Anytime Algorithms for Top-N Recommenders

David Ben-Shimon
Ben-Gurion University of the Negev, Beer-Sheva, Israel
dudibs@post.bgu.ac.il

ABSTRACT

Many small and mid-sized e-businesses use the services of recommender system (RS) provider companies to outsource the construction and maintenance of their RS. The fees that RS providers charge their clients must cover the computation costs for constructing and updating the recommendation model. By using anytime algorithms, a RS provider can control the computation costs and still offer a system capable of delivering reasonable recommendations. Thus, a RS provider should be able to stop the construction of a recommendation model once the cost for computing it reaches the amount the customer has agreed to pay.

In this research we suggest anytime algorithms as a possible solution to a problem that RS providers face. We demonstrate how certain existing recommendation algorithms can be adjusted to the anytime framework. We focus on the case of item-item algorithms, showing how the anytime behavior can be improved using different ordering methods of computations. We conduct a comparative study demonstrating the benefits of the proposed methods for top-N item-item recommenders.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Anytime Algorithms, Collaborative Filtering, Item-Based, Locality Sensitive Hashing

1. INTRODUCTION

Many e-commerce businesses record users' activities in order to enable personalization and to recommend items to view or purchase. The decision of an e-business to apply personalization and integrate recommendations into their existing systems can be challenging and even risky since the expected return on investment (ROI) is difficult to predict. As for small and medium e-businesses, the required knowledge, expertise, and resources needed for implementing a personalized recommender system may be beyond their reach.

To minimize the investment and risk, many small or medium e-businesses, prefer to purchase the recommender system as a service. Such services allow an e-business to request recommendations online for the active customer through a remote server, which the e-business neither owns nor operates. In case of an unsuccessful deployment in which the system does not meet the goals of the e-business owner, the service can be easily terminated without any major financial losses to the business owner. In other words, recommender system services can reduce the required initial investment and present very little risk to the e-business.

The providers of such services, often called recommender system (RS) providers, offer a set of pre-made solutions that can be adjusted to specific website needs at a low cost. The solutions offered by the RS providers are generally hosted on remote servers, where the website data is also maintained, prediction models are constructed, and recommendation queries are handled. While the RS provider may own and operate these servicers, growing cloud computing framework services [1], such as Amazon and Microsoft providing, are offering alternative storage and computation possibilities. In such a case, building models for customers may incurs substantial costs to the RS provider. We are motivated by that specific problem that RS providers face, namely, the balancing of model computation costs and quality.

One possible approach for controlling the balance between computation cost and recommendation quality is through the use of anytime algorithms (AA) [5]. An anytime algorithm provides a trade-off between computation time and the quality of the solution it provides. The more the computation time, the better the predictive performance that is achieved. Such algorithms can be stopped at any point in the course of the computation and still yield solutions. Given sufficient time, the solution becomes optimal.

This research makes two contributions to the field of RS. First, it is presenting the RS provider business model and the problems that these providers face. Second, it shows how anytime algorithms can address some of these problems by presenting two techniques for adapting nearest neighborhood CF algorithms to act as anytime algorithms. Experiments show that these techniques can be used to provide a better time-quality trade-off

2. BACKGROUND

2.1 Recommender Systems Providers

In the past, recommender systems were employed mainly by large scale e-businesses, such as Amazon. The main reasons for this phenomenon are that implementing recommendation algorithms can involve substantial investments; require certain expertise; programming effort; high computational power; and suitable infrastructure. In many cases, particularly for mid-sized and small e-businesses, these substantial investments cannot be justified by the added value. That is because the cost of realizing such a solution will be much larger than the income which will come out of it. On the other hand, the integration of a ready-made recommender engine into a website requires a set of skills that are commonly found in companies constructing web pages, and less initial investment, except for the needed integration, is required.

Consequently, a feasible RS alternative is to outsource the RS to a company that specializes in this area and consume it as a service. Companies providing recommendation systems as a service are often called RS providers. By exposing application program interface (API), the RS providers are storing recorded users' activities, computing models, and delivering list of

RecSys'13, October 12–16, 2013, Hong Kong, China.
Copyright © 2013 ACM 978-1-4503-2409-0/13/10…$15.00.
http://dx.doi.org/10.1145/2507157.2508074

recommendations on demand. Since recommendation services should be inexpensive, a RS provider must be able to support many clients to be profitable. It cannot afford to offer its clients a tailor-made solution. Providers typically select the most suitable solution for a customer from a set of pre-made generic methods.

While the quality of the recommendations may vary, an obvious restriction is that the on-line recommendations have to be provided very rapidly. It is unacceptable for a system to wait more than a fraction of a second for the computation of a recommendation. To ensure that, the RS providers is using Model-Based [2] approach and typically sign a service level agreement (SLA) that defines the required response time.

The RS provider may build its own infrastructure for computing recommendation models and answering recommendation requests rapidly. The cost of this infrastructure and its maintenance, however, may become too high for a RS provider given the low fees it charges its clients. Another alternative is to use cloud computing services [1]. Amazon, as well as other companies, maintain large server farms, and allow users to store data and to run computation processes on these servers. The users pay a fee proportional to the amount of storage, time and computation power they use. Given such cloud computing services, it is easy for the RS provider to estimate the cost of the required computations, such as models building, and charge the e-business accordingly.

2.2 Anytime Algorithms

To maintain profitability when using existing recommender methods, RS providers can offer either simple, low-cost models which are optimal but can be trained rapidly e.g., a Naïve Bayes Classifier, or expensive more popular models (such as SVD) that require heavy computational resources. However, those existing methods do not allow direct control over the accuracy-cost trade-off. Given the RS provider scenario, that provides RS as a service, it is difficult or even impossible to determine in advance the required time and computational resources needed to generate models offering a suitable level of cost and accuracy for handling unseen data of an e-business.

A more compelling approach would be to make use of an algorithm to be constructed in such a way so that it builds the best model possible under the constraint of limited computation time. Standard algorithms for model building do not offer this option. Many of the algorithms for model building in RS, when interrupted before termination, will not have an available model to return. To enable interruption, an algorithm must be designed to allow intermediate result snapshots. Such algorithms are typically called anytime algorithms [5].The accuracy that anytime algorithms provide always improve or stay the same over the time, but never decrease. In this research we propose that applying anytime algorithms to the recommendation problem is adequate for RS provider business model.

Anytime algorithms have been used in various areas such as data mining [4] decision support systems [8]; and for various intelligent systems [11]. Little research exists on the application of anytime algorithms to recommender systems. One particularly interesting line of research has been the application of efficient anytime methods to sequential association rules models [3]. In such a models, the anytime approach was used to limit the latency of computing recommendations online and not to the model building process. While this approach is interesting, especially in the case of memory-based algorithms, it is different from the scenario that we are interested in, where the approach is model-based and the recommendations must be computed in constant time.

3. CASE STUDY - ANYTIME TOP-N RECOMMENDERS OVER IMPLICIT FEEDBACK DATASETS

In the neighborhood-based item-item model, the goal is to find the nearest neighbors of each item. A neighborhood is defined as being based on similarity score between items. These similarities are cached in a model that is, in turn, used online to provide recommendations. In a CF approach for implicit datasets, two items may be considered to be similar, if users tend to consume them together. A simple example of a similarity function for implicit datasets is the Jaccard coefficient [9]:

$$J(i, j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|} \qquad (1)$$

where U_i and U_j are the sets of users that consumed items i, and j respectively.

3.1 Methods for Ordering Item Pairs in Nearest Neighbor Search Methods

In an anytime setting, we can compute item pair statistics until a predefined time limit has been reached. When computing recommendations online, the recommendation algorithm can only use the statistics it has computed within the given time. As we show below, in such a setting, the order by which item pairs are investigated can greatly influence the quality of the recommendations.

We suggest two methods for deciding on the order of pairwise similarity computation. An essential property of such methods is that they must be computed very rapidly. Otherwise, we might spend more time on the initialization of the method than on constructing the model. We use a data structure that allows us to rapidly access all the users who have used an item and all the items that a given user has used.

3.1.1 Locality Sensitive Hashing Tree

Locality sensitive hashing (LSH) [6] is a technique for rapidly computing the similarity of items. We apply the LSH technique to provide us with an anytime approach for the item pair computations. To generate our signatures matrix (reduced representation of the user-item input matrix) over the data, we adapted the idea of LSH Google news personalization [5] where the hash value for an item in a given dimension in the signature of an item, is the first user ID that consumed this item in a random permutation of the user ID list. We generated P permutations over the user IDs and applied this hash function on each permutation to end up with an item signatures matrix of the size PxN. This P defines the number of dimensions in the signatures matrix.

We now wish to split the item signatures set into two disjoint subsets such that the items in each subset will most likely be more similar to one another than to items in the other subset. We split the items in the signatures matrix based on the medians of the various dimensions. Given an item signature matrix, we first compute a median for each dimension independently. Then, for each item in the set, we compute the portion of its dimension entries whose value is higher than the median of that dimension. If this portion is lower than 0.5, the item is placed in the left child subset; otherwise it is placed in the right child subset. We use a top-down approach, starting with all item signatures and splitting them into equally sized subsets. We continue recursively until we reach a predefined minimum number of items in a subset. This splitting approach creates a balanced binary tree, where every two disjoint subsets created through the splitting process are always the full set of their related parent. Figure 1 illustrates the tree construction and the number of items in each node and leaf.

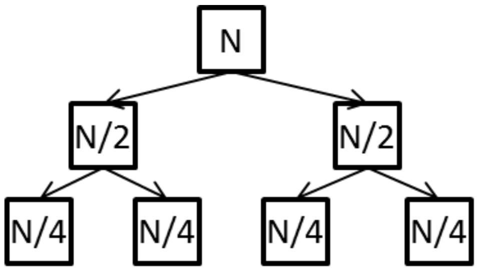

Figure 1: The N items divided evenly on each level using the medians.

Beginning with the leaves of the tree, we first compute for each leaf the pairwise Jaccard similarities of items in the leaf. After all leaves are computed, we move up a level, and compute the pairwise similarities of items that belong to sibling leaves. Moving up levels in the tree, the process is continued, until all pairwise similarities have been computed.

Assuming that the signatures matrix maintains the Jaccard similarities and that the median splits the items into two homogenous groups during the construction of the tree, we expect that items with high similarity will be computed first with high probability, and that a good anytime performance will be achieved.

3.1.2 The Most Popular First

The second method is based on the assumption that popular items provide information to the model, and thus should be computed first. As popular items appear in many users' profiles, they can be utilized for rapidly computing personalized recommendations for most users.

This approach can be implemented efficiently by sorting items according to decreasing consumption set cardinality (i.e., popularity). We then compute for each item its similarity to the most popular item, its similarity to the second most popular item, etc. Following is a pseudo code of the popular first approach.

```
L ← items sorted by decreasing popularity
For i = 1 to |L|
  For each user U who consumed item i
    For each item j consumed by user U
          compute $J(L_i, L_j)$
```

3.2 Initialization Complexity

Let *I* be the number of items, *A* be the average number of users who have chosen each item, and *P* be the reduced dimension in the LSH reduction.

The initialization complexity for the '*LSH Tree'* requires *O(P·I·A)* for computing the signatures matrix. Then, constructing the tree requires *O(I·PlogI)* operations, for finding the medians in each level in the tree, and there are at most *O(logI)* levels. Hence, assuming that *A* and *P* are much smaller than *I*, the computation time is dominated by *O(I·logI).*

The initialization complexity of the '*Most Popular*' method is *O(I·logI)* for sorting the items by the number of users who consumed them.

In the results below this initialization time is included, but for both methods it required no more than a few hundreds of milliseconds and is therefore negligible.

3.3 Comparative Experiments

To compare and analyze the anytime behavior of the '*LSH Tree*' and the *'Most Popular'* method, we carried out a series of experiments. As a baseline we present the Amazon approach [10] that computes item pairs in an arbitrary order.

All three methods for ordering the items pairs leverage the symmetry in the Jaccard coefficient. That is, if $J(i,j)$ was computed, then $J(j,i)$ is also known, and not computed again. The algorithms compute the similarities only for pairs of items which were consumed together by at least one user yielding a score greater than zero. The baseline method, the Amazon approach, is somewhat anytime in nature due to its iterative computation.

We conducted experiments on three datasets: buying events from a music web shop: buying events from a software games web shop; and Movie-Lens, described in Table 1.For Movie-Lens, we attempted to recommend movies that the user might rate, given other movies she has rated. To model this scenario, we ignored the actual ratings and used a dataset where users either rated or didn't rate a movie. The music dataset contains real data, capturing buying events in a music portal over several consecutive months during 2010. The games dataset is also a real dataset and derives from an online shop selling software games over the past few years. In the music and the games datasets we computed a list of recommended items given other items that the user has bought.

Table 1. The benchmark datasets used in the experiments

Name	#events	# items	# users	Sparsity
Movie-Lens	100,000	1682	943	0.9369
Music	458,131	116,583	9901	0.9996
Games	320,641	3304	72,347	0.9986

Each users' consumption set is divided randomly into a train and test set. We executed the three algorithms on all three datasets, stopping them at each predefined interval time to measure the performance of the currently computed model. The interval time was 5 seconds for the Movie-Lens; 30 seconds for the games dataset; and 60 seconds for the music dataset. Since the task is to provide top-N recommendations, we found *precision@N* [7] to be the most appropriate metric for evaluating model performances. Below, we provide results for *N=5*, i.e., we directed the algorithm to supply us with 5 recommendations. We also experimented with different values of *N* (10, 15, and 25), but found no sensitivity to *N*. We ran all algorithms 5 times, and report the average results. The standard error of the 5 runs was well below 10^{-5} and is not shown in the graphs. All algorithms use the same train-test split, allowing us to measure paired statistical significance over the different users.

3.4 Results

Figures 2, 3, and 4 present precision as a function of computation time for the Movie-lens, Music and Games datasets respectively.
Each point in a curve represents the average precision of a model at a certain time during the learning phase, and is averaged over five different train-test splits.

The *'Most Popular'* technique shows superiority in the Movie lens and in the music datasets, while the '*LSH Tree'* shows superiority also in the Movie-Lens and in the games datasets. The

Amazon baseline method demonstrates moderate anytime performance.

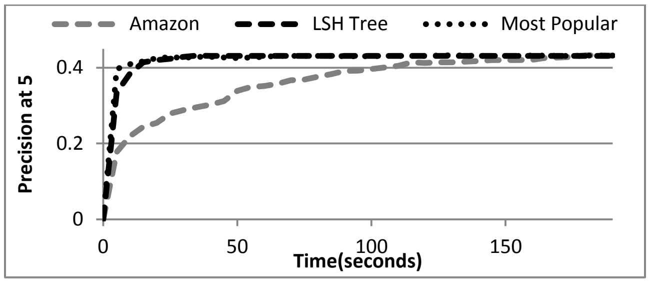

Figure 2: Anytime results for the Movie-Lens dataset

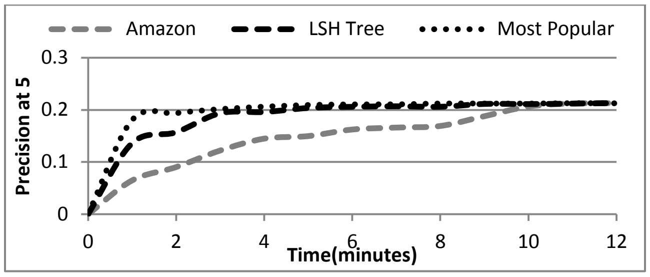

Figure 3: Anytime results for the music dataset

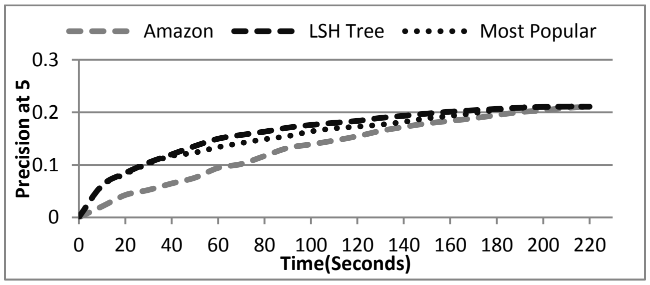

Figure 4: Anytime results for the software games dataset

Table 2 presents the area under the curve (AUC) for all three approaches on the three datasets. Each value in the table is the average over the five different train-test splits. The standard deviation of each value is also presented. AUC provides a comparison of the entire running process.

Table 2. Area under the curve (AUC) for both domains

Method	Movie-Lens	Music	Games
Amazon	68.18±1.71	1.78±0.02	29.4±1.48
Tree LSH	<u>79.18±1.44</u>	2.24±0.02	<u>36.1±0.5</u>
Most Popular	<u>80.34±1.47</u>	<u>2.36±0.03</u>	34.5 ±0.7

These results demonstrate that the anytime behavior of standard recommendation approaches can be improved. Simple technique, such as the *'Most Popular'* orders the item pairs so that the models present better precision faster than standard methods. While the '*LSH Tree*' method may not seem the best in our experiments, we believe that compared to the popularity technique, it is more general and less tailored to the Jaccard-based similarity algorithm that we employ.

4. CONCLUSIONS AND FUTURE RESEARCH

We described the problem of RS providers who need to balance the computation time of recommendation models with the fees they collect from customers. We have suggested that anytime algorithms, which support interrupting the computation after a given time, may provide an adequate solution to this problem. RS providers can stop the algorithm once the cost for computation time reaches the payment the RS provider agreed upon with the customer. We claim that typical model-based CF algorithms display poor anytime behavior. Using public and real life buying events datasets, we demonstrated this on the well-known Amazon approach. Consequently it is necessary to either construct new algorithms or to adapt existing algorithms to provide better anytime behavior. We view this research as the starting point for further exploration of anytime algorithms for constructing recommendations models. In the future we will investigate anytime techniques for the gradient descent SVD, improving on both the latent factors learning phase as well as the recommendation computation phase.

5. ACKNOWLEDGMENTS

I would like to thank my advisors, Prof. Lior Rokach and Prof. Bracha Shapira for their valuable instruction, ideas and feedbacks on this work, and to Dr. Guy Shani for the insightful discussions.

6. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia (2009). Above the Clouds: A Berkeley View of Cloud computing. Technical Report No. UCB/EECS-2009.

[2] J.S Breese. D. Heckerman, C. Kadie (1998). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98). G. F. Cooper, and S. Moral, Eds. Morgan-Kaufmann, San Francisco, Calif., 43–52.

[3] A. Brun, A. Boyer (2009). Towards Privacy Compliant and Anytime Recommender Systems. Proceedings of 10th International Conference on Electronic Commerce and Web Technologies. Pages 276-287.

[4] D. Dash, M. Druzdzel (1999). A hybrid anytime algorithm for the construction of causal models from sparse data, Proceedings of the fifteenth international conference on uncertainty in artificial intelligence.

[5] T. Dean (1987). Intractability and Time-Dependent Planning. In Proceedings of the 1986 Workshop on Reasoning about Actions and Plans. eds. M. P. Georgeff and A. L. Lansky. San Francisco, Calif.: Morgan Kaufmann.

[6] P. Gionis, P. Indyk, R. Motwani (1999). Similarity search in high dimensions via hashing. Proceedings of VLDB, pp. 518–529.

[7] J.L. Herlocker, J.A. Konstan, L.G Terveen, J.T. Riedl (2004). Evaluating Collaborative Filtering Recommender Systems. ACM Trans. Information Systems, vol. 22, no. 1, pp. 5-53, 2004.

[8] M.C. Horsch, D. Poole (1998). An anytime algorithm for decision making under uncertainty, in: Proceedings of the14th Annual Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, 1998, pp. 246–255.

[9] P. Jaccard (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin de la Société Vaudoise des Sciences Naturelles 37: 547–579.

[10] Linden, B. Smith, J. York (2003). Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, 7, 76-80.

[11] S. Zilberstein (1996). Using anytime algorithms in intelligent systems. AI Magazine, 17(3), 73-83.