Two-Tower Recommendation System

User Manual

Yinying Huo

April 2025

Contents

Preface **ii**

1 Introduction **1**
- 1.1 System Overview . . . 1
- 1.2 Key Features . . . 1
- 1.3 Architecture . . . 1

2 Installation **3**
- 2.1 Prerequisites . . . 3
- 2.2 Installation Steps . . . 3

3 Data Requirements and Processing **4**
- 3.1 Data Format . . . 4
- 3.2 Data Processing . . . 4

4 Usage Instructions **6**
- 4.1 Quick Start . . . 6
- 4.2 Using the User Interface . . . 6
- 4.3 Command-Line Arguments . . . 7
- 4.4 Training the Model . . . 7
- 4.5 Getting Recommendations for a Specific User . . . 8

5 Advanced Usage **9**
- 5.1 Model Parameters . . . 9
- 5.2 Customizing Neural Network Architecture . . . 9
- 5.3 Optimizing FAISS Index . . . 9

6 System Components **10**

7 Troubleshooting **11**
- 7.1 Common Issues . . . 11
- 7.2 Debugging Tips . . . 11

A Glossary **12**

Preface

This user manual provides comprehensive instructions for installing, configuring, and using the Two-Tower Recommendation System. This system is designed to provide personalized book recommendations based on user profiles using a deep learning approach.

The Two-Tower Embedding (TTE) architecture maps users and items into a shared embedding space, enabling efficient and accurate recommendation generation.

This document is intended for both end-users who want to use the recommendation system and developers who may want to extend or modify its functionality.

Chapter 1

Introduction

1.1 System Overview

The Two-Tower Recommendation System is a machine learning-based recommendation engine that suggests books to users based on their profile information and the characteristics of available books. The system uses a neural network architecture known as "Two-Tower Embedding" (TTE) to map both users and books into a shared embedding space. Recommendations are generated by finding books whose embeddings are closest to the user embedding in this space.

1.2 Key Features

- Deep learning-based recommendation system using PyTorch
- Highly efficient ANN (Approximate Nearest Neighbor) search using FAISS
- Personalized book recommendations based on user age, location, and other features
- Extensible architecture allowing for easy updates and modifications
- Command-line interface for easy interaction
- Fast inference time even with large book catalogs

1.3 Architecture

The system is based on a modular architecture that follows the principle of information hiding. It consists of three main layers:

1. **Hardware-Hiding Module**: Handles system interactions like file I/O
2. **Behavior-Hiding Modules**: Implement the core functionality
    - Data Processing Module
    - Model Training Module
    - Embedding Generation Module
    - Recommendation Module

3. **Software Decision Modules**: Implement algorithm-specific functionality
    - Neural Network Architecture Module
    - ANN Search Module
    - Vector Operations Module

Chapter 2

Installation

2.1 Prerequisites

Before installing the Two-Tower Recommendation System, ensure your system meets the following requirements:

- Python 3.9 or higher
- PyTorch 1.9 or higher
- FAISS for vector similarity search
- Other dependencies as listed in `requirements.txt`

2.2 Installation Steps

1. Clone the repository:

```
git clone https://github.com/V-AS/Two-tower-recommender-system.git
cd Two-tower-recommender-system
```

2. Create and activate a virtual environment:

```
python3 -m venv .venv
source .venv/bin/activate   # On Windows: .venv\Scripts\activate
```

3. Install dependencies:

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

4. Install FAISS:

```
# For CPU-only systems:
pip install faiss-cpu

# For systems with NVIDIA GPU:
pip install faiss-gpu
```

Chapter 3

Data Requirements and Processing

3.1 Data Format

The system expects data in a specific format to train the recommendation model. The required data consists of:

- **User information**: Including age, location
- **Book information**: Including title, author, publication year, publisher
- **Ratings**: Mapping users to books with numerical ratings

The system expects a CSV file with the following columns:

- User-ID
- Book-Rating
- Book-Title
- Book-Author
- Year-Of-Publication
- Publisher
- Age
- State
- Country

3.2 Data Processing

The system includes a data processing module that performs several important transformations:

1. **Data validation**: Checks if all required columns are present
2. **Feature engineering**: Creates derived features such as:

- Age groups
- State and country frequency
- Author and publisher frequency

3. **Missing value handling**: Imputes missing values with sensible defaults

4. **Data splitting**: Divides data into training and testing sets

If you need to preprocess your own data, the repository includes a Jupyter notebook in `src/utils/data_preprocessing.ipynb` that demonstrates how to merge separate user, book, and rating files into the required format.

Chapter 4

Usage Instructions

4.1 Quick Start

For users who want to immediately start getting recommendations, the system includes pre-trained models in the `output` folder. You can start the user interface with:

```
python src/user_interface.py
```

This will launch an interactive terminal where you can enter your information (age, state, country) to get personalized book recommendations.

For debugging information, use:

```
python src/user_interface.py --debug
```

4.2 Using the User Interface

The user interface provides a simple way to interact with the recommendation system:

1. When launched, the system will display the top states/countries in the dataset
2. Enter your age when prompted
3. Enter your state/province (or press Enter for "Unknown")
4. Enter your country (or press Enter for "Unknown")
5. Wait for recommendations to be generated
6. View your personalized book recommendations
7. Choose whether to get more recommendations

Example Session

```
$ python src/user_interface.py

====================
TOP 10 STATES/PROVINCES IN THE DATASET
--------------------
1. california: 8521 users (12.5%)
2. new york: 7234 users (10.6%)
3. texas: 4521 users (6.6%)
...

====================
TOP 10 COUNTRIES IN THE DATASET
--------------------
1. usa: 52341 users (76.8%)
2. canada: 5234 users (7.7%)
3. united kingdom: 4523 users (6.6%)
...

====================
BOOK RECOMMENDATION SYSTEM
====================

Please enter your information to get personalized book recommendations:
Your age (0-100): 32
...
```

4.3 Command-Line Arguments

The user interface supports the following command-line arguments:

- `--age`: Your age (e.g., `--age 32`)
- `--state`: Your state or province (e.g., `--state california`)
- `--country`: Your country (e.g., `--country usa`)
- `--debug`: Enable debug mode for additional information

If you provide the age, state, and country arguments, the system will generate recommendations without prompting for additional input.

4.4 Training the Model

If you want to train the model from scratch or on your own data, you can use the `main.py` script:

```
python src/main.py --mode train --epochs 3 --batch_size 10 --embedding_dim 32
```

The training process takes approximately 5 minutes with the recommended parameters, which have been optimized for the default dataset.

The training process involves:

1. Loading and preprocessing the data
2. Creating neural network architectures for the user and item towers
3. Training the model to predict ratings
4. Evaluating the model on a test set
5. Generating and saving embeddings
6. Building and saving an ANN index

4.5 Getting Recommendations for a Specific User

You can generate recommendations for a specific user ID using:

```
python src/main.py --mode recommend --user_id 12345 --num_recommendations 10
```

This will look up the user in the dataset, generate their embedding, and return personalized recommendations.

Chapter 5

Advanced Usage

5.1 Model Parameters

The system offers several configurable parameters for model training:

Parameter	Description	Default Value
`--epochs`	Number of training epochs	5
`--batch_size`	Training batch size	10
`--embedding_dim`	Dimension of embedding vectors	32
`--data_path`	Path to training data	`data/processed/recommender_data.csv`
`--output_dir`	Directory to save models	`output`
`--num_recommendations`	Number of recommendations to generate	10

5.2 Customizing Neural Network Architecture

The neural network architecture can be customized by modifying the `src/modules/neural_network.py` file. The default implementation uses a simple tower with one hidden layer, but you can add more layers or change activation functions as needed.

5.3 Optimizing FAISS Index

The FAISS index type can be modified in `src/modules/ann_search.py`. The default implementation uses a flat index, which provides exact search results but may be slower with very large datasets. For larger datasets, consider using IVF indices which trade some accuracy for significant speed improvements.

Chapter 6

System Components

The system design is described in detail in two documents: the Module Guide and the Module Interface Specification.

Chapter 7

Troubleshooting

7.1 Common Issues

Issue: `ImportError: No module named 'faiss'`
Solution: Make sure you have installed FAISS correctly for your system.

```
# For CPU-only systems:
pip install faiss-cpu

# For systems with NVIDIA GPU:
pip install faiss-gpu
```

Issue: `CUDA out of memory`
Solution: Try reducing the batch size or embedding dimension.

```
python src/main.py --mode train --batch_size 4 --embedding_dim 16
```

Issue: `File not found: data/processed/recommender_data.csv`
Solution: Make sure you're running the script from the project root directory, or specify the full path to the data file.

```
python src/main.py --data_path /full/path/to/data/processed/
    recommender_data.csv
```

7.2 Debugging Tips

1. Enable debug mode when running the user interface:

```
python src/user_interface.py --debug
```

2. Check the training history in `output/training_history.json` to see if the model is learning properly.

3. Run a specific test file to check if a particular component is working as expected:

```
pytest tests/unit/test_embedding_generation.py
```

Appendix A

Glossary

ANN (Approximate Nearest Neighbor) A technique for finding similar vectors in high-dimensional space, which trades some accuracy for improved speed and memory usage.

Embedding A low-dimensional, dense vector representation of a high-dimensional, sparse feature. In this system, both users and items are represented as embeddings in a shared vector space.

FAISS Facebook AI Similarity Search, a library for efficient similarity search and clustering of dense vectors.

Two-Tower Architecture A neural network architecture with two parallel networks (towers) that process different types of inputs (users and items) and map them to a shared embedding space.

Cosine Similarity A measure of similarity between two vectors based on the cosine of the angle between them. Used to rank items for recommendation.

Feature Engineering The process of creating new features from raw data to improve model performance.

Dot Product A vector operation that calculates the sum of the products of corresponding elements. Used to calculate similarity between embeddings.

Normalization The process of scaling values to a standard range, typically [0, 1].

Incremental Learning Updating a model with new data without retraining from scratch.