Hybrid-Movie Recommendation System

Pushpendra Singh - 190040082
Prerak Gandhi - 17B030004

Link to code: Movie_recommender_system.ipynb - Colaboratory (google.com)

Abstract

Whether we talk about products, movies, or any business, recommendation systems play a vital role in a company's growth, especially in an online scenario. It helps keep customers satisfied and engaged and encourages them to explore more products. A great Recommendation engine makes browsing content more accessible and helps users find things they may be interested in but would not have thought to look for on their own. Thus, it navigates users to useful content in large corpora. There is a constant need to evolve the recommendation systems and explore better approaches that combine with traditional algorithms.

Currently, the interest of big companies is shifted onto hybrid recommendation systems that provide much better recommendations than traditional approaches. It mainly combines a content-based recommendation approach and collaborative filtering based approach. This report describes the new recommendation system, a movie recommendation system based on a hybrid recommendation algorithm, which satisfies a user by providing the best and most efficient recommendations for movies.

Introduction

Basic steps involved in developing a product recommendations plan include collecting and compiling data, analysing the data, delivering content based on it, studying the results, and taking appropriate measures to improve your campaign.

Common architecture of recommendation system includes the following:

- **Candidate generation -** It is the first stage of recommendation. Given a query, the system generates a set of relevant candidates. Relevant candidates depend on the approach used for the recommendation system (e.g. content-based, collaborative filtering etc.)
- **Scoring -** Given a user or a movie, the Model provides scores to the relevant candidates based on sources like similarity measure, popularity, user history, the freshness of the item with respect to the user etc.
- **Ranking** - Candidates are arranged in descending order of their scores to allot rank. Candidates up to a particular rank are presented as recommendations to the user. Items explicitly disliked by users are removed.

Common Terminologies used:

- **Items** - The entities that the system recommends. Movies in our case.
- **Query** - Information used by the system to make recommendations. User_id and info or movie_id (if the user clicks a movie/searches for a movie).
- **Embeddings -** A mapping from a discrete larger set to a smaller embedding vector space.
- **Similarity measure** - It is a scalar measure of similarity between two vectors. Cosine, dot product and euclidean distance are some similar measures. Dot product also considers the magnitude of the vector, which is proportional to the movie's popularity.

The output of our model - Given user_id - Recommendation to the user .

Related Works

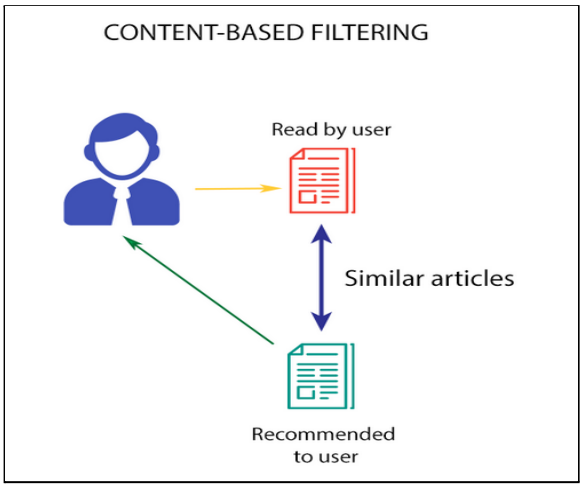
MOVREC (Kumar, 2015) is a movie recommendation system based on a collaborative filtering approach. The system also provides the user to select attributes on which he wants the movie to be recommended. (Campos, 2010) proposed a new system that combines the Bayesian network and collaborative filtering. The proposed method is optimised for the given problem and provides probability distributions to make valuable inferences. A hybrid system by (Kaur, 2015) uses a mix of content and a collaborative filtering algorithm. The context of the movies is also considered while recommending. The user-user relationship, as well as the user-item relationship, plays a role in the recommendation.

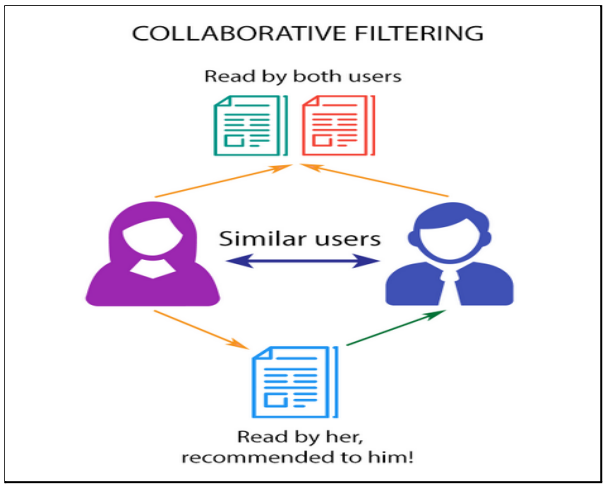
The papers - Neural Collaborative Filtering (He et al, 2017) and Deep Neural Networks for Youtube Recommendations (Covington et al, 2016) introduced ground-breaking papers in the field of recommendation systems. They used neural networks to solve the limitations of the existing systems.

Existing Approaches

There are various techniques for recommendation generation that have been proposed in the last decade. Many of them have been successfully deployed in commercial environments. The recommender systems are of 6 classes:

1. **Content-based:** These systems learn to recommend items similar to the ones the user liked. If a user clicks on a movie, such a system suggests movies with features identical to those selected.





2. **User-based Collaborative-Filtering:** This approach recommends to the user the items that other users with a similar taste or interest previously liked. It may use similarity measures like dot product to find users having similar rating histories.
3. **Knowledge-based:** This system recommends items based on specific domain knowledge about the user's needs and preferences.
4. **Demographic-based:** The user is recommended items using their demographic information such as age, gender, language, location, etc.
5. **Utility-based:** The system makes suggestions based on the computation of the utility of each object to the user.
6. **Hybrid:** This approach is the combination of multiple methods mentioned above. It is the most popular recommendation system as it shows better results than any individual approach.

Dataset Description

Although we have explored datasets from various sources and python libraries like IMDB, TMDB, Kaggle etc., the Movielens-100k dataset is used in this project as it is a simple, clean and efficient dataset.

Link for the dataset used - Index of /datasets/movielens (grouplens.org) We have used u.user, u.data and u.item files from the ml-100k zip file given in the link.

u.user file contains features/simple demographic information about the 943 users. New users can be added to the file. Data layout consists of attributes - user_id (unique identification number of user), age, sex (gender) occupation, zip_code. These attributes define the user characteristics.

The u.data file contains the 100,000 ratings (1 to 5) given by the 943 users to each movie in our dataset. The data is randomly ordered. It captures the user's characteristics based on their interest and the popularity/content of the movie (e.g. movies rated/watched by only users above age 18 must be adult movies). Each user has rated at least 20 movies. New users and movies data must also be updated in the u.data file and the u.user and u.item files. Data layout consists of attributes - user_id, movie_id, rating (1 to 5), timestamp (Unix seconds since 1/1/1970 UTC and not used in our model).

The u.item file contains features/information of 1682 movies used for the system. Data layout consists of attributes - movie_id (unique identification number of the movie), movie title, movie release_date, video_release_date (Nan/empty for all films), IMDB URL followed by 19 genres (unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western) indicated in binary form (1 means the movie is of a given genre, 0 shows it is not). Movies can be of more than one genre.

Proposed Approach for Hybrid Recommendation system

The rating matrix could be extensive, and most of the entries are unobserved since a given user will only rate a small subset of movies. For efficient representation, we will use a tf.SparseTensor. A SparseTensor uses three tensors to represent the matrix: tf.SparseTensor(indices, values, dense_shape) represents a tensor, where a value "Aij = a" is encoded by setting indices[k] = [i, j] and values[k] = a. The last tensor dense_shape is used to specify the shape of the full underlying matrix.

We have created a Collaborative filtering helper class, a simple class to train a model using stochastic gradient descent algorithm. The class constructor takes user embeddings U, movie_embeddings V, a loss to optimise (a tf.Tensor) and list of metrics dictionaries(optional), each mapping a string (the name of the metric) to a tensor. These are evaluated and plotted during training (e.g. training and test errors).

Two methods for hybrid based recommendation system are shown in the project:

Matrix Factorization method -

User embeddings and Movie embeddings are randomly initialized in this approach. Multiplication of user embedding and Movie embedding forms a representation matrix which should be ideally the same as rating matrix of movies by users. User and movie embeddings are updated so as to make the representation matrix similar to rating matrix by minimizing the loss between the representation matrix and rating matrix. Similarity measure (eg. dot product or cosine) between user embeddings and movie embeddings is used for movie recommendation. Loss is regularized with common L2 regularization and gravity term which pushes the rating prediction of any user and movie embedding pair towards zero.

Deep neural network -

Features of the user are used in this approach. Deep neural networks are used to predict output from these features. Output layers have nodes equal to the total number of movies in the dataset. Output layer represents probability of movies being rated by the user. Last hidden layer is mapped to probability distribution through the softmax layer. Last hidden layer of DNN thus represents the user embeddings and weights from it to probability distribution output represents movie embeddings. Weights of DNN are optimized to minimize the softmax cross-entropy loss by comparing the probability distribution output of the softmax model with the distribution of users' ratings for each movie in actual rating data

A function is defined to compute the scores using two similarity measures. We can use different similarity measures, and these can yield different results. We will compare the Dot product and the Cosine score of the user and movie embeddings. Model can output the recommendations for a user given its user id and can recommend movies based on the movie selected by the user.

Conclusion

Matrix factorisation is a simple embedding model. It is a hybrid Collaborative Filtering approach for recommender systems. Matrix factorisation typically gives a more compact representation than learning the full matrix. Matrix factorisation finds latent structure in the data, assuming that observations lie close to a low-dimensional subspace. The model's prediction for a given (user, item) pair is the dot product of the corresponding embeddings. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item. Side features are any features beyond the query or item ID. For movie recommendations, the side features might include country or age. Including available side features might improve the quality of the model.

References

- Deep Neural Networks for YouTube Recommendations
- https://d2l.ai/chapter_recommender-systems/index.html

- Xin et al., Folding: Why Good Models Sometimes Make Spurious Recommendations
- Bengio and Senecal, Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model
- Content-based Movie Recommender system (without ML)
- Classifying Different Types of Recommender Systems | BluePi
- Hybrid movie recommendation system article by Dhrumil D. Shah1 Dr Rajesh B. Ingle

Bibliography

- Luis M. de Campos, Juan M. Fernández-Luna *, Juan F. Huete, Miguel A. Rueda-Morales; “Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks”, International Journal of Approximate Reasoning, revised 2010
- Urszula Kuzelewska; “Clustering Algorithms in Hybrid Recommender System on MovieLens Data”, Studies in Logic, Grammar and Rhetoric, 2014
- He et al., “Neural Collaborative filtering”, Proceedings of the 26th International Conference on World Wide Web, 2017

- Paul Covington, Jay Adams, Emre Sagin, “Deep Neural Networks for Youtube Recommendations”, Proceedings of the 10th ACM Conference on Recommender Systems, ACM, New York, NY, USA (2016)