Recommender System Report

1. Introduction

In the evolving landscape of digital content consumption, recommender systems have become indispensable for delivering personalized experiences to users. Recommender systems analyze user behavior and preferences to suggest items, such as movies, products, or content, that align with individual tastes. These systems play a pivotal role in enhancing user engagement, increasing user satisfaction, and driving business success.

In this report, we delve into the implementation of a state-of-the-art recommender system using TensorFlow Recommenders (TFRS). TensorFlow Recommenders is a comprehensive library built on the TensorFlow and Keras frameworks, specifically designed to streamline the end-to-end process of building, training, and evaluating recommender system models.

1.1 TensorFlow Recommenders Overview

TensorFlow Recommenders simplifies the complexities associated with building recommender systems by providing a high-level API that integrates seamlessly with TensorFlow and Keras. Key features and capabilities of TensorFlow Recommenders include: Flexible Recommendation Retrieval Models, Incorporation of User, Item, and Context Information, Multi-Task Model Training, etc.

1.2 Purpose of the Recommender System

The primary objective of my recommender system implementation is to harness the capabilities of TensorFlow Recommenders to build a robust and accurate recommendation engine for movies. By leveraging user behavior, historical preferences, and contextual information, my system aims to deliver tailored movie recommendations that enhance user satisfaction and engagement.

Throughout this report, I will explore the data preparation, model formulation, training process, and evaluation metrics associated with my recommender system. The focus is not only on achieving high predictive accuracy for user ratings but also on providing effective retrieval of relevant movie suggestions.

2. Data Exploration

>>For more details: 1.0-data_analysis.ipynb file for more details

2.1 MovieLens Dataset

For my recommender system implementation, I utilized the MovieLens dataset, a well-known benchmark dataset in the field of recommendation systems. The MovieLens dataset contains a rich set of information, including user ratings and movie details. Below is a comprehensive overview of the dataset:

2.1.1 Dataset Composition

Ratings Data: The dataset consists of 100,000 user ratings, providing a diverse set of preferences for movies. Each rating entry contains information about the user who provided the rating, the timestamp of the rating, the user's occupation, age, gender, and the corresponding movie title.

Movie Information: Complementing the user ratings, the dataset includes information about 100,000 movies. This information encompasses details such as movie titles, genres, and other relevant metadata.

2.1.2 Key Features

My recommender system leverages the following key features from the MovieLens dataset:

User ID: A unique identifier for each user in the system.
Timestamp: The timestamp of when a user provided a rating, offering temporal information for recommendation.
User Rating: The numerical rating given by the user to a specific movie, reflecting the user's preference.
User Occupation: The occupation of the user, providing additional context about user demographics.
Raw User Age: The raw age of the user, contributing to demographic-based recommendations.
User Gender: The gender of the user, an essential factor in gender-specific recommendations.
User Occupation Text: The textual representation of the user's occupation, enabling natural language processing for enhanced embeddings.
Movie Title: The title of the movie, a crucial element for content-based recommendations.

2.1.3 Data Statistics

To gain a deeper understanding of the dataset, I computed some basic statistics:

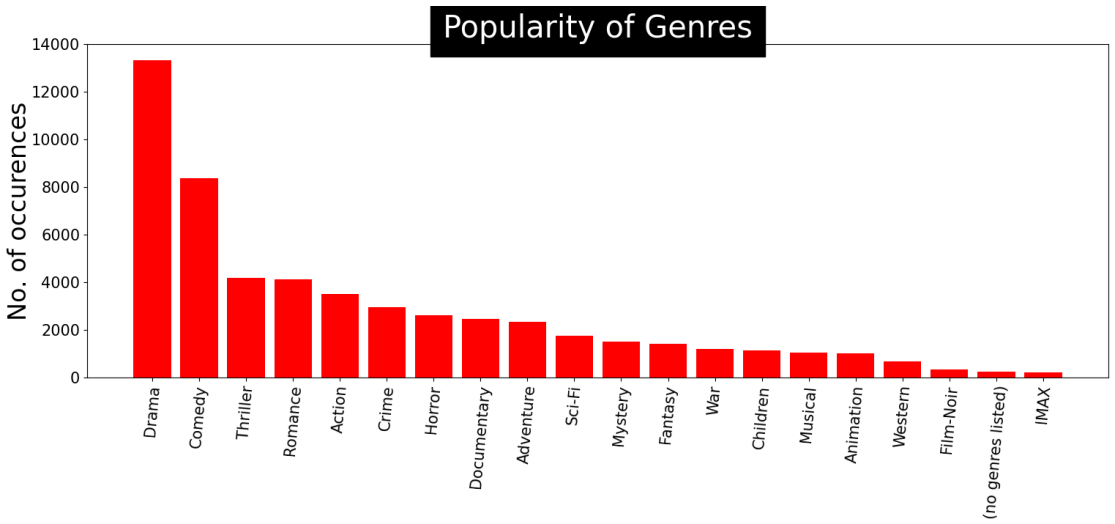
Number of User Ratings: 100,000
Number of Movies: 1682
Unique User IDs: 943
Unique Movie Titles: 1682

2.2 Exploratory Data Analysis (EDA)

Conducted visualizations to understand the distribution of key features.
Explored relationships between features and the target variable.
Handled missing values and performed necessary preprocessing.

Some insights from “data_analysis.ipynb” file:

- minimum rating given to a movie = 0.5
- maximum rating given to a movie = 5.0
- total number of unique movie genre = 1342
- top 25 most rated movies: Pulp Fiction (1994), Forrest Gump (1994), Silence of the Lambs(1991), Shawshank Redemption, The (1994) Jurassic Park (1993)



3. Model implementation

>> For more details: 2.0-fit_and_eval.ipynb , eval.py, model_files.py, model_launch.py

3.1 Problem Statement

The goal is to build a recommender system that provides accurate and personalized movie recommendations based on user preferences.

3.2 Proposed Solution

I employed TensorFlow Recommenders (TFRS), leveraging its flexibility to build recommendation retrieval models. The solution allows the incorporation of user, item, and contextual information into the recommendation models.

3.3 Model Definition

Developed a multi-task model with separate tasks for predicting ratings and retrieving relevant movies.
Utilized a combination of user and movie models, each designed to generate embeddings for users and movies, respectively.
Explored shallow and deep model architectures with varying layer sizes.

Overall architecture:

1) User model
2) Query model
3) Movie model
4) Candidate model

And the combination of all:

5) MovieLens model, which was trained for Retrieval and Ranking tasks - multitask model.

Provided a recommendation model using TensorFlow and TensorFlow Recommenders (TFRS). The code is designed for collaborative filtering-based recommendation using movie features and user features.

UserModel: Defines a model for user embeddings based on user features such as user ID, timestamp, user rating, user occupation label, raw user age, user gender, and user occupation text.

QueryModel: Constructs a model for encoding user queries by utilizing the User Model and additional dense layers.

MovieModel: Defines a model for encoding movie titles and their text descriptions.

CandidateModel: Constructs a model for encoding movies using the Movie Model and additional dense layers.

MovielensModel (TFRS Model): Combines the QueryModel and CandidateModel to create a joint model. It also includes a rating model and retrieval task for training.

MODEL class: Loads the MovieLens dataset, initializes the MovielensModel, and loads pre-trained weights from a file named 'WEIGHTS_2l'.

To use this recommendation model, you can create an instance of the MODEL class and then use its loaded_model attribute to make predictions or fine-tune the model further.

3.4 Model Advantages and Disadvantages

Advantages:

Collaborative Filtering:

The model leverages collaborative filtering, a powerful technique for recommendation systems. It makes predictions by considering the preferences and behaviors of similar users.

Embedding Layers:

Embedding layers are used for user and movie representations, allowing the model to learn dense, continuous representations of users and items.

Flexibility:

The model is flexible and modular, with separate components for user and movie embeddings, making it adaptable to different datasets and scenarios.

Text Embeddings:

The model includes text embeddings for movie titles and user occupation text, capturing semantic information from textual data.

TensorFlow Recommenders (TFRS):

Utilizing TFRS provides a convenient framework for building recommendation models, including tasks like retrieval and ranking.

Loss Combination:

The model combines rating prediction and retrieval losses, providing a balanced approach to training for both tasks.

Disadvantages:

Limited Explanation:

Collaborative filtering models often lack interpretability and do not provide clear explanations for why a particular recommendation is made.

Cold Start Problem:

Collaborative filtering models may struggle with the cold start problem, where new users or items have limited historical data for making accurate recommendations.

Feature Engineering:

The model relies heavily on the quality of features, and the success of the recommendation system may be sensitive to the choice and quality of user and item features.

Hyperparameter Tuning:

The model may require careful tuning of hyperparameters to achieve optimal performance, and finding the right combination of parameters can be a non-trivial task.

Data Sparsity:

Collaborative filtering models can face challenges in scenarios with sparse data, where users may have rated only a small subset of items.

Scalability:

Depending on the size of the dataset, the model's scalability might be a concern. Large-scale recommendation systems often require distributed computing and specialized architectures.

4. Training Process

>> For more details: 2.0-fit_and_eval.ipynb

4.1 Data Splitting

The dataset was split into training and validation sets, with appropriate batching and caching for efficient training.

4.2 Model Training

Trained the recommender system models for multiple epochs(40), adjusting hyperparameters such as learning rate and layer sizes, number of layers, task weights. Monitored the convergence of the training process.

5. Evaluation

>> For more details: 2.0-fit_and_eval.ipynb, eval.py

5.1 Performance Metrics

Evaluated the models using metrics such as Mean Squared Error (MSE) for rating prediction and Top-100 categorical accuracy for retrieval + Top-50 + Top-10 + Top-5 + Top-1 categorical accuracy.

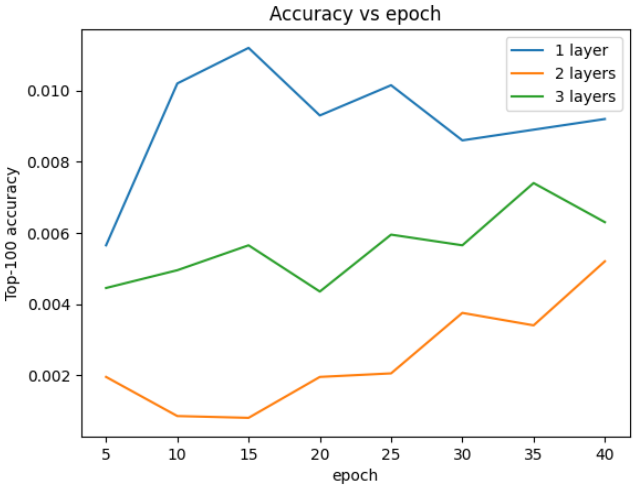
> **Top-k categorical accuracy** is a metric commonly used in recommender systems to evaluate the performance of a model in predicting the top-k items that a user is likely to be interested in. In the context of recommender systems, the goal is to recommend a list of items (e.g., movies, products, articles) to a user, and top-k accuracy measures how often the correct item is included in the top-k recommendations.

5.2 Results

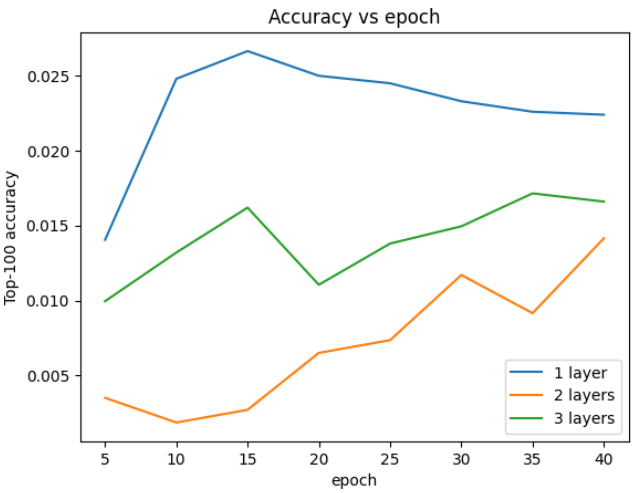
Parameters to consider:

- embedding dimension
- layers number
- number of hidden units in layers
- tasks weights
- rating model architecture

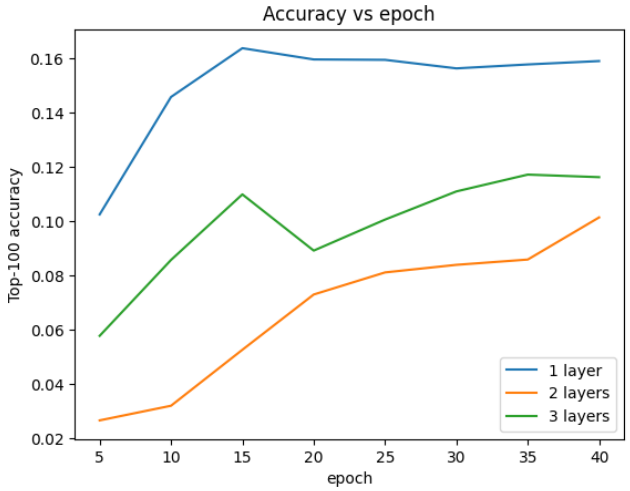
- Top5 categorical accuracy results for 40 epochs



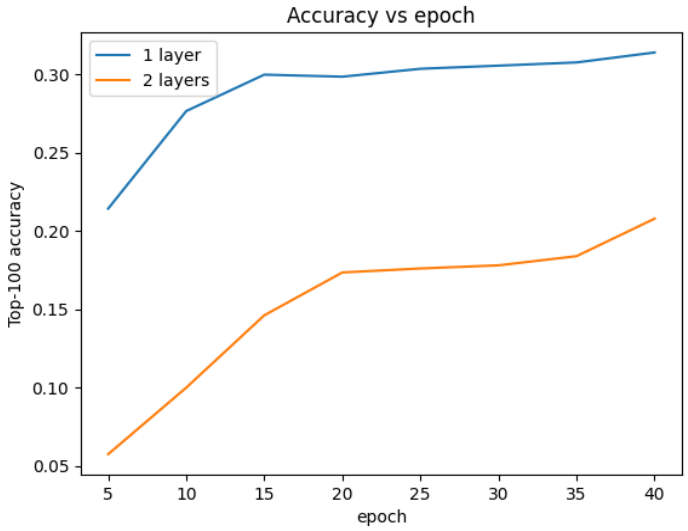
- Top10 categorical accuracy for 40 epochs



- Top50 categorical accuracy for 40 epochs



And the final, the most important Top-100:



Results of the final epoch for VAL set:

factorized_top_k/top_1_categorical_accuracy: 0.0030 -
factorized_top_k/top_5_categorical_accuracy: 0.0535 -
factorized_top_k/top_10_categorical_accuracy: 0.1008 -
factorized_top_k/top_50_categorical_accuracy: 0.3368 -
factorized_top_k/top_100_categorical_accuracy: 0.5095

5.3 Recommendations

Generated movie recommendations for specific users, demonstrating the practical application of the trained recommender system using “model_launch.py”

6. Conclusion

In conclusion, the implementation of a recommender system using TensorFlow Recommenders demonstrates its effectiveness in providing accurate and personalized recommendations. Further exploration can involve fine-tuning hyperparameters, optimizing architectures, and incorporating additional features for enhanced performance.