Capstone Project Proposal

Movie Similarity

David Pinto

2020-10-03

Contents

Introduction

The **Movie Similarity** project aims to recommend similar items given a particular movie, so that users that already watched the later are likely to enjoy the recommended movies. It involves **Recommender Systems**[1] theory and **Similarity Matching**[2] technique.

Recommending similar items is a very common task in many online selling platforms[3]. It provides recommendations like: **related to items you've viewed**, **recommended for you based on your previous purchase**, **customers who bought this item already bought**, so on. The main challenges involved are:

The cold start problem

How to recommend recent added products with few or no associated interaction data?

One way to cope with this consists in take advantage of product metadata (content features): `name`, `description`, `category`, `price level`, so on. Then we can link a product with others by content similarity. We call this as **content based** approach.

Item popularity

According to the **long tail phenomenon**[4], selling niche products has many advantages. Users are likely to pay more for niche products because they cannot find it everywhere. However, best seller products have more interaction data available to train a recommender system so that the fitted model may be subjected to popularity bias. In other words, the model will be more likely to recommend popular items.

Again, using content features to find similar items reduces popularity bias. But interaction data is very important to understand user behavior and cannot be discarded. So, in this project we will use **Collaborative Filtering** techniques that take into account item popularity.

Dataset

In this project we will combine two datasets, one that provides movie ratings to be used as interaction data, and another that provides movie metadata to be used as content features.

Movie Ratings

The full MovieLens (link) dataset, which contains `27M ratings` given to `58K movies` by `280K users`.

Movie Metadata

Data retrieved from the TMDB Open API (link) and available as a Kaggle Dataset (link). It contains information on 45k movies featured in the full MovieLens dataset. Features include title, description, genre and cast&crew information.

Movie Embeddings and Similarity Matching

First of all, we need to extract item embedding[5] features, which are low-dimensional dense vectors that captures some of the semantics of the item by placing semantically similar items close together in the embedding space. It allows efficient nearest neighbors search, so that we can find similar movies fast in application time.

Content Embeddings

Embedding features extracted from text fields like title and description, and from categorical fields like genre and cast&crew. More details in the Project Design section.

Collaborative Filtering Embeddings

Embedding features extracted from user-movie interactions given by the ratings. More details in the Project Design section.

Users Intersection as a Similarity Measure

Let's consider every rating (from 1 to 5) above 3.5 as an explicit feedback that the user liked the movie. Then we can measure the similarity between a pair of movies (x, y) as:

$$S(x, y) = \frac{N_{xy}}{\sqrt{N_x N_y}}$$

where N_x is the number of users that liked movie x, N_y is the number of users that liked movie y, and N_{xy} is the number of users that liked both x and y.

So, $S(x, y)$ close to 1 is a strong evidence that users who liked x will like y too.

But this kind of similarity is very sparse, providing a discrete recommendation. In other words, we will have many movies with few or none similar items to be recommended.

The embedding features solve this problem. Since we are computing distances between pairs of movies in a dense and low-dimensional space, the embedding features provide a continuous recommendation. We are able to find similar items to any movie, with high evidence (low distance) or low evidence (high distance).

Model Evaluation

How close the similarity match with embedding features can get to the similarity match given by the user's likes?

Ranking correlation

In a attempt to answer this question, we can randomly select n test movies and get their top N similar using the $S(x, y)$ measure. Then, for each test movie, we get the distances $D(x, y)$ from the top N similar using the embedding features. Finally, for each test movie, we apply the Spearman's Rank-Order Correlation Coefficient (ρ) between S and $1/D$, which is equal to the Pearson's correlation between the rank values of them.

For all test movies, we get an average rank correlation:

$$\frac{1}{n} \sum_{i=1}^{n} \rho(S_i, 1/D_i)$$

Precision and Recall at K

Given a threshold t_S for $S(x, y)$, we will assume that $S(x, y) \geq t_S$ indicates a strong/relevant similarity. So, given the K nearest neighbors from embedding features of a test movie:

$$\text{precision@k} = \frac{\#\text{relevants@k}}{K}$$

$$\text{recall@k} = \frac{\#\text{relevants@k}}{\#\text{relevants}}$$

For all test movies, we get an average Precision@K and an average Recall@K.

Project Design

Building Content Embeddings

For text features (title, description, keywords and user comments) we will concatenate them all and extract a sparse TF-IDF matrix using the `TfidfVectorizer` (link).

For the categorical features (genre, cast and crew) we will expand them into sparse binary features.

Both representations result in high-dimensional and sparse matrices. So, we will combine them and apply the sparse `TruncatedSVD` (link) algorithm to get low-dimensional and dense representations.

The `implicit` (link) package provides some fast factorization algorithms more powerful than SVD. They are great alternatives!

Building CF Embeddings

The `surprise` (link) package provides many classical factorization algorithms for explicit feedback problems. We can apply than in the user-movie ratings matrix to extract latent factors for the users and movies. Then, the movie factors can be used as embedding features. But we will try an approach based on likes and dislikes. Ratings above 3.5 will be considered likes (+1), while ratings below 3.5 will be considered dislikes (-1). From this assumption, we can build a user-movie matrix with (-1,+1) values and use a logistic loss to factorize this matrix. The `LighFM` (link) package provides a logistic loss factorization algorithm with additive bias. So, it takes into account the user and the movie bias, abstracting popularity away from the embedding features. This way it tries solve the item popularity problem.

Similarity Match

After computing the embedding features we will recommend similar movies using an approximate nearest neighbors search with cosine similarity. The `faiss` (link) package provides an extremely fast ANN algorithm.

Hybrid Approach

We will evaluate the recommendations from the content embedding features and compare it with the performance for the CF embedding features. Then, as a final attempt, we will try a hybrid approach, as follows:

1. Select a target movie for which we will recommend similar ones.

2. Compute the cosine similarity between the target and the candidates using the content embedding features.
3. Compute the cosine similarity between the target and the candidates using the CF embedding features.
4. Order the candidates by the sum between the computed similarities in the last 2 steps.

This approach combines content information with interaction information and may provide better recommendations.

Movie Similarity Dashboard

Finally, we will build a web application on which the user can choose a movie and get recommendations of similar movies. For this we will use the `Streamlit` (link) framework.

References

[1] Google's Introduction to Recommendation Systems (link).

[2] Building a real-time embeddings similarity matching system (link).

[3] The Amazon Recommendations Secret to Selling More Online (link).

[4] Anderson, Chris. The long tail: Why the future of business is selling more for less. Hyperion, 2006.

[5] Overview: Extracting and serving feature embeddings for machine learning (link).