PRODUCT RECOMMENDATION SYSTEM FOR SMALL ONLINE RETAILERS USING ASSOCIATION RULES MINING

Junnan Chen  
CIISE, Concordia University  
junn_che@encs.concordia.ca

Courtney Miller  
CIISE, Concordia University  
co_mille@live.concordia.ca

Gaby G. Dagher  
CIISE, Concordia University  
dagher@encs.concordia.ca

ABSTRACT

Recommendation systems in e-commerce have become essential tools to help businesses increase their sales. In this paper, we detail the design of a product recommendation system for small online retailers. Our system is specifically designed to address the needs of retailers with small data pools and limited processing power, and is tested for accuracy, efficiency, and scalability on real life data from a small online retailer.

KEYWORDS

Data Mining, Database Management, E-commerce, Performance

1. INTRODUCTION

The use of product recommendation systems is ubiquitous among large e-commerce companies today; some of the more famous product recommendation modules can be found on Amazon.com (Linden et al, 2003) and eBay. In many ways, the profits of a large e-commerce company can rise and fall on the efficacy of their product recommendation algorithms, which is why such companies often put much of their time and money into these algorithms.

Smaller e-commerce companies, however, often do not have the skill or the scale of resources to implement algorithms like those of Amazon, which has largely put effective product recommendation systems out of the reach of smaller retailers. In order for a small retailer to implement a product recommendation system, such a system must be efficient when running on a server machine with modest computing capability, as small businesses normally do not have the financial capacity to invest in a large infrastructure. The system must also make do with significantly less training data than a powerhouse like Amazon might have. In order to be of use to the company, however, this recommendation system must still be robust enough to make a difference in customer click-through on recommended products.

In this paper, we propose a recommendation system for a real life small retailer. To make the system more robust, we identify multiple product prediction criteria which might apply to any given customer and we weight each of these criteria such that they can be applied based on the current customer to result in a single product recommendation.

The contributions of this paper can be summarized as follows:

Contribution #1: Though much work has been done on data mining algorithms for large businesses with an excellent resource base, we have found relatively little work in the less glamorous area of developing data mining systems for small retail businesses. The constraints and focus points of a data mining solution scaled for small online retailers are much different, and we have implemented our solution with these constraints in mind.
Contribution #2: We have tested our solution on real data from a small online retailer which has transaction data dating back slightly less than three years. We show that the proposed solution is relatively accurate, as well as efficient and scalable.

The rest of the paper is organized as follows. Section 2 reviews the related literature. Section 3 provides the formal definition of the recommendation system problem. Section 4 describes the proposed solution. Comprehensive experimental results are presented in Section 5. Finally, we conclude the paper in Section 6.

2. RELATED WORK

In this section, we briefly overview some relevant work in recommendation systems.

There are five recommendation algorithms that are commonly used: content-based recommendation, collaborative filtering recommendation, association rule-based recommendation, utility-based recommendation and knowledge-based recommendation. In practice, hybrid recommendations, which consist of two or more above-mentioned algorithms, are usually used.

Collaborative filtering is so far the earliest and most successful recommendation technology. It filters the information presented to the user by using information about other users’ preferences (Herlocker et al., 2000). It shows good performance in using existing experiences, interests, and personal tastes to extract preference information which is more difficult to find intuitively. User-based, item-based, and model-based are three collaborative filter types. Despite their efficiency, scalability and sparsity are the main limitations of these types. Cacheda et al. proposed a new algorithm based on trends and user-item differences to improve sparsity by 20% (Cacheda et al., 2011) while Zhang et al. proposed an approach called Smoothing and Fusing (CFSF) strategies which construct a local item-user matrix from large-scale item-user matrices to improve on the scalability issue (Zhang et al., 2009). In recent years, a personalized one-to-one marketing strategy has caught researchers’ attention, along with the fast growth of e-commerce. Item-based and user-based collaborative filtering have proven to be efficient and successful for businesses. Wang et al. proposed an approach which fuses predictions between item ratings from other users, ratings of a different item from the same user, and other similar ratings from other similar users. The model gives better recommendations even on problems with sparsity (Wang et al., 2006). Using association rules is a traditional data mining method. Agrawal et al. proposed efficient algorithms that fast-mine and prune the generated association rules (Agrawal et al., 1993) (Agrawal and Srikant, 1994). Association rules-based recommendation usually implies the top-N items recommendation. Deshpande et al. developed a model to calculate the similarities between different items and then to output the top recommendations (Deshpande and Karypis, 2004). Ren et al. proposed a model which dynamic-learns using tendencies of the customers’ profile, and thus improves accuracy (Ren et al., 2012).

3. PROBLEM FORMULATION

In this section, we formally define the research problem. First, we present an overview of the problem of scalable product-recommendation in Section 3.1. Next, we define and explain the data inputs in Section 3.2. Finally, we present the problem statement in Section 3.3.

3.1. PROBLEM OVERVIEW

In this paper, we examine a small online retailer that wishes to recommend products to its customers based on certain characteristics mined from its data. This product-recommendation system requires the interaction of the retailer, all of the retailer’s past customer base, and the active customer who has currently made a selection for his or her shopping cart.

The retailer wishes to accurately predict which product recommendation will be most likely to result in the active customer making an extra purchase during this current transaction. The retailer's customer base and the retailer’s active customer balance two desires: firstly, customers wish for a certain level of privacy and anonymity; secondly, customers may wish to be offered products which better suit their needs and convenience.

Most product-recommendation systems take these priorities into account. However, we go a step further as small online retailers have the following limitations: low computing resources and small data pools. Thus, our algorithm must both restrict its resource needs and make do with less overall data.

3.2. DATA INPUTS

In this section, we give an overview of the input data and its formatting, and of our methodology in cleaning and extracting said data.

We started our project by identifying and extracting relevant data tables from our online retailer. This data comes from an open-source e-commerce database which has been running for slightly less than three years, though with a moderately large established customer base. We took this data entirely out of context by extracting only those columns of interest to our algorithm, many of
which contain data ids rather than descriptions or names. Since the source database is normalized, this data will be of great use when it is plugged back into the original database and given back its context.

With regard to product data, we extracted product category ids rather than product ids, in order to improve scalability. For the data set we selected, this should yield high-quality results, since this retailer’s product categories are highly specific. Another small retailer may require the use of product ids instead, if their products are cross-listed across many categories, or if their categories are too loosely generalized. Table-1 provides a snapshot of the ORDERS table in our dataset.

Table 1 - Snapshot of collated data of interest

<table>
<thead>
<tr>
<th>order_id</th>
<th>category_id</th>
<th>customer_id</th>
<th>country_id</th>
<th>order_total</th>
<th>order_month</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>140</td>
<td>9</td>
<td>36</td>
<td>22.5000</td>
<td>2011-09-21 15:52:37</td>
</tr>
<tr>
<td>11</td>
<td>140</td>
<td>9</td>
<td>36</td>
<td>22.5000</td>
<td>2011-09-21 15:52:37</td>
</tr>
<tr>
<td>11</td>
<td>140</td>
<td>9</td>
<td>36</td>
<td>22.5000</td>
<td>2011-09-21 15:52:37</td>
</tr>
<tr>
<td>11</td>
<td>140</td>
<td>9</td>
<td>36</td>
<td>22.5000</td>
<td>2011-09-21 15:52:37</td>
</tr>
<tr>
<td>11</td>
<td>140</td>
<td>9</td>
<td>36</td>
<td>22.5000</td>
<td>2011-09-21 15:52:37</td>
</tr>
<tr>
<td>12</td>
<td>223</td>
<td>0</td>
<td>36</td>
<td>19.9500</td>
<td>2011-09-24 08:50:07</td>
</tr>
<tr>
<td>12</td>
<td>223</td>
<td>0</td>
<td>36</td>
<td>19.9500</td>
<td>2011-09-24 08:53:30</td>
</tr>
<tr>
<td>14</td>
<td>223</td>
<td>572</td>
<td>36</td>
<td>63.9000</td>
<td>2011-09-24 09:18:04</td>
</tr>
<tr>
<td>15</td>
<td>223</td>
<td>572</td>
<td>36</td>
<td>63.9000</td>
<td>2011-09-24 09:18:41</td>
</tr>
<tr>
<td>16</td>
<td>223</td>
<td>572</td>
<td>36</td>
<td>63.9000</td>
<td>2011-09-24 09:18:58</td>
</tr>
<tr>
<td>17</td>
<td>223</td>
<td>572</td>
<td>36</td>
<td>63.9000</td>
<td>2011-09-24 09:24:45</td>
</tr>
<tr>
<td>18</td>
<td>223</td>
<td>572</td>
<td>36</td>
<td>63.9000</td>
<td>2011-09-24 09:32:37</td>
</tr>
</tbody>
</table>

3.3. PROBLEM STATEMENT

Given normalized data by id as above, the objective is to design a recommendation system which accepts a current product selection as input and returns one recommended product as output. The system must be able to do the following: (1) give an efficient recommendation response given the limitations in the computing resources. Specifically, the recommendation process must consume less than one tenth of a second of extra processing, and (2) the algorithm must be able to make use of a relatively small training dataset, and it must adapt to use different criteria if one criterion is not available.

4. PROPOSED SOLUTION

In this section, we present our proposed product-recommendation system for small retailers. We explain in detail how each part was designed to handle the problems previously mentioned—low computing resources and small data pools.

4.1. ASSOCIATION RULES EXTRACTION

We define in our system four different criteria to help us determine a good product recommendation. Those criteria are:

- The previous purchases of the customer in question.
- The home country of the customer in question.
- The month of the current purchase.
- The product which the customer has most recently selected.

The goal is to extract hidden patterns from the training dataset that correspond to the four defined criteria. Since one of our constraints is low computing resources, we store the set of patterns corresponding to each criterion into a separate table for fast access. The following algorithms illustrate how the hidden patterns are extracted and stored; the data mining tool \( R \)\(^1\) is utilized to perform the mining process.

Algorithm 1 association_rules: Association Rules by Category

Function association_rules

\$
\{
\text{Load database.csv;}
\text{Load library ’arules’;}
\text{Remove duplicate items in each order;}
\text{Read category_id, order_id as ‘Transactions’;}
\text{Generate apriori rules (min sup = 0.07, min confident =0.5);}
\text{Read apriori rules as ‘Data Frame’;}
\text{Export rules;}
\}\$

In Algorithm-1, we determine association rules between the category ids of products in each transaction within our dataset. We use the ‘arules’ library in \( R \) to remove duplicate category ids within the same orders; we also use it to regroup our data from multiple entries per order into transaction-type data, which creates a ‘shopping cart’ format for our data. This shopping cart data is now in the proper format to run through the Apriori algorithm which comes with the arules library. Because we are

\(^1\) http://www.rdatamining.com/
expecting a high spread of categories, we define our minimum support at 7%, but require the minimum confidence to be 50%. Lastly, we export the data to a new csv file so that it can be later read into a table.

**Algorithm 2** country_category_ranks: Most Popular Categories by Country ID

```r
Function country_category_ranks {
  Load database.csv;
  Load library 'plyr';
  Count frequency by country_id and category_id;
  Arrange country_id by descending order;
  Export country_category_ranks;
}
```

In Algorithm-2, we use frequency analysis to determine which category ids are most closely tied to each country id. We use the ‘plyr’ library in R, to do a frequency count of category ids by country id. We then order the data into a more intuitively useful format by ranking it in descending order of frequency. Lastly, we export this data to a new csv file as well, so that we can read it into a different table than the results of Algorithm-1.

**Algorithm 3** customer_category_ranks: Most Popular Categories by Customer ID

```r
Function customer_category_ranks {
  Load database.csv;
  Load library 'plyr';
  Count frequency by customer_id and category_id;
  Arrange customer_id by descending order;
  Export category_customer_ranks;
}
```

In Algorithm-3, we use frequency analysis to determine which category ids are most closely tied to each customer id. We use the ‘plyr’ library in R, to do a frequency count of category ids by customer id. We then order the data into a more intuitively useful format by ranking it in descending order of frequency. This data is also exported into a csv file.

**Algorithm 4** month_category_ranks: Most Popular Categories by Month ID

```r
Function month_category_ranks {
  Load database.csv;
  Load library 'plyr';
  Read category_id and order_time;
  Format order_time to POSIXlt timestamps;
  Count frequency by month and category_id;
  Arrange month by descending order;
  Export month_category_ranks;
}
```

In Algorithm-4, we use frequency analysis to determine which category ids are most closely tied to each month of the year. We load the ‘plyr’ library in R, then reformat our timestamp format into month ids from 1 to 12. We use the ‘plyr’ library in R to do a frequency count of category ids by this month id. We then order the data into a more intuitively useful format by ranking it in descending order of frequency. This data is the last to be exported into a csv file.

Later queries on these tables are made using a PHP interface which simulates integration with an e-
commerce software by sending a query to the cached table and requesting a category recommendation.

4.2. DETERMINE BEST CATEGORY
As mentioned in Section-4.1, each criterion has its own table, which directly relates information about the active customer or the last product selection to likely category recommendations. Each of these criteria suggests the top 5 product category ids based on the data of the active customer, starting with the most-suggested category.

Once each criterion’s algorithm has run, its results are weighted based on importance, and the weighted results of each algorithm are added together. The category id with the most support is then selected, and is given as the overall output. Since one of the constraints is small data pools, if a criterion has no relevant data available (for instance, if the customer is new and has made no previous purchases), that criterion will be ignored, and our system will make its recommendation decision based only on other criteria which do have relevant data available.

4.3. EXECUTING A RECOMMENDATION
Normally, the logic used in calculating a recommendation is done in the model layer of an e-commerce website. We used a simple set of PHP functions to handle this calculation; these functions can be added to the model layer of any e-commerce site which uses an MVC framework.

Figure-2 illustrates the overall process of our recommendation system. The process in its entirety consists of four major steps:

Step 1 – Accept Active Order Variables. The customer selects a product, and information from that active customer and his active order is sent to a PHP algorithm in the model layer.

Step 2 – Query Cached Tables. Each relevant table of cached data association rules is queried. The queries return up to 5 relevant category ids for potential recommendation, depending on availability. These category ids are returned in order of descending importance.

Step 3 – Weight Query Results. We exclude all criteria with no query results from its calculations. We define the importance (weight) of each result as follows:

- Customer Classification: 40%
- Category Associations: 30%
- Month Classification: 20%
- Country Classification: 10%

We determined the weights above based on the potential specificity of each criterion to the order.

That is, a customer’s previous history of purchases gives the most potentially relevant and specific data regarding his purchasing needs, while a customer’s country gives the least specific data regarding his purchasing needs.

Step 4 – Calculate the Best Recommendation. After weighting our query results, we assign each returned category id a ‘value’ commensurate with its ranking (1 through 5) and the weight of its criterion. We then add the value of any category ids which repeat across multiple criteria, and return the category id with the most aggregate value as the recommended category.

5. EXPERIMENTAL EVALUATION
We implemented our product recommendation system in the 64-bit Windows 7 environment on a machine equipped with an AMD E-450 APU 1.65 GHz Processor and 8GB DRAM. The training dataset comes from an active small online retailer whose database contains transactions dating back slightly less than three years. The number of records in the training dataset is 4000 records.

5.1. ACCURACY
To measure the accuracy of our solution, we built our test dataset from the same online retailer’s database by extracting all relevant orders which had been placed since the last order we included in our training dataset. We excluded orders which had only one item in their basket (which made them irrelevant to tests); we also excluded orders from new categories which had not been in the original algorithm’s dataset. The total number of records in the test dataset is 200 records. We start by setting the first listed category_id of each transaction in the test dataset as the ‘active category’ selection. Next, we extract the associated customer_id, country_id, and month_id as the other input variables, and then test to see if the category_id which our system recommends matches the actual one in the order, i.e., determine the percentage of true positives.

![Figure 4](chart.png) - Accuracy of recommendations given for increasing number of order records in the test dataset.
Figure-4 depicts the accuracy of our recommendation system, where the number of data records is 100 and 200 respectively. We observe that the accuracy can be as high as 56% when the number of records in the test dataset is 100, and 28% when the number of records in the test dataset is 200. Even though the accuracy decreased when the number of records increased from 100 to 200, we argue that this occurred because the size of the test dataset available to us is small, and this pattern will not bear out over a larger test dataset.

5.2. EFFICIENCY AND SCALABILITY
One major contribution of our work is the development of a scalable product recommendation system. We study the runtime required for recommending a product to an active customer, where the number of records in the test dataset ranges from 1,000 to 4,000, as depicted in Figure-5.

![Figure 5 - Scalability of product recommendation w.r.t. the number of records in the training dataset](image)

We observe that our proposed system is scalable as the runtime grows linearly when the number of records increases.

According to Jakob Nielsen, the author of *Usability Engineering* (Nielsen, 1993), 0.1 second is considered instantaneous from a user’s point of view. Therefore, we argue that our proposed recommendation system is also efficient, because our tests show that it takes on average 7 milliseconds to run a single recommendation.

6. CONCLUSIONS
In this paper, we address the problem of designing a recommendation system that addresses the needs of small online retailers. That is, it should effectively work with small data pools and execute relatively quickly. Our proposed solution makes use of cached association rules to cut down on runtime processing and uses multiple weighted criteria for recommendation in order to make good use of a small data pool. As a future work, more product prediction criteria can be added to yield higher recommendation accuracy.

7. REFERENCES