Book Recommendation System Setup

Requirements.txt

```
Flask==2.3.3
pandas==2.0.3
numpy==1.24.3
sentence-transformers==2.2.2
scikit-learn==1.3.0
torch==2.0.1
transformers==4.33.2
```

Directory Structure

```
book-recommender/
├── app.py                 # Main Flask application
├── requirements.txt       # Python dependencies
├── templates/
│   └── index.html        # HTML template
├── books.csv             # Your books dataset
├── book_embeddings.pkl   # Generated embeddings (auto-created)
└── README.md
```

Installation & Setup

1. Create Virtual Environment

```bash
python -m venv book_recommender_env
source book_recommender_env/bin/activate  # On Windows: book_recommender_env\Scripts\activate
```

2. Install Dependencies

```bash
pip install -r requirements.txt
```

3. Prepare Your Data

- Place your `books.csv` file in the project root
- The CSV should have columns: `ISBN`, `Book-Title`, `Book-Author`, `Year-Of-Publication`, `Publisher`

4. Create Templates Directory

```bash
mkdir templates
# Save the HTML template as templates/index.html
```

5. Run the Application

```bash
python app.py
```

The application will be available at `http://localhost:5000`

Features

🚀 Core Functionality

- **Embedding-based Recommendations**: Uses SentenceTransformers to create semantic embeddings
- **Cosine Similarity**: Finds books with similar content/themes
- **Smart Search**: Combines exact matching with semantic similarity
- **Caching**: Saves embeddings to avoid recomputation

🎯 How It Works

1. **Data Processing**: Combines book title, author, and year into descriptive text
2. **Embedding Generation**: Creates vector representations using `all-MiniLM-L6-v2` model
3. **Similarity Calculation**: Uses cosine similarity to find related books
4. **Ranking**: Returns top matches with similarity scores

🔧 API Endpoints

- `GET /` - Main interface
- `POST /recommend` - Get recommendations
- `GET /health` - Health check

📊 Sample Request

```json
POST /recommend
{
    "book": "Lord of the Rings"
}
```

📋 Sample Response

```json
{
    "query": "Lord of the Rings",
    "recommendations": [
        {
            "title": "The Hobbit",
            "author": "J.R.R. Tolkien",
            "year": 1937,
            "publisher": "Houghton Mifflin Harcourt",
            "similarity": 0.85
        }
    ],
    "exact_matches": []
}
```

Customization Options

1. Change the Embedding Model

```python
# In BookRecommendationSystem.__init__()
self.model = SentenceTransformer('all-mpnet-base-v2')  # Better quality, slower
# or
self.model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')  # Multilingual
```

2. Adjust Recommendation Count

```python
# In the /recommend endpoint
recommendations = recommender.find_similar_books(favorite_book, top_k=15)
```

3. Modify Text Combination

```python
# In load_data() method
self.books_df['combined_text'] = (
    self.books_df['Book-Title'].astype(str) + ' ' +
    self.books_df['Book-Author'].astype(str)  # Simplified version
)
```

Performance Tips

1. **Large Datasets**: For >10k books, consider using FAISS for faster similarity search
2. **Memory Usage**: The embeddings are cached - monitor memory usage for large datasets
3. **Startup Time**: First run takes longer due to embedding generation
4. **Model Selection**: Balance between speed and accuracy based on your needs

Troubleshooting

Common Issues

1. **Memory Error**: Reduce batch size or use a smaller model
2. **Slow Performance**: Ensure embeddings are cached properly
3. **No Recommendations**: Check if your CSV format matches expected columns
4. **Import Errors**: Ensure all dependencies are installed correctly

Debug Mode

The app runs in debug mode by default. Set `debug=False` for production.

Production Deployment

For production deployment:

1. Use a production WSGI server (e.g., Gunicorn)
2. Set up proper logging
3. Add input validation and rate limiting
4. Consider using Redis for caching embeddings
5. Add authentication if needed

```bash
pip install gunicorn
gunicorn -w 4 -b 0.0.0.0:5000 app:app
```