An Agent Framework for Recommendation

Zhonghang Xia
Western Kentucky University
Department of Computer Science
Bowling Green, KY 42101
USA

Guangming Xing
Western Kentucky University
Department of Computer Science
Bowling Green, KY 42101
USA

Xuejun Jiang
Shandong Institute of Economic
Department of Statistics
Jinan, Shandong 250014
China

Abstract: Internet users are becoming overwhelmed by rapidly growing Web information. Two commonly used technologies to solve the problem are *information retrieval* and *collaborative filtering*. Existing information retrieval methods have been mainly developed for handling flat documents and current collaborative filtering systems still suffer from the sparsity problem in which most users may rate very few items comparing with the large number of available items in the systems. Moreover, current methods usually cope with these issues separately. In this paper, we develop an intelligent agent framework that integrates document collection, information retrieval and recommendation. In order to improve the query performance, similar XML documents are grouped together based on structural information. Different with conventional methods, we use tree-edit distance to measure the similarity/dissimilarity among XML documents. In collaborative filtering, we convert the recommendation problem into a classification problem and solve it by multi-class support vector machines. Experimental studies show that the accurate rates of our recommendation method outperform existing ones.

Key–Words: Collaborative filtering, Tree edit distance, Classification, Support vector machines

1 Introduction

The rapid expansion of the World Wide Web has greatly facilitated the growth of e-commence. Each day, more and more movies, books, and articles are created and the information is posted online through Web pages. Consequently, users quickly become overwhelmed and are spending more and more time to search for their desired targets. To address this problem, we have to face two challenges: information retrieval and recommendation.

The semistructured data extracted from Web pages are usually maintained by relational database technologies [17], where Web documents are decomposed and inserted into a set of tables. Studies [17] show that grouping documents with similar structures together can reduce table fragmentation, and, thus, improve query performance. Recent research effort [10] has focused on categorizing XML documents by structures. Since an XML document can be represented by an ordered, labeled tree, tree edit distance is a common metric to measure the similarity/disimilarity between two documents. Various algorithms have been proposed to compute the tree-edit distance between trees [19, 6, 5]. However, the cost of computing the tree edit distances reported in current work is quadratic to the document size, which is not suitable for a collection of large documents. An algorithm for finding the minimum edit distance between an XML document and a Document Type Definition (DTD) is given in [15], and a faster algorithm is reported in [5]. However, these algorithms are not suitable for a large data set because of their high time complexities.

Besides extracting data from Web pages, efficient and effective tools to help users narrow down desired targets is required. Recommender systems provide automated methods for users to search for interesting items with respect to users' preferences. The underlying techniques used in today's recommender systems can be classified into *collaborative filtering* (CF) and *content-based filtering* (CBF). CF algorithms exploit similarities among users or items based on users' feedbacks. CBF systems, on the other hand, recommend items of interest to the active user by exploiting content information of the items already rated. Typically, a profile is formed for a user individually by analyzing information regarding the content of items, such as desired actors/actresses, title, and description, etc. Additional items can be inferred from this profile. Usually, the content is difficult to analyze, thus CF algorithms are more successful in a broad range of areas.

Current CF recommender algorithms suffer from a fundamental problem, called sparsity problem [14]. Since the set of all possible available items in a system is very large, most users may have rated very few items, and, hence, it is difficult to find the active user's neighborhood with high similarity. As a result the accuracy of the recommendations may be poor.

A lot of research have been launched to improve the quality of recommendation systems. Machine learning is a standard paradigm of predicting ratings and preferences for users' interests by casting the prediction problem as a classification problem. Compared to other machine learning methods, *support vector machines* (SVMs) can be analyzed theoretically using concepts from computational learning theory, and it has also been successfully applied to many applications, such as text categorization [8]. However, the standard SVM classifier is not very successful [19] when it is applied in recommender systems due to the sparsity problem.

In this paper,we propose an agent framework for users to integrate the functions of information retrieval and recommendation. We first present a practical algorithm that uses top-down tree edit operations. It is shown that the algorithm runs in $O(p \times \log p \times n)$, where p is the size of the schema and n is the size of the XML document.

Then, we address the sparsity problem by repeatedly estimating the missing ratings for the items which users have not rated. We first initialize these missing values with default values to provide enough training examples for learning machines, and, then, build classifiers based on these training examples. After the classifiers are obtained, those missing values are re-estimated. This procedure is repeated until the termination criterion is met.

The rest of the paper is organized as follows. Section 2 describe the overview of the agent framework and our solutions. In Section 3, we introduce the algorithm for computing tree-edit distance and classifying XML documents based the distance. Our recommendation method is presented in Section 4. Experimental studies are presented in Section 5. Section 6 states the conclusion of the paper.

2 Overview of the System Model and Techniques

Consider an Internet service system consisting of multiple proxy servers. Each proxy server hosts an agent to provide recommendation services for local users. The agent consists of several components, including information collection, caching, information retrieval, collaborative filtering, etc. The agent can collect Web documents from the Internet by crawlers or exchange information with other agents. Useful information are retrieved from these documents and stored in relational tables. These tables are maintained by cache managers at the proxy servers. Upon receiving requests from users, agents trigger CF systems and provide recommendations to users. In this paper, we will focus on some tasks in information retrieval and collaborative filtering.

Information retrieval involves the tasks of fulfilling ephemeral interest queries to find a particular information need. Due to its wide deployment in the commercial world, relational database is a common practice to store XML documents. In this case, these XML documents have to be decomposed into small pieces and each piece is inserted into a corresponding table. Since queries are usually constructed based on path expression, grouping documents with similar structures would improve the performance of query processing. In order to compute users' preference, agents construct a table of users and items. Note that some elements in the user-item matrix are missing because the users have not rated the corresponding items.

3 Document Classification

We now start to introduce the details of our technique. This section focuses on assigning a document into a predefined category, given a set of labeled documents at hand. We first define a metric to measure similarity between two documents and present an efficient algorithm to compute the similarity. Then, we employ SVM classifiers for the classification.

3.1 Tree Edit distance and Similarity

For the purpose of classification, a proper similarity/dissimilarity metric should be provided. An XML document can be represented as an ordered, labeled tree T. Each node in T represents an XML element in the document and is labeled with the element tag name. Each edge in T represents the element nesting relationship in the document.

Tree edit distance [19] which is a natural extension of string edit distance can be used to measure the structural difference between two documents. However, using the tree edit distance between two documents directly may not be a good idea of measuring the dissimilarity. Hence, a potential solution to measure the dissimilarity between two documents is using the cost that a document conforms to the DTD schema generating the other document. This cost is actually the edit distance between the document and DTD. Specifically, given two documents x_1 and x_2 and the corresponding DTD schemas $s(x_1)$ and $s(x_2)$, the dissimilarity between x_1 and x_2 is defined by $\delta(x_1, x_2) = \delta(x_1, s(x_2)) + \delta(x_2, s(x_1))$. As a DTD may be recursive, some nodes may lead to a infinite path. Instead of working on a DTD directly, we convert it to a normalized regular hedge grammar. Regular hedge grammars were introduced by Murata in [9] for schema representation of XML data.

Based on the definition of regular hedge grammar in [9], a normalized regular hedge grammar (NRHG) is defined as follows:

Definition A NRHG is a 5-tuple (Σ, V_T, V_F, P, s), where:

1. Σ is a finite set of terminals,
2. V_T is a finite set of tree variables,
3. V_F is a finite set of forest variables,
4. P is a finite set of production rules, each of which takes one of the four forms as below:
   (a) $v_t \rightarrow x$, where v_t is a tree variable in V_T, and x is a terminal in Σ. This rule is used to generate a tree with a single node.
   (b) $v_t \rightarrow a\langle v_f \rangle$, where v_t is a tree variable in V_T, a is a terminal in Σ and v_f is a forest variable in V_F. This rule is used to put a new node as a new root of the forest that is generated by forest variable
   (c) $v_f \rightarrow v_t$, where v_f is a forest variable and v_t is a tree variable. This rule is the base case to generate a tree for a forest.
   (d) $v_f \rightarrow v_t v_f'$, where v_f and v_f' are forest variables and v_t is a tree variable.
5. $s \in V_T$ is the starting symbol, which defines the tree pattern that can be generated by this grammar.

We have shown in [5] that the optimal edit sequences to transform an XML document to conform to a DTD can be computed in $O(n^2 p(n + \log p))$ time, where n is the size of the document, and p is the size of DTD. Although it is a polynomial time algorithm, it becomes inefficient to process XML document with more than $1M$ nodes, which is very common in E-commerce application. In the next section, a linear time algorithm to compute the top-down edit distance is presented. We also show that the algorithm can be used to compute the restricted top-down edit distance with just one change.

3.2 Tree Edit distance and Similarity

We present the recursion to calculate the distance between an ordered tree and a NRHG. We follow the same idea and notations as presented in [5].

Notations:

$t[i]$ represents the node of T whose post-order is i and it refers to the label of the node $t[i]$ when there is no confusion;

$T[i]$ represents the sub-tree rooted at node $t[i]$;

$F[i]$ represents the sub-forest obtained by deleting node $t[i]$ from the tree $T[i]$;

$p(i)$ refers to the order of the parent node of $t[i]$;

$n(i)$ refers to the order of the right sibling of $t[i]$;

$F_s[i]$ denotes the suffix-forest obtained by deleting the left sibling(s) of $t[i]$ from $F[p(i)]$.

$\delta(T_t, T_s)$: is the minimum cost to transform the source tree T_s to the target tree T_t;

$\delta(v_t, \lambda)$: is the minimum cost to construct a tree t such that $v_t \rightarrow^* t$;

$\delta(F_t, F_s)$: is the minimum cost to transform the source forest F_s to the target forest F_t;

$\delta(v_f, \lambda)$: is the minimum cost to construct a tree t such that $v_f \rightarrow^* f$;

For $v_t \in V_T$ in a NRHG, and a tree t, define:

$$C[v_t, T[i]] = \min\{\delta(t, T[i]) : v_t \rightarrow^* t\}.$$

Similarly, for $v_f \in V_F$ in a NRHG, and a forest f, define:

$$C[v_f, F[i]] = \min\{\delta(f, F[i]) : v_f \rightarrow^* f\}.$$

We state our main theorem as:

Theorem For each $v_t \in V_T$, and each sub-tree $T[i]$:

$$C[v_t, T[i]] = \min \left\{ \begin{array}{lll} v_t \rightarrow x & \delta(x, T[i]) & (1) \\ v_t \rightarrow a\langle v_f \rangle & \delta(\lambda, T[i]) + C[v_t, \lambda] & (2) \\ v_t \rightarrow a\langle v_f \rangle & C[v_f, F[i]] + \delta(a, t[i]) & (3) \end{array} \right\}$$

and for each $v_f \in V_F$ and sub-forest $F_s[i] = T[i]F_s[n(i)]$:

$$C[v_f, F_s[i]] = \min \left\{ \begin{array}{lll} v_f \rightarrow v_t & C[v_t, T[i]] + \delta(\lambda, F_s[n(i)]) & (4) \\ v_f \rightarrow v_t & \delta(\lambda, T[i]) + C[v_f, F_s[n(i)]] & (5) \\ v_f \rightarrow v_t v_f' & C[v_t, T[i]] + C[v_f', F_s[n(i)]] & (6) \\ v_f \rightarrow v_t v_f' & \delta(\lambda, T[i]) + C[v_f, F_s[n(i)]] & (7) \\ v_f \rightarrow v_t v_f & C[v_t, \lambda] + C[v_f', F_s[i]] & (8) \\ v_f \rightarrow v_f' & C[v_f', F_s[i]] & (9) \end{array} \right\}$$

Due to the space limit, the correctness of the above theorem is omitted from this paper.

There are at most $O(|V_F|)$ number of vertices and $O(|V_F|)$ number of edges in the graph, and there is no negative weight edge in this graph. The time needed to complete Dijkstra's shortest path algorithm is $O(V \log V + E)$, so the overall time to compute $C[v_f, F_s[i]]$ for all $v_f \in V_F$ is $O(|V_F| \log |V_F|)$.

For a tree with n nodes and a grammar with p rules, there are $O(n \times p)$ $C[v_t, T[i]]$ to compute, and it take constant time to compute each $C[v_t, T[i]]$.

Similarly, there are $O(n \times p)$ $C[v_f, F_s[i]]$ to compute. For each $F_s[i]$, it takes $p \log p$ time to compute $C[v_f, F_s[i]]$ for all the forest variables. So the above procedure can be completed in $O(n \times p \log p)$ time.

3.3 Linear Classifier

A general notion of the classification problem can be described as follows. Given a set of l training documents $(x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l) \in \chi \times \{1, \ldots\ldots, k\}$, we aim to estimate a prediction function f such that it can classify a new document x.

SVMs have been used successfully in the context of text classification. By introducing slack variables $\xi_n \geq 0, 1 \leq n \leq l$, SVMs allow some classification errors. In this case, the optimization problem can be written as follows:

$$\begin{aligned}
&\min\ \tfrac{1}{2}(||w||)^2 + C \sum_{n=1}^{l} \xi_n \\
&\text{subject to } y_n(w^T x + b) \geq 1 - \xi_n, 1 \leq n \leq l \\
&\quad \xi_n \geq 0, 1 \leq n \leq l
\end{aligned}$$

where C is a predefined constant to control the tradoff between the gap of two classes and errors of classification.

The above binary classification method can be extended for solving multi-class classification problems[7]. In paper we consider an one-against-one method, where a binary classifier is constructed for each pair of classes i and j, and totally $k(k-1)/2$ binary classifiers have to be constructed. If a classifier $w_{ij}^T x + b_{ij}$ assigns x to the ith class, then a vote for the ith class increases by one; otherwise, a vote for the jth class increases by one. Finally, the largest vote decides the class label of x.

4 Collaborative Filtering System

Consider a recommendation system consisting of M users and N items. There is a particular user called active user u_a. The task of collaborative filtering is to predict the preference of the active user based on the opinions of a set of similar users. Each user u_j has given opinions on a set of items I_j and its opinion on item n is given as numeric rating x_{jn}. Note that I_j can be empty. To predict the preference of the active user, we need to estimate its rating on item $n \notin I_a$. Let A be a user-item matrix, where the value of ith row and jth column is x_{ij}.

Current CF algorithms are usually divided into two categories: Memory-based and Model based algorithms [4]. Memory-based algorithms utilize the entire user-item database to make predictions. Model-based algorithms make predictions by first developing a model of user ratings and then predict according to this model. Our approach falls into the Model-based category.

4.1 Model-based collaborative filtering

Since memory-based algorithms seriously suffer from the sparsity problem, model-based approaches have been studied to overcome this problem by learning a model for predicting ratings of unobserved items. These approaches include item-based [14], clustering [16], and classification [3], etc. The item-based method assumes that users like to purchase items similar to those items they have selected in the history. To measure the similarity between two items, it first searches a set of users who have rated both of the two items and, then, compute the similarity with some techniques. Let $U_{in} = \{$users who have rated both item i and $n\}$. The similarity of item i and item n is computed by

$$S_{in} = \frac{\sum_{u \in U_{in}} (x_{ui} - \hat{x}_u)(x_{un} - \hat{x}_u)}{\sqrt{\sum_{u \in U_{in}} (x_{ui} - \hat{x}_u)^2} \sqrt{\sum_{u \in U_{in}} (x_{un} - \hat{x}_u)^2}}$$

where $\hat{x}_u$ is the average of the uth user's ratings, that is $\hat{x}_u = \frac{\sum_{u \in I_u} x_{un}}{|I_u|}$ After the similarity computation, we can predict the preference of u_a on item n. It is given as follows:

$$p_{an} = \frac{\sum_{i \in S} (S_{ni} \times x_{ai})}{\sum_{i \in S} |S_{ni}|}$$

where S is the set of items similar to item n.

4.2 Classification Methods

In this paper, we recast collaborative filtering as a classification problem. Based on its numeric rating, an item or a user can be classified into a corresponding class. There are two ways to cast the problem [1]. One way is to treat every item as a separate classification problem. Given an item n, one can build a classifier to predict which class the active user belongs to. Every user u_j is represented as a vector in the feature space by using u_j's ratings on items other than n. A more common way to cast the classification problem is to treat every user as a separate problem [3]. One can build a classifier for the active user u_a by using items as training instances. To be specific, training instance n is represented as a feature vector x_n in which elements are ratings provided by other users. Without loss of generality, we consider the first user u_1 as the active user and u_1 has rated the first l items, that is, $I_1 = \{1, \ldots, l\}$. Then, the feature vector of item

n,$1 \leq n \leq l$ is $x_n = (x_{2n}, x_{3n}, \ldots, x_{Mn})^T$ and its class label y_n is rating x_{1n} . We need to predict labels for all other feature vectors x_n,$l + 1 \leq n \leq N$.

As the number of items grows very fast, users usually just rate a smaller percentage of the item population. That is the so-called sparsity problem. As a result, the corresponding feature vectors have many empty elements. We overcome these problems by iteratively estimating missing elements in the user-item matrix A. For each element $a_{mn} \in A$, we have

$$a_{mn} = \begin{cases} x_{mn} & \text{if } n \in I_m; \\ p_{mn} & \text{otherwise.} \end{cases}$$

We will use SVM classifier introduced in the above section to solve the recommendation problem. Initially, We randomly assign 0 or 1 to p_{mn}. According to SVM classifier f_{mn}, a new p_{mn} is given. After each p_{mn} is re-computed, we test the current classifiers with the test data, denoted by T. Let T_c be the total number of correct labels computed with current classifiers. The accurate rate is defined as $|T_c|/|T|$. If the difference of accurate rates between two consecutive steps is less than a predefined value ϵ, the algorithm stops.

5 Experimental Studies

Our goal is to evaluate the performance of the above methods in agent systems. The experimental studies include two parts. We first compare the performances of our document classifier with tree edit distance, and, then, compare our heuristic recommender method with the user-based and item-based method.

We also tested the classification system on the benchmark data from XML Mining Challenge [18]. There are 3 sets of data for structure oriented classification in the benchmark. Each set consists of 11 classes of documents. The order of noises is increasing from set 1 to set 3. We use about 90% of the document to train the classifier, and the remaining 10% for testing. The number of correctly classified files and the number of total files are listed in Table 1. The results show that the distance between an XML document and a schema is a effective similarity measure for XML documents.

For the second part of experimental studies, we use a dataset from MovieLens[11]. In this database, there are about 43000 users who have given ratings on 3500 different movies. Before the training process, some data, e.g., some users who just rated on very few movies and some movies which were rated by very few users, have to be cleaned out. The remaining data were randomly divided into training set and test set according to 90/10 ratio. Two training set dataset A and dataset B were created.

	dataset A (%)	dataset B (%)
LIBSVM	75.35	74.81
User-based	64.23	63.57
Item-based	65.17	63.62

Table 2: Recommendation Results

We use LIBSVM [7] to solve the multiclass classification problem and compare it with the user-based and item-based algorithms aforementioned (all missing values are initialized with zeros). The metric we used to evaluate the performance of algorithms is the average accurate rate, defined as the percentage of correct ratings. ϵ is selected as 0.005 in the entire experiment. The results achieved by three algorithms for two datasets are given in Table 2. As we can see, the accurate rates of the SVM methods are higher than the user-based and item-based approaches.

5.1 Conclusion

In this paper, we proposed an agent framework to integrate services of information retrieval and recommendation. The proxy servers, which are close to the users, host agents to collect Web pages, decompose them, store useful information in relational tables, and then recommend interesting items to users. In order to improve the query performance, we categorized XML documents according to the structural information. The similarity of two XML documents is measured by tree edit distance. We provide an efficient solution for approximating matching between an XML document and a schema.

References:

[1] Justin Basilico, Thomas Hofmann, *Unifying collaborative and content-based filtering*,ICML 2004.

[2] Bharat, K., T. Kamba, and M. Albers, *Personalized, Interactive News on the Web*, Multimedia Systems,6(5), 1998, pp. 249-358.

[3] Billsus, D. and Pazzani, M. J., *Learning collaborative information filters*, In Proceedings of the 15th International Conference on Machine Learning. Morgan Kaufmann, San Francisco, CA, 46-54, 1998.

[4] Breese, J. S., Heckerman, D., and Kadie, C., *Analysis of Predictive Algorithms for Collaborative Filtering*, In Proceedings of the 14th Con-

Class	Set 1(correct/total)	Set 2(correct/total)	Set 3(correct/total)
1	99/99	99/99	99/101
2	64/70	63/66	48/75
3	71/96	66/86	50/85
4	81/81	81/90	63/72
5	116/116	109/109	104/120
6	29/29	27/28	25/26
7	73/73	72/73	71/73
8	10/13	16/25	6/15
9	44/44	43/43	43/47
10	125/125	128/128	118/130
11	55/55	55/56	53/59
Overall	767/801	759/803	680/803

Table 1: Results of classification algorithms based on edit distance

ference on Uncertainty in Artificial Intelligence, pp. 43-52,1998.

[5] Rodney Canfield, Guangming Xing, *Approximate XML Document Matching (Poster)*, Proceedings of ACM Symposium on Applied Computing, March, 2005, Santa Fe, NM.

[6] W. Chen, *New Algorithm for Ordered Tree-to-Tree Correction Problem*, J. of Algorithms, 40:135-158, 2001.

[7] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[8] T. Joachims, *Text categorization with support vector machines*, In European Conference on Machine Learning (ECML), 1998.

[9] M. Murata *Hedge Automata: A Formal Model for XML Schemata*, http://www.xml.gr.jp/relax/hedge_nice.html.

[10] A. Nierman, H. V. Jagadish, *Evaluating structural similarity in XML documents*, WebDB 2002, Madison, Wisconsin, June 2002.

[11] http://www.cs.umn.edu/Research/GroupLens/

[12] G. Myers *Approximately Matching Context Free Languages*, Information Processing Letters, 54, 2, pp. 85-92, 1995.

[13] Resnick, P., N. Iacovou, M. Sushak, P. Bergstrom, and J. Ried, *GroupLens: An open architecture for collaborative filtering of netnews*, In Proceedings of the ACM 1994 Conference on Computer Supported Collaborative Work (CSCW '94), Chapel Hill, NC, 1994, pp. 175-186.

[14] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl,J., *Item-based collaborative filtering recommendation algorithms*, In Proceedings of the 10th International World Wide Web Conference (WWW10), Hong Kong, 2001.

[15] Nobutaka Suzuki, *Finding an Optimum Edit Script between an XML Document and a DTD*, Proceedings of ACM Symposium on Applied Computing, pp. 647 - 653, March, 2005, Santa Fe, NM.

[16] Ungar, L. and Foster, D., *Clustering methods for collaborative filtering*, In Proceedings of the Workshop on Recommendation Systems,AAAI Press, Menlo Park California.

[17] Lian Wang, David W. Cheung, Nikos Mamoulis, and Siu Ming Yiu, *An efficient and scalable algorithm for clustering XML documents by structure*, IEEE Transactions on knowledge and data engineering, vol. 16, No 1, 2004.

[18] XML Document Mining Challenge, `http://xmlmining.lip6.fr/`

[19] D. Shasha, K. Zhang, *Approximate Tree Pattern Matching*, Chapter 14 Pattern Matching Algorithms (eds. Apostolico, A. and Galil, Z.), Oxford University Press, June 1997.