Introduction of Frequent Itemset Mining & Association Rules Homework

Course: Big Data and Business Intelligence - MI3600E

Phan Ngoc Anh, 20223464
Hanoi University of Science and Technology

A. Theoretical Background

1. Definition

A basket refers to a single transaction that contains a set of items purchased together. In market basket analysis, each basket represents one record in the transaction database.

An item is an individual product or object that may appear in a basket. Each basket consists of one or more items.

Support indicates how frequently an itemset occurs in the transaction database. Support reflects the level of importance or popularity of an itemset within the dataset.

$$\text{Support (X)} = \frac{\text{Number of baskets containing X}}{\text{Total number ofbasket}}$$

Confidence represents the strength of an association rule. For a rule X →Y, confidence is defined as:

$$\text{Confidence (X } \rightarrow \text{Y)} = \frac{\text{Support (X} \cup \text{Y)}}{\text{Support (X)}}$$

It expresses the conditional probability that itemset Y appears in a transaction, given that itemset X is present.

Interest measures the usefulness of an association rule by comparing its confidence with the overall probacurbility of the consequent. If the interest value is close to zero or negative, the rule is considered weak or uninformative. Interest (X → Y)= Confidence (X → Y)- Support(Y)

2. High Confidence but Uninteresting Rules

A rule may have high confidence when the consequent item is very common in the dataset. In such cases, the high confidence does not necessarily indicate a true dependency between the antecedent and the consequent.

For example, if an item appears in nearly every transaction, many rules predicting this item will have high confidence regardless of the antecedent. Therefore, confidence alone is not sufficient to evaluate the usefulness of a rule. Additional measures such as interest or lift are required to determine whether the rule provides meaningful insight.

3. Why Finding Frequent Pairs Is Harder than Finding Frequent Triples

Although the number of possible triples is larger than the number of pairs in theory, in practice, frequent pairs are more difficult to identify.

Many candidate pairs must be examined and counted in the early stages of the algorithm before pruning occurs. However, only a small number of triples remain frequent after the pruning process. As a result, computational effort is greater for frequent pairs because the algorithm must consider and count a much larger number of candidates before elimination.

4. Difference Between Basket and Bucket

A basket is a data object that contains items in a transaction. It represents real input data. In the meantime, a bucket is a hashing structure used to group item pairs in memory. Buckets are used in algorithms such as PCY to reduce the number of candidate pairs by filtering unlikely combinations.

5. Apriori Monotonicity Principle

The Apriori principle states that: If an itemset is frequent, all of its subsets must also be frequent.

Conversely: If an itemset is not frequent, then none of its supersets can be frequent. This property allows the Apriori algorithm to eliminate many candidate itemsets early, thereby improving efficiency and reducing the number of database scans.

B. Manual Computation[1]

1. Dataset Description

The dataset consists of 8 transaction baskets with five distinct items:

m = milk

c = coke

p = pepsi

b = beer

j = juice

The minimum support threshold is set to s = 3 baskets.

Transaction Table:

Basket	Items
B1	{m, c, b}
B2	{m, p, j}
B3	{m, b}
B4	{c, j}
B5	{m, p, b}
B6	{m, c, b, j}
B7	{c, b, j}
B8	{b, c}

2. Frequent 1-itemsets

The support count for each item is computed as follows:

Item	Support Count
m	5
c	5
b	6
j	4
p	2

Applying the support threshold (s ≥ 3), item p is removed: {m}, {c}, {b}, {j}

3. Frequent 2-itemsets

Itemset	Support Count
{m, b}	4
{b, c}	4
{c, j}	3
{b, j}	2
{m, c}	2
{m, j}	2
{m, p}	2

Applying the support threshold: {m, b}, {b, c}, {c, j}

4. Support of {m, b}

The pair {m, b} appears in the following baskets:

- B1
- B3
- B5
- B6

Support ({m, b}) = 4

5. Confidence of the Rule ({m, b} → {c})

Confidence is computed as:

$$Confidence(\{m, b\} \rightarrow \{c\}) = \frac{Support\ (\{m,b,c\})}{Support\ (\{m,b\})}$$

The itemset {m, b, c} appears in:

- B1

- B6

Support({m, b, c}) = 2

Confidence = 2: 4 = 0.5

6. **Interest of the Rule {m, b} → c**

Interest is calculated as:

$Interest =$ *Confidence({m, b} → {c}) - Support({c})*

The support of item c is:

Support({c}) = 5/8 = 0.625

Interest = 0.5 - 0.625 = -0.125

Since the interest value is negative, the rule is considered **uninteresting**, as the occurrence of coke is already very common and not significantly influenced by milk and beer.

7. **Association Rules from Frequent 3-itemsets**

No 3-itemsets satisfy the minimum support threshold of 3. Therefore: There are no frequent 3-itemsets. No association rules can be generated from 3-itemsets.

C. **Algorithms & Simulation**.

1. **Apriori Pass 1: Identifying Frequent Items**

In the first pass of the Apriori algorithm, the database is scanned to determine the support count of each individual item. The support counts obtained are

Item	Support Count
m (milk)	5
c (coke)	5
b (beer)	6
j (juice)	4
p (pepsi)	2

Given the minimum support threshold s = 3, item p (pepsi) is removed.

Frequent 1-itemsets (L1): L1 = {{m}, {c}, {b}, {j}}

2. **Generation of Candidate Itemsets C2, L2, and C3**

Candidate 2-itemsets (C2):

Using the frequent 1-itemsets L1, all possible pairs are generated:
C2 = {{m,b}, {m,c}, {m,j}, {c,b}, {c,j}, {b,j}}.
Their support counts are obtained in another database scan:

Candidate	Support
{m, b}	4
{m, c}	2
{m, j}	2
{b, c}	4
{c, j}	3
{b, j}	2

Applying the minimum support threshold:

Frequent 2-itemsets (L2):

L2 = {{m, b}, {b, c}, {c, j}}

Candidate 3-itemsets (C3) Using Pruning

Candidate 3-itemsets are generated by joining itemsets in L2.

Potential combinations: {m, b, c} and {b, c, j}

We apply the Apriori pruning rule:

{m, b, c}: Subsets: {m, b}, {b, c}, {m, c}. Since {m, c} is not frequent, this candidate is pruned.

{b, c, j}: Subsets: {b, c}, {b, j}, {c, j}. Since {b, j} is not frequent, this candidate is pruned.

C3 = ∅

3. **Why Apriori Requires Multiple Passes Over the Dataset**

The Apriori algorithm requires multiple scans of the database because frequent itemsets are discovered iteratively.

- In the first scan, it identifies frequent single items.
- In each subsequent scan, it generates candidate itemsets of increasing length and counts their support.
- Infrequent candidates are removed at each step using the Apriori principle.

Multiple passes are necessary because the frequency of larger itemsets depends on the frequency of smaller subsets. The algorithm cannot directly determine higher-order itemsets without first identifying lower-order frequent itemsets.

This level-wise approach ensures both correctness and efficiency by reducing the number of candidate itemsets through pruning.

D. PCY, Multistage, and Multihash

1. PCY Pass 1 Simulation

For this task, we simulate the first pass of the PCY algorithm using the hash function:

h(i, j) = (i *j) mod 5

Given the basket:

{1, 3, 4}

All possible pairs are generated:

(1, 3)

(1, 4)

cvb(3, 4)

Hash Computatio

Pair	Hash value
(1, 3)	(1 × 3) mod 5 = 3
(1, 4)	(1 × 4) mod 5 = 4
(3, 4)	(3 × 4) mod 5 = 2

Bucket Count After Pass 1

We maintain an array of 5 buckets (indexed 0–4).

The bucket count is incremented based on the hash values:

Bucket	Count
0	0
1	0
2	1
3	1
4	1

2. **Frequent Buckets and Candidate Pairs (s = 3)**

Given the support threshold s = 3:

All buckets have a count of 1, which is less than the required threshold.

There are no frequent buckets, and therefore:

- No candidate pairs are generated in Pass 2.
- All pairs are pruned immediately.

The bitmap will contain all zeros, meaning no candidate pairs survive this filtering step.

3. **Comparison of Multistage and Multihash**

Multistage Hashing

Multistage hashing applies multiple passes of hashing sequentially.

- In Pass 1, hashing determines frequent buckets.
- Only candidate pairs from frequent buckets are passed to the next round.
- A second hash function is applied in the next pass.

Advantages:

- Further reduces candidate pairs.
- Simple to implement.

Disadvantages:

- Requires additional database scans.
- Increases processing time.

Multihash

Multihash uses multiple hash functions simultaneously in one pass.

Each pair is hashed into several bucket arrays at the same time.

Advantages:

- Fewer passes over the dataset.
- Stronger filtering of candidate pairs.

Disadvantages:

- Requires more memory.
- More complex to implement.

4. PCY vs Multistage vs Multihash

Feature	PCY	Multistage	Multihash
Hash functions	1	Multiple (sequential)	Multiple (parallel)
Database scans	2	More than 2	Usually 2
Memory usage	Low	Medium	High
Pruning power	Moderate	Better	Strongest
Implementation	Simple	Medium	Complex

E. Discussion and Conclusion.

1. Discussion

Frequent itemset mining and association rule learning play a crucial role in extracting meaningful patterns from large transactional datasets. In this report, the Apriori algorithm and its optimizations were examined in order to understand how large-scale data mining problems are addressed under memory and performance constraints.

One of the primary limitations in frequent itemset mining is the main memory bottleneck. As datasets grow in size, it becomes infeasible to store all candidate itemsets and their counts in memory. Algorithms such as PCY attempt to mitigate this problem by using hashing techniques and bitmaps to reduce the number of candidate pairs stored explicitly. This significantly decreases memory usage by eliminating improbable candidates early in the process.

The choice of algorithm strongly depends on system constraints and application requirements:

- Apriori is suitable for small to medium datasets where memory is sufficient and interpretability is prioritized.
- PCY is preferred when memory is limited and the number of pair candidates is very large.
- Random Sampling is beneficial when a fast approximate result is acceptable and a full database scan is costly.
- SON algorithm is designed for distributed environments where data is stored across multiple nodes and parallel processing is required.

Additionally, the use of PySpark FP-Growth in this report demonstrates the significance of distributed computing frameworks in modern data analytics. Unlike Apriori, FP-Growth avoids generating candidate itemsets and instead constructs a compact FP-tree, which makes it more efficient for large-scale datasets.

Overall, this study emphasizes that no single algorithm is optimal in all situations. Instead, the effectiveness of each approach depends on available memory, dataset size, required accuracy, and computational resources.

2. Conclusion

This report explored the principles and applications of frequent itemset mining and association rule analysis using the Apriori algorithm, PCY optimization, and FP-Growth implementation. Through both manual calculations and algorithm simulation, the fundamental concepts of support, confidence, and interest were clearly demonstrated. The results show that frequent itemset mining requires careful consideration of memory limitations and computational efficiency. Hashing techniques, pruning principles, and distributed computing frameworks significantly enhance scalability and performance.

In conclusion, frequent itemset mining remains a vital technique in data mining and business intelligence applications such as market basket analysis, recommendation systems, and customer behavior analysis. Future improvements in this field will focus on scalable algorithms that can efficiently process massive data streams in real time using limited resources.

References

1. Agrawal, R., & Srikant, R. (1994). *Fast algorithms for mining association rules*. Proceedings of the 20th International Conference on Very Large Data Bases (VLDB).
2. Han, J., Pei, J., & Yin, Y. (2000). *Mining frequent patterns without candidate generation*. Proceedings of the ACM SIGMOD International Conference on Management of Data.
3. Park, J. S., Chen, M. S., & Yu, P. S. (1995). *An effective hash-based algorithm for mining association rules*. Proceedings of the ACM SIGMOD Conference.
4. Leskovec, J., Rajaraman, A., & Ullman, J. D. (2020). *Mining of Massive Datasets* (3rd ed.). Cambridge University Press.
5. Dean, J., & Ghemawat, S. (2008). *MapReduce: Simplified data processing on large clusters*. Communications of the ACM.
6. Zaharia, M., et al. (2016). *Apache Spark: A unified engine for big data processing*. Communications of the ACM.
7. Gibbons, P. B., & Matias, Y. (1998). *New sampling-based summary statistics for improving approximate query answers*. Proceedings of the ACM SIGMOD Conference.
8. Manku, G. S., & Motwani, R. (2002). *Approximate frequency counts over data streams*. Proceedings of the VLDB Conference.
9. Bloom, B. H. (1970). *Space/time trade-offs in hash coding with allowable errors*. Communications of the ACM.
10. Indyk, P., & Motwani, R. (1998). *Approximate nearest neighbors: Towards removing the curse of dimensionality*. Proceedings of the ACM Symposium on Theory of Computing (STOC).
11. Shukla, A., Marin, I., & Sarang, V. K. (2019). *Big Data Analysis with Python: Combine Spark and Python to Unlock the Powers of Parallel Computing and Machine Learning*. Packt Publishing