# Frequent itemset mining based on H-Mine,Apriori and FP-Growth

Zhang Hong
1024041135
Nanjing University of Posts and Telecommunications
School of Computer Science
Nanjing, China

***Abstract*—This paper introduces three algorithms for frequent itemset mining, namely Apriori Algorithm, FP-Growth Algorithm and H-Mine Algorithm, and uses each of the three algorithms to perform frequent itemset mining on the same dataset. By comparing the performance and effectiveness of these algorithms, we aim to explore their strengths and weaknesses when dealing with data of different sizes and complexities, and to evaluate their applicability in practical applications.**

***Keywords*—Association Rule Mining, Apriori Algorithm, FP-Growth Algorithm, H-Mine Algorithm, Frequent ItemSets**

## I. Introduction

Association rule mining is an important task in the research field of data mining to mine meaningful associations in transactional databases. The problem was first introduced in 1993 by Agrawal [1] et al. in their analysis of the Market Rule Analysis (MRA) problem to discover customer problems in merchandising. Association rules can discover interesting relationships between Items or Attributes present in a database that are unknown and hidden in advance, i.e., they cannot be derived by logical operations or statistical methods of the database. This means that they are not based on the inherent properties of the data itself, but on the concurrent features of the data items, and the rules discovered can assist people in marketing operations, decision support, business management and website design. Therefore the study of association rule algorithms is very important. Apriori algorithm of association rule mining is the most classic algorithm of database mining and widely used, but the study found that Apriori algorithm has the shortcomings of low efficiency of generating candidate itemsets and frequent scanning of data, etc. Various algorithms appeared in the future basically based on the Apriori algorithm to improve the algorithm.

## II. Association Rule Problem Description

The formal description of the association rule mining problem is: [2] set $\mathrm{I} = \{\mathrm{I}_1, \mathrm{I}_2, \ldots, \mathrm{I}_\mathrm{n}\}$. Let the task-related data D be a database of transactions, where each transaction T is a collection of items such that each transaction of $T \subseteq I$ has an identifier TID. a collection consisting of multiple items is called an itemset. A set of k items is called a k-item set. Let A be an itemset and a transaction T contains A if and only if $A \subseteq T$. An association rule is an implication expression of the form where $X \subseteq I$,$Y \subseteq I$ and $\mathrm{X} \cap \mathrm{Y} = \phi$. The strength of an association rule can be measured by its support and confidence, other metrics are described in the following articles. Support determines how frequently a rule can be used in a given data set, while confidence determines how frequently Y appears in a transaction containing X. The forms of the two metrics, support (s) and confidence (c), are defined below:

$$\mathrm{s}(\mathrm{X} \to \mathrm{Y}) = \frac{\sigma(\mathrm{X} \cup \mathrm{Y})}{\mathrm{N}} \quad \mathrm{c}(\mathrm{X} \to \mathrm{Y}) = \frac{\sigma(\mathrm{X} \cup \mathrm{Y})}{\sigma(\mathrm{X})} \tag{1}$$

where $\sigma(X) = |\{Ti|X \subseteq TiTi \in D\}|$,N is the total number of transactions.

An association rule is said to be a strong association rule if $\mathrm{s}(\mathrm{X} \to \mathrm{Y}) \geqslant \min$ sup and $c(\mathrm{X} \to \mathrm{Y}) \geqslant \min conf$, otherwise it is called a weak association rule. The association rule mining problem is to find all the strong association rules in D with a given support and confidence level. Therefore, mining association rules can be divided into two subproblems [3]:

- find so the set of frequent terms : calls$(\mathrm{X}) \geqslant \min$ sup as X the set of frequent terms.
- Strong association rules are generated from frequent itemsets: by definition, these rules must satisfy minimum support and minimum confidence.

## III. Mining frequent itemsets

### A. Apriori Algorithm

The Apriori algorithm is a basic algorithm for finding frequent itemsets, which is based on the use of a method of generation selection called layer-by-layer search [4], where the k-item set is used to explore the (k+1)-item set.The Apriori algorithm uses a priori knowledge of the nature of frequent itemsets to first find the set of frequent 1-item sets, which is denoted by L. L is used to find the set L of frequent 2-item sets, which in turn is used to find L and so on until no more frequent k-item sets can be found. and so on until no frequent k-term set can be found. In order to improve the efficiency of the layer-by-layer generation of frequent itemsets, an important property called the Apriori property is applied to the algorithm so that the search for frequent itemsets can be divided into two processes:

Step1:Connection In order to find $I_k$, a set of candidate k-itemsets is generated by connecting $I_{K-1}$ with self. The set of candidate k-itemsets is denoted as $C_k$. Let $I_1$ and $I_2$ be the

itemsets in $I_{K-1}$. The notation $I_i[j]$ denotes the jth top of $I_i$. Perform the join $\mathrm{I_{k-1}} >< \mathrm{I_{k-1}}$,where the elements of $I_{K-1}$ are joinable if their first k-2 terms are the same. The resultant itemset produced by joining $I_1$ and $I_2$ is $I_1[1]I_1[2]...I_1[k-1]I_2[k-1]$.

Step2:The pruning step $C_k$ is a superset of $I_k$. By the Aprior property: any infrequent (k-1)-item set cannot be a subset of a frequent k-item set. Therefore, if a (k-1)-subset of a candidate k-item set is not in $L_{K-1}$, the candidate cannot be frequent either, and thus can be removed from $C_k$. The database is then scanned to determine the support of each candidate itemset in $C_k$, and thus $I_k$.

Although the Apriori algorithm itself has been optimized to a certain extent, there is still the problem of inefficiency of the algorithm. The main shortcomings are the following three aspects:

- Generation of too many candidate sets
  Apriori algorithm in the generation of candidate itemsets, the use of two-two connection, the number of candidate itemsets may be exponential growth. For example, when the number of itemsets is large, the number of candidate itemsets generated may be very large, resulting in high computational complexity and great time consumption.
- Inefficient subset judgment
  When determining whether a candidate itemset satisfies the frequent condition, it is necessary to verify whether all the subsets of each candidate itemset are frequent. This process of subset judgment may require a large number of subset generation and scanning operations, resulting in inefficiency.
- Multiple scans of the database
  In order to calculate the support of a candidate itemset, Apriori algorithm needs to scan the database several times, the number of scans is proportional to the length of the itemset. For large-scale datasets, scanning the database multiple times increases the I/O cost and further reduces the efficiency of the algorithm.

### *B. FP-Growth Algorithm*

The basic idea of FP-Growth algorithm is to utilize a tree structure to compress the transaction while preserving the relationships between attributes in the transaction [5]. This algorithm does not generate candidate itemsets, but uses the method of growing frequent sets for data mining.The important step of the FP-Growth algorithm is the FP tree construction process, which requires scanning the set of transactions twice: the first time scanning the set of transactions T to find the frequent 1-item set $l_{\mathrm{NULL}}$ , and arranging the $l_{\mathrm{NULL}}$ in descending order according to the support counts: the second time scanning the set of transactions T to find the frequent 1-item set $l_{\mathrm{NULL}}$ , which is ordered by "Null" as the root node, and construct the FP tree based on $l_{\mathrm{NULL}}$ . In order to facilitate the traversal of the FP tree, it is also necessary to create a header table. Each row of the table represents a frequent item and has a pointer to its corresponding node in the FP tree, as in Fig.1.The specific algorithm flow is as follows:

Step1:create the original FP tree,

- Scan the transaction set T, find the items whose support counts satisfy the condition, and combine these items into a frequent item 1-set L. Based on the support counts, sort L in descending order to get $l_{\mathrm{NULL}}$.
- Create the original FP tree with "Null" as the root node.
- Create a header table. In order to facilitate traversal of the original FP tree, each row in the header table represents a frequent item and has a corresponding pointer to its node in the FP tree.
- Traverse the transaction set T once and adjust the item order of all transactions in T according to $l_{\mathrm{NULL}}$. Create a transaction branch for each transaction with adjusted item order. If the branch can share the path it does so and the number of shared transactions is recorded at each node.

Step2:recursively find the set of all maximal frequent items in the original FP tree.

- Assign the initial value of the suffix pattern $\alpha$ to the root node Null,i.e.$\alpha = Null$.Search for frequent itemsets on the FP tree using a recursive approach.If the FP tree has only one branch, then one combination of nodes on the branch path is a prefix pattern $\beta$. Tick Excluding the nodes on the branch path that do not satisfy the minimum support, any set $\beta$ consisting of the remaining node values is taken and merged with the suffix pattern $\alpha$ to obtain all the corresponding frequent itemsets. where $\beta_i$ is a prefix pattern that otherwise grows $\alpha$,$\alpha = \alpha \cup \{\mathbf{Ei}\}$ (where Ei is the last term in $l_{\mathrm{NULL}}$, i.e., the smallest term in the support count). Then construct the conditional pattern base of the suffix pattern a with the conditional FP tree. (where the conditional pattern base of a refers to all branches in the FP tree with Ei as a leaf node.) The conditional FP tree of A. refers to a new FP subtree created in the manner in step 1 with the conditional pattern base of $\alpha$ as the transaction.
- Recursively search the set of frequent items (suffix pattern $\alpha$ is (Ei) at this point) on the conditional FP tree using the same method.
- For each maximal frequent itemset, take all its subsets, where each subset is a frequent itemset.

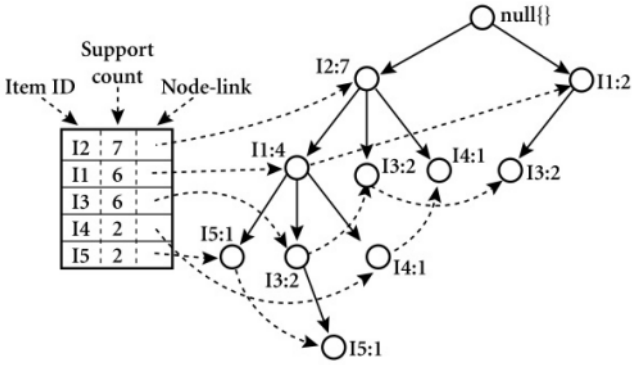


Fig. 1. FP Tree and Header Table

The FP-Growth algorithm, despite solving some of the problems of the Apriori algorithm and showing significant efficiency advantages in frequent pattern mining, has some drawbacks. First, the construction process of FP-Tree is complex and has a high memory dependency, and when the dataset size is very large or the transaction length is long, the FP-Tree may have too much depth, leading to memory overflow problems. Secondly, the algorithm performs poorly when dealing with sparse data because there are fewer transaction overlapping terms in sparse data, and the node reuse rate of FP-Tree is low, which prevents it from compressing data effectively. In addition, FP-Growth recursively generates a large number of conditional FP-Trees, which leads to too many intermediate results and further exacerbates the consumption of memory and computational resources. Finally, FP-Growth itself is a single-threaded algorithm, which cannot fully utilize the computational power of multi-core or distributed environments, and has limited applicability to ultra-large data sets.

### C. H-Mine Algorithm

H-mine uses a new data structure to process the data by generating an H-struct, similar to the FP-growth algorithm, using frequent itemset growth without having to generate a large number of candidate sets. In addition to this, compared to FPgrowth, H-mine does not generate FP-tree and the iterative database required to generate FP-tree, thus saving storage space and time in many cases: compared to Apriori, which does not have to traverse the total dataset many times to generate a lot of candidate sets, the algorithm only traverses the database twice to generate the H struct, and each subsequent scan is performed only in H-struct [6].

The following is a concrete example to illustrate the principle of the H-mine algorithm, setting the minimum support count to 2. The transaction set of the known database TDB and the filtered frequent items are shown in Table I.

TABLE I
TRANSACTION SET

| Trans ID | Items | Frequent-item projection |
|---|---|---|
| 1 | $A, B, D, G, I, F$ | $B, D, G, F$ |
| 2 | $B, C, D, G, H$ | $B, C, D, G$ |
| 3 | $C, E, F, G, K, M$ | $C, E, F, G$ |
| 4 | $B, D, E, N$ | $B, D, E$ |

From Table 1, it can be seen that A, K, M, N, H, I do not meet and support requirements, which is the first scanning, the transaction set will be filtered to get the table $\mathrm{H} : \{\mathrm{B} : 3, \mathrm{C} : 2, \mathrm{D} : 3, \mathrm{E} : 2, \mathrm{F} : 2, \mathrm{G} : 3\}$. And the article needs to mine the frequent item set is divided into: contains B items, contains C items but does not contain B items, contains D items but does not contain C items and B items, contains E items but does not contain D, C, and B items, contains F items but does not contain E, D, C, and B items, as well as only contains G.

The elements in the H-table are used as header pointers to build the H-struct. as shown in Fig.2. Table H is also known as

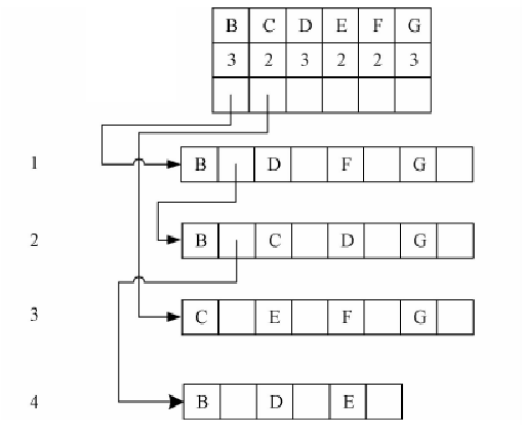


Fig. 2. H-struct

Header table H. It consists of three regions which are frequent items, support counts and their pointers. The nodes in table H are connected by pointers to all the transactions with the same first item, such as item B in the above figure, which links the first items are all 1, 2 and 4 transactions of B, item C links 3 transactions, and items D, E, F and G do not have to establish links because there is no first item that is a transaction of these items.

A second scan is performed when building the H-struct. Once the H-struct is built, the data mining work is then performed only on the H-struct. Firstly, the five 1-item sets in table H are mined in the H-struct, traversing the B-queue to find out all the frequent items in the B-queue, and the table HB is built, which is structured in the same way as the table H. However, the support counts are recorded based on the B elements in the queue are recorded. The output frequent 2-item set can be derived as follows. As shown in Fig. 3.

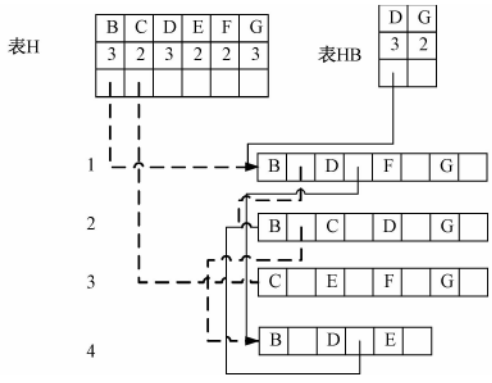


Fig. 3. Header table HB

By analogy, digging for items containing B and D. This time, not only the BD queue is to be mined, but also the 2-transaction in which the pointer has been created in table H is to be inserted, since it contains both B and D, thus obtaining the complete BD dataset, and this time, a frequent 3-item set is mined. Finally, the transactions with BG as the first

item are also searched, but since there is no pointer to BG in table H and table HB, no frequent itemsets are generated. To summarize the above is the process of mining the transaction with B as the first item, the next step is to mine the frequent itemsets that contain C but not B. This process needs to mine the C queue and the B queue that contains the C queue, so it is necessary to insert the B queue into the appropriate position, as shown in Fig. 4, the BCDG is inserted after the CEFG. It is important to note that when mining the C queue, no frequent itemsets about B are generated. The other frequent itemsets are mined in the same way.

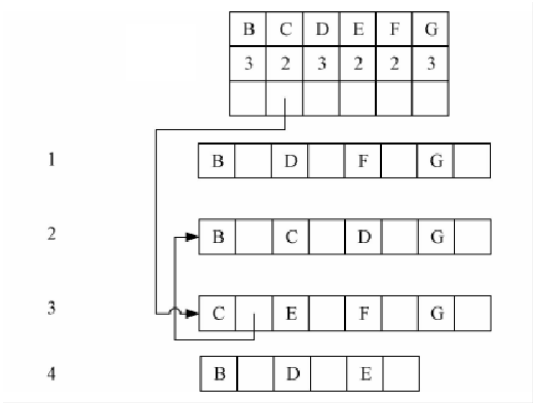


Fig. 4. Process chart of establishing c queue

## IV. Generate association rules

Association rules are assessed for their quality by two main metrics: support and confidence. Support indicates how often the itemsets in a rule occur in all transactions, while confidence measures the probability of a rule's conclusion occurring if the rule's preconditions are met. When generating rules, we first select the itemsets from the frequent itemsets that meet the minimum support and confidence thresholds, and then transform them into rules in the form of "premise-¿conclusion" by partitioning the frequent itemsets. The generated association rules can help us understand the association relationship between different items, which can then be applied in the fields of business decision-making and recommender systems, such as joint purchase prediction of goods and customer behavior analysis.

For some datasets, support and confidence have limitations in association rule mining, mainly because they ignore the independence between terms, cannot handle the effect of rare terms, and may lead to overfitting. To compensate for these shortcomings, other metrics such as lift, leverage, and belief have been introduced to evaluate the effectiveness of rules more comprehensively. Boosting degree reveals the true association between terms, leverage measures the difference between actual and expected frequencies, and belief degree helps to exclude chance rules, thus ensuring that the mined rules have higher practical value and reliability.

- Association Rule Lift

  Lift is a measure of the correlation between two items (e.g., goods or events). It measures the ratio between the frequency with which the antecedent and the consequent of a rule occur together, and the expected frequency with which they would occur independently of each other. In short, lift tells us the strength of the rule: if lift is greater than 1, the antecedent and the consequent occur more often than their probability of occurring independently, indicating that they are somehow related; if it is less than 1, there is no correlation between the antecedent and the consequent, and there may even be a negative correlation.

$$\text{Lift}(A \to B) = \frac{P(A \cap B)}{P(A) \cdot P(B)} \tag{2}$$

- Leverage Ratio

  Leverage is a measure of the difference between the actual frequency of occurrence and the expected frequency of occurrence, and is primarily used to assess the actual strength of a rule. It helps us distinguish rules that are frequent but may be caused by chance, and emphasizes those that are truly meaningful. By calculating the difference between actual and expected frequencies, Leverage can reveal hidden underlying relationships in the data, especially for highly correlated rules with low-frequency items.

$$\text{Leverage}(A \to B) = P(A \cap B) - P(A) \cdot P(B) \tag{3}$$

- Kulczynski Index

  The Kulczynski coefficient is a measure of the similarity of two sets and is often used to assess the similarity between the antecedent and the consequent in an association rule. It measures the relevance of the antecedent and the consequent by comparing their occurrences in the dataset. Kulczynski focuses more on the proportion of occurrences of the antecedent and the consequent than other similarity measures, and is therefore suitable for assessing the similarity and strength of association between two terms in a rule.

$$Kulc\,(A, B) = \frac{Con\,(A \Rightarrow B) + Con\,(B \Rightarrow A)}{2} \tag{4}$$

## V. Frequent pattern mining

This experiment aims to perform frequent itemset mining on the same dataset using three different frequent itemset mining algorithms - Apriori, FP-Growth and H-Mine. By comparing the performance and effectiveness of these algorithms, we aim to explore their strengths and weaknesses when dealing with data of different sizes and complexities, and to assess their applicability in real-world applications. The experiments will analyze the computational efficiency, memory consumption, and quality of the generated frequent itemsets of each algorithm, so as to provide valuable references for practical data mining tasks.

### A. Datasets

The dataset contains 1892 rows of transaction records, with each row representing a list of purchases made by one customer at a time. The numbers in each data row represent different item IDs, and the amount of data (number of items) in each row varies, reflecting the inconsistency in the number of items purchased by different customers in a single shopping trip. For example, a row of data may be "1332, 971, 167, 60, 9, 75, 700, 349", which means that one customer purchased items with IDs 1332, 971, 167, etc., while other customers purchased different quantities and types of items.

From a practical application point of view, by numbering the item names, this data can be used for frequent itemset mining to discover which items are often purchased together. For example, if frequent item set mining results show that item ID 971 (milk) and item ID 167 (bread) appear together frequently in transactions, the merchant can hypothesize that customers tend to buy these two items together. Based on this insight, the merchant can optimize the display layout of the products, design bundling strategies, or provide related product recommendations to increase sales and customer satisfaction.

### B. Data preprocessing

Before applying the frequent itemset mining algorithm, the data needs to undergo some preprocessing. The raw data is usually provided in the form of transactions, where each row represents a transaction and each transaction contains the IDs of multiple commodities.In order to make the data suitable for algorithmic processing, it is first necessary to convert the raw transaction data into a format that can represent the occurrences of commodities.

In this process, the CSV file is read line by line, and the commodity IDs of each line are comma-separated and saved as a list that constitutes a collection of transactions. Then, these transaction data are converted using the TransactionEncoder class.TransactionEncoder converts each transaction into a binary matrix where each column represents an item ID, and the corresponding column value is 1 if the item appears in the current transaction, and 0 otherwise.In this way, the transaction data are converted into a format suitable for the the format of the algorithm input.

The converted data is organized as a Pandas::DataFrame, where each column represents a commodity ID and the rows represent a transaction, with a value of 0 or 1 indicating whether or not the commodity appears in the transaction. Data in this format facilitates frequent itemset mining by the Apriori algorithm.

### C. Experimental design

This experiment aims to perform frequent itemset mining on the same dataset using three different frequent itemset mining algorithms - Apriori, FP-Growth and H-Mine. By comparing the performance and effectiveness of these algorithms, it aims to explore their strengths and weaknesses when dealing with data of different sizes and complexities, and to assess their applicability in real-world applications. The experiments will analyze the computational efficiency, memory consumption, and the quality of the generated frequent itemsets of each algorithm, so as to provide valuable references for practical data mining tasks. Two experiments are conducted here, one is to use H-mine to conduct experiments with different support levels of the dataset, to observe the number of frequent itemsets mined and the time spent under different support levels, and to explore the impact of the support level of frequent itemsets on the mining results; the second experiment compares the performance and memory consumption of the three algorithms mentioned above.

## VI. Data analysis

### A. Impact of support of frequent itemsets on mining results

Firstly, we keep statistics on the frequency of occurrence of individual data items, and the results are shown in Fig.5, where the frequency distribution of the data is sparse, with most of the data items having a very low frequency, and only a few itemsets having a high frequency.Therefore, choosing a higher support threshold can help filter out the vast majority of infrequent itemsets, thus focusing only on those itemsets that occur frequently. Setting a support threshold too low may result in finding a large number of sparse, useless frequent itemsets, while setting a support threshold too high may miss some valuable itemsets.

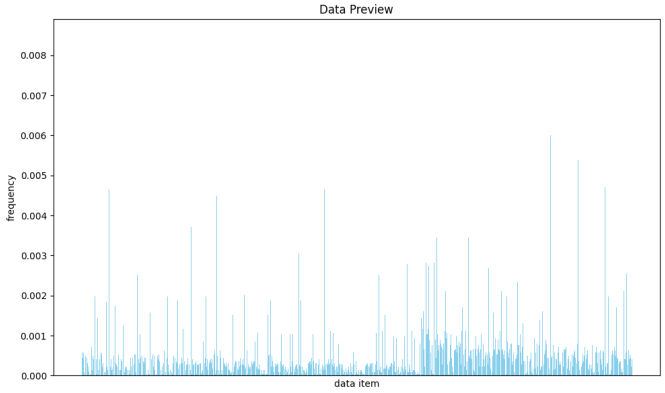


Fig. 5. Data Preview

The results of the runs for different support degrees are shown in Table II. The experimental results show that the lower the support degree, the higher the number of frequent itemsets, and the longer the computation time consumed. For example, when the support level is 0.001, the number of frequent itemsets is 211571, and as the support level increases, the number of frequent itemsets decreases sharply; there are no frequent itemsets when the support level is 0.10, which is in line with the expectation, indicating that the dataset is very sparse, and the high support level means that only a very few itemsets occur frequently, which are relatively rare in the data, and show a long-tailed distribution: a few itemsets occur frequently, and most itemsets occur less frequently. most itemsets occur less frequently.

The balance in practice: low support (e.g. 0.001) finds more frequent itemsets, but it takes longer to compute and may contain irrelevant or noisy itemsets, affecting the quality of the results. A higher support level (e.g. 0.01 or higher) significantly reduces the computational effort and improves the computational efficiency, but reduces the number of frequent itemsets found. Therefore, the selection of the optimal support degree should be balanced according to the requirements: when choosing the support degree, it is recommended to adjust it to the specific application scenario, consider the computational resource limitations, and make sure that valuable frequent itemsets are mined.

| sub | num_item | time |
|---|---|---|
| 0.001 | 211571 | 6.1404 |
| 0.002 | 6524 | 0.244 |
| 0.003 | 2769 | 0.124 |
| 0.004 | 1644 | 0.078 |
| 0.005 | 1115 | 0.05 |
| 0.006 | 800 | 0.04 |
| 0.007 | 602 | 0.029 |
| 0.008 | 467 | 0.021 |
| 0.009 | 374 | 0.018 |
| 0.010 | 335 | 0.016 |
| 0.012 | 260 | 0.014 |
| 0.014 | 192 | 0.011 |
| 0.016 | 151 | 0.009 |
| 0.018 | 114 | 0.008 |
| 0.02 | 96 | 0.007 |
| 0.03 | 36 | 0.005 |
| 0.04 | 19 | 0.004 |
| 0.05 | 14 | 0.004 |
| 0.06 | 7 | 0.005 |
| 0.07 | 5 | 0.005 |
| 0.08 | 2 | 0.003 |
| 0.09 | 2 | 0.004 |
| 0.10 | 0 | 0 003 |

TABLE II
IMPACT OF SUPPORT OF FREQUENT ITEMSETS ON MINING RESULTS

### B. Algorithm comparison

Fig.6, as well as Fig.7, show some of the experimental data, excluding some points with unusually large values, which could not be plotted on a single graph. For example, the Apriori algorithm consumes 10 seconds for a support of 0.002. The icon data for time consumption was chosen for points where the time consumption was one second as well as one second or less. In terms of memory and time consumption for frequent itemset mining, all three algorithms show a decreasing trend in memory and time consumption as support increases. Specifically, the memory consumption of the Apriori algorithm is higher at low support (e.g., 5.9 MB for 0.002), but decreases with increasing support to 1.6 MB. Its time consumption shows a similar decreasing trend, especially at low support (0.005), where the running time is 1.1434 seconds, but has dropped to 0.068 seconds for 0.014 support. 0.014 has decreased to 0.068 seconds. In contrast, FP-Growth has higher memory consumption at low support (11.6 MB for 0.002), which may be due to the fact that the FP tree it builds is abnormally large at low support, resulting in an increased memory footprint. However, as the support level increases, both the memory and time consumption of FP-Growth decreases, showing better performance advantages. In particular, its time consumption is significantly lower than Apriori at higher support levels, e.g., at a support level of 0.005, its running time is 1.002 seconds, compared with 1.1434 seconds for Apriori. The H-Mine algorithm consumes the least amount of memory and time at all levels of support, especially at low levels of support, where the memory consumption remains low (e.g., 5.5 MB at 0.002), and the time consumption is only 0.05 seconds at 0.005, and decreases to 0.011 seconds at 0.014, which shows very high computational efficiency and memory utilization. Although H-Mine's memory consumption is higher than Apriori's at low support, its efficient indexing technique and mining pool structure make it especially superior in time consumption. Overall, the memory and time consumption of the three algorithms decreases with increasing support, while at low support, FP-Growth has significantly higher memory consumption, Apriori has greater time consumption, and H-Mine performs consistently more efficiently.

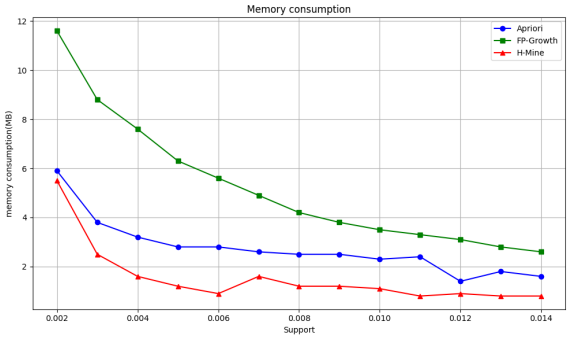


Fig. 6. memory consumption

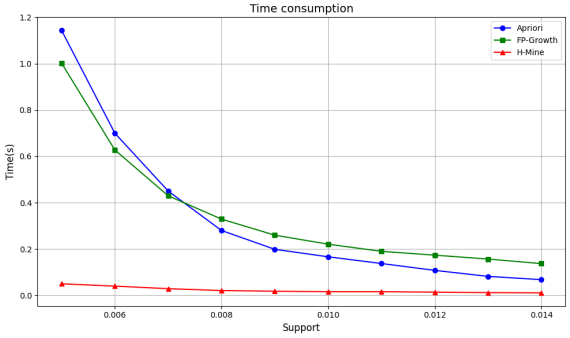


Fig. 7. time consumption

## SUMMARY

After a semester in the Big Data Analytics course, I was introduced to a new field, the science of working with data. Thanks to Prof. Zou's lectures, I felt the pure spirit of learning and research, and had the opportunity to time some knowledge in the field of data mining.

## REFERENCES


[1] R. Agrawal, T. Imielinski, and A. Swami.Mining association rules between sets of items in large databases.Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD'93), ACM Press, 1993,207-216.

[2] WurS Y, Leu Y H. An effective Boolean algorithn formin ing association nules in large databases[C]. Database Systens for Advanced Applications 1999: Proceedings 6 th ntemational Conference April 1999: 19-21

[3] Park JS Chen M S Yu P S Using a hashbased method with transaction trimm ing form ining assoc iation nules[J]. Know ledge and Data Engineering IEEE Transactions 1997, 9(5): 813 825.

[4] Harish Kumar Pamnani,Linesh Raja Thom Ives.(2024).Developing a novel H-Apriori algorithm using support-leverage matrix for association rule mining.International Journal of Information Technology(prepublish),1-11.

[5] Praveen Kumar B.,Padmavathy T.,Muthunagai S.U. Paulraj D..(2024).An optimized fuzzy based FP-growth algorithm for mining temporal data.Journal of Intelligent Fuzzy Systems(1),41-51.

[6] Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D. (2001). H-mine: hyper-structure mining of frequent patterns in large databases. Proceedings 2001 IEEE International Conference on Data Mining, 441-448.