Outlook Recommendation Engine Project

# Project Overview Introduction

The Article Recommendation System is a web application that leverages Natural Language Processing (NLP) techniques to provide personalized article recommendations to users. The system aims to enhance user engagement by presenting relevant and interesting articles based on their preferences and past reading history.

## Objectives and Goals

The main objectives of the project are as follows:

1. Personalized Recommendations: Provide users with personalized article recommendations based on their interests, reading history, and behaviour on the platform.
2. Article Similarity: Implement NLP algorithms, such as BERT and TF-IDF, to calculate article similarity and identify related articles.
3. Subcategory Extraction: Utilize NLP techniques to automatically extract subcategories from articles, enhancing the granularity of article classification.
4. Efficient Search: Enable users to search for articles by subcategory, title, or keyword to find articles of interest quickly.
5. User Interface: Develop an intuitive and user-friendly web interface to facilitate seamless interactions with the application.
6. Scalability: Design the system to handle a large volume of articles and users efficiently.

## Scope and Limitations

The scope of the Article Recommendation System includes the following aspects:

1. Article Collection: The system will collect article data from various online sources, focusing on news articles from diverse categories.
2. NLP Techniques: The system will utilize NLP techniques like BERT and TF-IDF to analyze and process article content for recommendation and subcategory extraction.
3. User Profiles: User profiles will be created to store preferences, reading history, and other relevant information for personalized recommendations.
4. Article Recommendation: The system will provide personalized article recommendations based on user profiles and article similarity scores.
5. Web Scraping: A web scraping module will be developed to extract article content and metadata from external websites.
6. Database: The system will use an SQL database to store article data, user profiles, and recommendation history.

However, the project has some limitations:

1. Real-time Updates: The system will not provide real-time updates of articles, and new articles may not be immediately available for recommendation.
2. User Data Privacy: While the system will store user preferences and reading history, measures will be taken to ensure data privacy and security.
3. Language Support: The initial version of the system will focus on English language articles, limiting multilingual support.
4. Content Filtering: The system will not implement content filtering to restrict specific types of articles.

The Article Recommendation System aims to offer users an engaging and tailored reading experience by harnessing the power of NLP and intelligent recommendation algorithms. With a user-friendly interface and a robust recommendation engine, the application aims to become a go-to platform for discovering relevant and captivating articles from a wide range of categories.

# Technologies Used

## Front-End Technologies

1. **HTML, CSS, JavaScript, PHP:** The front-end of the application is built using these core web technologies to create the user interface and provide an interactive user experience.
2. **React.js:** A popular JavaScript library used to build the dynamic user interface and manage the application's state effectively.

## Back-End Technologies

1. **Node.js:** A JavaScript runtime used on the server-side to execute server code, handle requests, and build APIs.
2. **Express.js:** A minimalistic web application framework for Node.js, used to build robust and scalable APIs.
3. **Python:** Used to implement natural language processing (NLP) functionalities, including BERT-based article similarity and subcategory extraction.

## Database Technologies

1. **MySQL:** A relational database management system (RDBMS) used to store and manage article data efficiently.

## Natural Language Processing (NLP) Technologies

1. spaCy: An NLP library used for text processing tasks, such as tokenization, named entity recognition, and part-of-speech tagging.
2. Hugging Face Transformers: A popular library for utilizing pre-trained transformer models like BERT for natural language understanding tasks.

# Data Collection and Preprocessing

## Data Collection

The article data for the project was obtained directly from Outlook India's webpages. We accessed the HTML content of the articles and extracted the required information, such as the article title, text, meta tags, and other relevant details.

The data collection process involved accessing the articles programmatically using HTTP requests and parsing the HTML content to extract the desired information. The data collection was focused on articles from various categories, including National, Sports, Business, Entertainment, and more, to ensure a diverse dataset.

## Data Preprocessing

Before using the collected article data for analysis and recommendation, we performed the necessary preprocessing steps to clean and prepare the text. The preprocessing steps included:

1. Text Cleaning: Removing any HTML tags, special characters, and non-alphanumeric characters from the article text.
2. Tokenization: Breaking down the article text into individual words or tokens.
3. Stopword Removal: Removing common stopwords such as "the," "is," "and," etc., which do not contribute much to the meaning of the text.
4. Lowercasing: Converting all words to lowercase to ensure case-insensitive analysis.
5. Lemmatization: Reducing words to their base or root form to avoid duplicate words with similar meanings.
6. Sentence Segmentation: Splitting the article text into sentences for more granular analysis.
7. Named Entity Recognition (NER): Identifying and categorizing entities such as organizations, people, and locations mentioned in the article using NLP techniques.

## Data Storage

The preprocessed article data was stored in a MySQL database hosted on the local server. The database schema was designed to accommodate key fields such as article title, text, category, subcategory, and content tags. Each article was assigned a unique identifier (article ID) and marked as unclassified initially.

The stored data serves as the foundation for the article recommendation algorithm.

# Natural Language Processing

Natural Language Processing (NLP) is a crucial aspect of this project, enabling us to process and analyze textual data from articles to extract meaningful information, such as article similarity and subcategory identification. In this section, we will explore the NLP techniques and libraries utilized in our project.

## Text Preprocessing

Before performing any NLP tasks, we preprocess the raw text data to clean, tokenize, and normalize it. The following preprocessing steps are applied:

1. Text Cleaning: Removal of HTML tags, special characters, and non-alphanumeric characters to obtain clean text.
2. Tokenization: Splitting the text into individual words or tokens.
3. Lowercasing: Converting all text to lowercase to ensure case-insensitive comparison.
4. Stopword Removal: Eliminating common words like "the," "and," "is," etc., which do not contribute significantly to the meaning.
5. Stemming/Lemmatization: Reducing words to their base or root form to handle different inflexions.

## Spacy for Named Entity Recognition (NER)

Spacy, a popular NLP library, is employed for Named Entity Recognition (NER). NER helps in identifying entities like organizations, persons, locations, and other meaningful entities in the article's content. These entities are then used as potential content tags and added to the stored article data.

## TF-IDF for Article Similarity

To calculate article similarity, we employ the Term Frequency-Inverse Document Frequency (TF-IDF) technique:

1. Term Frequency (TF): TF measures the frequency of a term (word) in a specific article. It indicates how important a word is to an article.
2. Inverse Document Frequency (IDF): IDF measures the rarity of a term across all articles in the dataset. It gives lower weight to common words.
3. TF-IDF Score: The TF-IDF score is the product of TF and IDF. It assigns a higher score to words that are frequent in a specific article but rare across all articles, making them potentially indicative of the article's content.
4. Cosine Similarity: After obtaining the TF-IDF vectors for two articles, we calculate their cosine similarity. The cosine similarity measures the cosine of the angle between the two vectors. A value closer to 1 indicates high similarity, while a value closer to 0 indicates dissimilarity.

By using TF-IDF and cosine similarity, we can quantitatively measure the similarity between pairs of articles, enabling us to recommend articles to users based on the article they are currently reading.

# Database Schema

The project uses a relational database to store and manage data efficiently. Three primary tables have been designed to organize and retrieve relevant information: articles, organizations, and recommendations.

## Articles Table

The articles table is central to the project, containing detailed information about the articles scraped from various sources. Below are the key attributes of the articles table:

* **title: (Primary Key)** The title of the article.
* **description:** A brief description or summary of the article content.
* **tag:** A JSON array containing tags or keywords associated with the article.
* **summary:** A concise summary or snippet of the article's content.
* **body:** The full text or body of the article.
* **publish\_date:** The date when the article was published.
* **update\_date:** The date when the article was last updated.
* **author:** The author or authors of the article.
* **category:** The primary category of the article (e.g., National, Sports).
* **subcategory:** A subcategory determined through NLP techniques to enhance recommendations.
* **slug:** A unique identifier for the article derived from its URL.
* **client\_id:** The client or user identifier associated with the article.
* **user\_id:** The user identifier who added or interacted with the article.

## Organizations Table

The ‘organizations’ table stores information related to organizations and users using the recommendation engine. Attributes include:

* **orgKey:** (Primary Key) A unique alphanumeric key for each organization.
* **name:** The name of the organization.
* **address:** The address or location of the organization.
* **phone:** The contact phone number of the organization.
* **email\_id:** The email address associated with the organization.
* **orgDomain:** The domain or website of the organization.

## Recommendations Table

The ‘recommendations’ table plays a vital role in generating article recommendations for users. It stores information about recommended articles for each user based on their preferences and interactions. Key attributes include:

* **user\_id:** The unique identifier of the user.
* **article\_id:** The identifier of the recommended article.
* **score:** A numerical score representing the relevance or suitability of the recommendation.
* **timestamp:** The timestamp when the recommendation was generated.

Each article\_id will have its top 3 recommendations stored in this table.

# API Implementation

## Introduction

The API implementation section provides an overview of the RESTful API endpoints, their functionalities, and the database operations carried out by the Express.js application.

## API Endpoints

### 1. GET /data

* **Description:** This endpoint is used to fetch data associated with a specific client ID.
* **Request Method:** GET
* **Request Parameters:**
  + `clientID`: The unique client identifier.
* **Response:**
  + If the client ID is found in the database, the API responds with the client's data.
  + If the client ID is not found, a 404 Not Found status is returned.
* **Example**:
  + Request: `GET /data?clientID=12345`
  + Response:

```json

{

"clientId": "12345",

"name": "Client Name",

"email": "client@email.com",

...

}

```

### 2. POST /api/insertArticle

* **Description:** This endpoint allows the insertion of article data into the database.
* **Request Method:** POST
* **Request Body:** 
  + - Article data in JSON format, including attributes such as `title`, `description`, `tag`, `summary`, `body`, `publish\_date`, `update\_date`, `author`, `category`, `subcategory`, `slug`, `client\_id`.
  + **Response:**
    - If the article data is successfully inserted, a success message is returned.
    - If there's an error during insertion, a 500 Internal Server Error response is returned.
  + **Example:**
    - Request:

```json

POST /api/insertArticle

{

"title": "Sample Article",

"description": "A brief description of the article.",

"tag": ["tag1", "tag2"],

...

}

```

* + - Response:

```json

{

"message": "JSON data stored successfully!"

}

```

### 3. GET /api/getRelatedStories

* **Description:** This endpoint calculates and retrieves related stories based on the Jaccard Index and similarity scores.
* **Request Method:** GET
* **Request Parameters:**
  + `collectionID`: The unique identifier for the collection of articles to analyze.
* **Response:**
  + A list of related story articles sorted by their similarity scores.
* Example:
  + Request: `GET /api/getRelatedStories?collectionID=123`
  + Response:

```json

{

"relatedStories": [

{

"title": "Related Article 1",

"score": 0.85

},

{

"title": "Related Article 2",

"score": 0.78

},

...

]

}

```

## Database Operations

### Connection Pooling

To manage database connections efficiently, the application uses connection pooling with a limit of 10 concurrent connections.

### Duplicate Record Check

Before inserting a new article record, the API checks if a record with the same title already exists. If found, it returns a message indicating that the record already exists.

### Jaccard Index Calculation

The `/api/getRelatedStories` endpoint calculates the Jaccard Index between the tags of the current article and all other articles in the database. This index is used to measure similarity.

### Top Related Stories

The API identifies and returns the top three related stories based on the calculated similarity scores.

## Conclusion

The API implementation section highlights the endpoints, request-response examples, and key database operations of the Express.js application. These API endpoints facilitate data retrieval, insertion, and related story generation.

# Article Recommendation Algorithm

The core functionality of our project revolves around recommending articles to users based on their preferences and the content of the articles. This recommendation system is powered by a combination of Natural Language Processing (NLP) techniques and data analysis. Here's an overview of how the recommendation algorithm works:

## 1. Content-Based Recommendation

Our recommendation algorithm primarily relies on content-based filtering, which recommends articles to users based on the similarity between articles in the dataset and the user's historical interactions. We consider the following components for content-based recommendation:

### Current Article Similarity

**Text Similarity:** We compute text similarity between the current article and other articles in the database using various NLP techniques such as BERT embeddings and TF-IDF vectors. This allows us to measure how closely related articles are in terms of their textual content.

**Keyword Matching:** We calculate the overlap of the keywords of each article compared to the current article. This helps in recommending articles with similar themes or topics.

## 2. Generating Recommendations

Once we have calculated the similarity scores between articles, we proceed to generate recommendations:

### a. Article Ranking

**Score Calculation:** We calculate a score for each article based on its similarity to the user's preferences and historical interactions. This score combines text similarity, keyword matching, and user profile matching.

**Ranking:** Articles are ranked based on their scores in descending order, with the highest-scored articles ranked at the top.

### b. Filtering and Personalization

**Filtering:** We filter out articles that the user has already interacted with to avoid recommending duplicate content.

**Personalization:** The system adapts to the user's evolving preferences. If a user interacts with certain articles, the system will learn from this behaviour and adjust recommendations accordingly.

## 3. Implementation Details

### a. NLP Techniques

**BERT Embeddings:** We utilize BERT embeddings to represent articles and user profiles in a dense vector space. BERT captures contextual information and semantic meaning, enhancing the accuracy of similarity calculations.

**TF-IDF Vectorization:** TF-IDF is employed for keyword extraction and matching. It helps identify important terms within articles and user profiles.

### b. Database Interaction

We interact with the database to fetch article data.

## 4. Continuous Improvement

Our recommendation algorithm is designed to continuously learn and improve over time. It adapts to user behaviour and incorporates user feedback to refine recommendations.

## 5. Future Enhancements

We are actively exploring ways to enhance the recommendation system. Potential future enhancements include maintaining and updating user profiles, collaborative filtering to incorporate user-user interactions, implementing reinforcement learning for real-time recommendation, and exploring deep learning models for improved content understanding to provide users with personalized and relevant articles based on their interests and interactions.