Low-Level Design (LLD) Document

Project Title: Book Recommendation System Using K-Means Clustering

---

1. Introduction

This document outlines the low-level design for the Book Recommendation System project, detailing individual components and their implementation. It includes the specific data structures, API routes, algorithm details, and deployment configuration for the system. The project leverages a K-Means Clustering model to recommend books to users based on various attributes such as title, author, genre, and publisher.

---

2. Data Model Design

The recommendation system uses a CSV file as the data source. The primary attributes of the dataset are:

2.1 Data Attributes (Columns in CSV)

- Title: The title of the book.
- Author: The name of the author.
- Publisher: The name of the book's publisher.
- Genre: The primary genre of the book (e.g., Fiction, Non-Fiction).
- SubGenre: A subcategory under the primary genre (e.g., Fantasy, Biography).

2.2 Data Structure in CSV File

plaintext

Copy code

```
Title, Author, Publisher, Genre, SubGenre

"Book Title 1", "Author Name", "Publisher Name", "Fiction", "Fantasy"

"Book Title 2", "Author Name", "Publisher Name", "Non-Fiction", "Biography"
```

2.3 Preprocessing Steps

- Data Cleaning: Remove duplicates and handle missing values.
- TF-IDF Vectorization: Convert text features (Title, Author, Publisher, Genre, SubGenre) into numerical vectors using TF-IDF.
- Feature Matrix: After vectorization, the features are combined into a matrix where each book is represented by a vector of numerical values.

Example feature matrix after vectorization:

plaintext

Copy code

```
[ [0.12, 0.3, 0.56, ...],  # Book 1
  [0.4, 0.5, 0.6, ...],   # Book 2
  ...
]
```

---

3. K-Means Clustering Model

3.1 Clustering Logic

- **Model: K-Means clustering is applied to the TF-IDF feature matrix. It clusters books based on similarity in attributes like genre, author, and publisher.**
- **Number of Clusters: The optimal number of clusters (k) is selected through experimentation and evaluation using inertia or the Elbow Method.**

python

```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF vectorization
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(book_data['combined_features'])

# K-Means Clustering
kmeans = KMeans(n_clusters=10, random_state=42)
book_data['cluster'] = kmeans.fit_predict(X)
```

3.2 Model Output

- **The output of the clustering model assigns a cluster to each book in the dataset.**
- **Inertia: Model quality is evaluated using inertia, which represents the sum of squared distances of samples to their closest cluster center.**

---

4. Detailed API Routes

4.1 Route: / (Homepage)

- **Method: GET**
- **Description: Renders the home page where the user inputs a book title.**

- **Implementation:**
    - **The user inputs a book title.**
    - **Autocomplete functionality helps to suggest book titles based on partial input.**

python

```
@app.route('/')
def index():
   book_titles = get_all_book_titles()
   return render_template('index.html', book_titles=book_titles)
```

4.2 Route: /recommend (Book Recommendation)

- **Method: POST**
- **Input: Book title entered by the user.**
- **Output: List of recommended books from the same cluster.**
- **Description: Fetches the recommendations based on the book title input by the user.**

python

```
@app.route('/recommend', methods=['POST'])
def recommend():
   title = request.form['title']
   recommendations = get_recommendations(title)
   return render_template('recommendations.html', recommendations=recommendations)
```

Function: get_recommendations(title)

- **Description: Takes the book title as input, retrieves its cluster, and returns books from the same cluster.**
- **Input: Book title.**
- **Output: List of recommended book titles.**

python

```
def get_recommendations(title):
   # Retrieve the cluster of the input book
   book_cluster = data[data['Title'] == title]['cluster'].values[0]

   # Retrieve similar books from the same cluster
```

```
    similar_books = data[data['cluster'] == book_cluster]['Title'].tolist()
    return similar_books
```

4.3 Route: /get_book_titles (Autocomplete Feature)

- Method: GET
- Output: JSON list of all book titles in the dataset.
- Description: Provides a list of book titles for the autocomplete functionality.

python

```
@app.route('/get_book_titles')
def get_book_titles():
    titles = get_all_book_titles()
    return jsonify(titles)
```

---

5. Error Handling

5.1 Invalid Book Title

- If the user enters an invalid or unrecognized book title, the system displays an error message.

python

```
@app.route('/recommend', methods=['POST'])
def recommend():
    try:
        title = request.form['title']
        recommendations = get_recommendations(title)
        return render_template('recommendations.html', recommendations=recommendations)
    except KeyError:
        return render_template('error.html', message="Book not found")
```

---

6. Front-End (UI) Design

6.1 HTML Templates

- index.html: Contains the input form for the user to enter the book title.

html

```
<form method="POST" action="/recommend">
  <input type="text" name="title" placeholder="Enter a book title">
  <button type="submit">Get Recommendations</button>
</form>
```

- recommendations.html: Displays the list of recommended books.

html

Copy code

```
<h2>Recommended Books</h2>
<ul>
  {% for book in recommendations %}
    <li>{{ book }}</li>
  {% endfor %}
</ul>
```

- error.html: Displays an error message when the book is not found.

html

```
<h2>{{ message }}</h2>
<a href="/">Go back to search</a>
```

---

7. Deployment Configuration

7.1 GitHub Integration

- Connect the repository to Azure Web App using the Deployment Center.
- Configure GitHub Actions for CI/CD:
  - Build the application.
  - Run tests.
  - Deploy to Azure Web App upon code push to the main branch.

7.2 Azure Web App Deployment

- Publish Profile: Add the Azure Web App publish profile to GitHub Secrets under the name AZURE_WEBAPP_PUBLISH_PROFILE.

yaml

```
name: Deploy to Azure WebApp
```

```
on:
  push:
    branches:
      - main

jobs:
  build_and_deploy:
    runs-on: ubuntu-latest
    steps:
    - name: 'Checkout GitHub Action'
      uses: actions/checkout@v2

    - name: 'Set up Python'
      uses: actions/setup-python@v1
      with:
        python-version: '3.x'

    - name: 'Install dependencies'
      run: |
        pip install -r requirements.txt

    - name: 'Deploy to Azure Web App'
      uses: azure/webapps-deploy@v2
      with:
        app-name: 'your-app-name'
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: .
```

---

8. Testing and Validation

8.1 Unit Testing

- Unit tests are implemented to verify core functionalities like:

    - Book clustering.
    - Recommendation retrieval.
    - API response for /recommend and /get_book_titles.

8.2 Integration Testing

- Test the system end-to-end by submitting a book title and verifying that correct recommendations are displayed.

---

9. Conclusion

This low-level design document provides an in-depth look at the implementation of the Book Recommendation System. The design encompasses data handling, API routes, clustering logic, and deployment strategy. The system is ready for deployment on Azure, and continuous integration is handled via GitHub Actions.

End of Document