Abstract:

This technology review compares collaborative filtering approaches in recommender systems, briefly describes different ways those approaches are implemented and offers some guidance for how to select an appropriate method for an application.

Background:

In the field of information retrieval, there are two primary approaches for meeting a user's information requirements. The first approach is search (also known as 'pull'), which meets ad hoc information needs by responding to user-initiated queries. The second approach is recommender systems (also known as 'push'), which meets the long-term information needs of a user and is initiated by the application. Recommender systems can be further grouped into two sub-categories: content-based filtering and collaborative filtering. In the broadest sense, content-based filtering relies on the internal content of items along with a user's preferences in order to assess the relevance of an unseen item, whereas collaborative filtering utilizes information from the entire user-base to assess the relevance of unseen items.

Technology Review: Collaborative Filtering Algorithms

Within the ecosystem of information retrieval, collaborative filtering provides a distinct utility to the user, which is the ability to "discover" relevant items that would otherwise be missed by content-based filtering. This is partly because the universe of potentially relevant items is expanded in proportion to the user-base rather than relying solely on information gathered from a single user. This paper will explore and compare the most common approaches in collaborative filtering, then offer some guidance on how to select an appropriate method for an application.

Under the umbrella of collaborative filtering, the two main approaches for filtering recommendations are item-item and user-item. In the simplest sense, item-item filtering defines the similarity of items, whereas user-item filtering defines the similarity of users. Put another way, item-item filtering might say, "viewers who liked 'Game of Thrones' also liked 'His Dark Materials,'" while user-item filtering might say, "viewers who are very similar to you liked 'Breaking Bad.'"

It is worth mentioning that Item-item filtering is actually quite similar to the idea of content-based recommendation systems in that both systems are attempting to define the similarity of unseen items to a knowledge base of preferred items. The obvious difference is that content-based filtering relies on the internal content of items to define similarity, while item-item collaborative filtering relies on the user base to score relevance for unseen items. Both approaches have merit, and a robust recommendation system may well incorporate multiple methods to best support users' information needs.

Both item-item filtering and user-item filtering approaches are achieved either through a memory-based approach or through parametric modeling. The memory-based approach is a bit more straightforward in that it relies solely on recorded data and utilizes a similarity function with normalization. The three most common measures are cosine similarity, Pearson correlation coefficients and K-nearest neighbors. The alternative to memory-based approaches is parametric modeling, which predicts the relevance of unseen items. The latter tends to perform better at scale or when data is sparse. Parametric filtering models either rely on some form of matrix factorization or employ deep learning with neural nets.

First, let's consider two memory-based approaches for item-item filtering in greater detail. The cosine similarity and Pearson similarity coefficient follow the same two-step algorithm, with the main difference being how normalization is defined and applied. The input is a collection of vectors that capture each user's item-ratings and the output is a vector of item recommendation scores for each user. The first step of the algorithm draws data from the entire collection of vectors to compute the similarity of each item to every other item and stores these values in an N x N matrix (N being the union of rated items over all user vectors). The second step of the algorithm takes as input the N x N matrix from step one as well as the original item-ratings vectors for all users and computes a recommendation score for each unrated item in each user's vector.

A similar scheme is applied to compute user-item relevance. Specifically, the algorithm determines the cosine similarity of each user to every other user based on

mutually reviewed item-ratings. Then, unseen items for each user are assigned a relevance score based on a weighted score that is derived from other users' scores for that item combined with the similarity of those users to the target user. This approach captures the intuition that users most similar to the target user will be able to offer the most relevant recommendations to the target user.

K-nearest neighbors is the other common method for either item-item or user item approaches. As an unsupervised learning model, k-NN seeks to cluster similar users or similar items together geometrically, then computes recommendation scores for unseen items by combining the results of the clustering with the original item-ratings vectors.

Parametric models such as matrix factorization and deep learning rely on latent features that are shared by users and items. The basic idea behind matrix factorization is to use these latent features to formulate a model that most accurately predicts the items that a given user will like. The learning step of matrix factorization optimizes a predictive model for each user by minimizing the error between predicted ratings for user-rated item versus the actual ratings those same items. The optimized model is then applied to map out predicted relevance scores of unseen items. There are many variations of matrix factorization that are used in recommender systems today which allow developers to manage overfitting and tweak latent factors, etc. These variations and models are beyond the scope of this review, but they do represent the state of the art in recommender systems today and are a very active research area. Similarly, deep

learning models have also been developed for recommender systems and are yet another promising line of research in the recommender systems space.

When it comes to selecting a filtering approach, there are no hard and fast rules, but the following recommendations offer some generic guidance for selecting an appropriate choice. First, it is best to not use a model unless one is intimately familiar with the mechanics of a given algorithm, the underlying mathematics and the governing theoretical justification for a particular approach. Modern libraries make it relatively easy to implement cutting-edge models, but that doesn't mean it is necessarily the best choice for a given application. Second, the type of items being recommended should inform one's selection of a filtering approach. For example, text documents lend themselves well to content-based filtering, whereas retail items like socks and shoes probably are best filtered collaboratively. Third, the density of data and physical resources should also inform the developer's choice since memory-based approaches are not as scalable as parametric modeling approaches. Fourth and most importantly, the context should inform one's selection of recommender systems. For example, item-item filtering would be appropriate alongside a user query to best support browsing similar items, whereas user-item filtering might be the best way to recommend personalized news briefings in a daily news digest.

In conclusion, one's choice of model will largely determine a recommender system's performance, and the performance of a recommender system can drastically impact an application's performance. The success of leading internet firms such as

Amazon, Netflix, Spotify, Google and Facebook largely depends on the efficiency of their recommender systems, which play a major role in driving user-engagement and sales. Fortunately, sample data sets, tutorials and a wealth of material on this topic are widely available and make it relatively easy to explore different algorithms in similar use cases to one's target development project.

References:

Grover, P. (2017, December 28). Various Implementations of Collaborative Filtering. Retrieved November 15, 2020, from https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

Le, J. (2020, February 23). Recommendation System Series Part 4: The 7 Variants of MF For Collaborative Filtering. Retrieved November 15, 2020, from https://towardsdatascience.com/recsys-series-part-4-the-7-variants-of-matrix-factorization-for-collaborative-filtering-368754e4fab5

Li, S. (2017, September 20). How Did We Build Book Recommender Systems in An Hour Part 2 - k Nearest Neighbors and Matrix Factorization. Retrieved November 15, 2020, from https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c

Qutbuddin, M. (2020, March 07). Comprehensive Guide on Item Based Collaborative Filtering. Retrieved November 15, 2020, from https://towardsdatascience.com/comprehensive-guide-on-item-based-recommendation-systems-d67e40e2b75d

Seo, J. (2018, November 22). [Paper Summary] Matrix Factorization Techniques for Recommender Systems. Retrieved November 15, 2020, from https://towardsdatascience.com/paper-summary-matrix-factorization-techniques-for-recommender-systems-82d1a7ace74

Zhai, C., & Massung, S. (2016). Chapter 11. In *Text Data Management and Analysis* (pp. 221-237). San Rafael, California: Morgan & Claypool.