Discovering Treasure Island: Building Literature Recommendation System

Alex Liu

MPCS53112 - Advanced Data Analytics

Dec. 2, 2017

Contents

1. OVERVIEW

The total amount of information in this world is growing so quickly that the problem of information overload is getting worse and worse. As more and more information starts to be published and distributed on the internet, it has become increasingly difficult for people to find useful information quickly.

It's specifically tedious for scientists to search for related work as millions of scientific papers are published every year [1]. Thus, technologies that can help scientists sift through all papers quickly and find valuable work are needed.

One of the most widely used technology is collaborative filtering. Collaborative filtering works by matching users based on the similarities of users preferences [2]. Each user will have a neighborhood of other users with similar preferences. These other users will then be used to suggest items that might be interest of the user.

In this project, we will explore classic collaborative filtering algorithms, and use them to build a recommendation engine in the domain of scientific literature using Apache Mahout, which is a machine learning library on Hadoop. The input data for the engine will be a high energy physics theory citation network from Stanford's snap lab [3].

Specific to this course, before mid-term presentation, I plan to have a script that can send request to Google scholar and parse the return value. I will also get the algorithms working with toy examples. In the final presentation, I will demonstrate the entire system including the recommendation engine and the evaluation of their performance .

2. RELATED WORK

In this section I will briefly presents some of the related work in the field of recommendation system and collaborative filtering. I will also give a short introduction to the library I am using, Apache Mahout.

2.1. User-based and Item-based Collaborative Filtering

In the field of recommendation system, many algorithms have been proposed. These include item-based algorithms and user-based algorithms.

User-based recommendations are derived from how similar users to users are, which means to make recommendations for a user, users who shares similar tastes are taken into consideration, and based on the items they possess we recommend items to user. Similarly, item-based recommendations are derived

from how similar items are to items, which means based on the items a user has already rated, more similar items are recommended.

2.2. Collaborative Filtering Algorithms Design

As is described above, the goal of a collaborative filtering algorithm is to suggest new items for a user based vector similarity.

The first step of collaborative filtering algorithm is to obtain the users history profile, which can be represented as a ratings matrix with each entry the rate of a user given to an item. A ratings matrix consists of a table where each row represents a user, each column represents a specific item, and the number at the intersection of a row and a column represents the user's rating value. The absence of a rating score at this intersection indicates that user has not yet rated the item. Owing to the existence problem of sparse rating, we use the list to replace the matrix [4].

The second step is to calculate the similarity, depending on the algorithm we can either compare the similarities between items or users. There are many different ways to this, including cosine similarity, correlation based similarity and adjusted cosine similarity [2].

The last step is to provide predictions or recommendations. Techniques such as weighted sum or regression are usually used.

Researchers have experimented with collaborative filtering systems in a wide variety of domains, including jokes, movies and music. Collaborative filtering has succeeded in helping users in all of these domains. Other technologies have also been applied to recommendation systems, including Bayesian networks and clustering, which are also very successful. But these technologies are beyond the scope of this project. So they are not discussed in detail here.

2.3. Apache Mahout

Apache Mahout is an algorithm library for scalable machine learning on Hadoop. It implements popular machine learning techniques such as recommendation, classification and clustering. Most of the algorithms in Apache Mahout are written on top of Hadoop, so it works well in distributed environment. In this project we only use a single node machine rather than Hadoop cluster as the size of our data is limited.

In terms of building recommendation engine, Mahout has a non-distributed, non-Hadoop-based recommendation engine. A text document containing user preferences for items can be fed to the engine, and the output of the engine would be the estimated preferences of a particular user for other items.

3. RECOMMENDATION ENGINE DESIGN

In this section, I will discuss in detail how all the different components of a recommendation engine are designed.

3.1. Gather Data

In order to perform collaborative filtering on the domain of research papers, we need to collect data and process it so that it fits the input format of our recommendation engine. The dataset I am using is high energy physics theory papers as is mentioned above. It contains 27770 papers. The papers in the dataset are firstly given artificial ids and mapped into a citation web. Next, the citation web is mapped onto a collaborative filtering ratings matrix. There are several ways to create a collaborative filtering ratings matrix.

One approach is to make paper authors the 'users' and keep citations as 'items'. In this ratings matrix, each author would "vote" for the papers that he/she has cited. The problem with this approach is that it suffers from a generality problem. Many authors have written papers in several different fields over the course of their careers. Say a computer scientist can publish in the field Biology, Computer Science and Data Science. Then it might be difficult to find a set of authors who have worked on all these fields too. [5]

Thus, instead of using an author's list of citations, we chose another mapping which uses the citation lists from individual papers. Here, a paper would represent a 'user' in the matrix and a citation would represent an 'item'. Each paper would then vote for the citations found in its references list. This mapping also does not suffer from the loss of specificity that could occur with the author-citation mapping [5].

3.2. Measure Similarity

Given the ratings matrix discussed earlier, we then need to decide how we want to measure the similarity between two rows/columns in the matrix.

There are several similarity measures supported in Mahout, such as Euclidean distance similarity, Pearson correlation similarity, Tanimoto coefficient similarity and Log likelihood similarity. Different similarity measures have different pros and cons. For example, Pearson correlation similarity is good for measuring how much two vectors are correlated while Tanimoto coefficient similarity is good for measuring the similarity between two vectors that have binary values.

In this project, the values in the matrix are binary as a paper can either cite or note cite another paper. Therefore, I have decided to use Tanimoto coefficient to measure the similarity between two vectors.

3.3. Choose User Neighborhood

User neighborhood is applicable for user-based recommendations. As is mentioned before, user-based recommendations are made based on user to user similarity. Thus, a neighborhood of most similar users that share almost same tastes is form so that we get better recommendations.

Mahout provides two algorithms to select user neighborhood, namely Nearest N User Neighborhood and Threshold User Neighborhood. In Nearest N User Neighborhood, the number of most similar uses to be considered for generating recommendations are specified as N. On the contrary, in Threshold User Neighborhood, the neighborhood size is not specified, rather a similarity threshold is specified. Any item pair with similarity greater than the threshold is taken into consideration. In my implementation Threshold User Neighborhood is used with a low threshold to take into account as many similar items as possible.

3.4. Build Computing Engine

After collecting data and deciding how we measure the similarity between two vectors. the last thing to do is to build the engine. A computing object that couples together the data model, similarity algorithm (and neighborhood if needed) is finally created to generate recommendations. One such computing engine is created for each of the algorithm mentioned above.

With both engines running properly, we then need to test the engines with test set and evaluate the performance of them. The methods used to achieve this are detailed in section 4.

4. EVALUATION

Sean [5] proposes quite a few good ways to evaluate the performance of a recommendation system, by doing online and offline experiments. Inspired by his work, I will use the following approaches to evaluate the system.

4.1. Offline Experiment

The offline experiment tests whether our algorithms are useful for predicting specific relevant citations for a given paper.

In this experiment, first we divide our data into test set and train set at a 90% to 10% ratio. For each paper in the test dataset, we randomly removed one citation from that paper's references list. The remaining citations were used to generate a list of recommended citations. We perform a 10-fold cross validation for this experiment, where in each fold we randomly assigned data into either the test or training dataset.

In this experiment we examine two metrics: Average rank and coverage percentage. We define rank as the position of the removed citation in the recommendation list. Coverage percentage is the percentage that an algorithm is able to successfully recommend the missing citation in the recommendation list.

4.2. Online Experiment

We can also use Google as a benchmark, where the title of the paper is sent to the Google scholar search engine, and we restrict the returned results to those appear in the input dataset. Then we compare the results from the recommendation engine with the results from Google.

In this experiment we only examine one metric: Average rank. We define rank as the the position of the related paper returned by Google in the recommendation list.

5. WHAT DID AND DID NOT WORK

The offline experiment worked well. The online experiment did not. I have created a scrapping tool to send request to Google scholar and parse the returned result. However, I failed to foresee that Google would block your IP when you exceeded a certain amount of requests. It took a day for the IP to be unblocked, so I have very limited amount of data to analyze. Another issue with online experiment is that there are hundreds of thousands physical papers that might be related to each paper. We only have 27770 papers in our dataset. So it is unlikely that the returned result from Google is also in our dataset. Thus, the online experiment has not been successful as a result of lacking sufficient data.

6. RESULTS AND DISCUSSION

6.1. Results

As is mentioned in section 5, the online experiment is not very successful. Thus, we will only discuss offline experiment here.

Figure 1 shows for removed citations, the percentage of missing citations recommended in a recommendation list of size 1, 10, 20 and 40. Figure 2 shows the average rank of missing citations in the recommendation list of size 1, 10, 20 and 40 (the lower the better).

As can be seen from both figure 1 and 2, Item-Based algorithm performs slightly better User-Based algorithm. We also see that in both Item-Based and User-Based algorithms, missing citations are mostly ranked in top 10 (as is indicated

by figure 2), which means that both algorithms perform well in identifying missing citations.

6.2. Discussion

As can be seen from above, both algorithms are able to provide high quality recommendations with a reasonably good coverage percentage (mostly likely over 50%). However, the coverage percentage and average rank can be further improved.

First, the dataset has the only a small number of papers, which means lot of the citations and papers are not included in this dataset. Given larger citation network and more papers, it is possible that the recommendation algorithm are able to produce better result.

Second, some of the data in the dataset only has a few citations in the network. When one citation is removed, it is unlikely to recover the missing citation as not enough information is given to the recommendation engine. So removing these papers in the test set could also improve the coverage percentage.

Finally, a paper could not reference papers that are published later than itself. However, when the engine recommends similar papers, it does not take into consider of this fact. So removing papers that are published later than the target paper in the recommendation list could further improve the coverage percentage.

Although there are several issues with the offline experiment design, both algorithms are still able to achieve around 60% coverage percentage with recommendation list of size 40. It is reasonable to conclude that these recommendation engines are efficient in recommending literature papers.

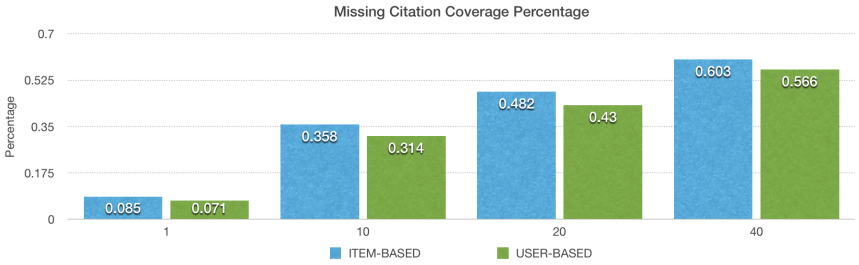

Figure 1: Coverage Percentage

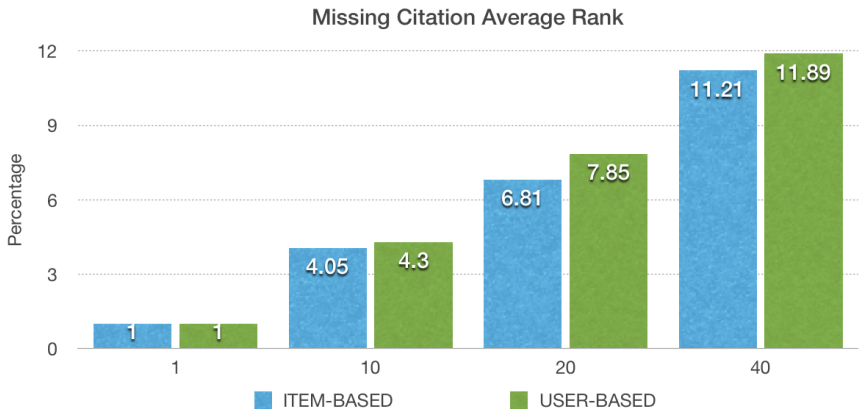

Figure 2: Average Rank

References

[1] E. Garfield, "Citation indexes for science. a new dimension in documentation through association of ideas," *International journal of epidemiology*, vol. 35, no. 5, pp. 1123–1127, 2006.

[2] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, ACM, 2001.

[3] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data, June 2014.

[4] Z.-D. Zhao and M.-S. Shang, "User-based collaborative-filtering recommendation algorithms on hadoop," in *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*, pp. 478–481, IEEE, 2010.

[5] S. M. McNee, I. Albert, D. Cosley, P. Gopalkrishnan, S. K. Lam, A. M. Rashid, J. A. Konstan, and J. Riedl, "On the recommending of citations for research papers," in *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 116–125, ACM, 2002.