Recommender System Project - Item based Collaborative filtering

Team: Noufan, Subramanian, Akshaya

Objective

This documentation is to describe the use and operation of an item-item based collaborative filtering build for movie recommendation engine.

Recommender System

Recommender systems can be loosely broken down into three categories: content based systems, collaborative filtering systems, and hybrid systems (which use a combination of the other two).

Collaborative Filtering

Collaborative filtering approach builds a model from a user's past interactions with the items (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. Based on this the two types can be user-user collaborative filtering and item-item collaborative filtering. This model is then used to predict items (or ratings for items) that the user may have an interest in.

For this project we've used Collaborative filtering which is by Item-Item based recommendation.

Algorithms Used

- K – Nearest Neighbors (KNN)
- Local Sensitive Hashing (LSH) using Jaccard Distance
- LSH using Cosine Similarity

K-Nearest Neighbors [KNN]

To implement an item based collaborative filtering, KNN is a perfect go-to model and also a very good baseline for recommender system development. KNN does not make any assumptions on the underlying data distribution but it relies on item feature similarity. When KNN makes inference about a movie, KNN will calculate the "distance" between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest neighbor movies as the most similar movie recommendations.

Working Algorithm: KNN

- First, we transform the dataframe of ratings into a proper format that can be consumed by a KNN model. We want the data to be in an m x n array, where m is the number of movies and n is the number of users. To reshape dataframe of ratings, we'll pivot the dataframe to the wide format with movies as rows and users as columns. Then we'll fill the missing observations with 0s since we're going to be performing linear algebra operations (calculating distances between vectors).
- Our dataframe is an extremely sparse matrix, We definitely don't want to feed the entire data with mostly 0s in float32 datatype to KNN. For more efficient calculation and less memory footprint, we transform the values of the dataframe into a Scipy Sparse Matrix.
- Now our training data has a very high dimensionality. KNN's performance will suffer from curse of dimensionality if we use "euclidean distance" in its objective function. Euclidean distance is unhelpful in high dimensions because all vectors are almost equidistant to the search query vector (target movie's features). Instead, we've used cosine similarity for nearest neighbor search.

Locality Sensitive Hashing [LSH]

Locality Sensitive Hashing (LSH) is a hashing based algorithm to identify approximate nearest neighbors. LSH is a generic hashing technique that aims, as the name suggests, to preserve the local relations of the data while significantly reducing the dimensionality of the dataset. This is a popular approach to handle nearest neighbor search in high dimensional data.

LSH works on the principle that if there are two points in feature space closer to each other, they are very likely to have same hash (reduced representation of data). LSH primarily differs from conventional hashing (aka cryptographic) in the sense that cryptographic hashing tries to avoid collisions but LSH aims to maximize collisions for similar points. The hash collisions make it possible for similar items to have a high probability of having the same hash value. The popular approaches for implementing LSH are discussed below

MinHash

minHash is suitable hash function for Jaccard similarity that is usually used to measure how close sets are, although it is not really a distance measure. That is, the closer sets are, the higher the Jaccard similarity. It is defined as the ratio of the sizes of the intersection and union of two sets. The idea of the MinHash scheme is to reduce this variance by averaging together several variables constructed in the same way.

SimHash

With SimHash, documents are represented as points in a multi-dimensional space. The similarity sim(x,y) of two documents x and y is the cosine of the angle between them – the cosine distance. In simHash, it divides the space with hyperplanes. For a given plane, each document ends up in one of the two regions which the plane separates. the closest two documents are, the more likely they will be on the same region for a random plane.

Working Algorithm: LSH

- We have implemented LSH using random projection method i.e. Simhash. We consider each user rating for a movie as individual features and generate hash tables by binning observations with similar hash values together.
- To generate the hash, we dot product between a random vector and the feature vector and convert to binary hash form. Because of splitting using a random hyperplane we it is likely that many of the similar items may not fall in same bin. So we compute multiple hash tables with different random projections to improve the performance.
- When a new observation comes in, we generate its feature vector and query the hash tables to get the similar candidates. After this step we evaluate the similarity between the candidate and the new observation using Jacard and cosine similarity and recommend the most similar items.

- Identify the weaknesses of the item-based approach and illustrate them with some examples from the recommender you've implemented.

General weakness of item based collaborative filtering:

- popularity bias: recommender is prone to recommender popular items
- item cold-start problem: recommender fails to recommend new or less-known items because items have either none or very little interactions

Illustration of weakness:

Item based recommenders are static and give obvious recommendations. It might not interest all users who will expect variety and freshness in their recommendations.

```
Query movie name The Number 23
Movie suggestion by cosine Die Hard
Movie suggestion by Jacard Die Hard
```

User's varied interests is not captured fully by the algorithm.

Problems and reflections:

- There is no direct package for implementing locality sensitive hashing that suit our problem requirements. We used random projection method and implemented the algorithm from scratch.
- Using cosine similarity, the similarity measures are very similar. Using pearson correlation i.e. centered cosine solves this issue.
- Using a single hash table does not generalize well. We generated multiple hash tables but this increased the time for hash table generation.
- Memory issues can be solved using SparkRDD or using a cloud environment.

EVALUATION:

Evaluation is done by testing rating prediction performance of the three algorithms against randomly selected test movies in terms of execution time and RMSE in the Evaluation.ipynb file.

Sources:

- https://medium.com/engineering-brainly/locality-sensitive-hashing-explained-304eb39291e4
- https://towardsdatascience.com/locality-sensitive-hashing-for-music-search-f2f1940ace23
- https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134
- https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea
- Class slides of Collaborative Filtering