Algorithms for Data Science: Frequent Itemset Mining & Apriori Algorithm

Pablo Mollá Chárlez

October 20, 2024

Contents

1 Frequent Itemset Mining

1.1 Introduction to the Market Basket Model

The Market Basket Model is a fundamental concept in data mining and analysis, particularly within the realm of retail and e-commerce. It is used to discover patterns or associations between items that frequently appear together in transactions or "baskets."

The idea is that by examining the contents of customer's shopping baskets, businesses can identify which products are often purchased together and use this information for various purposes such as inventory management, marketing strategies, and personalized recommendations.

This model is based on the concept of frequent itemsets and association rules, which help identify and measure the relationships between different items.

1.2 Frequent Itemsets

A frequent itemset is a set of items that appear together in a large number of baskets or transactions. In mathematical terms:

Let $\mathcal{I}$ be the set of all items, and $\mathcal{B}$ be the set of all baskets.
An itemset I is any subset of $\mathcal{I}$ and a basket B is a subset of $\mathcal{I}$.

The support of an itemset $\mathcal{I}$ is defined as the number of baskets that contain all the items in $\mathcal{I}$. In other words:

$$Supp(I) = |\{B \in \mathcal{B} \mid \mathcal{I} \subseteq B\}|$$

An itemset I is considered frequent if its support is greater or equal to a specified threshold. Frequent itemsets help identify combinations of products that are often bought together.

Frequent itemsets satisfy a property called monotonicity, meaning that when an itemset is frequent, then so is every subset of it

1.2.1 Example

A store has the following items and the transations (baskets):

$$\mathcal{I} = \{Bread, Milk, Diaper, Coke, Eggs, Beer\}$$
Basket 1: {Bread, Milk}
Basket 2: {Bread, Diaper, Beer, Eggs}
Basket 3: {Milk, Diaper, Beer, Coke}
Basket 4: {Bread, Milk, Diaper, Beer}
Basket 5: {Bread, Milk, Diaper, Coke}

Let's say the threshold for a frequent itemset is set at 3. The itemset {Bread, Milk} appears in Baskets 1, 4, and 5, so its support is 3. Thus, {Bread, Milk} is a frequent itemset. As it can be seen, {Milk} and {Bread} are as well frequent itemsets due to monotonicity.

1.3 Association Rules

Association rules describe correlations or patterns in the contents of baskets. An association rule is typically written in the form:

$$\{i_1, i_2, ..., i_n\} \longrightarrow j$$

This rule suggests that if a basket contains items $\{i_1, i_2, ..., i_n\}$, it is likely to contain item j as well. There can be many possible rules, but we are generally interested in those that are considered "interesting".

An association rule is considered interesting if it meets a certain level of confidence. The confidence of an association rule measures the likelihood that item j is present in a basket, given that the basket already contains items $\{i_1, i_2, ..., i_n\}$. It is calculated as:

$$conf(\{i_1, i_2, ..., i_n\} \longrightarrow j) = \frac{Supp(\{i_1, i_2, ..., i_n, j\})}{Supp(\{i_1, i_2, ..., i_n\})}$$

The interest of an association rule is determined as follows:

$$interest(\{i_1, i_2, ..., i_n\} \longrightarrow j) = conf(\{i_1, i_2, ..., i_n\} \longrightarrow j) - \frac{Supp(\{j\})}{|\mathcal{B}|}$$

Generally, if the interest produces a low negative value, then it means there is a negative correlation between items, meaning that the presence of one item in a basket decreases the likelihood of the other. The interest value, ideally, should be either a high positive value, either a low negative value.

For example, if you find that customers who buy "Vegetarian Cookbooks" rarely buy "Steak", this can provide valuable insights into customer behavior, product placement or marketing strategy.

1.3.1 Example

Continuing with the previous example, consider the rule:

$$\{Milk, Diaper\} \longrightarrow Beer$$

To calculate the confidence of the rule, we need to apply the formula:

$$\text{conf}(\{Milk, Diaper\} \longrightarrow Beer) = \frac{Supp(\{Milk,Diaper,Beer\})}{Supp(\{Milk,Diaper\})} = \frac{2}{3} \approx 0.67$$

This means that in about 67% of the baskets containing Milk and Diaper, Beer is also present. The interest of the rule would be:

$$interest(\{Bread, Milk\} \longrightarrow Beer) = conf(\{Bread, Milk\} \longrightarrow Beer) - \frac{Supp(\{Beer\})}{|\mathcal{B}|} = \frac{2}{3} - \frac{2}{5} = \frac{4}{15}$$

1.4 Exercise

Let's consider the following set of all itemsets $\mathcal{I} = \{m, c, p, b, j\}$ and baskets:

Basket 1: {m, c, b}
Basket 2: {m, p, j}
Basket 3: {m, b}
Basket 4: {c, j}
Basket 5: {m, p, b}
Basket 6: {m, c, b, j}
Basket 7: {c, b, j}
Basket 8: {b, j}

The goal is to **determine all the association rules with a confidence above** $c = 0.75$ and we consider an itemset to be frequent when its support is greater or equal to $s = 3$.

1. **Determine all the frequent itemsets**: Proceed in a clever way, use the property of monotonicity to ease the verifications. Start with triplets, if you find one you will know for sure that every subset of it is a frequent itemset as well.

   - Single items: {m}, {c}, {b}, {j}
   - Pairs: {m, b}, {c, b}, {c, j}, {b, j}
   - Triplet: None

2. **Determine all association rules given the frequent itemsets**: Single items don't have association rules. For every frequent itemset I, generate rules of the form $A \longrightarrow (I - A)$ where $A \subset I, A \neq \emptyset$.

   - From {m, b}: $\text{conf}(m \longrightarrow b) = \frac{Supp(\{m,b\})}{Supp(\{m\})} = \frac{4}{5} \approx 0.8$ ✓
   - From {b, m}: $\text{conf}(b \longrightarrow m) = \frac{Supp(\{b,m\})}{Supp(\{b\})} = \frac{4}{6} \approx 0.67$ ✗
   - From {c, b}: $\text{conf}(c \longrightarrow b) = \frac{Supp(\{c,b\})}{Supp(\{c\})} = \frac{3}{4} = 0.75$ ✓
   - From {b, c}: $\text{conf}(b \longrightarrow c) = \frac{Supp(\{b,c\})}{Supp(\{b\})} = \frac{4}{6} \approx 0.67$ ✗
   - From {c, j}: $\text{conf}(c \longrightarrow j) = \frac{Supp(\{c,j\})}{Supp(\{c\})} = \frac{3}{4} = 0.75$ ✓
   - From {j, c}: $\text{conf}(j \longrightarrow c) = \frac{Supp(\{j,c\})}{Supp(\{j\})} = \frac{3}{5} = 0.6$ ✗
   - From {b, j}: $\text{conf}(b \longrightarrow j) = \frac{Supp(\{b,j\})}{Supp(\{b\})} = \frac{3}{6} = 0.5$ ✗
   - From {j, b}: $\text{conf}(j \longrightarrow b) = \frac{Supp(\{j,b\})}{Supp(\{j\})} = \frac{3}{5} = 0.6$ ✗

3. **Filter the association rules with tue suitable confidence**:

   The association rules greater or equal to the given confidence are: $\boxed{m \longrightarrow b}$, $\boxed{c \longrightarrow b}$ and $\boxed{c \longrightarrow j}$.

1.5 Computational Model & Cost

Generally, the data storage is kept in a disk file, processed basket by basket, and usually, the data does not fit in the main memory, necessitating the use of disk storage.The cost is measured in terms of the number of disk accesses required, and, since disk access is much slower than memory access, minimizing the number of accesses is crucial.

The data is read in manageable batches to fit into the main memory, and subsets (as single items, pairs triples, etc) are checked while in-memory:

- For pairs of items, this is computationally feasible using **nested-loop processing**, with a complexity of $O(n^2)$.
- For larger sets (more than two items), it becomes infeasible with a complexity of $O(\frac{n^k}{k!})$.

In practice, frequent itemsets are mostly limited to pairs or triples due to the computational constraints.

1.5.1 Example: Counting Pairs vs. Larger Sets

Let's explore an example to illustrate why counting pairs of items is computationally feasible, while counting larger sets (more than two items) becomes infeasible.

Suppose you have a dataset of $1,000,000$ baskets, each containing a varying number of items. Let's assume there are 1,000 different unique items (e.g., products in a supermarket).

1. **Counting Pairs**: To find frequent pairs of items (i.e., two items that appear together in a significant number of baskets), we need to check all possible combinations of two items from the set of 1,000 unique items.

   - **Total Pairs to Consider**: The number of all possible pairs of items can be calculated using the combination formula:

   $$\binom{n}{2} = \frac{n(n-1)}{2} \underbrace{=}_{n=1000} \frac{1000 \times 999}{2} = 499,500 \text{ pairs}$$

   - **Computational Feasibility**: Checking 499,500 pairs is feasible in terms of memory and time complexity, especially if we do it in batches and use efficient data structures like hash tables or triangular arrays to keep track of the counts. The time complexity of this operation is $O(n^2)$, where n represents the number of unique items. Since we only have to consider pairs, this is manageable for modern computers.

2. **Counting Larger Sets (e.g., Triples)**: Now, consider counting triples (sets of three items that appear together frequently). To do this, we need to check all possible combinations of three items

   - **Total Triples to consider**: The number of possible triples can be calculated using the combination formula:

   $$\binom{n}{3} = \frac{n(n-1)(n-2)}{6} \underbrace{=}_{n=1000} \frac{1000 \times 999 \times 998}{6} \approx 166,167,000 \text{ triples}$$

   - **Computational Complexity**: Now we have around 166 million triples to consider. This is already significantly larger than the number of pairs. The time complexity for checking all combinations of three items is $O(n^3/3!)$. The factorial in the denominator represents the increasing complexity as the size of the subsets grows. In terms of memory constraints, storing these 166 million counts is already demanding on memory, and the time required to process each basket and update counts is substantial.

1.6 Triangular Array

A triangular array is an efficient way to store the counts of pairs of items while saving memory. This method avoids storing redundant or unnecessary data by keeping only the necessary elements in a **1-dimensional array**. When counting pairs of items, you could use a 2-dimensional array to keep track of every possible pair of items. However, this would waste a lot of space because:

- The pairs (i, j) and (j, i) are essentially the same.
- You don't need to store pairs where $i = j$ (i.e., an item paired with itself).

Instead of using a 2-dimensional array, a triangular array uses a single-dimensional array to store the counts of pairs more efficiently. The idea is to store only pairs where the first item's index (i) is less than the second item's index (j).

To store pairs efficiently, a formula is used to map a 2D index (i, j) into a 1D index k in the array:

$$\boxed{k = (i-1) \cdot [n - \tfrac{i}{2}] + (j - i) \text{ where } i < j \text{ and } n \text{ is the total number of unique items}}$$

1.6.1 Example

Let's say we have 5 unique items: $\mathcal{I} = \{A, B, C, D, E\}$, and we want to count the occurences of each pair in multiple baskets. We will use the indices 1 through 5 to represent these items.

1. **Unique Pairs to consider**: We only need to consider pairs where the first item's index is less than the scond item's index. The pairs are:

   - $(1,2)$, $(1,3)$, $(1,4)$, $(1,5)$
   - $(2,3)$, $(2,4)$, $(2,5)$
   - $(3,4)$, $(3,5)$
   - $(4,5)$

   The number of elemens in the triangular array is $\binom{5}{2} = \frac{5(5-1)}{2} = 10$.

2. **Storing the Pairs in a Triangular Array**: We can calculate the index k in the 1-dimensional array for each pair (i, j):

   - For $(1,2)$ the index is: $k = (1-1) \cdot [5 - \frac{1}{2}] + (2-1) = 0 * \frac{9}{2} + 1 = 1$
   - For $(1,3)$ the index is: $k = (1-1) \cdot [5 - \frac{1}{2}] + (3-1) = 0 * \frac{9}{2} + 2 = 2$
   - For $(1,4)$ the index is: $k = (1-1) \cdot [5 - \frac{1}{2}] + (4-1) = 0 * \frac{9}{2} + 3 = 3$
   - For $(1,5)$ the index is: $k = (1-1) \cdot [5 - \frac{1}{2}] + (5-1) = 0 * \frac{9}{2} + 4 = 4$
   - For $(2,3)$ the index is: $k = (2-1) \cdot [5 - \frac{2}{2}] + (3-2) = 1 * 4 + 1 = 5$
   - For $(2,4)$ the index is: $k = (2-1) \cdot [5 - \frac{2}{2}] + (4-2) = 1 * 4 + 2 = 6$
   - For $(2,5)$ the index is: $k = (2-1) \cdot [5 - \frac{2}{2}] + (5-2) = 1 * 4 + 3 = 7$
   - For $(3,4)$ the index is: $k = (3-1) \cdot [5 - \frac{3}{2}] + (4-3) = 2 * \frac{7}{2} + 1 = 8$
   - For $(3,5)$ the index is: $k = (3-1) \cdot [5 - \frac{3}{2}] + (5-3) = 2 * \frac{7}{2} + 2 = 9$
   - For $(4,5)$ the index is: $k = (4-1) \cdot [5 - \frac{4}{2}] + (5-4) = 3 * 3 + 1 = 10$

   Therefore, when we encounter a pair in a basket (e.g., (A, C) corresponding to $(1, 3)$), it suffices to update the count at index 2 in the triangular array.

1.7 Hash Table

When dealing with triples of items (i.e., combinations of three items from a set), the storage approach differs from the triangular array used for pairs. Instead of using a mathematical formula to compute the index in a 1-dimensional array, hash tables are used to efficiently store and access triples. This approach is particularly effective when the counts of triples are sparse.

In many cases, only a small fraction of all possible triples may actually appear frequently in the data (baskets), this is called sparse data. The vast majority of possible triples will not be frequent, resulting in many zero or negligible counts.

Just like with the triangular array, each unique triple (i, j, k) (where $i < j < k$) is stored as a key in the hash table. The order is maintained (i.e., $i < j < k$) to avoid duplicates (e.g., (A, B, C) is the same as (B, A, C), (C, B, A), etc.). The value corresponding to each key is the count of that triple (i.e., how often this combination of three items appears together in the baskets).

The hash table only stores the triples that have appeared in the data, avoiding the need to allocate memory for all possible combinations. Hash tables offer $O(1)$ average time complexity for insertions, deletions, and lookups, making it efficient to update and query the counts of the triples.

2 Apriori Algorithm

Previously we presented two data structures (1D arrays and Hash tables) that can be employed to efficiently store and count either pairs or larger sets of data. Now, we introduce the Apriori Algorithm, which is a fundamental method in data mining for discovering frequent itemsets in a large dataset, such as transactions or baskets of items and is widely used for association rule learning.

The purpose of the algorithm is to reduce the number of itemsets that need to be examined for frequency by leveraging the principle of monotonicity.

2.1 Algorithm Steps

The algorithm makes two passes over the data to efficiently identify frequent itemsets.

2.1.1 Pass 1

- **Read the Baskets**: Count the support (frequency) of each individual item across all baskets.
- **Filter Items by Support**: Retain only the items that have a support greater than or equal to a predefined threshold s. These items are called frequent itemsets.

2.1.2 Pass 2

- **Read the Baskets Again**: Count only the pairs of frequent items (those identified in pass 1). This significantly reduces the number of pairs we need to examine, as we only consider combinations of items that have already been determined to be frequent.
- **Filter Pairs by Support**: Retain only the pairs whose support is above or equal to the threshold s.

The algorithm uses the monotonicity property to eliminate unnecessary computations: if an itemset (like a pair) does not meet the support threshold, none of its supersets (e.g., triples that include that pair) can be frequent. This principle allows the algorithm to efficiently prune the search space.

The triangular array and hash tables can be used during the second pass of the Apriori Algorithm to drastically reduce memory usage, optimize space and fast access. Finally, some notations that might be useful to ease the resolution of exercises:

- C_k denotes the candidate itemsets of size k
- L_k represents the truly frequent itemsets among those candidates of sike k.

2.2 Example

Let's apply the Apriori Algorithm given a support threshold of 2 and the following transactions/baskets:

- B1 = {m, c, b}
- B2 = {m, p, j}
- B3 = {m, b}
- B4 = {m, j}

1. **Generate Candidate 1-Itemsets**: The itemset candidates for C_1 are: {m}, {c}, {b}, {p}, {j}. The truly frequent 1-itemsets are the itemsets that meet the support threshold $s = 2$, therefore, after counting the occurences, L_1 is composed of {m}, {j} and {b}.

2. **Generate Candidate 2-Itemsets**: To form the itemset candidates for C_2, we pair the items from L_1: {m, j}, {m, b}, {b, j}. The truly frequent 2-itemsets are the itemsets that meet the support threshold $s = 2$, therefore, after counting the occurences, L_2 is made up of {m, b} and {m, j}.

3. **Generate Candidate 3-Itemsets**: To form the itemset candidates for C_3, we pair the items from L_2: {m, b, j}. The truly frequent 3-itemsets are the itemsets that meet the support threshold $s = 2$, therefore, $L_3 = \emptyset$.