Market Basket Analysis

Algorithms for massive data

Mariasilvia Pancione - 32440A
mariasilvia.pancione@studenti.unimi.it

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.

Contents

1 Introduction

This project focuses on market-basket analysis with the goal of identifying frequent itemsets in a large-scale dataset of book reviews. Transactions are defined at user level, where each basket represents the set of books reviewed by a given user. The analysis is carried out by implementing and comparing two classical algorithms for frequent itemset mining, namely Apriori and SON, in a distributed computing environment using Apache Spark. The methodology includes data preprocessing, basket construction, frequent itemset extraction, and association rule mining, with particular attention to scalability and result consistency on large transactional datasets.

2 Data Pre-processing

The dataset used in this project is the *Amazon Books Review Dataset*, publicly available on Kaggle under the CC0 license. The dataset is downloaded at execution time through the Kaggle API, using authenticated access. It consists of two main files:

- `books_data.csv`, containing metadata about books (e.g., title, description, author, cover image);
- `Books_rating.csv`, containing information about user reviews, including book identifiers, user identifiers, review scores, and review texts.

For the purposes of this project, only the `Books_rating.csv` file is considered. This choice is motivated by the objective of identifying frequent sets of books reviewed by the same users, which requires user–book interaction data rather than book metadata. The original dataset contains 3,000,000 reviews.

Handling missing values and duplicates. As a first preprocessing step, only the columns relevant to market-basket analysis are retained, namely the user identifier, the book identifier, and the book title. Rows containing missing values in these fields are removed to ensure data consistency. In addition, duplicate reviews are eliminated by removing multiple occurrences of the same book reviewed by the same user. This step is particularly important to avoid counting the same item multiple times within a single basket, especially in cases where the same book appears under different identifiers.

After data cleaning, the dataset contains 2,114,559 rows, corresponding to 1,008,423 distinct users and 208,865 distinct books.

Subsampling To reduce the computational load during development and experimentation, a subsampling strategy is introduced through the configuration of a global variable. When this variable is set to `true`, a random subset of users is selected and only the reviews associated with these users are retained; when it is set to `false`, the full dataset is used without any reduction. The subsampling is performed at user level rather than at review level, in order to preserve the internal structure of each basket. Sampling is performed without replacement and controlled by a fixed random seed, ensuring reproducibility across different executions. In the experiments presented in this report, a sampling fraction of 5% is used when subsampling is active, resulting in a reduced dataset of 108,961 rows.

Basket construction. Finally, baskets are constructed by grouping the data by `User_Id` and aggregating the corresponding book identifiers, so that each basket represents the set of distinct books reviewed by a given user. A minimum basket size constraint is then applied, and all users associated with fewer than two distinct books are filtered out. This step removes trivial transactions (singletons) that do not contribute to the discovery of itemsets of size ≥ 2 and would otherwise introduce unnecessary computational overhead without providing meaningful co-occurrence information.

The resulting collection of baskets is then converted into an RDD (Resilient Distributed Dataset), Spark's core data abstraction for representing immutable and distributed collections that can be processed in parallel. Since the frequent itemset mining algorithms require multiple passes over the same baskets, the RDD is persisted in memory to avoid recomputing the basket construction pipeline at each action. The final number of baskets obtained after this step is 14,239.

3 Algorithms Implementation

For this project, two algorithms for frequent itemset mining were implemented: Apriori and SON.

3.1 Apriori Algorithm

The Apriori algorithm is a classical approach for frequent itemset mining based on the monotonicity property, which states that if an itemset is frequent, then all of its non-empty subsets must also be frequent. Conversely, if an itemset is infrequent, all of its supersets can be safely pruned. This property allows to reduce significantly the search space by iteratively generating and filtering candidate itemsets of increasing size.

In this project, Apriori is implemented in a distributed setting using Apache Spark, which allows computations to be parallelized across data partitions and enables the scalability of frequent itemset mining algorithms on large-scale transactional data. The algorithm proceeds in multiple passes over the data, where at each iteration k frequent itemsets of size k are identified. The process starts from frequent singletons ($k = 1$) and progressively considers itemsets of increasing size until no new frequent itemsets are found or a predefined maximum size is reached.

Frequent singleton discovery. In the first pass, all distinct items (books) appearing in the baskets are extracted and counted. Items whose support exceeds the given support threshold are retained as frequent 1-itemsets and stored in a dictionary for subsequent iterations.

Candidate generation and pruning. For $k \geq 2$, candidate k-itemsets are generated from the frequent $(k-1)$-itemsets identified in the previous iteration. For $k > 2$, candidate generation follows the Apriori join-and-prune strategy: two $(k-1)$-itemsets are joined if they share the same prefix of length $k-2$, and the resulting candidate is retained only if all of its $(k-1)$-subsets are frequent. This pruning step avoids generating candidates that cannot possibly be frequent.

For the case $k = 2$, candidate generation is optimized by directly enumerating all unordered pairs of frequent singletons appearing together within each basket, reducing the overhead of explicit candidate construction.

Support counting. Once the candidate set C_k is generated, their support is computed by scanning the basket RDD. To efficiently distribute the candidate information across workers, the candidate set is broadcast to all executors. Each basket is then checked for candidate containment, and partial counts are aggregated using a reduce operation. Only candidates whose support exceeds the specified threshold are retained as frequent itemsets.

Termination and output. The algorithm iterates until either no new frequent itemsets are found or a predefined maximum itemset size is reached. The final output is a collection of frequent itemsets grouped by their cardinality, along with their absolute support counts.

3.2 SON Algorithm

While Apriori is effective, its scalability is limited by the need to perform multiple full passes over the entire dataset and by the potentially large number of candidates to be counted. To address these issues, this project implements the SON algorithm (Savasere–Omiecinski–Navathe), a distributed frequent itemset mining technique designed for large-scale data.

The key idea of SON is to exploit data partitioning. Let the dataset be split into multiple partitions. In the *first phase*, each partition is treated as a sample of the full dataset and a local frequent itemset mining procedure is executed on each partition using a scaled support threshold. Any itemset that is globally frequent must be frequent in at least one partition when the threshold is scaled proportionally. Therefore, the union of all itemsets found frequent in at least one partition forms a set of *global candidates*. In the *second phase*, these candidates are counted on the full dataset to remove false positives and produce the final set of globally frequent itemsets.

Phase 1: local mining and candidate generation. The first phase of the SON algorithm corresponds to a Map–Reduce computation aimed at generating candidate itemsets. In the *Map* step, each mapper processes a single partition of the basket RDD using `mapPartitions`. Each partition is treated as an independent chunk (or sample) of the dataset, and a local Apriori algorithm is executed on the baskets contained in that partition up to a maximum itemset size `max_k`.

If a partition contains n baskets and the full dataset contains N baskets, the local support threshold is computed as:

$$s_{\text{local}} = \left\lceil s \cdot \frac{n}{N} \right\rceil,$$

where s is the global minimum support. This scaling ensures that any itemset that is globally frequent must appear as frequent in at least one partition, thus guaranteeing the absence of false negatives in the candidate generation step.

Each mapper outputs the locally frequent itemsets found in its partition as key-value pairs. In the *Reduce* step, these itemsets are simply deduplicated by key, ignoring their associated values. The result of the first Map–Reduce phase is the union of all itemsets that are frequent in at least one partition, which constitutes the global candidate set.

Phase 2: global counting of candidates. The second phase performs another Map–Reduce computation to validate the candidate itemsets generated in Phase 1. Before the Map step, the candidate itemsets are grouped by cardinality and broadcast to all workers to avoid repeated data transfers.

In the *Map* step, each mapper scans its assigned partition of baskets and counts how many times each candidate itemset appears in that partition. This produces partial support counts for each candidate. In the *Reduce* step, partial counts corresponding to the same candidate itemset are summed across all partitions to obtain the global support.

Finally, only candidate itemsets whose global support is at least the minimum support threshold s are retained. This step removes all false positives introduced in Phase 1, yielding the final set of globally frequent itemsets.

Output. The SON implementation outputs an RDD containing the frequent itemsets along with their absolute support counts. By decomposing the computation into two Map–Reduce phases, SON reduces the candidate search space and improves scalability compared to a direct application of Apriori on the full dataset.

4 Experimental Results

4.1 Frequent Itemsets

The experiments were conducted by applying both the Apriori and the SON algorithms with a maximum itemset size of $k = 3$ and using a support threshold equal to 0.3% of the total number of baskets.

Both algorithms successfully identified frequent itemsets of size 1, 2, and 3. Table 1 reports the top–5 frequent itemsets for each cardinality k, together with their absolute support counts, as obtained by Apriori and SON.

The results obtained with Apriori and SON are identical for all itemset sizes and support values. This outcome is consistent with the theoretical guarantees of the SON algorithm: while Phase 1 may generate additional candidates, Phase 2 removes all false positives through global counting, yielding exactly the same frequent itemsets that would be found by Apriori applied to the full dataset.

An interesting observation is that several singleton, pair, and triplet itemsets exhibit identical or nearly identical support values. This indicates that certain items consistently appear together in the same set of baskets. Such behavior may occur when items correspond to books belonging to the same series, closely related editions, or alternative catalog entries referring to the same underlying product.

To better understand this behavior, the titles associated with the most frequent itemsets were inspected. Table 2 reports the book identifiers and titles corresponding to the two most frequent triplets.

The analysis reveals that these identifiers correspond to different IDs of the same book, differing only in punctuation, spelling, or subtitle formatting. As a consequence, users who reviewed *The Hobbit* may

k	Apriori Itemset	Support	SON Itemset	Support
1	(B000ILIJE0)	173	(B000ILIJE0)	173
1	(B000Q032UY)	173	(B000Q032UY)	173
1	(B000NWU3I4)	171	(B000NWU3I4)	171
1	(B000NDSX6C)	161	(B000NDSX6C)	161
1	(B000GQG5MA)	158	(B000GQG5MA)	158
2	(B000ILIJE0, B000Q032UY)	173	(B000ILIJE0, B000Q032UY)	173
2	(B000ILIJE0, B000NWU3I4)	171	(B000ILIJE0, B000NWU3I4)	171
2	(B000NWU3I4, B000Q032UY)	171	(B000NWU3I4, B000Q032UY)	171
2	(B000NDSX6C, B000Q032UY)	161	(B000NDSX6C, B000Q032UY)	161
2	(B000ILIJE0, B000NDSX6C)	161	(B000ILIJE0, B000NDSX6C)	161
3	(B000ILIJE0, B000NWU3I4, B000Q032UY)	171	(B000ILIJE0, B000NWU3I4, B000Q032UY)	171
3	(B000ILIJE0, B000NDSX6C, B000Q032UY)	161	(B000ILIJE0, B000NDSX6C, B000Q032UY)	161
3	(B000ILIJE0, B000NDSX6C, B000NWU3I4)	159	(B000ILIJE0, B000NDSX6C, B000NWU3I4)	159
3	(B000NDSX6C, B000NWU3I4, B000Q032UY)	159	(B000NDSX6C, B000NWU3I4, B000Q032UY)	159
3	(B000GQG5MA, B000ILIJE0, B000Q032UY)	158	(B000GQG5MA, B000ILIJE0, B000Q032UY)	158

Table 1: Comparison of the top–5 frequent itemsets for each cardinality k obtained with Apriori and SON.

Book ID	Title
B000ILIJE0	The Hobbit There and Back Again
B000NDSX6C	The Hobbit
B000NWU3I4	The Hobbitt, or there and back again; illustrated by the author
B000Q032UY	The Hobbit or There and Back Again

Table 2: Different catalog entries referring to the same book.

appear to have reviewed multiple distinct items, even though they refer to the same underlying title. This leads to artificially strong co-occurrence patterns and identical support values across singletons, pairs, and triplets.

Although the preprocessing step removes exact duplicates based on user identifier and title, it does not resolve near-duplicate or aliased representations of the same book. Consequently, some frequent itemsets reflect inconsistencies in the catalog rather than genuine co-reviewing behavior across distinct books.

4.2 Association Rules

Once frequent itemsets have been identified, they can be used to extract *association rules*, which provide a more interpretable representation of co-occurrence patterns in transactional data. Association rules describe directional relationships between items and are commonly expressed in the form

$$A \Rightarrow B,$$

where A and B are disjoint itemsets. In the context of market-basket analysis, such rules highlight dependencies between items and can be used to infer which items tend to appear together within the same transaction.

The strength of an association rule is evaluated using the following measures:

- **Support**:
$$\text{support}(A \Rightarrow B) = \frac{\text{count}(A \cup B)}{N},$$
where N is the total number of baskets. Support measures how frequently the itemset $A \cup B$ appears in the dataset.

- **Confidence**:
$$\text{confidence}(A \Rightarrow B) = \frac{\text{support}(A \cup B)}{\text{support}(A)},$$
which represents the conditional probability of observing B in a basket given that A is present.

- **Lift**:

$$\text{lift}(A \Rightarrow B) = \frac{\text{confidence}(A \Rightarrow B)}{\text{support}(B)},$$

which measures how much more likely A and B are to appear together compared to the case in which they are statistically independent.

Association rules were generated from the frequent pairs obtained using both Apriori and SON. Tables 3 and 4 report the top–10 rules ranked by lift.

Item A	Item B	Support	Conf($A \Rightarrow B$)	Conf($B \Rightarrow A$)	Lift
0451519329	1582790485	0.00302	1.000	1.000	331.14
0451519329	B0008588BU	0.00302	1.000	1.000	331.14
0451519329	B000FC1R6O	0.00302	1.000	1.000	331.14
1582790485	158726398X	0.00302	1.000	1.000	331.14
158726398X	B0008588BU	0.00302	1.000	1.000	331.14
158726398X	B000FC1R6O	0.00302	1.000	1.000	331.14
0451519329	158726398X	0.00302	1.000	1.000	331.14
1582790485	B0008588BU	0.00302	1.000	1.000	331.14
1582790485	B000FC1R6O	0.00302	1.000	1.000	331.14
B0008588BU	B000FC1R6O	0.00302	1.000	1.000	331.14

Table 3: Top–10 association rules derived from frequent pairs using Apriori.

Item A	Item B	Support	Conf($A \Rightarrow B$)	Conf($B \Rightarrow A$)	Lift
0451519329	1582790485	0.00302	1.000	1.000	331.14
0451519329	B000FC1R6O	0.00302	1.000	1.000	331.14
1582790485	158726398X	0.00302	1.000	1.000	331.14
0451519329	B0008588BU	0.00302	1.000	1.000	331.14
158726398X	B000FC1R6O	0.00302	1.000	1.000	331.14
158726398X	B0008588BU	0.00302	1.000	1.000	331.14
0451519329	158726398X	0.00302	1.000	1.000	331.14
1582790485	B000FC1R6O	0.00302	1.000	1.000	331.14
1582790485	B0008588BU	0.00302	1.000	1.000	331.14
B0008588BU	B000FC1R6O	0.00302	1.000	1.000	331.14

Table 4: Top–10 association rules derived from frequent pairs using SON.

The association rules obtained using Apriori and SON are identical in terms of support, confidence, and lift, since both algorithms identify exactly the same frequent itemsets with identical support values.

Most of the extracted rules exhibit confidence values equal (or very close) to 1 and extremely high lift values. A confidence of 1 indicates that whenever the antecedent item appears in a basket, the consequent item is always present as well. Similarly, very large lift values indicate a strong positive dependency between the items, meaning that their co-occurrence is far more frequent than would be expected if the items were statistically independent.

However, these strong associations must be interpreted with caution. As discussed in the previous subsection, several frequent itemsets correspond to different catalog entries or various editions of the same book, or to multiple volumes belonging to the same book series. As a consequence, two items may co-occur frequently not because they represent distinct, independently reviewed books, but because they are alternative identifiers of the same product or closely related titles within the same saga. In such cases, association rules do not capture genuine co-reviewing patterns between unrelated books, but rather reflect catalog aliasing, duplication effects, or systematic co-occurrence within a series. This leads to artificially high confidence and lift values, since the presence of one identifier almost deterministically implies the presence of the others.

5 Conclusions and Limitations

This project applied market-basket analysis to a large dataset of book reviews by implementing the Apriori and SON algorithms. The results show that Apriori and SON identify the same frequent itemsets and derive identical association rules, confirming the correctness of both implementations and the theoretical guarantees of SON as a scalable alternative to Apriori. Strong co-occurrence patterns were observed among subset of books, reflected by high support values and association rules with very high confidence and lift. However, several limitations must be considered when interpreting these results. A closer inspection revealed that many frequent itemsets correspond to different catalog entries, editions, or volumes belonging to the same book series. As a consequence, some frequent itemsets do not capture meaningful co-occurrence patterns between distinct books. Although exact duplicates were removed during preprocessing, near-duplicate titles were not resolved. Addressing this limitation would require additional entity resolution techniques to merge multiple identifiers referring to the same underlying item.