AN EFFICIENT AND SCALABLE LOCATION-AWARE RECOMMENDER SYSTEM.

Gomathy M  
Lecturer, Departament of Computer Science, Shrimarhi Indra Gandhi College, Trichy  
Email: m.gomathy.phd@gmail.com

Received:09, Mar, 2014 Accepted: 15, May, 2014

Abstract
LARS, a location-aware recommender system that uses location-based ratings to produce recommendations is proposed. Traditional recommender systems do not consider spatial properties of users nor items; LARS, on the other hand, supports a taxonomy of three novel classes of location-based ratings, namely, spatial ratings for non-spatial items, non-spatial ratings for spatial items, and spatial ratings for spatial items. LARS exploits user rating locations through user partitioning, a technique that influences recommendations with ratings spatially close to querying users in a manner that maximizes system scalability while not sacrificing recommendation quality. LARS exploits item locations using travel penalty, a technique that favors recommendation candidates closer in travel distance to querying users in a way that avoids exhaustive access to all spatial items. LARS can apply these techniques separately, or together, depending on the type of location-based rating available. LARS is efficient, scalable, and capable of producing recommendations twice as accurate compared to existing recommendation approaches.

Keywords: LARS, Querying, Techniques and Scalable.

INTRODUCTION
Recommender systems make use of community opinions to help users identify useful items from a considerably large search space. The technique used by many of these systems is collaborative filtering (CF), which analyzes past community opinions to find correlations of similar users and items to suggest k personalized items (e.g., movies) to a querying user u. Community opinions are expressed through explicit ratings represented by the triple (user, rating, item) that represents a user providing a numeric rating for an item. Currently, myriad applications can produce location-based ratings that embed user and/or item locations. For example, location-based social networks allow users to “check-in” at spatial destinations (e.g., restaurants) and rate their visit, thus are capable of associating both user and item locations with ratings. Such ratings motivate an interesting new paradigm of location-aware recommendations, whereby the recommender system exploits the spatial aspect of ratings when producing recommendations.

LITERATURE REVIEW
Location-based services
Current location-based services employ two main methods to provide interesting destinations to users. (1) KNN techniques and variants (e.g., aggregate KNN) simply retrieve the k objects nearest to a user and are completely removed from any notion of user personalization. (2) Preference methods such as skylines and location-based top-k methods require users to express explicit preference constraints. Recent research has proposed the problem of hyper-local place ranking. Given a user location and query string, hyper-local ranking provides a list of top-k points of interest influenced by previously logged directional queries. Hyper-local ranking does not personalize answers to the querying user, i.e., two users issuing the same search term from the same location will receive exactly the same ranked answer set.

Traditional recommenders
A wide array of techniques are capable of producing recommendations using non-spatial ratings for non-spatial items represented as the triple (user, rating, item). The closest these approaches come to considering location is by incorporating contextual attributes into statistical recommendation models. Some existing commercial applications make cursory use of location when proposing interesting items to users. For instance, Netflix displays a “local favorites” list containing popular movies for a user’s given city. However, these movies are not personalized to each user (e.g., using recommendation techniques); rather, this list is built using aggregate rental data for a particular city.

Location-aware recommenders
The CityVoyager system mines a user’s personal GPS trajectory data to determine her preferred shopping sites, and provides recommendation based on where the system predicts the user is likely to go in the future. The spatial activity recommendation system mines GPS trajectory data with embedded user-provided tags in order to detect interesting activities located in a city. It uses this data to answer two query types: (a) given an activity type, return where in the city this activity is happening, and (b) given an explicit spatial region, provide the activities available in this region. Geo-measured friend-based collaborative filtering produces recommendations by using only ratings that are from a querying user’s social-network friends that live in the same city. This technique only addresses user location embedded in ratings.

Amazon.com Recommendations Item-to-Item Collaborative Filtering
Greg Linden, Brent Smith, and Jeremy York • Amazon.com Recommendation algorithms are best known for their use on e-commerce Web sites, where they use input about a customer’s interests to generate a list of recommended items. Many applications use only the items that customers purchase and explicitly rate to represent their interests, but they can also use other attributes, including items viewed, demographic data, subject interests, and favorite artists.

E-commerce recommendation algorithms often operate in a challenging environment. For example: A large retailer might have huge amounts of data, tens of millions of customers and millions of distinct catalog items. Many applications require the results set to be returned in realtime, in no more than half a second, while still producing high-quality recommendations. New customers typically have extremely limited information, based on only a few purchases or product ratings. Older customers can have a glut of information, based on thousands of purchases and ratings. Customer data is volatile. Each interaction provides valuable customer data, and the
algorithm must respond immediately to new information. There are three common approaches to solving the recommendation problem: traditional collaborative filtering, cluster models, and search-based methods. Here, we compare these methods with our algorithm, which we call item-to-item collaborative filtering.

Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions

Gediminas Adomavicius and Alexander Tuzhilin;
The paper presents an overview of the field of recommender systems and describes the current generation of recommendation methods that are usually classified into the following three main categories: content-based, collaborative, and hybrid recommendation approaches. The paper also describes various limitations of current recommendation methods and discusses possible extensions that can improve recommendation capabilities and make recommender systems applicable to an even broader range of applications. These extensions include, among others, improvement of understanding of users and items, incorporation of the contextual information into the recommendation process, support for multi-criteria ratings, and provision of more flexible and less intrusive types of recommendations.

SINA: Scalable Incremental Processing of Continuous Queries in Spatiotemporal Databases

Mohamed F. Mokbel, Xiaopeng Xiong, Walid G. Aref, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398.

This paper introduces the Scalable INcremental hash-based Algorithm (SINA, for short); a new algorithm for evaluating a set of concurrent continuous spatio-temporal queries. SINA is designed with two goals in mind: (1) Scalability in terms of the number of concurrent continuous spatiotemporal queries, and (2) Incremental evaluation of continuous spatio-temporal queries. SINA achieves scalability by employing a shared execution paradigm where the execution of continuous spatio-temporal queries is abstracted as a spatial join between a set of moving objects and a set of moving queries. Incremental evaluation is achieved by computing only the updates of the previously reported answer. We introduce two types of updates, namely positive and negative updates. Positive or negative updates indicate that a certain object should be added to or removed from the previously reported answer, respectively. SINA manages the computation of positive and negative updates via three phases: the hashing phase, the invalidation phase, and the joining phase. The hashing phase employs an in-memory hash-based join algorithm that results in a set of positive updates. The invalidation phase is triggered every T seconds or when the memory is fully occupied to produce a set of negative updates. Finally, the joining phase is triggered by the end of the invalidation phase to produce a set of both positive and negative updates that result from joining in-memory data with in-disk data. Experimental results show that SINA is scalable and is more efficient than other index-based spatio-temporal algorithms.

Evaluating Collaborative Filtering Recommender Systems

Jonathan I. Herlocker, Oregon State University and Joseph a. Konstan, loren g. Terveen, and John t. Riedl, University of Minnesota

Recommender systems have been evaluated in many, often incomparable, ways. In this article, we review the key decisions in evaluating collaborative filtering recommender systems: the user tasks being evaluated, the types of analysis and datasets being used, the ways in which prediction quality is measured, the evaluation of prediction attributes other than quality, and the user-based evaluation of the system as a whole. In addition to reviewing the evaluation strategies used by prior researchers, we present empirical results from the analysis of various accuracy metrics on one content domain where all the tested metrics collapsed roughly into three equivalence classes. Metrics within each equivalency class were strongly correlated, while metrics from different equivalency classes were uncorrelated.

Optimal aggregation algorithms for middleware

Ronald Fagin, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA; Amnon Lotem, Department of Computer Science, University of Maryland-College Park, College Park, MD 20742, USA; and Moni Naor, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel

Assume that each object in a database has m grades, or scores, one for each of m attributes. For example, an object can have a color grade, that tells how red it is, and a shape grade, that tells how round it is. For each attribute, there is a sorted list, which lists each object and its grade under that attribute, sorted by grade (highest grade first). Each object is assigned an overall grade, that is obtained by combining the attribute grades using a fixed monotone aggregation function, or combining rule, such as min or average. To determine the top k objects, that is, k objects with the highest overall grades, the naive algorithm must access every object in the database, to find its grade under each attribute. Fagin has given an algorithm ("Fagin's Algorithm", or FA) that is much more efficient. For some monotone aggregation functions, FA is optimal with high probability in the worst case. We analyze an elegant and remarkably simple algorithm ("the threshold algorithm", or TA) that is optimal in a much stronger sense than FA. We show that TA is essentially optimal, not just for some monotone aggregation functions, but for all of them, and not just in a high-probability worst-case sense, but over every database. Unlike FA, which requires large buffers (whose size may grow unboundedly as the database size grows), TA requires only a small, constant-size buffer. TA allows early stopping, which yields, in a precise sense, an approximate version of the top k answers. We distinguish two types of access: sorted access (where the middleware system obtains the grade of an object in some sorted list by proceeding through the list sequentially from the top), and random access (where the middleware system requests the grade of object in a list, and obtains it in one step).

METHODOLOGY

To design a location-aware recommender system (LARS), which support three types of query retrieval in a single framework. To design the framework in JAVA. LARS produces recommendations using location-based ratings within a single framework: Spatial ratings for non-spatial items, non-spatial ratings for spatial items, and spatial ratings for spatial items. To arrive top-k results for user specific query.

Proposed Solution

LARS produces recommendations using spatial ratings for non-spatial items, by employing a user partitioning technique that exploits preference locality. This technique uses an adaptive pyramid structure to partition ratings by their user location attribute into spatial regions of varying sizes at different hierarchies. The LARS pyramid dynamically adapts to find the right pyramid shape that balances scalability and recommendation locality.

Existing System

Existing recommendation techniques assume ratings are
represented by the (user, rating, item) triple, thus are ill-equipped to produce location-aware recommendations.

**Location-based services**

Current location-based services employ two main methods to provide interesting destinations to users. (1) KNN techniques and variants (e.g., aggregate KNN) simply retrieve the k objects nearest to a user and are completely removed from any notion of user personalization. (2) Preference methods such as skylines and location-based top-k methods require users to express explicit preference constraints.

**Traditional recommenders**

A wide array of techniques are capable of producing recommendations using non-spatial ratings for non-spatial items represented as the triple (user, rating, item). The closest these approaches come to considering location is by incorporating contextual attributes into statistical recommendation models. However, these are not personalized to each user; rather, this list is built using aggregate rental data for a particular city.

**Location-aware recommenders**

The CityVoyager system mines a user's personal GPS trajectory data to determine her preferred shopping sites, and provides recommendation based on where the system predicts the user is likely to go in the future. The spatial activity recommendation system mines GPS trajectory data with embedded user-provided tags in order to detect interesting activities located in a city. It uses this data to answer two query types: (a) given an activity type, return where in the city this activity is happening, and (b) given an explicit spatial region, provide the activities available in this region. Geo-measured friend-based collaborative filtering produces recommendations by using only ratings that are from a querying user's social-network friends that live in the same city. This technique only addresses user location embedded in ratings

**Disadvantages**

- Does not personalize answers to the querying user
- No traditional approach has studied explicit location-based ratings.

**PROPOSED SYSTEM**

- LARS, a novel location-aware recommender system built specifically to produce high-quality location-based recommendations in an efficient manner.
- LARS produces recommendations using a taxonomy of three types of location-based ratings within a single framework.
- Spatial ratings for non-spatial items, represented as a four-tuple (user, ulocation, rating, item), where ulocation represents a user location, for example, a user located at home rating a book.
- Non-spatial ratings for spatial items, represented as a four-tuple (user, rating, item, ilocation), where ilocation represents an item location, for example, a user with unknown location rating a restaurant.
- Spatial ratings for spatial items, represented as a five-tuple (user, ulocation, rating, item, ilocation), for example, a user at his/her office rating a restaurant visited for lunch.

**Advantages**

- Helps users discover new and interesting items.
- LARS produces personalized recommendations influenced by location-based ratings and a querying user location.

**Design of system**

Item details are added to the application database by the administrator. Item id will be generated automatically and item location is defined while giving the item input to the application. These items are queried by the user, according to the location, user preference and rating the item data will be retrieved.

The traditional item-based collaborative filtering (CF) method is a special case of LARS. CF takes as input the classical rating triplet (user, rating, item) such that neither the user location nor the item location are specified. In such case, LARS directly employs the traditional model building phase to calculate the similarity scores between all items. Moreover, recommendations are produced to the users using the recommendation generation phase.

Item ratings are given by the administrator. Rating will be given from 0.1 to 0.5 according to the preferences. Each item added to the database will be chosen and given a rating based upon the item id. Specification will be given for item rating.

**Spatial ratings for non-spatial items**

LARS produces recommendations using spatial ratings for non-spatial items represented by the tuple (user, ulocation, rating, item). The idea is to exploit preference locality, i.e., the observation that user opinions are spatially unique. Three requirements for producing recommendations using spatial ratings for non-spatial items are (1) Locality: recommendations should be influenced by those ratings with user locations spatially close to the querying user location (i.e., in a spatial neighborhood); (2) Scalability: the recommendation procedure and data structure should scale up to large number of users; (3) Influence: system users should have the ability to control the size of the spatial neighborhood that influences their recommendations. LARS achieves its requirements by employing a user partitioning technique that maintains an adaptive pyramid structure, where the shape of the adaptive pyramid is driven by the three goals of locality, scalability, and influence. The idea is to adaptively partition the rating tuples (user, ulocation, rating, item) into spatial regions based on the ulocation attribute. Then, LARS produces recommendations using any existing collaborative filtering method over the remaining three attributes (user, rating, item) of only the ratings within the spatial region containing the querying user.

**Non-spatial ratings for spatial items**

LARS produces recommendations using non-spatial ratings for spatial items represented by the tuple (user, rating, item, ilocation). The idea is to exploit travel locality, i.e., the observation that users limit their choice of spatial venues based on travel distance. Traditional LARS produces recommendations within reasonable travel distances by using travel penalty, a technique that penalizes the recommendation rank of items the further in travel distance they are from a querying user. Travel penalty may incur expensive computational overhead by calculating travel distance to each item. Thus, LARS employs an efficient query processing technique capable of early termination to produce the recommendations without calculating the travel distance to all items.

**Spatial ratings for spatial items**

LARS produces recommendations using spatial ratings for
spatial items represented by the tuple (user, u-location, rating, item, i-location). A salient feature of LARS is that both the user partitioning and travel penalty techniques can be used together with very little change to produce recommendations using spatial user ratings for spatial items. However, the only difference is that the item-based collaborative filtering prediction score \( P(u, i) \) used in the recommendation score calculation is generated using the (localized) collaborative filtering model from the partial pyramid cell that contains the querying user, instead of the system-wide collaborative filtering model.

**ARCHITECTURE DIAGRAM**

**Gantt Chart**

<table>
<thead>
<tr>
<th>Phase Description:</th>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1 Analysis</td>
<td>Analysis</td>
<td>Analyze the information user requirement.</td>
</tr>
<tr>
<td>Phase 2 Literature survey</td>
<td>Collect raw data and elaborate on literature surveys.</td>
<td></td>
</tr>
<tr>
<td>Phase 3 Design</td>
<td>Assign the module and design the process flow control.</td>
<td></td>
</tr>
<tr>
<td>Phase 4 Implementation</td>
<td>Implement the code for all the modules and integrate all the modules.</td>
<td></td>
</tr>
<tr>
<td>Phase 5 Testing</td>
<td>Test the code and overall process weather the process works properly.</td>
<td></td>
</tr>
<tr>
<td>Phase 6 Documentation</td>
<td>Prepare the document for this project with conclusion and future enhancement.</td>
<td></td>
</tr>
</tbody>
</table>

**Survey on Phase Diagram**

**CONCLUSION**

LARS, proposed location-aware recommender system, tackles a problem untouched by traditional recommender systems by dealing with three types of location-based ratings: spatial ratings for non-spatial items, non-spatial ratings for spatial items, and spatial ratings for spatial items. LARS employs user partitioning and travel penalty techniques to support spatial ratings and spatial items, respectively. Both techniques can be applied separately or in concert to support the various types of location-based ratings. Experimental analysis using real and synthetic data sets show that LARS is efficient, scalable, and provides better quality recommendations than techniques used in traditional recommender systems.

**References**


