Machine Learning - Final Project

Sara Nasr, Hosein Zarrabi

Abstract

In this report, we first discuss what recommender systems are and what are the main types of such systems, and introduce a typical dataset used in recommender systems research, namely The Movies Dataset. We then go over various algorithms, concisely overviewing their mathematical details. Finally, we evaluate the models on the movies dataset using a variety of metrics.

1 Introduction

The goal of recommender systems is to offer users customized recommendations of items that match their needs.

There are three main approaches to designing and implementing recommender systems: collaborative filtering, content-based filtering, and hybrid filtering.

- Collaborative filtering leverages the feedback of a group of users to generate recommendations for a target user. The main assumption behind Collaborative filtering is that users who have similar preferences in the past will also have similar preferences in the future. Collaborative filtering can be further divided into two subtypes: memory-based and model-based.
  - Memory-based Collaborative filtering uses similarity measures to compute recommendations based on the ratings matrix, such as k-nearest neighbors (kNN) or cluster-based approaches.
  - Model-based Collaborative filtering uses machine learning algorithms to learn a latent representation of users and items from the rating matrix, such as matrix factorization, singular value decomposition, or neural networks.
- Content-based filtering relies on the features of the items to generate recommendations for a target user. The main assumption behind Content-based filtering is that users will like items that are similar to the ones they have liked in the past. Content-based filtering can also use machine learning algorithms to learn a user profile that captures the user's preferences based on their feedback history.

- Hybrid filtering combines Collaborative filtering and Content-based filtering. Examples include Factorization Machines and Graph-based approaches.

It is important to note that a family of algorithms can have members in multiple approaches. For instance, only clustering the items based on their features can be considered as a content-based approach, meanwhile clustering the users can be considered a collaborative filtering approach.

In this report, we implement multiple recommendation systems and evaluate them on The Movies Dataset. The Movies Dataset is a collection of data on 45,000 movies and contains information such as budget, revenue, release dates, languages, production countries, and companies for each movie. The dataset also includes ratings of 270,000 users for these movies. This dataset consists of multiple .csv files, describing the movies and the ratings that the users gave to them. We detail how we cleaned, integrated, and preprocessed the data in subsection 3.1.

2 Methods

We now briefly go over the implemented algorithms.

2.1 Collaborative Filtering

2.1.1 k-nearest neighbors

K-nearest neighbors collaborative filtering uses the ratings of the k most similar users or items to make recommendations. For example, if you want to recommend a movie to a user, you can find the k users who have rated the movie and have the most similar ratings to the user, and then take the average of their ratings as the predicted rating. Alternatively, you can find the k movies that have the most similar ratings to the movie, and then take the average of their ratings by the user as the predicted rating.

The similarity between users or items can be measured by different metrics, such as cosine similarity or Euclidean distance.

More formally, Let R be the matrix of user-item ratings, where R_{ui} is the rating of user u for item i. Let S be the matrix of user-user or item-item similarities, where S_{uv} is the similarity between user u and user v, and S_{ij} is the similarity between item i and item j. Let k be the number of nearest neighbors to consider.
The predicted rating of user u for item i using user-based k-nearest neighbors collaborative filtering can be calculated as:

$$\hat{R}_{ui} = \frac{\sum_{v \in N_u(i)} S_{uv} R_{vi}}{\sum_{v \in N_u(i)} |S_{uv}|},$$

where $N_u(i)$ is the set of k users who have rated item i and have the highest similarities with user u.

The predicted rating of user u for item i using item-based k-nearest neighbors collaborative filtering can be calculated as:

$$\hat{R}_{ui} = \frac{\sum_{j \in N_i(u)} S_{ij} R_{uj}}{\sum_{j \in N_i(u)} |S_{ij}|},$$

where $N_i(u)$ is the set of k items that have been rated by user u and have the highest similarities with item i.

2.1.2 Matrix Factorization

Matrix factorization decomposes a large matrix of user-item ratings into smaller matrices of user factors and item factors. These factors represent the latent features of the users and items. By multiplying these factors, the technique can estimate the rating a user would give to an item they haven't yet rated.

More formally, let R be the matrix of user-item ratings, where R_{ui} is the rating of user u for item i. Let P be the matrix of user factors, where P_u is the vector of factors for user u. Let Q be the matrix of item factors, where Q_i is the vector of factors for item i. The predicted rating of user u for item i using matrix factorization collaborative filtering can be calculated as:

$$\hat{R}_{ui} = P_u^T Q_i.$$

The matrices P and Q are learned from the data by minimizing a loss function that measures the difference between the observed ratings and the predicted ratings, such as MSE.

2.1.3 Factorization Machines

Factorization machines generalize matrix factorization approaches.

Formally, Let $x \in \mathbb{R}^d$ be a feature vector, where d is the number of features. Let $y \in \mathbb{R}$ be the target variable.
A linear model, such as matrix factorization, predicts y as:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{d} w_i x_i,$$

where $w_0 \in \mathbb{R}$ is the global bias, and $w_i \in \mathbb{R}$ are the feature weights.
A factorization machine extends the linear model by adding pairwise interactions between features:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} v_i^T v_j x_i x_j,$$

where $v_i \in \mathbb{R}^k$ are the factor vectors for feature i, and k is the dimensionality of the factorization. The factor vectors capture the characteristics of the features, and their inner product measures the interaction strength between features. The parameter k controls the expressiveness and complexity of the model.

2.1.4 Deep Factorization Machines

Deep factorization machines are models that combine factorization machines and deep learning.

Deep factorization machines consist of two components: an FM component and a deep component. The FM component is the same as factorization machines, which model the pairwise interactions between features. The deep component is an MLP, which models the higher-order and nonlinear interactions between features. These two components share the same input embeddings and their outputs are summed up as the final prediction.

Formally, Let $x \in \mathbb{R}^d$ be a feature vector, where d is the number of features. Let $y \in \mathbb{R}$ be the target variable. A deep factorization machine predicts y as:

$$\hat{y}(x) = w_0 + w^T x + x^T V V^T x + h(W^{(L)}(h(W^{(L-1)}(...h(W^{(1)}x + b^{(1)})...)) + b^{(L)}),$$

where $w_0 \in \mathbb{R}$ is the global bias, $w \in \mathbb{R}^d$ is the vector of feature weights, and $V \in \mathbb{R}^{d \times k}$ is the matrix of factor vectors, $W^{(l)} \in \mathbb{R}^{m_l \times m_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{m_l}$ are the weights and biases of the MLP, and h is an activation function.

2.1.5 Neural Graph Collaborative Filtering

Neural Graph Collaborative Filtering (NGCF) is a graph neural network model for recommender systems. It is designed to learn user and item embeddings from the user-item interaction graph, by performing embedding propagation on it.

Neural Graph Collaborative Filtering consists of three components: an embedding layer, a graph convolutional layer, and a prediction layer. The embedding layer initializes the user and item embeddings from a Gaussian distribution. The graph convolutional layer propagates the embeddings over the user-item interaction graph, which is treated as a bipartite graph. The prediction layer computes the inner product between the user and item embeddings to obtain the predicted rating.

Formally, Let $G = (U, I, E)$ be the user-item interaction graph, where U and I are the sets of users and items, and E is the set of edges representing the interactions. Let $n = U + I$ be the number of nodes in the graph. Let $e_i \in \mathbb{R}^d$ be the embedding vector of node i, where d is the embedding dimension. The embedding layer initializes the embeddings of all nodes from a Gaussian distribution:

$$e_i^{(0)} \sim \mathcal{N}(0, \sigma^2 I_d), \quad i = 1, ..., n.$$

The graph convolutional layer propagates the embeddings over the graph using a message-passing scheme:

$$e_i^{(l+1)} = \text{LeakyReLU}\left(W^{(l)} e_i^{(l)} + \sum_{j \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_j|}} W^{(l)} e_j^{(l)} \right), \quad i = 1, ..., n,$$

where l is the layer index, $W^{(l)} \in \mathbb{R}^{d \times d}$ is the weight matrix at layer l, N_i is the set of neighbors of node i, and LeakyReLU is the activation function.

The prediction layer computes the inner product between the user and item embeddings to obtain the predicted rating or preference:

$$\hat{y}_{ui} = (e_u^L)^T e_i^T, \quad (u, i) \in E,$$

where e_u^L and e_i^L are the last layer embeddings of user u and item i.

2.1.6 LightGCN

LightGCN is a simplified version of NGCF, which removes the weight matrices and the activation functions from the graph convolutional layer. This makes the model simpler and more efficient, without sacrificing much performance. There are some other minor changes as well, such as changing the loss function, regularizing the embeddings instead of the parameters, and summing up the embeddings of all the layers, not just the last.

2.2 Content-based Filtering

2.2.1 Cluster-based

Content-based clustering approaches in recommender systems use the features of the items to group them into clusters, and then use the cluster membership to generate recommendations. Our specific approach is summarized in algorithm 2.2.1. In a nutshell, we calculate an embedding for a user based on the features of the movies that they like and then find new movies to recommend to them based on minimizing some distance criteria between the embedding and the movie features.

Algorithm 1 Content-based clustering recommender system

Require: *movies*, *ratings*, *min_rating*, *n_clusters*, K

1: Perform k-means clustering on *movies* with *n_clusters* as the number of clusters, and obtain the cluster labels C
2: **for** each user u **do**
3: Find the movies m_u that u has rated higher than or equal to *min_rating* in the rating matrix *ratings*
4: Find c_u, the cluster indices of m_u
5: For each cluster index in c_u, find the movie(s) that u likes the most, h_u
6: Normalize the ratings of h_u and obtain the weights w_u
7: Calculate the user embedding e_u by performing a weighted summation over the features of the movies in h_u, with weights being w_u
8: Recommend K unseen movies to u based on the distance between their features and e_u, using a distance measure such as Euclidean distance, and select the movies with the smallest distance
9: **end for**

2.2.2 Similarity-based

The idea of this algorithm is to recommend items that are similar to the items that the user liked the most. To do this, we need to have some information about the ratings of the users and the features of the items. First, we find the items that each user rated the highest. Next, for each top-rated item, we look for the most similar item among the unseen items, based on their features. Finally, we recommend the most similar items to the user, in the same order as their top-rated items. This approach is summarized in algorithm 2.2.2.

Algorithm 2 Simple Content-Based Filtering

Require: $R \in \mathbb{R}^{m \times n}$, $X \in \mathbb{R}^{n \times d}$, k
Ensure: $Y \in \mathbb{R}^{m \times k}$
for $u = 1, ..., m$ **do**
  $T_u \leftarrow \{i : R_{ui} \geq R_{uj}, \forall j \neq i, i = 1, ..., n\}$
  $U_u \leftarrow \{i : R_{ui} = 0, i = 1, ..., n\}$
  for $i \in T_u$ **do**
    $S_i \leftarrow \arg\max_{j \in U_u} \text{sim}(X_i, X_j)$
    $Y_{ul} \leftarrow S_i$, where l is the index of i in T_u
  end for
end for
return Y

3 Experiments

We now provide an overview of the implementation details of this project.

3.1 Data Preprocessing

1. first, we convert the 'id' values into numeric data type, replacing any values that cannot be converted with NaN, and also convert the values in the 'tmdbId' column of the 'links' dataset to a specific data type (int64) and also we rename the 'id' column to 'tmdbId'.

2. we merge the links DataFrame based on the 'tmdbId' creating a new merged DataFrame assigned to 'movies-metadata', we filter the rows in merged-data frame to exclude any rows where the 'title' column value is "Unknown", also we convert the 'tmdbId' column values to integer data type, and finally reset the index to a default numeric index, dropping the previous index.

3. we fill any missing values in the 'genres' column with an empty list. Then, parse the strings in the 'genres' column to convert them into a list of dictionaries. Extract the 'name' values from the dictionaries in the list, creating a new list of genre names. Perform similar transformations for the

'production-companies' and 'production-countries' columns, filling missing values with an empty list, parsing strings to convert them into a list of dictionaries, and extracting the 'name' values from them.

4. we apply a custom function (try-join) to each element in the 'genres' column. This function joins the list of genre names into a single string. we apply this step for 'production-companies' and 'production countries'.

5. we remove the 'belongs-to-collection', 'poster-path', 'homepage', 'video', 'imdb-id' and 'original-title' columns from the DataFrame, then we get the unique values in the 'adult' column of the DataFrame. we remove the 'adult' column from the DataFrame.

6. we fill any missing values in the 'overview' and 'tagline' column of the DataFrame mm2 with an empty string, and create a new 'content' column in mm2 by concatenating the values from the 'overview' and 'tagline' columns, then we remove the 'overview' column from the DataFrame. Remove the 'tagline' column from the DataFrame mm2.

7. we convert the 'release-date' column to a datetime data type then we extract the year from the datetime values and store them in a new 'release-year' column. If a value cannot be converted, it will be replaced with NaN.

8. we convert the 'release-date' column to a datetime data type and extract the day name of each date. Store the day names in a new column called 'release-day' after that we remove the 'release-date' column from the DataFrame and also we convert the 'popularity' column values to float data type, then we Create a list called 'popularity-list' by selecting the non-null values from the 'popularity' column, after all, we replace any missing values (NaN) in the 'popularity' column of mm3 with the median value from the 'popularity-list'.

9. now we create for each 'voteavg-list', 'vote-average','votecnt-list' column by selecting the non-null values from these columns. we sort the DataFrame based on the columns 'popularity', 'vote-average', and 'vote-count' in descending order, and finally, we reset the index of mm3 to a default numeric index, dropping the previous index.

10. we do preprocess such as removing invalid and NaN values or filling any missing values with median and mean of columns, converting to the proper type for 'budget' and 'spoken-languages' columns.

11. we use 'translator-API' to convert the extracted language to their English translation. We do the preprocessing process for the remaining columns, the final data frame contains these columns: 'tmdbId', 'title', 'release-year', 'release-day', 'genres', 'original-language', 'runtime', 'content', 'production-companies', 'budget', 'revenue', 'production-countries', 'status', 'popularity', 'vote-average', 'vote-count'.

12. We extract features of the movies using PCA on the numerical columns, and MCA on the categorical columns, concatenating the resulting features.

3.2 Models Evaluation

We construct a *pandas* dataframe with n rows and $m + 2$ columns, where n is the number of users and m is the number of models. The two columns that do not correspond to an implemented model are $userId$ and $actual$. Each entry under the model columns and actual column stores a list of K movie Ids, where we set $K = 10$. We calculate various metrics using the *recmetrics* library. We refer the reader to this post for an explanation of the metrics mentioned here.

1. Mean Average Precision @ K (Figure 1)

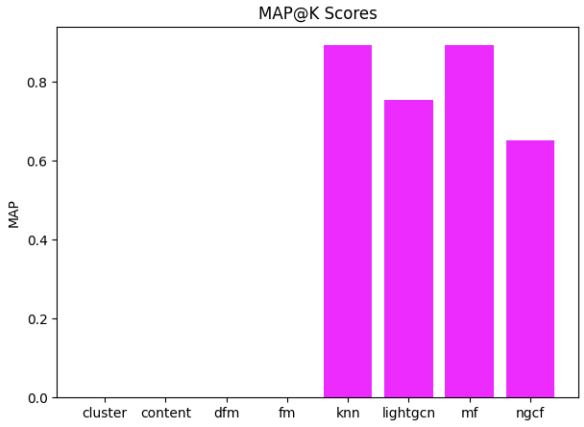

Figure 1: Mean Average Precision @ K

2. Mean Average Recall @ K (Figure 2)
3. Novelty (Figure 3)

   Note that the content-based model has a novelty of inf, because it recommends a movie that has no ratings! This can be an advantage of content-based models, as they can better handle the cold-start problem.

4. Coverage (Figure 4)
5. Personalization (Figure 5)
6. Intra-List Similarity (Figure 6)
7. Summary (Figure 7)

   This plot summarizes Coverage, Personalization, and Intra-List Similarity.

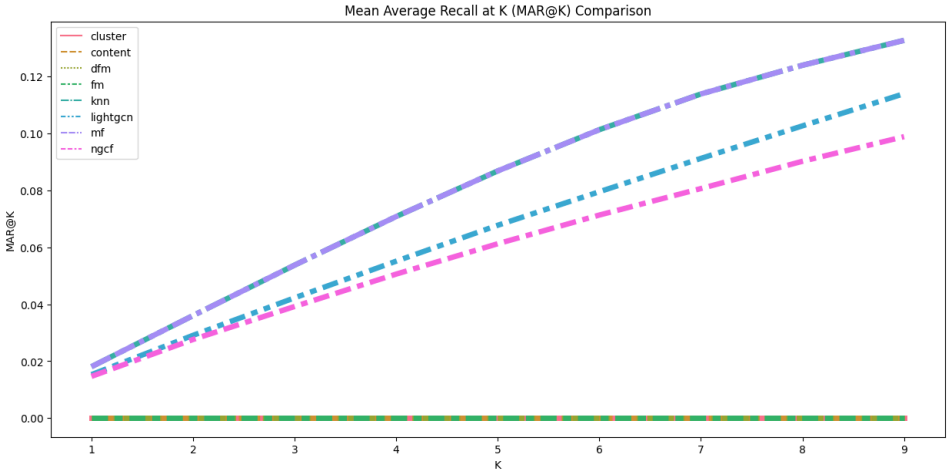

Figure 2: Mean Average Recall @ K

4 Extra Points

1. implemented algorithms:
   (a) k-nearest neighbors
   (b) Matrix Factorization
   (c) Factorization Machines
   (d) Deep Factorization Machines
   (e) Neural Graph Collaborative Filtering
   (f) LightGCN
   (g) Content-based Clustering (Not Extra)
   (h) Content-based using item similarity
2. All the evaluation metrics mentioned in this post are implemented
3. Made the user interface more attractive

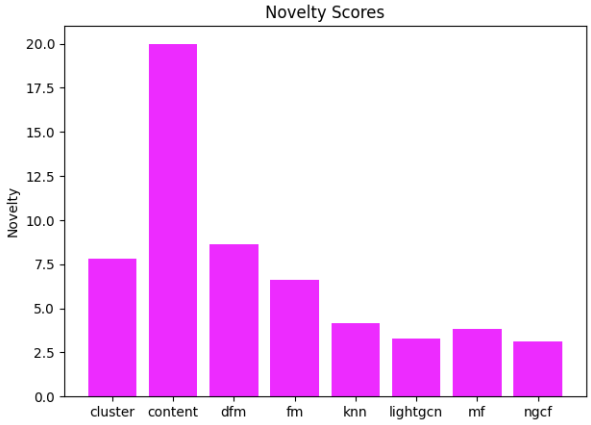

Figure 3: Novelty

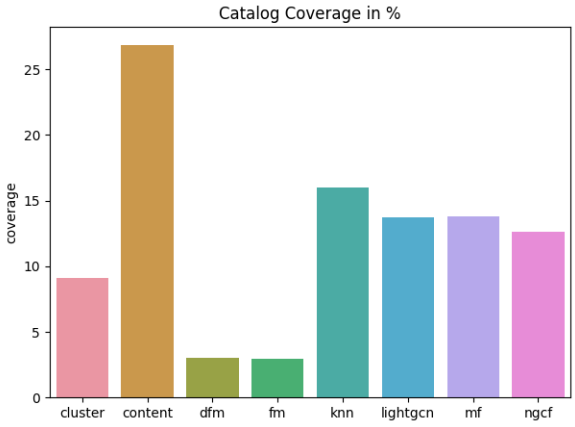

Figure 4: Coverage

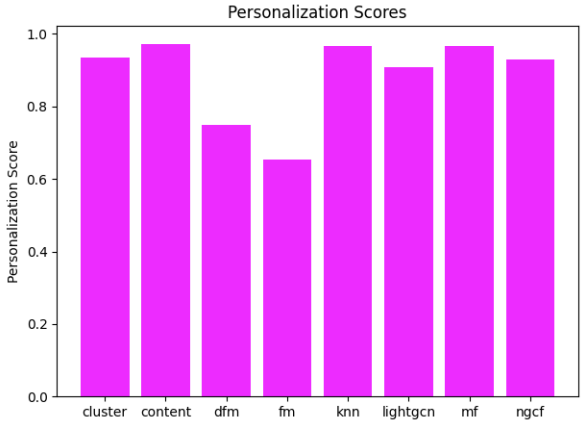

Figure 5: Personalization

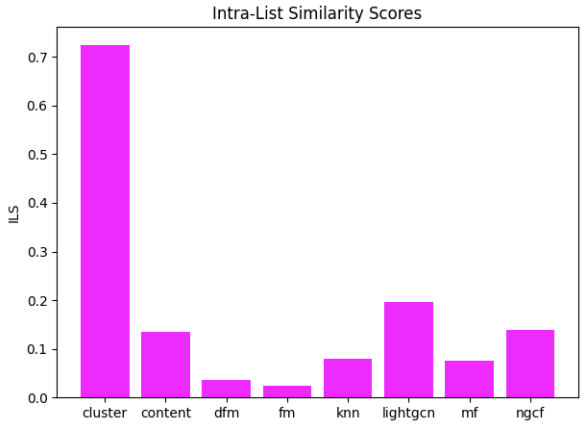

Figure 6: Intra-List Similarity

Figure 7: Summary