Item-based Incremental Top-K Recommendation System

Aritra Kumar Lahiri
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
Tempe, US
aklahiri@asu.edu

***Abstract*—**The rise of e - commerce and the rapid growth of the World Wide Web has led to the growth of recommendation system. In this project we have proposed and developed an incremental approach to Collaborative filtering Algorithm for predicting binary ratings of an item by a user. The algorithm works by predicting the rating of top k restaurants with its place and type of cuisines. This algorithm was implemented as a web service for restaurant recommender.

***Keywords* —** Collaborative filtering algorithm, Item based recommendation, Jaccard Index, Cosine similarity index, user based recommendation, binary rating based recommendation.

I. INTRODUCTION

Item-based incremental Top-K recommendation system is the approach that we initiated to implement and design a restaurant recommendation system. Recommender systems has emerged very popular in recent years, and has been in applied to various applications and already existing system. The most popular examples of such system are promoted by Amazon and various other website, movies by Netflix, music by Spotify and Youtube, news, books, search queries, social tags and products in general. However, there are also recommender systems for experts, jokes, restaurants, financial services, and Twitter followers. A system could produce a list of recommendations through combination of content or content-based filtering or an one of them. Here the Collaborative filtering approach are based on filtering a large amount of information based on user's behaviours, preferences and predicting back to the user based on the recommended items that people with similar preferences liked in the past. On the other hand Content-based recommendation approach is based on the filtering method of recommending items to the user similar to the ones the user preferred in the past These approaches are often combined in Hybrid Recommender Systems.

II. PROBLEM DESCRIPTION AND SOLUTION

The problem that we have worked upon is developing an application that can efficiently recommend a set of top - k restaurant. We generated a similarity matrix on the basis of user rating and cuisine. We have collected the dataset from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Restaurant+%26+consumer+data) and used this large dataset so that we can develop the application more efficiently and can recommend according to the following parameter -

1. The reviews and rating of restaurant from past users' predict user recommendation for restaurant.
2. The type of cuisine a user wants.

A. *Algorithm implementation*

The algorithm uses an incremental collaborative filtering technique. In the building phase of the model, for each item j, the k most similar items*{ j1, j2, ..., jk }*are computed, and their corresponding similarities *{sj1 ,sj2 ,...,sjk}* are recorded. Now, for each user that has viewed a set (i.e., basket) U of items, this information is used to compute the top-K recommended items. This proposed approach will analyze the user-item matrix and determines the relations among different items. These results help in computing the list of top-K recommendations. This approach leads the users to view the items similar to those they have already viewed.Thus it will produce faster recommendation engines since there is no need of computation of the neighbourhood of similar users when suggesting a recommendation.

We implemented the algorithm to maintain a "co-concurrency matrix". This is an item x item matrix which is incrementally updated. The matrix stores the count of appearances of the combination of two items in an interaction set. We processed these co-concurrence counts with both jaccard and cosine similarity measures to fetch another item x item similarity matrix. This matrix is used to recommend the K most similar items for every item.

B. *Algorithm Steps*

I. Form the interaction sets by grouping the pairs of input user->item by user-id.
II. Increment the corresponding element in co-concurrency matrix for all the item<->item combinations in the interaction set.
III. Calculate the item<->item similarity for all the combinations of item<->item in the co-concurrency matrix.
IV. Fetch the K most similar items for every item in the corresponding output set.

C. *Incremental Collaborative Filtering Algorithm Steps:*

In this approach, we implemented a user-item similarity matrix that is updated using an incremental collaborative algorithm. The incremental item based algorithm that we proposed is for binary ratings. In case of non incremental approach the similarity matrix is constructed from time to time from the scratch. In the incremental approach, the similarity matrix is updated after each new session. Each incremental updates is necessary for affecting only the particular row of the similarity matrix under process.
The algorithm takes in a database D of pairs < u; i> (user u saw the item i). It assumes of computing the similarity between pairs of items or users (similarity matrix S). It takes parameters N_{eib} (number of neighbours to be found for a given user/item) and N (number of recommendations to be made). All the algorithms maintain a set of known Users and Items. The active user is denoted by U_a.

Steps:
1. Update Int.u and S
2. Determine the activation weight of each item never seen by Ua and recommend the N items with highest activation weight.
Let I be the set of items in the active session. The updating of Int.u and S is done as follows:
1. Add Ua to Users and the session to D
2. If Ua is a new user, a row and a column are added to Int.u and S .
3. The row and column of Int.u corresponding to Ua are updated using the new D.
4. Add new items in I to Items
5. Update the row(column) of S corresponding to Ua using Int.u.
The activation weight of an item for the user-based algorithm is W(i) =

$$\sum\nolimits_{\text{users near } u_a \text{ who saw } i} S[u_a,.] / \sum\nolimits_{\text{all the users near } u_a} S[u_a],.$$

D. *Jaccard and Cosine Similarity Index*

The Jaccard coefficient measures similarity as the intersection divided by the union of the objects. For text document, the Jaccard coefficient compares the sum weight of shared terms to the sum weight of terms that are present in either of the two documents but are not the shared terms. Mathematically it can be represented as -

$$\mathrm{SIM}_j(\vec{t_a}, \vec{t_b}) = = \frac{\vec{t_a} \cdot \vec{t_b}}{|\vec{t_a}| + |\vec{t_b}| - \vec{t_a} \cdot \vec{t_b}}$$

The Jaccard coefficient is a similarity measure and ranges between 0 and 1.

The Cosine similarity measure can be defined as the computation of the similarity between two items is to treat each item as a vector and use the cosine measure between these vectors as a measure of similarity.
Unlike Jaccard similarity measure, cosine similarity measure can also be applied to non-binary ratings. If R is the n × m user-item matrix, then the similarity between two items v and u is defined as the cosine of the n dimensional vectors corresponding to the vth and uth column of matrix R. The cosine between these vectors
is given by the below equation (where '·' denotes the vector dot-product operation).

$$sim(v,u) = cos(\vec{v}, \vec{u}) = \frac{\vec{v} \cdot \vec{u}}{||\vec{v}||_2 ||\vec{u}||_2},$$

If the ratings are binary, they can be represented as

$$sim(i,j) = cos(\vec{i}, \vec{j}) = \frac{\#(I \cap J)}{\sqrt{\#I} \times \sqrt{\#J}}$$

E. *Implementation details*

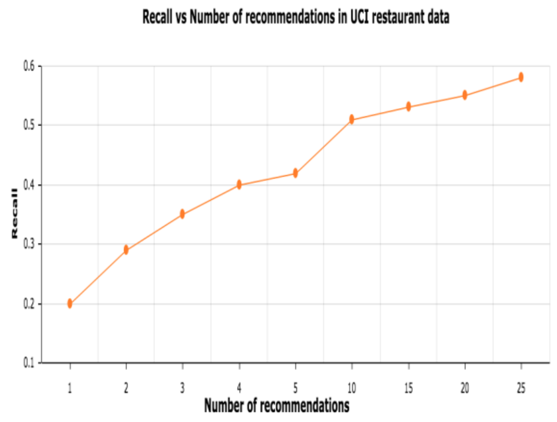
The proposed algorithm was implemented into code to test over the data set specified earlier.
As the final result for the restaurant application is expected to be a web application, we implemented the algorithm and necessary data cleaning and data loading functionalities in the form of a web service application. The web application for the algorithm implementation and other related jobs was done with Ruby On Rails(ROR) web application framework. The algorithm was built into the ruby on rails as a gem file. The basic implementation of the algorithm is in Ruby. To support the scalability and efficiency over a large data, we used Redis key value pair NoSQL database. Our idea was to develop each item-item pair that creates a key value in redis. The value of item-item increment count is stored as the corresponding value for the item-item key. In this way we generate a sparse matrix. This saves a large amount of memory and processing time, considering the fact that the number of seen item-item pairs is much less compared to the number of unseen item-item pairs. This implementation brings down the complexity from $O(n^3)$ complexity to O(n). The same technique is used to store the similarity matrix, which saves spaces and increases scalability during incremental updation of items or recommendation generation.
Ruby On Rails has an in built active record mechanism that helps in easy retrieval and updation of the data from this database setup. The loaded data is kept same such that the data may not be loaded each time when a recommendation request is processed. During this process we can either process the whole data together to create a similarity matrix or each row by row incrementally. Once the similarity matrix has been created, we can

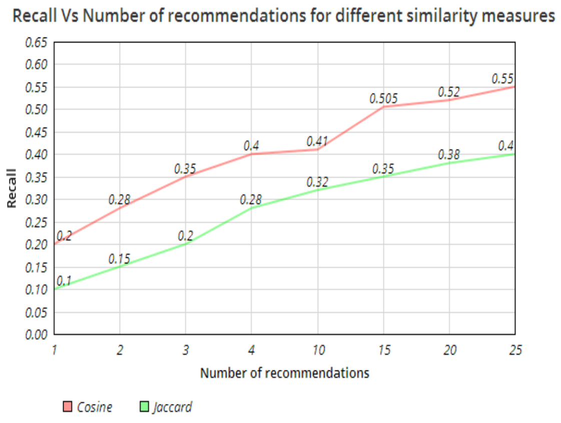
send an item to get recommendations for. The algorithm selects the top K similarity scores from the row corresponds to that item. This list of top K picks is the result of the algorithm.

F. *Experimental results and analysis*

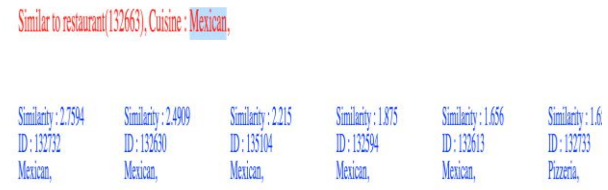
From the top K items returned by algorithm, we check for hits. To get compare the accuracy, we use the recall parameter. The recall is calculated a s follows

Recall = Number of hits / total number of users

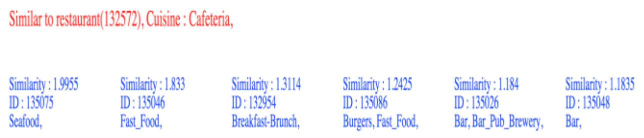




The result window as displayed on the part of the web application is as follows in the given diagram -


Similar to restaurant(132663), Cuisine : Mexican,

Similarity : 2.7594	Similarity : 2.4909	Similarity : 2.215	Similarity : 1.875	Similarity : 1.656	Similarity : 1.6
ID : 132732	ID : 132630	ID : 135104	ID : 132594	ID : 132613	ID : 132733
Mexican,	Mexican,	Mexican,	Mexican,	Mexican,	Pizzeria,


Similar to restaurant(132572), Cuisine : Cafeteria,

Similarity : 1.9955	Similarity : 1.833	Similarity : 1.3114	Similarity : 1.2425	Similarity : 1.184	Similarity : 1.1835
ID : 135075	ID : 135046	ID : 132954	ID : 135086	ID : 135026	ID : 135048
Seafood,	Fast_Food,	Breakfast-Brunch,	Burgers, Fast_Food,	Bar, Bar_Pub_Brewery,	Bar,

III. MY CONTRIBUTIONS

In this project I have at first cleaned the data set that is being downloaded from the UCI repository and then developed the incremental algorithm that is an item based collaborative filtering algorithm in Ruby. After that my duty was to interface the algorithm with the Redis database. I also made a comparative analysis of both the user based and item based approaches and calculated the recommendation time and rate of both the approaches using the restaurant - consumer dataset.
Finally after the entire completion of the application I have debugged and tested application before it is submitted.

IV. NEW SKILLS, TECHNIQUES OR KNOWLEDGE ACQUIRED

While doing this project I learnt how to code using Ruby On Rails web application framework which I did not have experience before. Also I learnt how to simulate and interface the gem file that is obtained with the database by making the algorithm compatible. Also I gained the knowledge of Redis, a NoSQL database that works on the basis of a key value pair. So these are the technologies that I learnt while developing this project.

V. CONCLUSION

The proposed algorithm, on the experiment with restaurant data proved that, it could give descent item based recommendations based past user visits to the item or restaurants. The algorithm is configurable with any similarity measure. But our results show that the cosine similarity function performs better with the algorithm. The algorithm is disk efficient and scalable as it uses a sparse matrix implementation and the algorithm can be updated incrementally without recalculating the entire model.

VI. TEAM MEMBERS

1. Aritra Kumar Lahiri
2. Ashish Mahajan
3. Arun Scaria
4. Pavan Kumar Akilla.

VII. REFERENCES

[1] Marko Balabanovic and Yoav Shoham. FAB: Content-based collaborative recommendation. *Communications of the ACM*, 40(3), March 1997. ___

[2] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: Using social and content-based infor- mation in recommendation. In *Proceedings of the 1998 Workshop on Recommender Systems*, pages 11–15. AAAI Press, 1998. ___

[3] DougBeefermanandAdamBerger.Agglomerativeclusteringofasearchenginequerylog.In*ProceedingsofAC MSIGKDD International Conference*, pages 407–415, 2000. ___

[4] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proceedings of ICML*, pages 46–53, 1998. ___

[5] P. Chan. A non-invasive learning approach to building web user profiles. In *Proceedings of ACM SIGKDD International ___Conference*, 1999. ___

[6] N. Good, J. Scafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with ___personal agents for better recommendations. In *Proceedings of AAAI*, pages 439–446. AAAI Press, 1999. ___

[7] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In ___*Proceedings of CHI*, 1995. ___

[8] BrendanKitts,DavidFreed,andMartinVrieze.Cross-sell:Afastpromotion-tunablecustomeritemrecommendationmethod ___based on conditional independent probabilities. In *Proceedings of ACM SIGKDD International Conference*, pages 437–446, 2000. ___

[9] Miranda C. and Alipio J. (2008). Incremental collaborative filtering for binary ratings (LIAAD - INESC Porto, University of Porto)

[10] George Karypis (2000) Evaluation of Item-Based Top-N Recommendation Algorithms (University of Minnesota, Department of Computer Science / Army HPC Research Center)

[11] Shiwei Z., Junjie W. Hui X. and Guoping X. (2011) Scaling up top-K cosine similarity search (Data & Knowledge Engineering 70)

[12] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993, pp. 207–216, Washington, DC.

[13] C. Alexander, Market Models: A Guide to Financial Data Analysis, John Wiley & Sons, 2001.

[14] A. Arasu, V. Ganti, R. Kaushik, Efficient exact set-similarity joins, Proceedings of the 32th International Conference on Very Large Data Bases, 2006.

[15] A. Awekar, N.F. Samatova, P. Breimyer, Incremental all pairs similarity search for varying similarity thresholds, Proceedings of the 3rd Workshop on Social Network Mining and Analysis, 2009.

[16] J. Blanchard, F. Guillet, R. Gras, H. Briand, Using information-theoretic measures to assess association rule interestingness, Proceedings of the Fifth IEEE International Conference on Data Mining, 2005, pp. 66–73, Houston, TX.

[17] S. Brin, R. Motwani, C. Silverstein, Beyond market basket: generalizing association rules to correlations, Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, 1997, pp. 265–276, Tucson, AZ.

[18] S. Chaudhuri, V. Ganti, R. Kaushik, Efficient set joins on similarity predicates, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004.

[19] Asmuth Paul, The recommender gem Skelton code. https://github.com/paulasmuth/recommendify.

[20] S. Chaudhuri, V. Ganti, R. Kaushik, A primitive operator for similarity joins in data cleaning, Proceedings of the 22th International Conference on Data Engineering 2006.