

# Library Recommendation System

Chad Hansen Dave Frogley Reed McGrew Phuong Pham  
Brigham Young University

## Abstract

We present our efforts to create a system that recommends additional books for library patrons at the moment of checkout based on the checkout history of the books the patron has selected. We measured our success by comparing with Amazon.com and by surveying students.

We acquired a large volume of data for training our system from the library itself. After filtering and formatting the data, we set about designing an algorithm similar to Apriori that could handle the large amounts of data we had gathered in the format we had chosen.

After implementing our algorithm, we used it to generate sets of rules with varying thresholds of support and confidence. We split up the results and validated the rules.

## 1 Introduction

The library recommendation system recommends books for library patrons based on the transaction history of a book,  $b_t$ , in the current transaction. The rule generator uses a simple counting system to analyze the transaction history of book  $b_t$  from 2001 through 2008 and selects as recommendations those books  $B$  that were checked out within a short time of book  $b_t$  on multiple occasions as recommendations. Patrons looking for information on a given topic can be directed to other books relevant to the topic.

### 1.1 Pros and Cons of Our Project

Our project helps patrons doing research to find additional and/or related material. It also helps patrons interested in simple pleasure reading to find books and authors similar to the ones they already enjoy. Additionally, our project helps the library itself by tracking the popularity or usefulness of the books it keeps in circulation.

We based the recommendations in our system solely on the transaction histories of the books. Therefore, even a book that is extremely related to the patron's book will not be recommended unless the two have been checked out together frequently; for the same reason, patrons checking out rarely used or new books will not get any recommendations for the rare or new book.

### 1.2 Overview of Success Criteria

To measure the success of our project, we selected two simple tests: 1) we selected the best rules generated by our system and compared the recommendations from those rules to those generated by Amazon.com for the selected book; 2) we selected a subset of our best rules and surveyed various people around campus, asking them their opinions of the rules generated (i.e., do the recommended books appear to be related to the book the rule is based on). Our goal was 80% positive responses.

## 2 Approach

To build a library recommendation system, we needed data from a library. The Harold B. Lee Library (HBL), on campus, was willing to let us use their item checkout data. This data was initially in large log files that contained all the checkout information from January 2001 through November 2008. Each entry, which we refer to as checkouts, in these log files represents a single book being checked out by a patron. If the same patron were to checkout two books at the same time than two entries would be made in the log files, one for each book.

### 2.1 Initial Data Collection and Preparation

The data originally contained the BYU Net ID numbers for many students and faculty on campus as well as some Social Security numbers that helped to identify unique patrons of the library. The HBL requested that we privatize these unique numbers before we did anything with the data so that they could not be traced back to the original patrons (to protect their privacy). We privatized the patron IDs and maintained their uniqueness by feeding them through an md5 hash.

After the patron ID numbers were privatized we needed to determine what pieces of information from the checkout logs would be useful in building our recommendation system. We decided only three values from the checkouts would be of use: the privatized unique patron ID number, the item bar code, and the date of the checkout.

Our goal in creating recommendations was to look at each transaction made by each patron to see if there were any emerging patterns. For example, if book  $b_r \in B$  showed up frequently (as defined in Section 2.2) with book  $b_t$ , then book  $b_r$  might make a good recommendation for patrons checking out book  $b_t$ .

Due to holes and inconsistencies in the data it was difficult to determine the equivalence of two books. It was impossible to determine book equivalence based on bar codes, because individual copies of the same book do not share bare codes. For example, if the library were to buy three copies of the same book there would be three different bar codes that would each reference three copies of the same book.

Similarly, ISBNs can not be used to determine book equivalence. If a book is printed by two different publishers, each publisher will use a different ISBN. Also, if a book has multiple editions, each edition may have a different ISBN. Some older books don't have ISBN. In order to do this we needed more information about the checkouts. After cleaning up our initial data we were able to obtain from library personnel the following additional fields for each checkout: *author*, *title*, *call number* (number used by the library to shelve the books), *type* (book, audio, sampler, etc.), and *ISBN* (if available).

After obtaining this new information we observed that many of the items being checked out were not books. Audio CDs and headphones are just two of the different non-book items we saw frequently. Using the *type* field we were able to filter non-book items out of the checkout history.

Next, we created unique IDs for each book by combining the *author* and *title* fields, stripping all punctuation except apostrophes. We removed the spaces from the *author* portion and separated it from the *book* portion with a forward slash to make it easier to read the resulting rules generated by our model. Here is an example of a unique book ID:

*Ender's Game* by *Orson Scott Card* would look like *cardorsonscott/ender's game*.

Next we created pseudo transactions by combining all of the books that were checked out by the same patron within a specific time interval into a comma separated list. For example, every book that a single patron checked out during a 24 hour period was combined into one transaction. We created several sets of pseudo transactions using time intervals of one day, three days, seven days and 14 days.

## 2.2 Our Algorithm : Little Apriori

We implemented a simplified version of the Apriori algorithm. For each pseudo transaction  $t_b \in T$  the following process is repeated for each book  $t_{b,j} \in t_b$ : increment the antecedent<sup>1</sup> support count,  $\text{support}[t_{b,j}]$ , and then increment the consequent<sup>2</sup> support count for every other book in the transaction. Our actual implementation stores these counts by hashing each new book encountered in the pseudo transaction into an associative array<sup>3</sup> of objects containing the antecedent support counts, which constitute the spine of the data structure (see Figure 1). Also included in the objects of the spine are references to other associative arrays which contain support counts for all consequents belonging to their corresponding antecedent (as illustrated by the ribs of the data structure

<sup>1</sup>The left side of an association rule. In the context of our system this is the book being checked out by a patron.

<sup>2</sup>The right side of an association rule. Our system uses the consequent as the book being recommended to a patron.

<sup>3</sup>A dynamically growing hash table that doesn't waste space on empty slots.

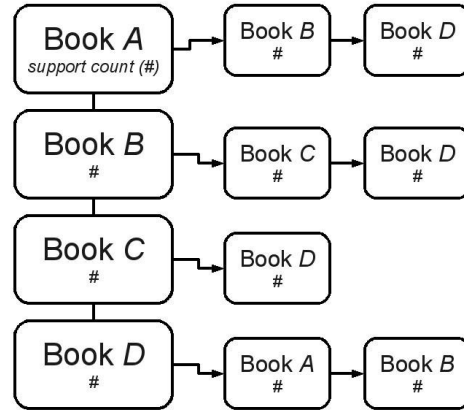


Figure 1: Representation of data structure used.

in Figure 1).

From the support count information, our algorithm extracts a list of all possible one-antecedent, one-consequent rules. Our data structure also contains all information necessary to calculate one-antecedent, multiple-consequent rules, though we chose not to calculate them. In the context of a library recommendation system, multiple-consequent rules do not add additional information that single-consequent rules cannot already provide. We are not trying to create market baskets, and we intended to lump all consequents for all rules with a common antecedent into one recommendation pool anyway, so it did not make sense to incur the additional computational and spatial costs associated with calculating multiple-consequent rules.

From the resulting data structure, individual rules can be derived cheaply by simply choosing one antecedent and a corresponding book from its consequent list. Support for a rule is simply the count associated with the consequent. Confidence is the consequent count divided by the antecedent count.

## 2.3 First Attempt

Once we had our transaction files and our model generator working we attempted to create rules. After running the one day time interval transactions through Little Apriori (see Algorithm 1) with a minimum support of 0.0005 and a minimum confidence of 0.2, we only had 30 rules. Any higher thresholds obviously yielded even fewer rules. These results were not acceptable to us, so we computed rules again using a three day time interval with the same minimum support and minimum confidence. We obtained 35 rules, which was also not very satisfying.

When we tried generating rules with a seven day time interval we found that our program (PHP script) ran out of allowed memory before it could finish. We had already encountered this problem with the three day time interval and had increased PHP's allowed memory to 2 GB. We increased it again to 3 GB, which still was not sufficient to allow our script to finish with the seven day time interval.

---

**Algorithm 1 : Little Apriori** – **Input:** pseudo transaction file  $T$ ,  $minsupport$ ,  $minconfidence$  – **Output:** support counts for all book pairings

---

```
for  $t_b \in T$  do {Iterate through each pseudo transaction}
  for  $t_{b,j} \in t_b$  do {Iterate through each book in a pseudo transaction}
    support[ $t_{b,j}$ ]++
     $k = 0$ 
     $n = \text{sizeof}(t_b)$ 
    for  $t_{b,k}$  to  $t_{b,n}$  do {Iterate through every other book}
      if  $k! = j$  then
        support[ $t_{b,j}$ ][ $t_{b,k}$ ]++
      end if
    end for
  end for
end for
for  $t_b \in support$  do
  for  $t_{b,j} \in support[t_b]$  do
     $confidence = \text{support}[t_b][t_{b,j}] / \text{support}[t_b]$ 
    if  $\text{support}[t_b][t_{b,j}] \geq minsupport \ \&\& \ confidence \geq minconfidence$  then
      rules.add( $t_b \Rightarrow t_{b,j}$ )
    end if
  end for
end for
```

---

We do not know where exactly to ascribe blame for this first failure. It may have failed because our program was written in PHP, or possibly because of our brute force implementation of the algorithm. We do know that we had an enormous amount of data, and we were reluctant to discard any of it, seeing as we wanted to generate recommendations for as many books as possible.

## 2.4 Second Attempt

We decided that if we were to store all of the checkout information as well as the unique book information in a database, the information would be easier to work with. We hoped to be able to generate rules more easily as well as retrieve unique book information more quickly. So we created a database to house the unique book information as well as the checkout information. We were able to generate SQL insert statements easily from each of the checkouts. We even used a nice little MySQL trick to determine if a book was already in the database before the insert statements ran. We generated eight SQL files (one for each year of data that we had) and ran the scripts to insert unique books into the database.

The first two scripts would have taken well over 48 hours to complete before we canceled them. After about 48 hours of running these conditional SQL insert statements we only had 287,913 unique books represented in the database and we hadn't even started inserting the checkouts. We also had another six conditional SQL insert statement files to run. We decided SQL was not going to work.

We determined later that there were around 800,000 unique books that we would have had to insert.

## 2.5 Third Attempt

Our third attempt to generate rules was another complete failure. Instead of inserting the items into a database we decided to keep the transaction files separated into years, storing them as serialized JSON objects and then retrieve each year transaction file one at a time and combine the results to generate rules.

We serialized the transaction information for 2008 and found that we couldn't even read the JSON object back into PHP to start parsing it again. Once again we didn't have enough memory allotted to the PHP script.

## 2.6 Other Attempts

We briefly entertained the idea of rewriting our algorithm in C++. We felt the PHP scripts were running too slowly and took up too much memory, and we knew C++ had the potential to do much better on both accounts. However, the PHP scripts at the time employed JSON serialization, which is not yet well supported in C++. Also, C++ and PHP behave so differently that much of the correct functionality was "lost in translation." To make the C++ program behave correctly, we would have needed to do a complete rewrite. In the end, we solved our speed and memory problems with smarter scripts, so the C++ implementation became unnecessary.

## 2.7 Final Solution

Realizing that we did not need to produce a large result set, we decided that having a few quality rules and validating those rules would fulfill our original success criteria. In the initial definition of our success criteria, which we put on our project wiki, we specifically stated, "We know we will not be able to offer recommendations for each book, because some of them appear very infrequently in transactions."

We still needed something to trim down our data in order to obtain the best rules and still be able to run our Little Apriori implementation. We ended up doing essentially the same thing that was described in Section 2.3, with the addition of an intermediate step.

Before we converted the checkout data to transaction data we pruned it by removing any book that did not show up more times in one year than a given threshold. We experimented with this threshold and ended up using three as the pruning threshold value (if a book did not show up at least three times in a given year, it was removed from all transactions for that year).

After pruning the checkout books and creating the transactions for each year, we were easily able to combine the transactions into a single file and run them through our rule generator. After throwing out books that did not show up at least three times in one year, we generated 101 rules. Our minimum support count was 15 (a rule needed to show up at least 15 times in all eight years) and our minimum confidence was 0.7.

## 3 Results

Our 101 rules seemed to make sense through visual inspection of the titles.

The Cat in the Hat Comes Back (20)  $\Rightarrow$  The Cat in the Hat (15)

The subjects of most antecedents and their corresponding consequents generally correlated well. However, to verify our initial impressions in a more objective fashion, we designed two different validation schemes.

### 3.1 Design of Verification Approaches

To validate our 101 rules, we compared them recommendations from Amazon.com, and (probably more accurately) we handed out surveys to random people to see whether or not our recommendations made sense to them.

#### Amazon Approach

First, we used Amazon.com for our first verification because it is a large on-line bookstore and many books are bought and sold by different customers. With Amazon.com's recommendation system, we verified our work based on a real and functional application, and we also saved time.

**Procedure:** We divided the 101 rules into 4 sections and each member of the team performed verification on the assigned section.

- Select one book from our rules and search for it on Amazon.com
- If Amazon finds the book, click the title of the book and go to the recommendation section
- Search for the book our system recommended in Amazon's recommendation list
- Record the result: if the books match, mark "good," otherwise mark "ignore"

**Result:** After doing the verification with Amazon.com, we had 27 out of 75 rules that were confirmed to be "good." Therefore, our system agreed with Amazon.com for 36.0% of the rules.

#### Survey Approach

To check our system's usability, we surveyed random people to see if our recommended books made sense. We preferred this recommendation method because our recommendation system is for people who use the library to check out books based on the patrons' interests.

**Procedure:** We took a random sample of 50 rules to perform survey verification, each team member taking a portion of the sample.

- Make surveys that contain our chosen rules
- Randomly select participants from different genders, locations, and ages
- Ask participants if they think those recommended books go well with those books being checked out
- Record the result: if the books make sense, mark "good", otherwise mark "ignore"

**Result:** After the survey, we had 43 out of 50 rules that participants said made sense to them. That means we had an 86.0% positive response rate.

### 3.2 Issues with Verification

After comparing the results from verifying our rules, we took time to discuss and investigate the differences in the results from both verification approaches. We thought of several factors that might have effected our results.

#### Buying vs. Borrowing

Buying from a store takes a much larger commitment than borrowing from the library. A library patron is free to borrow many books without having to pay any additional charge. The differences between these two paradigms are worth exploring and very well could have a large effect on book associations.

#### Customers and Patrons

The people who buy books from Amazon.com are from many different areas and professions; their interests are more complex and various. On the other hand, the participants who took the survey were mostly BYU students; their interests are based mostly upon doing research for classes and using books from the library to aid their learning.

#### Location of Books

Amazon is a on-line bookstore where customers do not physically search for books. They buy books based on what is displayed through search algorithms provided by Amazon.com. In contrast, books in the library are located next to each other; they are sorted by categories. This can impact the way patrons check out their books. One patron might be prone to check out several books that are physically located next to each other on the shelves, but this is not a factor when shopping on-line.

#### Other Possible Concerns with Survey Verification

When considering only the surveys we gave, one flaw was that many of those surveyed were completely unfamiliar with the subject matter some of the rules contained. For example:

Advances in Microstrip and Printed Antennas (30)  
 $\Rightarrow$  Semi conducting Transparent Thin Films(27)

The subject matter of these two books are fundamentally related, and a physicist or a mechanical engineer might recognize this fact immediately. However, the rule received hardly any positive votes on the survey because the group member who passed it out only surveyed people with backgrounds in mathematics, computer science, computer engineering, elementary education, and dance. In addition, one-antecedent, one-consequent rules give little context with which a person being surveyed can make a solid judgment about a rule. For example, the following two rules were judged independently, and therefore received few positive votes:

Prime Numbers (32)  $\Rightarrow$  Linear Programming and Extensions(30)

Prime Numbers (32)  $\Rightarrow$  Primes and Programming(28)

Had the people surveyed been directed to consider these rules together, the second consequent clearly would have provided a logical bridge between the subject of Primes and the subject of Linear Programming.

need a machine more powerful than the machines we used, which only had 4 GB of RAM.

## 4 Conclusion

We believe that our project was a success according to our initial success criteria. We did not implement an enterprise-scale recommendation system, but we successfully carried out a proof of concept. We set a goal of receiving 80% positive responses on our surveys. We got a positive response rate of 86% (43 of 50), much to our delight. We did not originally think of comparing our results to Amazon.com, so we did not set a goal for that test. However, considering the fact that we did not use metadata in our recommendations (author, genre, patron's history, etc.), and considering the large number of books the HBLL circulates that Amazon.com does not, we were very pleased with our 36% success rate for that test. In other words, for the rules having an antecedent that appeared on Amazon.com with at least one recommendation, Amazon.com recommended the book that we recommended 36% of the time.

Ultimately, though Amazon.com is well known for its recommendation system, we decided that our survey validation test was more meaningful. Our surveys were filled out by humans (not algorithms) at BYU, and ultimately a human opinion matters more (since the books will be recommended to people).

## 5 Future Work

As one of our major constraints during this project was a lack of main memory, there are several things we can do in the future to help us better cope with spatial complexity. There is one other association mining algorithm and two other implementations of Apriori that we would like to try on our data. They are, respectively, FP-Tree based Association mining, Bayesian based Apriori, and a compact C implementation of Apriori done on binarized, attribute vectors where each book is represented concisely as one bit. Our hope is that these algorithms will allow us to mine multiple-antecedent rules more easily by taking less space and, in the case of Bayesian based Apriori and FP-Trees at least, less time. It would also be worthwhile to try to re-factor these algorithms to take advantage of multiple processors.

We believe that one of our greatest limitations in creating recommendations is that we based them solely on checkout history. Including richer metadata in the process might aid in generating better recommendations by allowing us to abstract our several hundred-thousand book space in to a smaller meta-space. Richer metadata might include: book abstracts, book subject headings, and some patron specific data such as gender, major, etc.

Mining association rules from the vast amount of data that we have is an enormously computationally intensive task, and as such we would also like to try our algorithms on a super-computer or a high-performance server of some kind. If the library is going to maintain a recommendation system, it will