Final report
Analysis and Evaluation of Different Movie Recommendation System approaches

Christian Francesco Russo, Lorenzo Spagnolo, Matteo Spinato

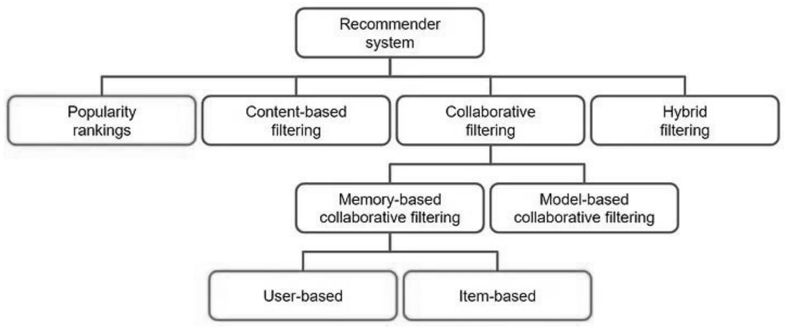
Project repository link: https://gitlab.com/public_unipd_projects/movie_recommendation_system

Abstract: This paper presents a comprehensive analysis and evaluation of diverse movie recommendation system approaches. The project explores classical methods, (unsupervised) shallow node embeddings-based approaches and supervised graph neural network (GNN)-based approaches. The study employs the well-known MovieLens $100k$ dataset, encompassing various sources such as movie metadata, users' ratings, movie keywords, movie credits and movie links. The implementation details, model architectures, and experimental results are discussed, providing valuable insights into the strengths and limitations of each approach.

1. Introduction

In the rapidly evolving landscape of digital entertainment, movie recommendation systems have emerged as indispensable tools for enhancing user experience and engagement. With an ever-expanding array of movies available, personalized recommendation systems play a crucial role in guiding users toward content tailored to their preferences.

Movies recommendation systems can generally be categorized into four main categories [4]:



Popularity-based filtering: This approach recommends movies based on their popularity among a large user base. This approach is simple and efficient, but it can overlook niche or less popular movies that may be of interest to specific users.

Content-based filtering: This approach utilizes movie metadata, such as genres, actors, directors, and plot summaries, to recommend movies that are similar to those a user has previously liked. This approach is effective for users who have well-defined preferences, but it can struggle to capture subtle nuances in user tastes.

Collaborative filtering: This approach relies on the interactions between users and movies to identify patterns and make recommendations. Two main subcategories exist:

User-based collaborative filtering: This approach identifies users with similar preferences to the target user and recommends movies that those users have liked.
Item-based collaborative filtering: This approach analyses the ratings given to a particular movie by different users and recommends movies that have been highly rated by users who have similar tastes in other movies.

Hybrid filtering: This approach combines aspects of two or more of the aforementioned techniques to leverage the strengths of each method. For instance, a hybrid system might employ content-based filtering to identify a set of relevant movies and then refine the recommendations using collaborative filtering based on user interactions.

This project undertakes an exhaustive exploration of various movie recommendation approaches of the different types just described, encompassing classical methods, shallow node embeddings, and supervised Graph Neural Networks (GNNs). The goal is that of analyzing, evaluating, and comparing advantages, disadvantages, trade-off and performances among these approaches.

2. Dataset

The project utilizes the MovieLens 100k dataset, which includes the following files:

- **`movies_metadata.csv`:** contains information on 45,000 movies.
- **`keywords.csv`:** contains movie plot keywords in JSON format.
- **`credits.csv`:** contains the cast and crew Information of all movies.
- **`links_small.csv`:** contains the TMDB and IMDB IDs of a small subset of 9,000 movies.
- **`links.csv`:** contains the TMDB and IMDB IDs of all movies.
  Note: this file is not used in practice since the dataset generated from it does not fit into our computer memory!
- **`ratings_small.csv`:** contains a subset of 100,000 ratings from 700 users on 9,000 movies.

This dataset and additional information are available here [1].
Note: for larger versions of the MovieLens dataset you can check here [2].

3. Project description

In this section are schematically described the different classes of approaches that we implemented and, for each of them, are reported: the types of filters implemented, the format of the dataset used, the model adopted, and an analysis on the advantages and the disadvantages. The implementation of these approaches will be described in more details in the section **Methods**.

Classical approaches:

- **Recommendation system types:** popularity ranking, content-based filtering, collaborative filtering, hybrid filtering.
- **Dataset format:** tabular dataset.
- **Task:**
  - **Movie suggestion:** for popularity ranking, content-based filtering, and hybrid filtering:
    - No ML/DL.
    - **Metrics:** None.
  - **Movie prediction:** for collaborative filtering:
    - Surprise library: SVD (Matrix-Factorization based algorithm).
    - **Metrics:** MAE, RMSE:

- **Pros/Cons:**
  - No ML on graphs.
  - This is only the basic point of our work from which we started implementing the next approaches.

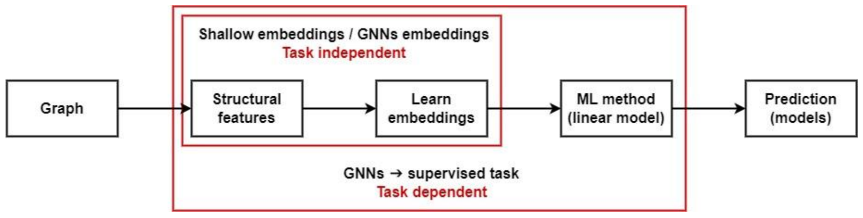
(Shallow) node embeddings-based approach:

- **Recommendation system types:** collaborative filtering.
- **Dataset format:** graph dataset.
- **Model:**
  - **Encoder:** Node2Vec.
  - **Decoder:** `Torch.nn` linear layer.
- **Task:**
  - **Movie prediction:** for collaborative filtering:
    - **Metrics:** MAE, RMSE.
    - **Framework:** PyTorch Geometric.
  - **Movie suggestion:** for collaborative filtering, makes use of cosine similarity.
- **Pros/Cons:**
  - No external features can be added to node embedding.
  - Shallow node embedding is trained for unsupervised task (no edge regression task specific), only to learn nodes embedding representation.

Supervised GNNs-based approaches:

- **Recommendation system types:** collaborative filtering and hybrid filtering.
- **Dataset format:** graph dataset + tabular dataset (for nodes external features).
- **Model:**
  - **Graph Auto Encoder (GAT):**
    - **Encoder:** GCN | GraphSAGE | GAT.
    - **Decoder:** for edge regression task.
- **Task:**
  - **Movie prediction:** for collaborative filtering:
    - **Metrics:** MAE, RMSE.
    - **Framework:** Pytorch Geometric to implement the graph dataset and to implement the model.
  - **Movie suggestion:** for collaborative filtering and hybrid filtering.
- **Pros/Cons:**
  - External features can be added to node embedding.
  - GNNs are trained for the specific supervised task: **edge regression**.

What described so far for graph-based approaches can be summarized in the following schema:



4. Project structure

The code of this project is contained in the folder `MovieRecommendationSystem/Src`, which contains the following subfolders:

`datasets`:

- **`raw`:** **the code on the GitLab repository does not contain the dataset**, therefore you will need to download the MovieLens 100k dataset from here [1] and then add all its .csv files in this folder!
- **`processed`:** this folder contains the processed tabular dataset and the graph dataset generated starting from the raw MovieLens dataset.

`scripts`:

- **`data`:** `tabular_dataset_handler.py`, `graph_dataset_handler.py`.
- **`approaches`:**
    - **filters:** `popularity_rankings.py`, `content_based_filtering.py`, `collaborative_filtering.py`, `hybrid_filtering.py`.
    - **`collaborative_filters`:** `SVD_based_CF.py`, `GNN_based_CF.py`.
    - **`models`:** `GNN_regression_model`.
- **`trained_models`:** this folder contains the trained models saved, and a subfolder `runs` which contains the training information obtained by means of TensorBoard.
- **Main notebooks files:** `main_classical_approaches.ipynb`, `main_GNN_approaches.ipynb`, `main_embedding_approach.ipynb`.
  Note: the main notebooks provide a full demonstration of how the previous files work.

5. Methods

5.1 Dataset handlers

In this section are described the files which handle the tabular and the graph MovieLens datasets:

`tabular_dataset_handler.py`: (References: [3])
This file contains the class `TabularDatasetHandler` for handling the tabular MovieLens datasets. In particular, this class performs the following operations: data loading, data cleaning, data aggregation, data manipulation, data formatting, etc.

`graph_dataset_handler.py`: (References: [6][7][8][9][10][11])
This file contains the class `HeterogeneousGraphDatasetHandler` which allows to build a heterogeneous graph dataset, using the Pytorch Geometric library, starting from the tabular dataset. In particular, this class provides functions for performing: nodes features building, edges features building, graph building, updating, saving, loading and visualization.
The heterogeneous graph obtained has the following features:

- **Nodes:** users and movies (hence, the graph is heterogeneous).
- **Edges:** if a user has rated a movie.
- **Nodes features:**
    - **Movie:** in addition to the node embedded features some external features, extracted from the tabular dataset, are added (concatenated). In particular, we tested the following combinations of (concatenated) external features:
        - `[titles, genres]`
        - `[titles, collections, genres]`
        - `[titles, genres, production_companies]`
        - `[titles, collections, genres, production_companies]`

Note: `[titles]` was built by means of `SentenceTransformer` to obtain a fixed-size vector representation of the movie title for each movie node, `[collection]` was built through labeling, and `[genres]` and `[production_companies]` were built thorough one-hot encoding.

  - **User:** only graph embedded features, since we do not have data about users.
- **Edge label:** the user-movie rating. This is what we want to predict.

5.2 Classical approaches

In this section are described the files which contain the classical approaches implementation (check the relative references for more details).

`popularity_rankings.py`: (References: [3])

- **Top movies (function):** utilizes the IMDB's formula to do a weighted rating of movies based on the distribution of users' rankings counts and means.
- **Top movies + genre (function):** utilizes the previous approach on a selected movie genre.

`content_based_filtering.py`: (References: [3])

- **Description-based (function):** utilizes the cosine similarity scores on vectors of tokens extracted from movies: description, overview, and taglines.
- **Metadata-based (function):** utilizes the cosine similarity scores on vectors of tokens extracted from movies: title, cast, director, keywords, and genres.
- For both previous methods, an improved version has been proposed, using also the IMDB's formula.

`collaborative_filtering.py`: contains the abstract class `CollaborativeFilteringInterface` which defines a common interface for collaborative filters.

`SVD_based_CF.py`: (References: [3][5])

- **SVD model-based (class `SVD_Based_CollaborativeFilter` implementing `CollaborativeFilteringInterface`):** utilizes the SVD model (Matrix-based Factorization) from the Surprise library. This is what we consider the reference model for comparing the performance of the other collaborative filters!

`hybrid_filtering.py`: (References: [3][4])

- **Ensemble** of model-based **collaborative filtering** (SVD-based and GNN-based) + **content-based filtering** (metadata-based) recommenders**.**

5.3 Shallow node embedding-based approach

In this section is described the file which contains the shallow node embedding approach implementation.

`main_embedding_approach.ipynb`: (References: [17][18])

- **Encoder**: the Node2Vec library to generate walks and train the model. The resulting embeddings of users and movies are used as input features for the recommendation model, the decoder.
- **Decoder:** a regression model in PyTorch that takes as input the combined feature vector of users and movies, used to predict the ratings.
- **Model architecture:** the model is a regression model, inheriting from `nn.Module`. It takes concatenated user and movie embeddings (output from Node2Vec) as input. A linear layer (fully connected) is used for predicting the user ratings.
- **Model evaluation:** Mean Squared Error (MSE), Mean Absolute Error (MAE).

- **Loss Function and Optimizer:**
    - **Mean Squared Error (MSE)** is used as the loss function
    - **The Adam optimizer** model is used for model parameter updates
- **Challenges:** Generating walks and training Node2Vec embeddings is computationally expensive, for this reason, using this approach on much larger datasets would have taken a long time and caused slowdowns.

5.4 Supervised GNNs-based approaches

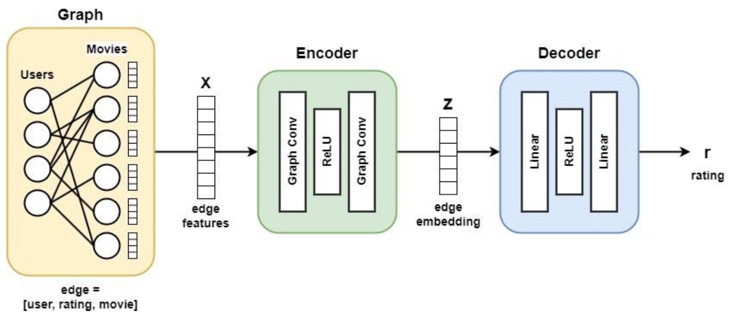
In this section are described the files containing the GNNs-based approaches implementation.

`GNN_regression_model.py`: (References: [6][7][8][9][10][11][11][12][13][14][15])

- **Encoders:** we implemented 3 types of encoders:
    - **GCN (class `GCNEncoder` implementing `GNNEncoderInterface`)**
    - **GraphSAGE (class `GraphSAGEEncoder` implementing `GNNEncoderInterface`)**
    - **GAT (class `GATEncoder` implementing `GNNEncoderInterface`)**

  All encoders have the same structure with two message passing layers:
    - Graph Conv
    - ReLU
    - Graph Conv

  where GAT in addition to each Graph Conv has a Linear layer.
- **Decoder (class `EdgeDecoder`):** the task is **edge regression** (i.e., the prediction of a rating for each user-movie relation). The decoder has the following structure:
    - Linear
    - ReLU
    - Linear
- **Model (class `Model`):** Graph Auto Encoder (GAE) model for edge regression task:
    - **Encoder:** (one of the 3 aforementioned encoders) converted into an equivalent heterogeneous encoder model:
        - **Input:** a dictionary X_{dict} containing edge features.
        - **Output:** a dictionary Z_{dict} containing edge embedding.
    - **Decoder:**
        - **Input:** a dictionary Z_{dict} containing edge embedding.
        - **Output:** the rating r for that edge.



Note: edges have the structure `(user, rating, movie)` (and `(user, rev_rating, movie)` for allowing message passing in both directions between nodes).
Important note: the edge regression can also be done between nodes that do not have an edge in common: negative edges (i.e., for movies the user has never rated before).

`GNN_based_CF.py`: (References: [6][7][8][9][10][11][12][11])

- **GNN model-based (class `GNN_Based_CollaborativeFilter` implementing `CollaborativeFilteringInterface`):** utilizes the GNN-based model (with the preferred encoder) for implementing a collaborative filter for movie prediction and new movie suggestion tasks. In particular, it provides the functions for training and evaluating the GNN-based model on the graph dataset and for doing predictions (on both positive and negative user-movie edges) and user-based suggestion of new movies.

6. Results

The following results are referred to the implemented collaborative filters for the user-movie rating prediction with the different approaches proposed:

SVD-based collaborative filter: this library was designed and built specifically for recommendation systems and represents the state of the art in this domain, especially for movie recommendations. From our test on the preprocessed tabular dataset that we built, and that we used as basing point for building and training all the approaches we have proposed in this paper, we have extracted the following performance: RMSE: 0.8937 and MAE: 0.6875.

Shallow node embedding-based collaborative filter: tests results for 10 epochs trainings:

External features	MRSE	MAE
None	1.0795	0.8298

Supervised GNNs-based collaborative filters: test results for 500 epochs trainings:

External features	GCN		GraphSAGE		GAT	
	MRSE	MAE	MRSE	MAE	MRSE	MAE
[titles, genres]	1.5674	1.2493	**1.0439**	**0.7863**	1.0143	0.8111
[titles, collections, genres]	1.8819	1.5021	1.0487	0.8180	1.0879	0.8348
[titles, genres, production_companies]	1.5287	1.2013	1.0301	0.8042	1.0263	0.7909
[titles, collections, genres, production_companies]	1.5932	1.2691	1.0465	0.8095	1.0591	0.8207

7. Conclusion

In this paper we have implemented different types of movie recommendation systems approaches for the tasks of movie prediction and movie suggestion, encompassing classical methods, shallow node embeddings, and supervised Graph Neural Networks (GNNs) based approaches. We have analysed differences, implementations, advantages, and disadvantages of each implemented approach.

From the result obtained we can see that none of the proposed approaches have beaten the SVD approach from Surprise library, however the results are still very good for GraphSAGE and GAT GNNs-based approaches. Surprisingly, unlike what expected, the results are very good also for the shallow node embedding-based approach, with performance really close to the one obtained for GNNs.

For this reason, these three approaches could easily be used within a real movie recommendation system application.

Lastly, we notice that, although GraphSAGE has obtained better performance, it is worth mentioning that GAT turned out to be appreciably faster than GraphSAGE and extremely faster than GCN, which any way achieved the worst performance. Hence, since GAT has slightly worst performance than GraphSAGE but it is faster, it may be preferrable in (almost) real-time applications.

Future works:
For future works it would be interesting to test an ensemble of approaches for trying to obtain better performance in the user-movie rating prediction task (extending to movie suggestion) than the SVD model, as the average of the outputs of: (shallow node embedding | GAT | GraphSage, SVD).

Other interesting tests would be to replicate the implemented approaches for more complex networks, such as the user-movie-actor network, or to integrate users' data from a user tabular dataset in order to improve the movie suggestion for users, exploiting also the connection between users' nodes in the graph (this would require to integrate these approaches in a real movie recommendation system application, or to use non-opensource datasets).

8. References

[1] MovieLens dataset from Kaggle [link]
[2] MovieLens dataset official web site [link]
[3] Movie Recommender Systems (Kaggle) [link]
[4] Recommender System: User based Collaborative filtering [link]
[5] Surprise library [link]
[6] Link Regression on MovieLens [link]
[7] Link Prediction on MovieLens [link]
[8] Link Prediction on Heterogeneous Graphs with PyG [link]
[9] Heterogeneous Graph Learning (Pytorch-geometric) [link]
[10] GRAPH Link Prediction w/ DGL on Pytorch and PyG Code Example | GraphML | GNN [link]
[11] Colab Notebooks and Video Tutorials [link]
[12] Source code for torch_geometric.nn.models.autoencoder [link]
[13] PyTorch Geometric tutorial: Graph Autoencoders & Variational Graph Autoencoders [link]
[14] Temporal Edge Regression with PyTorch Geometric [link]
[15] Graph Neural Networks with PyG on Node Classification, Link Prediction, and Anomaly Detection [link]
[16] Movie Recommendation Engine with GNN [link]
[17] Recommender System with Node2vec Graph Embeddings [link]
[18] Graph representation learning with node2vec [link]

9. Member contribution

- **Christian Francesco Russo:** (fraction of work: 65%)
  - Designed and implemented the structure/organization of the project.
  - Fully designed, implemented and written report about:
    - **Classical approaches:** files:
      - `main_classical_approaches.ipynb`
      - `tabular_dataset_handler.py`
      - `popularity_rankings.py`, `content_based_filtering.py`, `collaborative_filtering.py`, `hybrid_filtering.py`,
      - `SVD_based_CF.py`

        - **Supervised GNN-based approaches:** files:
            - `main_GNN_approaches.ipynb`
            - `graph_dataset_handler.py`
            - `GNN_regression_model.py`
            - `GNN_based_CF.py`
- **Lorenzo Spagnolo:** (fraction of work: 15%)
    - Tests on GNN embeddings.
    - Written report about the shallow node embedding-based approaches.
    - Tests on classical approaches.
- **Matteo Spinato:** (fraction of work: 20%)
    - Implemented the classical approaches (not present on git).
    - Fully designed, implemented and written report about:
        - **Embeddings approach:** files:
            - `main_embedding_approaches.ipynb`
    - Experimented the unsupervised approach.