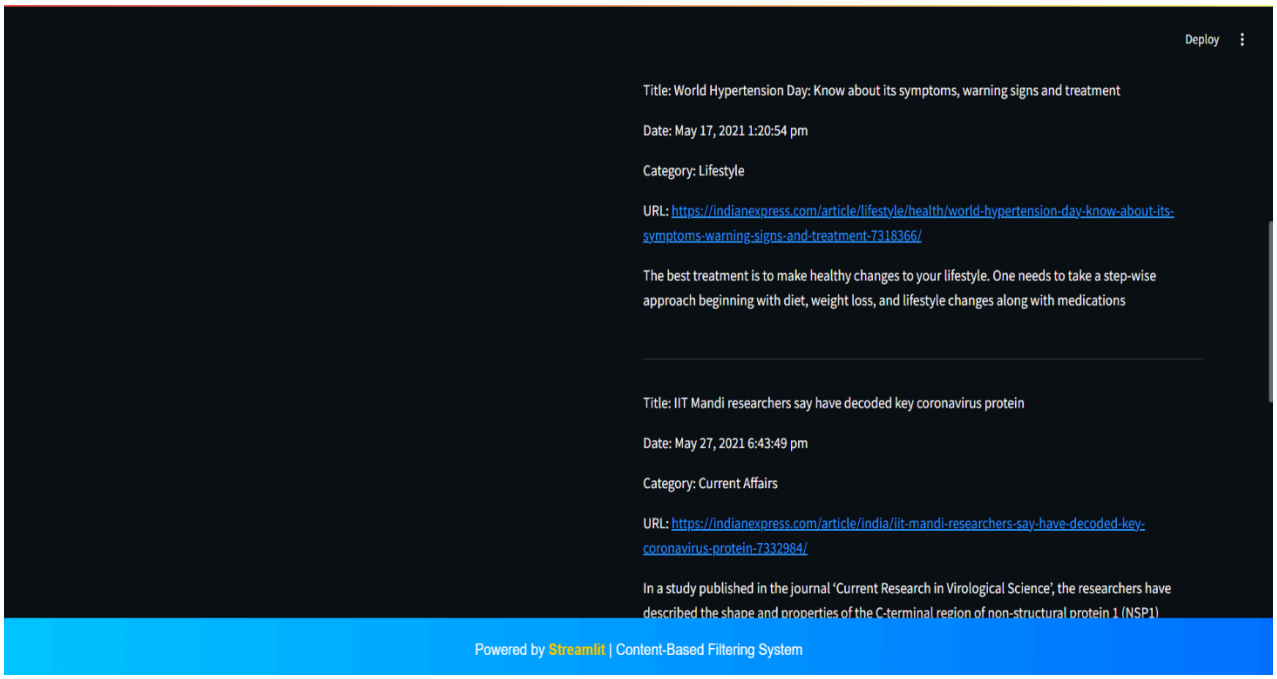

Fig 9. Output SS-3

VII. Conclusion

In summary, the development of a content-based news recommendation system marks a significant stride towards addressing the challenges associated with information overload in the digital news landscape. Through the application of advanced natural language processing techniques and machine learning algorithms, the system is able to analyze and interpret the content of news articles, offering users personalized recommendations that align with their unique preferences. The results indicate that users benefit from a more streamlined and relevant news experience, leading to increased engagement and satisfaction with the content consumed.

VIII. References

[1] StrataScratch. "Step-by-Step Guide to Building Content-Based Filtering." [Online]. Available: https://www.stratascratch.com/blog/step-by-step-guide-to-building-content-based-filtering/. [Accessed: 29-Sept-2024].

[2] J. Mahadevan. "API Chart Visualisation with Python Pandas - FAUN—Developer Community," [Online]. Available: https://faun.pub/create-chart-visualisation-with-python-pandas-using-apis-in-5-steps-eefaf42f3f64. [Accessed: 29-Sept-2024].

[3] S. Ai. "Sentiment Analysis — using NLTK Vader - Skillcate AI," [Online]. Available: https://medium.com/@skillcate/sentiment-analysis-using-nltk-vader-98f67f2e6130. [Accessed: 29-Sept-2024].

[4] pyngrok. "pyngrok."[Online]. Available: https://pypi.org/project/pyngrok/. [Accessed: 29-Sept-2024].

[5] DataCamp. "A Step-by-Step Guide to Streamlit." [Online]. Available: https://www.datacamp.com/tutorial/streamlit. [Accessed: 29-Sept-2024].