Paola Loi (33088A)
Loris Palmarin (32349A)

Algorithms For Massive Datasets

Market Basket Analysis

Amazon Book Reviews

March 2025

Contents

1 Introduction

This project has been developed for the *Algorithms for Massive Datasets* course, taught by Professor Malchiodi during the 2024/25 academic year. The objective is to implement a market-basket analysis using the Amazon Books Reviews dataset. The goal is to extract insights into book co-purchasing patterns by identifying frequent itemsets and generating association rules.

To achieve this, we deploy two well-established algorithms for frequent itemset mining:

- *A-Priori*, a two-pass approach designed to efficiently find frequent itemsets while reducing memory usage.
- *SON (Savasere, Omiecinski, and Navathe)*, a MapReduce-based algorithm designed for distributed environments, which finds frequent itemsets in chunks of data before validating them globally.

The dataset consists of book reviews, where each user represents a basket, and the books they reviewed form the items. The dataset requires preprocessing to remove inconsistencies and ensure proper structuring for association rule mining. The analysis will focus on book pairs that frequently appear together, measuring their support, confidence, and lift to determine meaningful associations.

The implementation is carried out using Apache Spark to handle large-scale data efficiently, leveraging distributed computing for scalability. The two algorithms will be compared in terms of efficiency and scalability, evaluating their performance as the dataset size increases.

In the subsequent sections, we will discuss the preprocessing steps, algorithm implementation details, experimental results, and a critical analysis of the findings.

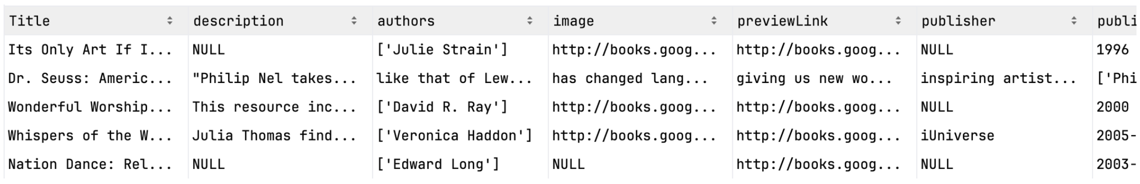
2 Data

2.1 Data Upload

The dataset used in this project is automatically downloaded at runtime via the Kaggle API. Two files are retrieved:

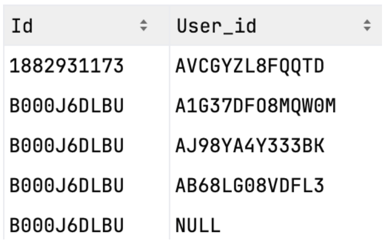
- `books_data.csv`: contains metadata about the books, including title, author, edition, publication year, cover image, and more.
- `books_ratings.csv`: contains user reviews, including `user_id`, `book_id`, numerical ratings, and review texts.

The dataset is based on Amazon book reviews. Once loaded, the tables appear as follows:



Title	description	authors	image	previewLink	publisher	publi
Its Only Art If I...	NULL	['Julie Strain']	http://books.goog...	http://books.goog...	NULL	1996
Dr. Seuss: Americ...	"Philip Nel takes...	like that of Lew...	has changed lang...	giving us new wo...	inspiring artist...	['Phi
Wonderful Worship...	This resource inc...	['David R. Ray']	http://books.goog...	http://books.goog...	NULL	2000
Whispers of the W...	Julia Thomas find...	['Veronica Haddon']	http://books.goog...	http://books.goog...	iUniverse	2005-
Nation Dance: Rel...	NULL	['Edward Long']	NULL	http://books.goog...	NULL	2003-

Figure 1: Preview of books_data.csv

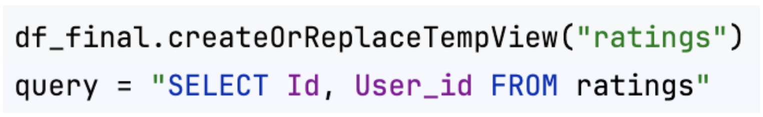


Id	User_id
1882931173	AVCGYZL8FQQTD
B000J6DLBU	A1G37DF08MQW0M
B000J6DLBU	AJ98YA4Y333BK
B000J6DLBU	AB68LG08VDFL3
B000J6DLBU	NULL

Figure 2: Preview of books_ratings.csv

2.2 Data Preprocessing

The tables are loaded into Spark as DataFrames via the SQLContext interface. A SQL query is used to select only the relevant columns for market basket analysis—namely, `user_id` and `book_id` from the `books_ratings` table.



```
df_final.createOrReplaceTempView("ratings")
query = "SELECT Id, User_id FROM ratings"
```

Figure 3: Column selection query

The full dataset contains approximately 3 million reviews. To reduce computational load during development, a variable `sampling` is introduced: when set to `True`, it enables working with just 1% of the dataset for testing purposes.

The pre-processing phase also includes standard data cleaning steps:

- Removal of duplicate entries
- Elimination of null values
- Merging of books with the same title but different IDs

After preprocessing, the dataset is reduced to approximately 2 million cleaned and usable reviews.

3 Algorithms Implementation

For the purposes of this project, I implemented two different algorithms to find frequent itemsets: A-Priori and SON.

3.1 A-Priori Algorithm

3.1.1 In theory

The A-Priori algorithm is a classical method for discovering frequent itemsets in transactional datasets, particularly suitable when the number of possible item combinations is too large to be held entirely in memory. A-Priori proceeds in two sequential passes over the dataset to limit the number of candidate itemsets that need to be counted.

First Pass. In the first pass, the algorithm focuses on identifying frequent singleton items:

1. Each item is optionally mapped to a unique integer ID.
2. An array of counts is initialized, where each position corresponds to a specific item.
3. As the dataset is scanned, each item's count is incremented whenever it appears in a basket.

At the end of this phase, only items whose counts exceed the support threshold s are retained as *frequent singleton items*.

Between Passes. A new mapping is created that assigns new consecutive integer IDs (from 1 to m) exclusively to the frequent items identified during the first pass. This enables the algorithm to reduce the memory footprint by focusing only on the relevant items.

Second Pass. During the second pass, the algorithm counts the occurrences of item pairs, but only those composed of items that are already known to be frequent:

1. For each basket, the frequent-items mapping is used to filter out infrequent items.
2. A nested loop is used to generate all possible unordered pairs of the remaining items in the basket.
3. Each pair is counted using a suitable data structure (e.g., triangular matrix or hash-based table).

At the end of this second pass, the support counts of all candidate pairs are evaluated, and only those that satisfy the minimum support threshold s are retained as frequent pairs.

3.1.2 Implementation

The A-Priori algorithm has been implemented using the `PySpark` package in order to leverage distributed computing and parallelization. Given the size of the dataset and the computational complexity of frequent itemset mining, the use of Spark allows for an efficient and scalable approach to processing large volumes of data.

To begin, the data has been organized into an RDD named `book_baskets`. Using PySpark's `map` and `groupByKey` functions, the ratings have been transformed so that each entry in the RDD corresponds to a single user and a list of all book identifiers they have rated. This list of books for each user represents a *basket* in the context of market basket analysis.

Once the baskets were created, we computed the total number of baskets, which resulted in `1,008,423` entries. This constitutes the dataset over which the A-Priori algorithm is applied in the subsequent steps.

First Pass: Frequent Singletons The first pass of the A-Priori algorithm aims to identify the frequent singleton itemsets, i.e., individual books that appear frequently across user baskets. To accomplish this, we define an RDD named `single_counts`, which is derived from `book_baskets` using PySpark's `flatMap` and `reduceByKey` transformations.

Initially, each book within every basket is isolated and mapped to a key-value pair `(book_id, 1)`. Then, using the `reduceByKey` function, the algorithm aggregates these pairs by summing up the occurrences of each book across all baskets, effectively counting how many times each book appears in the dataset.

To determine which books are frequent, we introduce a support threshold. This threshold is computed as the product between a predefined percentage of the total baskets and the total number of baskets itself. In this implementation, we chose a threshold of 0.001, i.e., books that appear in at least 0.1% of the baskets are considered frequent.

After applying this filtering step, 81 frequent singleton books were identified in the complete dataset. These were then sorted in descending order of frequency. Notably, the most frequent book appeared in more than 3,000 baskets.

Second Pass: Frequent Pairs The second pass replicates the structure of the first pass but focuses on identifying frequent *pairs* of items instead of singletons. To ensure computational efficiency, only the frequent singleton items obtained from the first pass are considered when forming pairs. This step is crucial since, according to the A-Priori principle, a pair can only be frequent if both of its items are themselves frequent.

Using `flatMap`, all possible unordered pairs of frequent singleton books within each basket are generated. These pairs are mapped to key-value pairs of the form `((book_i, book_j), 1)` and then aggregated via `reduceByKey` to count their occurrences across all baskets.

The resulting counts are filtered using the same support threshold adopted for the singletons (0.001 of the total number of baskets), yielding 84 frequent pairs. As in the first pass, the final list is sorted in descending order by frequency. The most frequent pair appears 3,574 times in the dataset.

Third Pass: Frequent Triplets In the third pass, the algorithm extends the previous procedure to compute frequent *triplets* of books. Again, combinations are generated exclusively from the frequent singleton items, applying the `flatMap` and `reduceByKey` operations analogously to the previous passes.

The same support threshold is applied to identify frequent triplets. After filtering and sorting, a total of 111 frequent triplets were found. The most common triplet appears 3,561 times in the dataset.

The results will be used in the last section when we will compute the association rules to identify patterns and purchase behavior.

3.2 SON Algorithm

3.2.1 In theory

The SON (Savasere, Omiecinski, Navathe) algorithm is a two-pass, distributed approach to finding frequent itemsets in large-scale datasets. It is designed to work efficiently in parallel computing environments and eliminates both false positives and false negatives by structuring the process into two key MapReduce phases.

In the first pass, the input dataset is divided into smaller chunks (e.g., via partitions in a distributed file system). For each chunk, a local version of the frequent itemset mining algorithm is executed using a lowered support threshold ps, where p is the fraction of data represented by the chunk and s is the global support threshold. This phase yields locally frequent itemsets which are considered as candidate itemsets.

In the second pass, the global support for these candidate itemsets is computed by scanning the entire dataset. Only the itemsets that meet or exceed the original support threshold s across the full dataset are retained as globally frequent itemsets.

This two-pass structure ensures that:

- All globally frequent itemsets are discovered (no false negatives).
- The process can be parallelized, improving computational efficiency on large datasets.

3.2.2 Implementation (Map-Reduce version)

The SON algorithm was implemented using the `PySpark` framework in order to exploit distributed computing and achieve efficient processing of large-scale datasets. The overall procedure follows the canonical structure of the algorithm and is divided into two MapReduce phases, each executed across multiple data partitions.

First Phase – Local Frequent Itemsets In the first phase, the dataset containing user baskets was divided into 10 partitions. Each partition was treated as an independent sample of the global dataset. A local support threshold was computed as:

$$s_{\text{local}} = s \cdot p$$

where s is the global support threshold, and p is the fraction of the dataset in a single partition.

Within each partition, the algorithm:

- Counted the occurrences of singleton and pair itemsets.
- Selected locally frequent itemsets, i.e., those whose count exceeded s_{local}.
- Returned these itemsets as candidates for the second phase.

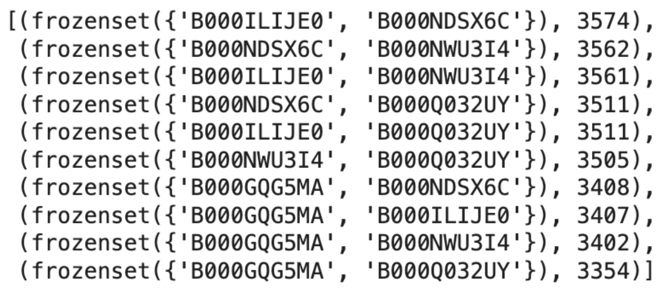
At the end of the first MapReduce phase, the set of global candidate itemsets was obtained by computing the union of all locally frequent itemsets. This guarantees that all globally frequent itemsets are preserved, with no false negatives.

Second Phase – Global Support Counting In the second phase, the algorithm measured the actual support of each candidate itemset across the full dataset. Each partition contributed to the total count of each candidate.

The second phase involved:

- Scanning each basket to check whether it contains each candidate itemset.
- Aggregating partial counts across partitions.
- Filtering itemsets whose total count exceeded or equaled the global support threshold s.

The output of this phase is the final set of globally frequent itemsets.



```
[(frozenset({'B000ILIJE0', 'B000NDSX6C'}), 3574),
 (frozenset({'B000NDSX6C', 'B000NWU3I4'}), 3562),
 (frozenset({'B000ILIJE0', 'B000NWU3I4'}), 3561),
 (frozenset({'B000NDSX6C', 'B000Q032UY'}), 3511),
 (frozenset({'B000ILIJE0', 'B000Q032UY'}), 3511),
 (frozenset({'B000NWU3I4', 'B000Q032UY'}), 3505),
 (frozenset({'B000GQG5MA', 'B000NDSX6C'}), 3408),
 (frozenset({'B000GQG5MA', 'B000ILIJE0'}), 3407),
 (frozenset({'B000GQG5MA', 'B000NWU3I4'}), 3402),
 (frozenset({'B000GQG5MA', 'B000Q032UY'}), 3354)]
```

Figure 4: Frequent pairs

4 Association Rules

Association Rules

Association rules are a fundamental tool in market-basket analysis, used to uncover relationships and dependencies between items in large transactional datasets. Their primary purpose is to identify patterns in purchasing behavior, enabling insights into how the presence of certain items in a basket affects the likelihood of other items being present.

In this project, the association rules are derived from the output of the Apriori algorithm, applied frequent item pairs. By analyzing these frequent itemsets, we can generate rules of the form $A \Rightarrow B$, where A and B are itemsets, and evaluate their strength based on the following key metrics:

- **Support:** Measures how frequently an itemset appears in the dataset.

$$\text{Support}(A) = \frac{\text{Count}(A)}{\text{Total Transactions}}$$

- **Confidence:** Represents the conditional probability that itemset B appears in a transaction given that A is also present.

$$\text{Confidence}(A \Rightarrow B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)}$$

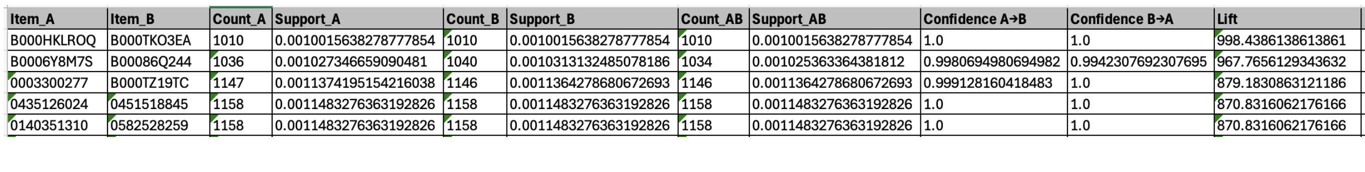
- **Lift:** Indicates how much more likely itemset B is to appear with A compared to if B were independent of A.

$$\text{Lift}(A \Rightarrow B) = \frac{\text{Confidence}(A \Rightarrow B)}{\text{Support}(B)}$$

These metrics help identify the most meaningful and non-trivial associations in the data. Association rules with high confidence and lift are especially useful for practical applications such as recommendation systems and targeted marketing strategies.

4.1 From pairs

After computing the metrics described above, this is the final output (only first 5 rules selected:



Item_A	Item_B	Count_A	Support_A	Count_B	Support_B	Count_AB	Support_AB	Confidence A→B	Confidence B→A	Lift
B000HKLROQ	B000TKO3EA	1010	0.0010015638278777854	1010	0.0010015638278777854	1010	0.0010015638278777854	1.0	1.0	998.4386138613861
B0006Y8M7S	B00086Q244	1036	0.001027346659090481	1040	0.0010313132485078186	1034	0.001025363364381812	0.9980694980694982	0.9942307692307695	967.7656129343632
0003300277	B000TZ19TC	1147	0.0011374195154216038	1146	0.0011364278680672693	1146	0.0011364278680672693	0.999128160418483	1.0	879.1830863121186
0435126024	0451518845	1158	0.0011483276363192826	1158	0.0011483276363192826	1158	0.0011483276363192826	1.0	1.0	870.8316062176166
0140351310	0582528259	1158	0.0011483276363192826	1158	0.0011483276363192826	1158	0.0011483276363192826	1.0	1.0	870.8316062176166

Figure 5: Column selection query

We analyze the top three association rules derived from the most frequent item pairs identified through the *Apriori* algorithm. These rules demonstrate very strong dependencies between items, as indicated by their high confidence and lift values.

Rule 1: `B000HKLROQ` ⇒ `B000TKO3EA`

- **Support**: 0.0010015
- **Confidence (A → B)**: 1.0
- **Confidence (B → A)**: 1.0
- **Lift**: 998.44

This rule is perfectly symmetric and deterministic — whenever `B000HKLROQ` appears in a transaction, so does `B000TKO3EA`, and vice versa. The lift of nearly 1000 confirms an extremely strong association, likely indicating complementary or co-bought items.

Rule 2: `B0006Y8M7S` ⇒ `B00086Q244`

- **Support**: 0.001025
- **Confidence (A → B)**: 0.9981
- **Confidence (B → A)**: 0.9942
- **Lift**: 967.77

This rule is nearly deterministic, with both directions showing very high confidence. The slightly lower values compared to Rule 1 still reflect a strong connection. A lift of over 960 suggests a very strong co-occurrence relationship.

Rule 3: `0003300277` ⇒ `B000TZ19TC`

- **Support**: 0.001136
- **Confidence (A → B)**: 0.9991
- **Confidence (B → A)**: 1.0
- **Lift**: 879.18

This rule also indicates a strong association. The asymmetry in confidence values suggests that `B000TZ19TC` appears every time `0003300277` does, but not necessarily vice versa. Nonetheless, the high lift confirms a non-random pattern.

All three rules demonstrate:

- High **support**, indicating the pairs are relatively frequent in the dataset.
- Near-perfect **confidence**, meaning strong directional associations.
- Extremely high **lift**, validating these associations as useful for recommendations and insights.

5 Conclusions and Limitations

In this project, we successfully implemented both the Apriori and SON algorithms using PySpark to perform frequent itemset mining on a large-scale dataset of book reviews. Despite following different computational strategies—Apriori relying on multiple well-structured passes over the data, and SON leveraging distributed computing and local thresholds—both algorithms produced consistent results in terms of identified frequent itemsets. This consistency reinforces the correctness and reliability of the implementations.

5.1 Limitations

While the association rules extracted are statistically very strong, as shown by metrics such as support, confidence, and lift, one significant limitation lies in the quality of the raw data. Although extensive preprocessing was carried out—including converting all book titles to lowercase and merging identical titles under a unique identifier—some discrepancies still remain.

These inconsistencies are mainly due to:

- Slight textual variations in book descriptions (e.g., presence of edition notes or formatting differences).
- Typographical errors or non-standard characters.
- Metadata that causes otherwise identical books to be treated as different items.

As a consequence, many of the strongest association rules discovered—including the top three analyzed in the previous section—do not actually capture associations between different books. Instead, they link together multiple representations of the same book.

Here are a few illustrative examples from the top association rules:

- `‘middlesex’` ⇔ `‘middlesex [unabridged audiobook]’`
- `‘how to win friends & influence people’` ⇔ `‘how to win friends & influence people (cardinal editions, c 303)’`
- `‘fahrenheit 451 (cascades s.)’` ⇔ `‘fahrenheit 451’`

These cases demonstrate that although the association metrics are high, the semantic interpretation of the rules may be limited. Future work could involve more advanced text normalization techniques, such as fuzzy matching or natural language processing-based clustering, to better unify equivalent items before analysis.

5.2 Final remarks

Despite these limitations, the project effectively demonstrates how distributed computing frameworks like Spark can be leveraged to extract actionable insights from large-scale data through classic frequent pattern mining algorithms. The results can serve as a solid foundation for recommendation systems and further analytical developments in the domain of book consumption behavior.