Algorithm for Massive Dataset report
Finding Frequent Itemsets

Mubarak Babslawal

July 2025

1 Introduction

One of the early implementations of modern data science was in market basket analysis: i.e. which items are likely to be bought together. This analysis is done by identifying which items are commonly found in the same basket, and recommending one of them to users who purchase the other. There are several algorithms that have been developed to tackle this common problem. The earliest of them is the apriori algorithm, which others built on top. This paper will implement the a priori algorithm on the Amazon book review dataset evaluate how it performs on this dataset, and discuss how this algorithm scales to larger datasets.

2 Chosen Dataset

The chosen dataset for this experiment is the Amazon Book review dataset. The dataset is a pretty large one, about 2.55GB in data. It contains 300000 rows and 20 columns. However, the focus of the analysis is on two columns: User_id and Title. The data is summarized to capture the list of books reviewed by each User_id. The collection of books reviewed by each user is thus a basket.

3 Pre-processing

The following preprocessing steps were performed:

1. Grouping of data: The data was grouped into lists of books per user_id. (Paste code) 2. String cleaning: The baskets required a number of cleaning steps. This is because a number of books were really the same book but with slight differences in punctuation. To fix this, all book titles were converted to lower case, all punctuation marks were removed, and spaces were trimmed. This allowed for uniformity among book titles 3. Creation of transactions: To convert the list of baskets into transactions, distinct values were taken per basket. i.e., each basket contains any given book title only once. This list of transactions became the final dataset to be worked upon by the algorithm

4 Algorithms

4.1 A priori algorithm

The a priori algorithm has been described thoroughly by MMD. The implementation linked is simply a step-by-step implementation of what was described in the aforementioned book. The algorithm works based on the simple assumption that all items in a frequent pair must be frequent, and all pairs in a frequent triplet must also be frequent and so on. This logic easily scales to massive datasets because each pass of the algorithm reduces the number of candidates to examine. Hence this ensures that the runtime of the algorithm doesn't necessarily scale linearly with data size. Implementation The implementation of the algorithm using python 3 is pretty straightforward. It is divided into three main functions: 1. First pass: First a hash table was generated from unique items in all transactions, thus mapping each item to an integer. This integer is then used to generate the transactions object as a list of sets, where in each basket each item appears only once. The first pass then loads the transaction data into a single list and counts the number of times each item appears. Remember that each item appears only once per basket, so the number of appearances equals the number of different baskets that have the item. The dataframe is sorted from highest counts to lowest, making the hashes reusable in the next pass and avoiding the overhead of calculating new hashes from only the frequent single items.

2. Between passes: The between passes calculates the support threshold as a number of books based on the percentage defined as part of the function call. It then filters the dataframe using this threshold. The threshold is stored to be reused in the second pass, and the filtered dataframe gives the list of frequent singletons. As mentioned earlier, there is no need to generate new hashes from these due to the sort that has been applied. For example, if we assume that all the items have been assigned indexes from 1 to n. If the cutoff is at m, the filtered dataframe will be left with hashes from 1 to m, leaving us with no need to generate new hashes for the frequent singletons

3. Second pass: The second pass goes on to filter each basket to have only frequent singletons, then generates the pairs of singletons found in each. If there is no more than one singleton in a basket, the basket is removed from transactions. While it is possible to generate and count the pairs without this step, this step is useful as it prunes down the size of transactions for potential future passes, ensuring that only baskets containing at least pairs of frequent singletons will be considered. After counting the number of times each pair appears, these are evaluated against the support threshold, and the result of this filtering gives us frequent pairs.

PCY Algorithm This algorithm aims to reduce the memory requirements for the second pass by adding one more step between passes. It stores each pair of items into a hash bucket, taking less space than storing the count of each pairs. During the second pass, each pair is then checked up against the hash bucket, and if the bucket is found to be infrequent, then it's not possible for the pair

step	**a priori**	**pcy**
first pass	995ms	1.012s
between passes	30ms	43.8s
second pass	40.6s	38s

Table 1: Performance of a priori vs PCY

to be frequent even if it is made up of two frequent singletons. Under certain conditions, this can help reduce memory requirements during the second pass

Experiments Both the apriori algorithm and PCY algorithms were evaluated against the Amazon books dataset. Only frequent pairs were considered for this experiment, as beyond k=2, both algorithms are practically the same.

Using python's logging features, it was possible to identify how much time each step of the process took.

For this dataset, a priori performed better. This is because PCY performs particularly well when frequent items don't commonly appear together in the same bucket, thus where there is a high probability that a frequent singleton doesn't belong to a frequent pair. This isn't true for the amazon dataset, as frequent singletons tend to be similar books/sequels. So it is highly likely that they appear together in the same bucket. Due to this, creating the bitmap becomes an additional overhead for PCY, without the added benefits of reducing the number of items evaluated in the second pass, and thus no significant performance increase is observed by using the PCY compared to the a priori algorithm.

5 Declaration

"I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work, and including any code produced using generative AI systems. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study."