A Scalable Approach for Data Mining – AHUIM

Vandna Dahiya
Research Scholar, Maharshi Dayanand University, Rohtak, India.
E-mail: vandanadahiya2010@gmail.com

Sandeep Dalal
Assistant Professor, Maharshi Dayanand University, Rohtak, India.
E-mail: sandeepdalal.80@gmail.com

Received December 04, 2020; Accepted February 10, 2021
ISSN: 1735-188X
DOI: 10.14704/WEB/V18I1/WEB18029

Abstract

Utility itemset mining, which finds the item sets based on utility factors, has established itself as an essential form of data mining. The utility is defined in terms of quantity and some interest factor. Various methods have been developed so far by the researchers to mine these itemsets but most of them are not scalable. In the present times, a scalable approach is required that can fulfill the budding needs of data mining. A Spark based novel technique has been recommended in this research paper for mining the data in a distributed way, called as Absolute High Utility Itemset Mining (AHUIM). The technique is suitable for small as well as large datasets. The performance of the technique is being measured for various parameters such as speed, scalability, and accuracy etc.

Keywords

Utility Mining, Spark, Big Data, ARM.

Introduction

Frequent Itemset Mining (FIM) (Agrawal, 2003) is a technique focused on the frequency of occurrence of items to mine the datasets. It explores the itemsets built on the support and confidence standards of the itemsets. But FIM does not pay attention, however, to the value of the objects. For example, sales of bread and butter in a general store can be more frequent, but a microwave oven would produce more profit. In a specific transaction, FIM often considers the existence of the items only once, whereas the items with more benefit value but with fewer occurrences may be crucial for the users. In order to address these FIM limitations, high-utility itemset mining has emerged as a superior data mining technique that mines itemsets based on their utility factors. Two terminologies - internal

utility and external utility - compose the utility factor of products. The internal utility depends on the quantity of the product, while the external utility may be in the form of cost, weight, benefit, impact or some other target or desire for the consumer. Utility mining thus discovers the most valuable or useful goods that are difficult to mine using periodic itemset mining. An itemset, if and only if its usefulness is greater or equal to a predefined threshold value, is discovered and termed as a HUI. For example, in the case of retail stores, while the FIM only discovers the customer's regular purchased items, the HUIM discovers the item sets that are even more lucrative for the store when they are purchased together. In the store, these itemsets are then shoved together, recommended to the clients and some discount can also be given. Recommender systems can therefore be established based on the customers' buying experience. Click-stream analysis is another instance of HUIM. FIM would suggest all of the related links to the user if a user clicks on different links on the Internet. But only a few of those clicked links can be of interest to users. The sort of links where users have spent more time will be suggested by HUIM. The utility factor here is time spent on a specific tab. Other utility mining applications include cross-marketing, smarter healthcare, e-commerce, drug design, etc.

Preliminaries and Problem Statement

Utility Itemset Mining (UIM) has developed itself as an integral type of data mining, which identifies itemsets based on utility factors. The utility is specified in terms of quantity and some interest factor and the algorithm seeks to classify the set of items for which the utility exceeds a predefined threshold. Conventional High Utility Item Set (HUI) mining methods are not suitable for mining the HUIs on an individual device with limited processing resources in the big data age.

Different HUI techniques for itemset mining have recently been suggested, but most of them are for small datasets and based on single machines (Zida, 2016), (Vincent, 2013), (Liu, 2005), (Chon, 2018), (Zihayat, 2017). As the scale of datasets increases, the quality of mining begins to deteriorate. The computational resources of one computer are not enough to mine vast datasets and impose constraints on the algorithm's scalability. A parallel or distributed system is required to mine the large datasets. The algorithms for these frameworks are in their early stages of research because big data mining is a novel environment. In mining massive databases, the problems are different from conventional data mining (Sethi, 2018), (Nguyen, 2018), (Tamrakar, 2017), (Jimmy, 2019).

In this research work, a distributed algorithm has been recommended to mine the HUIs for big data, called the Absolute High Utility Itemset Mining Algorithm or AHUIM

Algorithm. Using the Apache Spark platform (Apache Spark, 2014), AHUIM mines the data effectively on a multi-node cluster. Different tests are conducted to measure the performance of the algorithm for execution time, scalability, and accuracy. For the fundamentals of HUI, the authors recommend their earlier work in this field (Dalal, 2018), (Dalal, 2019), (Dahiya, 2020).

Proposed Work - AHUIM

The proposed algorithm, Absolute High Utility Itemset Mining extends the fastest small-scale algorithm EFIM. The algorithm has been sculpted on a parallel framework and Apache Spark has been used for the distributed processing. The algorithm works by following the steps described below.

Tree Structure of the Items

First of all, Utility lists are built for the items of the dataset based on the quantity and weightage values. These lists are then used to measure the transaction weighted utilization (TWU) values and unpromising items with TWU values below the minimum threshold are being discarded at this step. The transactions are then revised according to the sorted order of the items based on the TWU values. An enumerated tree is constructed with these sorted items.

Search Space Division

The enumerated tree is used for search space division using the technique recommended by (Y. Chin, 2008), which ensures uniform division of the workload. The process of division can be understood with the help of figure 1.

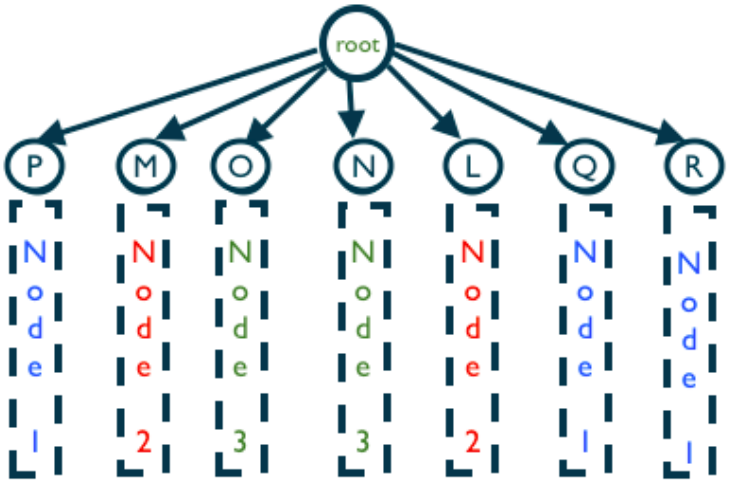

Figure 1 Distribution of Items Among the Nodes

Proposed Method

The proposed AHUIM algorithm takes two inputs - the transactional dataset and the minimum utility threshold. An itemset F is considered to be an empty itemset. For all of the items, the method then calculates the local utility, which is initially equal to their TWU values. To compute the extension of F by comparing it with the minimum utility threshold, the original local utilities or TWU values of items are used. These F items are then arranged with their TWU values sorted out. And the items that are not part of the F extension (i.e. the unpromising TWU value-based items) are discarded here only because they cannot be part of any larger set of HUIs. By organizing all the remaining items of a transaction in order of their rising TWU values, the dataset is then re-checked. Any transaction that is null is removed here. For all the elements of F, the algorithm then calculates the subtree utility. To build larger itemsets based on the local and subtree utilities, the idea of succeeding and following items of an itemset is being used. To avoid the traversal of unnecessary and unpromising items, search space is being pruned using two novel strategies – Absolute Local Utility (ALU) and Absolute Subtree Utility (ASU) (Dalal, 2020). Both the strategies ensured tighter upper bounds to local utility and subtree utility.

In the cluster configuration, the worker nodes are responsible for mining the HUIs for their allocated search space. Node 1 mines the HUIs for items z and x, for instance. For recursively defining the HUIs from a node, the binary recursive search technique is used. The working algorithm, which initiates the process and returns the final HUIs is shown below as Absolute_HUIM.

Algorithm: Absolute_HUIM
Input: DS-Transactional dataset,
I-List of items
Th- minimum threshold rate
Output: Itemsets with high utility

1. F=empty itemset
2. Compute locU (F, I_i) ∀ items $I_i \in I$ by DS scan
3. Compute fi (F) = {I_i |$I_i \in I \wedge$ locU (F, I_i) ≥ Th }
4. For fi (F), arrange according to increasing TWU values
5. Prune items $I_i \notin$ fi (F) and remove null transactions
6. Sort the remaining transactions by ~
7. Compute the SU (F, I_i) for every item $I_i \in$ fi (F) by DS scan
8. The succeeding_items for F, si (F) = { $I_i \in$ fi (F) ∧ SU (F, I_i) ≥ Th }
9. Return **Search** (F, DS, si (F), fi (F), Th)

For performing the depth-first search of F, a recursive search procedure is then carried out. The process is represented in component **Search**. The input parameters are the present item set F, the projected dataset of F, the following and succeeding items of the item set F and the minimum threshold Th. A loop is named for another item set for each item belonging to succeeding items of F, such that Ω = F {Ii}. To compute the utility of the itemset, a dataset scan is performed. In the next step, the projected dataset of Ω is generated. For the itemset, if the utility is greater than or equal to the minimum threshold defined, it can be considered as a HUI. Another time to compute absolute sub tree utility and absolute local utility for each item q belongs to Ω is then visualized in the dataset. The method is then recursively called to resume the search process by expanding it. High utility items are returned as output at the end of the algorithm.

Component Search
Input: F: itemset
F –DS: the projected dataset of F
si (F): succeeding items of F
fi (F): following items of F
Th: minimal threshold
Output: I_{HUI}: itemsets of high utility

1. I_{HUI} = emptyset;
2. **for** every item $I_i \in$ si (F)
   a. Ω = F ∪ { I_i }
   b. Scan F − DS
   c. Compute U(Ω),
   d. Construct Ω –DS // projected dataset of Ω
   e. if U(Ω) ≥ Th,
   f. Ω⟶ I_{HUI}
   g. end
   h. if Ω – DS = empty,
   i. Compute ASU (Ω, q)
   j. Compute ALU (Ω, q) ∀ items q ∈ fi (F) from Ω –DS scan;
   k. si (Ω) = q ∈fi (F) ASU (Ω, q) ≥Th
   l. fi (Ω) = q ∈ fi (F) ALU (Ω, q) ≥ Th ;
   m. I_{HUI} ∪ **Search** (Ω, Ω –DS, si (Ω), fi (Ω), Th);
   n. end
3. end
4. return I_{HUI}

The flow graph for the whole process runs in parallel on various nodes of the Spark cluster.

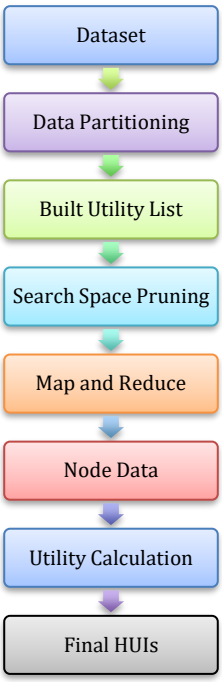

Figure 2 Flow Graph of the Process Working in Parallel Using Spark

Experimental Setup

For experiments, Spark cluster is built with a main node and ten slave nodes on a device with 32 GB main memory with 2 processors x Intel(R) Xeon(R) CPU E5-2620, 6 cores per processor with windows 10 operating system. The following configurations are available for all nodes: Apache Spark 3.0, Python 3.7 and Spyder4 as the IDE.

Datasets

Three real-world datasets are used for the experiments: Chess, Connect, and Mushroom. Chess and Connect are game data sets. Mushroom is a dataset for various varieties of fungi. These datasets have been taken from the SPMF Database (SPMF, 2016), which is a library for multiple algorithms and codes for data mining. In the table 1, the attributes of the datasets are shown. #Transactions, #Items, #AverageItems, reflecting the total transactions in the dataset, the number of specific items, and the mean number of items for the transaction respectively.

Table 1 Various Datasets from SPMF Library

Dataset	#Transactions	#Items	#AverageItems
Chess	3196	75	37
Connect	67557	129	43
Mushroom	8416	119	23

Algorithm Evaluation

The performance of the algorithm AHUIM is being measured on following parameters: A. Execution Time, B. Scalability, C. Accuracy, and D. Stability.

A. *Execution Time:* The time-performance of the algorithm AHUIM on distinctive datasets with various values of threshold can be seen in table 2. AHUIM is around 8 times faster (figure 3) than the conventional Two-Phase algorithm, (Liu, 2005), which is in-built in library of Python as *itemset_mining.two_phase_huim.TwoPhase.*

Table 2 Execution Time (Seconds) of AHUIM for Different Thesholds

Threshold / Dataset	Th = 2	Th = 3	Th = 4
Chess	1.844	1.78	1.32
Connect	6.65	5.44	5.2
Mushroom	4.238	4.62	3.98

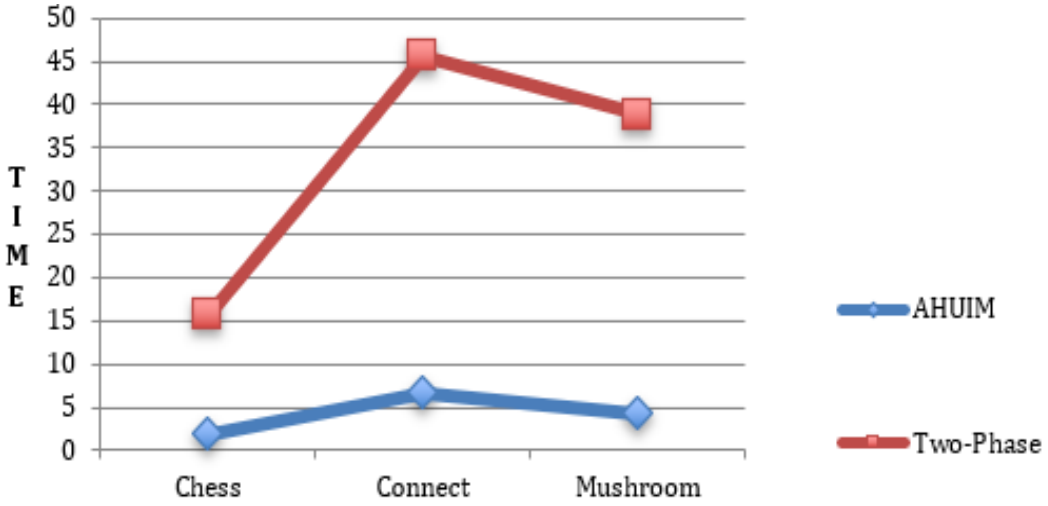

Figure 3 Execution Time (Seconds) for AHUIM and Two-Phase (for Th=2)

B. Scalability: To examine the scalability, two methods have been used. First, the AHUIM algorithm operates on various fixed-size-datasets, but with a varying number of nodes (machines). The effect can be shown with nodes 1, 5 and 10 in figure 4. The execution time of the algorithm decreases approximately linearly with an increase in the number of working nodes. This confirms that the algorithm has the ability to effectively divide the search space between different nodes and independently perform the mining operation.

Table 3 Execution Time (Seconds) with Different Nodes

Nodes / Dataset	Node 1	Node 5	Node 10
Chess	17.97	5.32	1.844
Connect	57.65	29.76	6.65
Mushroom	38.874	18.78	4.238

Table 4 Execution Time (Seconds) of AHUIM on Various Sizes of Datasets

Dataset	AHUIM
Chess	1.844
Chess10x	13.423
Chess20x	24.54
Chess30x	52.546

The second approach for testing the scalability of the algorithm is used by linearly increasing the size of the database to some degree (like 100 percent, 200 percent, 300 percent, etc.). The dataset can be replicated with a scalar d, such as experimental dataset = initial dataset*d, and d = 1, 2, 3, etc. For each of the datasets, the execution time is then calculated, as shown in table 3. It can be perceived from figure 5 that AHUIM's running time races very slowly and almost flat with a rise in datasets. The scalability of the algorithm is verified by both the approaches.

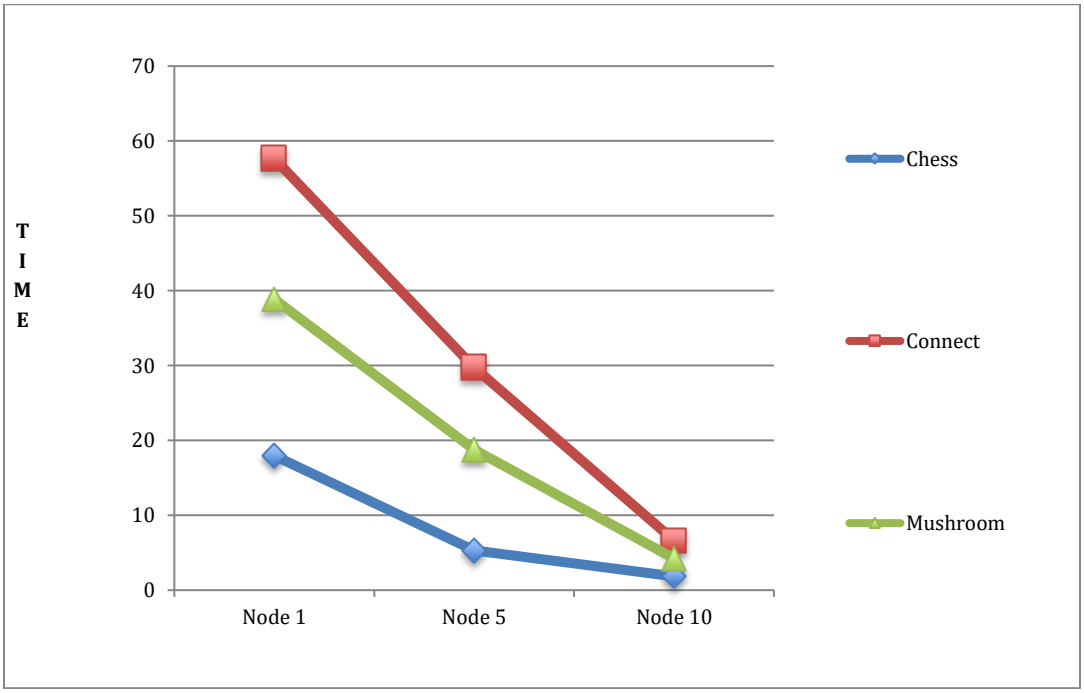

Figure 4 Execution Time (Seconds) with Varying Nodes

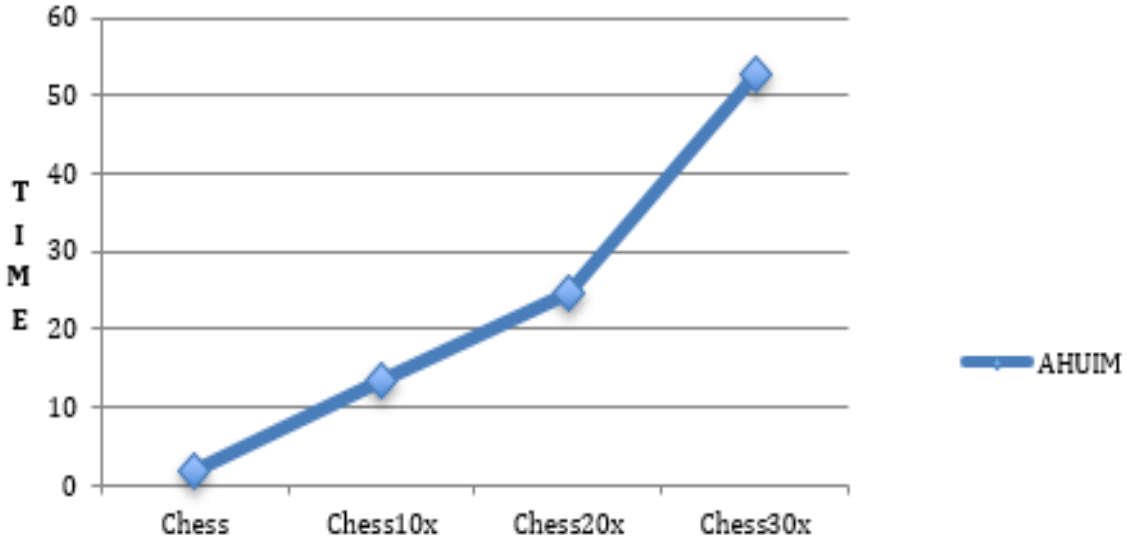

Figure 5 Execution Time (Seconds) for Various Size of Chess Dataset

C. Accuracy: The accuracy in itemset mining algorithms is evaluated by comparing the number of itemsets generated by the algorithm. For that reason, here, an algorithm without any search space pruning strategy is used as an accurate algorithm and the performance of recommended algorithm is compared with it. Here, the Two-Phase algorithm is taken as the ground truth for accuracy. The HUIs generated from the three datasets are shown in the table 5 for the threshold value 2.

Table 5 Numbers of HUIs for Different Datasets

Dataset / Algorithm	Chess	Connect	Mushroom
AHUIM	21	71	42
Two-Phase	22	78	46

From the above table, it can be concluded that the accuracy of the algorithm is around 91.32% (based on the average values of three datasets.)

D. Stability: The stability of the algorithm in itemset mining can be interpreted from the fact as how it executes with various values of thresholds. The algorithm AHUIM performs well for all three-threshold values, demonstrating the algorithm's stability.

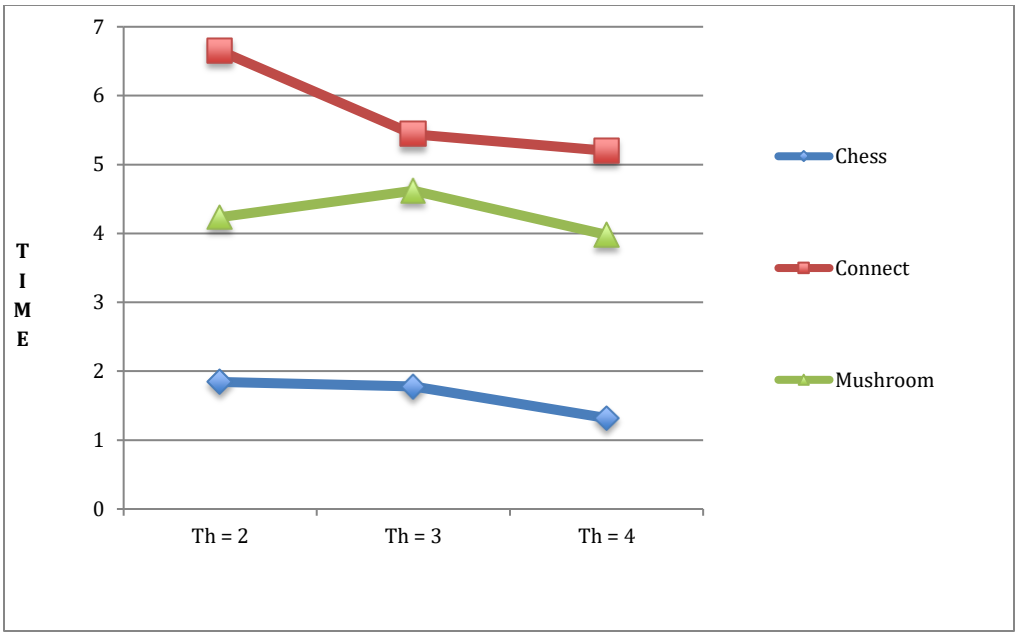

Figure 6 Execution Time (Seconds) of AHUIM with different values of Thresholds on various datasets

Conclusion and Future Work

The conventional UIM algorithms are not suitable for large datasets. A scalable algorithm has been projected in this research work, which mines the itemsets based on some utility factors and is suitable for small as well as large datasets. These itemsets are then used to generate association rules for a database. The system's standalone architecture has been customized with distributed processing. The consistency of AHUIM for time, scalability, accuracy, and stability is advocated by systematic experiments. The novel algorithm can be improved with other characteristics of UIM as part of further analysis, such as mining top-k itemsets, uncommon itemsets, negative utility itemsets, etc.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *In Proceedings of the 20th International Conference on very large data bases, VLDB, 1215*, 487-499.

Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., & Tseng, V.S. (2017). EFIM: a fast and memory efficient algorithm for high-utility itemset mining. *Knowledge and Information Systems*, *51*(2), 595-625.

Tseng, V.S., Shie, B.E., Wu, C.W., & Philip, S.Y. (2012). Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE transactions on knowledge and data engineering*, *25*(8), 1772-1786.

Liu, Y., Liao, W.K., & Choudhary, A. (2005). A two-phase algorithm for fast discovery of high utility itemsets. *In Pacific-Asia Conference on Knowledge Discovery and Data*

Mining, Springer, Berlin, Heidelberg, 689-695.

Chon, K.W., & Kim, M.S. (2018). BIGMiner: a fast and scalable distributed frequent pattern miner for big data. *Cluster Computing*, *21*(3), 1507-1520.

Zihayat, M., Chen, Y., & An, A. (2017). Memory-adaptive high utility sequential pattern mining over data streams. *Machine Learning*, *106*(6), 799-836.

Sethi, K.K., Ramesh, D., & Edla, D.R. (2018). P-FHM+: Parallel high utility itemset mining algorithm for big data processing. *Procedia computer science*, *132*, 918-927.

Nguyen, T.D., Nguyen, L.T., & Vo, B. (2018). A parallel algorithm for mining high utility itemsets. *In International Conference on Information Systems Architecture and Technology*, *Springer, Cham,* 286-295.

Ashish, T. (2017). *High Utility Itemsets Identification in Big Data*. M.S. thesis, University of Nevada, Las Vegas.

Wu, J.M.T., Lin, J.C.W, & Tamrakar, A. (2019). High-utility itemset mining with effective pruning strategies. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, Article, *58*, *13*(6), 1-22.

Dalal, S., & Dahiya, V. (2018). Review of High Utility Itemset Mining Algorithms for Big Data. *Journal of Advanced Research in Dynamical and Control Systems–JARDCS*, *10*(4), 274-283.

Dalal, S., & Dahiya, V. (2019). Various Research Opportunities in High Utility Itemset Mining. *International Journal of Recent Technology and Engineering (IJRTE)*, *8*(4), 2455-2461.

Vandna, D., & Sandeep, D. (2020). Big data Mining: Current Status and Future Prospects. *International Journal of Advanced Trends in Computer Science and Engineering*, *29*(3), 4659- 4670.

Sethi, K. K., Ramesh, D., & Sreenu, M. (2019). Parallel high average-utility itemset mining using better search space division approach. *In International Conference on Distributed Computing and Internet Technology*, *Springer, Cham,* 108-124.

Zihayat, M., & An, A. (2014). Mining top-k high utility patterns over data streams. *Information Sciences*, *285*, 138-161.

Vo, B., Nguyen, H., Ho, T.B., & Le, B. (2009). Parallel method for mining high utility itemsets from vertically partitioned distributed databases. *In International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, Springer, Berlin, Heidelberg, 251-260.

Subramanian, K., Kandhasamy, P., & Subramanian, S. (2013). A novel approach to extract high utility itemsets from distributed databases. *Computing and Informatics*, *31*(6+), 1597-1615.

Lin, J.C.W., Li, T., Fournier-Viger, P., Hong, T.P., Zhan, J., & Voznak, M. (2016). An efficient algorithm to mine high average-utility itemsets. *Advanced Engineering Informatics*, *30*(2), 233-243.

Dahiya, O., Solanki, K., Dalal, S., & Dhankhar, A. (2020). Regression Testing: Analysis of its Techniques for Test Effectiveness. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(1), 737-744.

Ryang, H., Yun, U., & Ryu, K.H. (2016). Fast algorithm for high utility pattern mining with

the sum of item quantities. *Intelligent Data Analysis*, *20*(2), 395-415.

Zida, S., Fournier-Viger, P., Wu, C.W., Lin, J.C.W., & Tseng, V.S. (2015). Efficient mining of high-utility sequential rules. *In International workshop on machine learning and data mining in pattern recognition*, Springer, Cham, 157-171.

Fournier-Viger, P., Wu, C.W., Zida, S., & Tseng, V.S. (2014). FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. *In International symposium on methodologies for intelligent systems, Springer, Cham*, 83-92.

Li, H., Wang, Y., Zhang, D., Zhang, M., & Chang, E.Y. (2008). Pfp: parallel fp-growth for query recommendation. *In Proceedings of the 2008 ACM conference on Recommender systems*, 107-114.

Sandeep, D., & Vandna, D. (2020). A Novel Technique - Absolute High Utility Itemset Mining (AHUIM) Algorithm for Big Data. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(5).

https://hadoop.apache.org/

https://spark.apache.org/

https://www.philippe-fournier-viger.com/spmf. An open source Data Mining Library.