# Midterm Essay

Course: Mining Massive Data Sets

1st Chau Bao Nhan
*Department of Computer Science*
*Ton Duc Thang University*
522h0093@student.tdtu.edu.vn

2nd Nguyen Thanh An
*Department of Computer Science*
*Ton Duc Thang University*
nguyenthanhan@tdtu.edu.vn

***Abstract*—This document is written to report on the midterm assignment for the Mining Massive Data Sets.It includes four tasks involving data manipulation on the ”basket.csv” using method through Pyspark library. We have implement four core taskes:(1) is Apriori algorithm using Hadoop MapReduce to identify frequent pairs, (2) is PCY algorithm for itemset mining and association rule generation,(3) a comparative analysis of MinHashLSH and ... methods for date similarity detection using jaccard similarity and the visualization of the execution time for both algorithms on the provided dataset.**

***Index Terms*—Big data, Mining Massive Data Sets**

## I. Introduction

Task 1 implements the A-Priori algorithm using Hadoop MapReduce (Java) to identify frequent customer pairs shopping together on the same date, executed through two MapReduce passes for candidate generation and pruning.

Task 2 applies the PCY algorithm via PySpark, optimizing frequent itemset discovery with hash-based compression. We generate association rules from baskets stored in Google Drive, detailing our hashing function and bucket management strategy.

Task 3 compares MinHashLSH (PySpark's built-in) and brute-force approaches to detect dates with 0.5 Jaccard similarity in purchased items, analyzing runtime performance across thresholds s [0.0, 1.0].

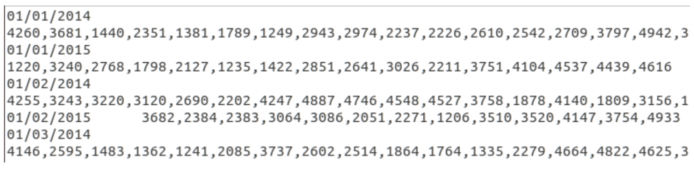
## II. Task 1: A-Priori Algorithm for Frequent Customers

### *A. Generating Group of Customers Going Shopping on the Same Date*

In this section, the A-Priori algorithm has been applied to the 'basket.csv' dataset. The dataset has been processed to create customer groups by day. After grouping the customers by day, the results were recorded in a Hadoop directory using MapReduce programs written in Java. These intermediate results were then utilized to implement the A-Priori algorithm, structured into two layers, each representing one of the two runs of the algorithm.

To design a program that groups customers by date, a MapReduce model has been implemented. The mapper class processes the data by breaking down each line of data, removes the header, and maps it into structured key-value pairs in the form of (date, id_customer). After mapping the data into (date, id_customer) pairs, we process to reduce them by using a set to ensure that only unique key-value pairs are retained, ensuring that no duplicate customers appear on the same day.

In the final step, we pass the input and the output path parameters on the HDFS system to execute the MapReduce program and observe the results from the file part-r-00000. The structure of the results is as follows:

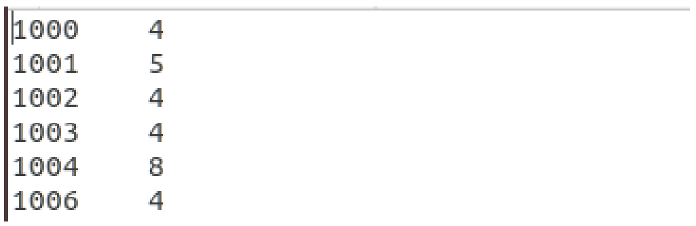


```
01/01/2014
4260,3681,1440,2351,1381,1789,1249,2943,2974,2237,2226,2610,2542,2709,3797,4942,3
01/01/2015
1220,3240,2768,1798,2127,1235,1422,2851,2641,3026,2211,3751,4104,4537,4439,4616
01/02/2014
4255,3243,3220,3120,2690,2202,4247,4887,4746,4548,4527,3758,1878,4140,1809,3156,1
01/02/2015      3682,2384,2383,3064,3086,2051,2271,1206,3510,3520,4147,3754,4933
01/03/2014
4146,2595,1483,1362,1241,2085,3737,2602,2514,1864,1764,1335,2279,4664,4822,4625,3
```

Fig. 1. Structure of group of customer going shopping on the same date

### *B. First Pass of A-Priori: List of all the frequent items in the data set*

After extracting the data, we implement a MapReduce program to perform the First Pass of the Apriori algorithm. Next, the CustomerCountMapper class processes each group of customers by separating them individually and mapping them into key-value pairs. After the mapping step, CustomerCountMapper reads the minimum support threshold of 0.0001. If the total count of an item meets or exceeds the threshold, the customer will be identified as a frequent customer. In the final step, to run the program, the input and the output paths are provided as parameters. The results obtained can be viewed in the file part-r-00000 in the HDFS system. A sample result structure is shown below:



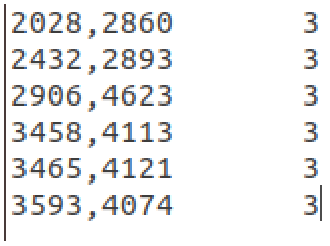
```
1000    4
1001    5
1002    4
1003    4
1004    8
1006    4
```

Fig. 2. Result of Pass1

## C. Second Pass of A-Priori: List of all the frequent pairs in the data set

After obtaining frequent customers from Pass1, we implement a MapReduce program to perform Pass2 of the A-priori algorithm. The PairMapper class begins to reading the file containing the frequent items, determined base on the support threshold in Pass1. These frequent items are stored in a list for further processing. In the mapping phase, the customers are processed by reading their values. Then Nested loops are used to create customer pairs. If both customers are frequent customers, they will be paired together and mapped as key-value pairs in the format id_cus1, id_cus2, support_count.

After mapping, the PairReducer class processes the pairs to reduce them to the format pair_id, pair. Each customer is identified by a unique Member_number. To run the program, the input path, the output path and the path of regular customers (generated from Pass1) are passed in as parameters. The final result is stored in the file part-r-00000 in the HDFS system. The structure of the sample result is shown below:



```
2028,2860	3
2432,2893	3
2906,4623	3
3458,4113	3
3465,4121	3
3593,4074	3
```

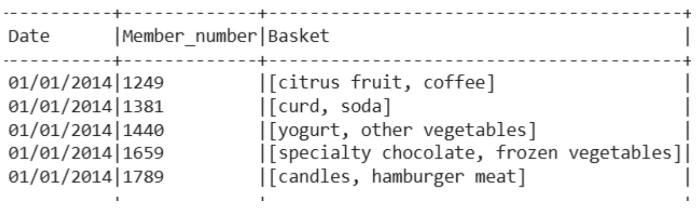
Fig. 3. Result of Pass 2

## III. Task 2: PCY algorithm for Frequent Items

In this section, we apply the PCY algorithm to use PySpark DataFrame on the dataset ”basket.csv”. We process the data to create baskets that represent items purchased by individual customers on specific dates. We then develop a PCY algorithm class that is based on an object-oriented programming model, allowing for flexible definitions of support threshold S and confidence threshold C to identify frequent Item pairs and extract association rules.

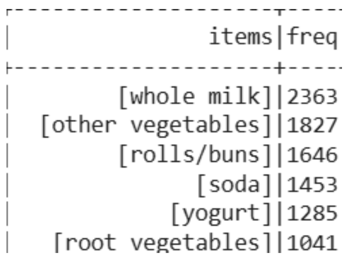
### A. Use Pyspark DataFrames to identify baskets

To find all the baskets, we used PySpark DataFrame to read the data file from the specified path.After that, we grouped the two columns Member-number and Date, aggregating all the items into a list named items. This is .Finally, here is the final output structure:



```
-----------+-------------+--------------------------------------+
Date       |Member_number|Basket                                |
-----------+-------------+--------------------------------------+
01/01/2014|1249          |[citrus fruit, coffee]                |
01/01/2014|1381          |[curd, soda]                          |
01/01/2014|1440          |[yogurt, other vegetables]            |
01/01/2014|1659          |[specialty chocolate, frozen vegetables]|
01/01/2014|1789          |[candles, hamburger meat]             |
```

Fig. 4. Structure of Basket



```
+--------------------+----+
|               items|freq|
+--------------------+----+
|        [whole milk]|2363|
|  [other vegetables]|1827|
|        [rolls/buns]|1646|
|              [soda]|1453|
|            [yogurt]|1285|
|   [root vegetables]|1041|
```

Fig. 5. Structure of the Frequent 1 item

### B. Implement the PCY class

*a) Initialize and fit function:* In the ”init” method, the PCY algorithm is initialized with the transaction dataset, the number of hash buckets, the minimum support and confidence thresholds. These parameters determine how frequent itemsets and association rules are identified. The class maintains several attributes, including the total number of transactions ”num baskets”, a computed minimum support threshold min support, and storage structures such as buckets for hashed item pairs and frequent 1 item for frequent single items. Additionally, the frequent attribute pairs and association rules store the frequent item pairs and generated association rules, respectively. Through the ’fit’ method, the class reads the transactions file mentioned above and converts the item column into an array of strings. Additionally, the class attributes are set to the values returned by the functions defined below.

*b) Item counting function* : The count items function is responsible for counting the occurrences of each item in the transaction dataset. It takes a preprocessed dataframe (customer buy items on a date), which contains to group transaction baskets, and returns a new dataframe with each unique item and its corresponding frequency. To achieve this, the function first ensures that the number of baskets has been set; otherwise, it raises a error, prompting the user to run the *run* method beforehand. The counting process involves exploding the basket column to extract individual items, grouping them by item name, and aggregating their counts. Items appearing at least twice are retained for further processing. This function plays a crucial role in identifying frequent single-item occurrences, which serve as the foundation for subsequent steps in the PCY algorithm, such as frequent pair detection and association rule generation.

*c) Filtering frequent items*: This function filters items base on the minimum support threshold. It calculates the probability of each item by dividing its occurrence count by the total number of transactions. Items that meet or exceed the support threshold are retained. This step ensures that only frequent items are considered for further processing in the pcy algorithm, reducing the number of candidate pairs in the next stage.

*d) Creating the hash table*: This function constructs a hash table to store the frequency of bucket occurrences. It generates item pairs from each transaction using a hash function and assigns them to specific buckets. For each unique pairs, the corresponding bucket's support count is incremented. This step helps to optimize the processing of frequent pairs by reducing the number of candidate pairs that need to be examined in the later stages of the pcy algorithm.

item1id + item2id mod number of buckets

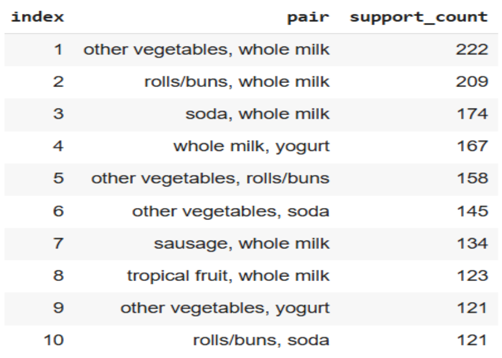
*e) Creating bit map*: This function generates a bit vector for each bucket, where the bit is set to 1 if the bucket's support is greater than or equal to the minimum support threshold, and 0. Otherwise, the result going to be a binary representation of the buckets, where most of the buckets will likely be marked with a 1, indicating that they meet the minimum support requirement. This step further refines the bucket data, facilitate the identification of frequent item pairs and improve the efficiency of subsequent steps in the pcy algorithm.

*f) Linked filter*: This function acts as a link to the filtering process without directly applying the filter itself. It first calls the filter function to retain only the items that meet the support threshold. Then, it creates a dataframe which contain the frequent items along with their support count and probability. The filtered items are sorted in descending order of their probability, ensuring that the most frequent items are prioritized for further processing in the pcy algorithm. This step helps refining the list of candidate items, making the algorithm more efficient by focusing on the most promising candidates.

*g) Count of candidate pairs*: this function counts the frequent pairs in the dataset. It follows several steps to identify and count item pairs that meet the support threshold. Firstly, it generates all possible pairs of items within each transaction, ensure that each pair has a defined order (i.e., x¡y) and remove duplicate pairs where both items are the same. Secondly, it filters these pairs by checking whether they are presented in the set of previously identified frequent pairs. after that, It counts the occurrences of each candidate pair in the dataset. Finally, the function calculates the probability of each pair by dividing its support count by the total number of transactions, filtering out pairs with a probability lower than the support threshold. The resulting frequent pairs are sorted by their support count in descending order, ready for use in the subsequent stages of the pcy algorithm.

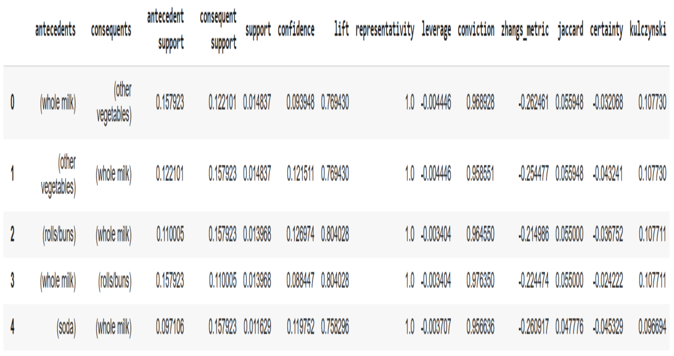
*h) Creating Frequent Pairs*: This function generates a set of frequent item pairs and their corresponding support values. It begins by identifying pairs of items that appear in frequent buckets (where the bit vector is set to 1). Each pair is extracted from the bucket, then sorted and stored as a candidate frequent pair. The function filters out pairs where both items are not frequent, ensuring that only pairs of frequently occurring items are considered. After that, it separates the pair into two distinct columns (item1 and item2) and counts the occurrences of these pairs in the dataset. The resulting frequent pairs are stored with their support counts and are ready for the next stage in the PCY algorithm.

*i) Generate Association Rules*: This function generates association rules from the frequent itemsets. It begins by splitting each frequent pair into two components: the antecedent and the consequent, and then creates the reverse pair (antecedent and consequent swapped) to account for both possible rule directions. The function combines both sets of rules, ensuring that both antecedent → consequent and consequent → antecedent are considered. Next, it calculates the support for both the antecedent and consequent by joining the frequent items dataset and computes the support and confidence for each rule. Finally, the function filters the rules based on the confidence threshold and sorts them in descending order of confidence. The result is a set of association rules, which are ranked by their confidence, ready to be applied in the next steps of the algorithm.



| index | pair | support_count |
|---|---|---|
| 1 | other vegetables, whole milk | 222 |
| 2 | rolls/buns, whole milk | 209 |
| 3 | soda, whole milk | 174 |
| 4 | whole milk, yogurt | 167 |
| 5 | other vegetables, rolls/buns | 158 |
| 6 | other vegetables, soda | 145 |
| 7 | sausage, whole milk | 134 |
| 8 | tropical fruit, whole milk | 123 |
| 9 | other vegetables, yogurt | 121 |
| 10 | rolls/buns, soda | 121 |

Fig. 6. Structure of the Frequent Item Pairs



| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | representativity | leverage | conviction | zhangs_metric | jaccard | certainty | kulczynski |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | (whole milk) | (other vegetables) | 0.157923 | 0.122101 | 0.014837 | 0.093948 | 0.769430 | 1.0 | -0.004446 | 0.968928 | -0.262461 | 0.055948 | -0.032068 | 0.107730 |
| 1 | (other vegetables) | (whole milk) | 0.122101 | 0.157923 | 0.014837 | 0.121511 | 0.769430 | 1.0 | -0.004446 | 0.958551 | -0.254477 | 0.055948 | -0.043241 | 0.107730 |
| 2 | (rolls/buns) | (whole milk) | 0.110005 | 0.157923 | 0.013968 | 0.126974 | 0.804028 | 1.0 | -0.003404 | 0.964550 | -0.214986 | 0.055000 | -0.036752 | 0.107711 |
| 3 | (whole milk) | (rolls/buns) | 0.157923 | 0.110005 | 0.013968 | 0.088447 | 0.804028 | 1.0 | -0.003404 | 0.976350 | -0.224474 | 0.055000 | -0.024222 | 0.107711 |
| 4 | (soda) | (whole milk) | 0.097106 | 0.157923 | 0.011629 | 0.119752 | 0.758296 | 1.0 | -0.003707 | 0.956636 | -0.260917 | 0.047776 | -0.045329 | 0.096694 |

Fig. 7. Structure of the Association Rules

*j) Saving the results*: This function saves the results of the algorithm into multiple files for further analysis. Firstly it stores the frequent individual items along with its support counts in a csv file. Secondly it saves the frequent item pairs, ensure that they are sorted in descending order of support count. In addition, the function exports the association rules in two formats: one as a csv file containing antecedent, consequent, and their respective support and confidence values, and another as a plain text file that lists rules in a more readable format. All files are saved in overwrite mode to ensure that updated results are recorded each time the function is executed.

## *C. Conclusion*

- Task 1: : Implement the A-Priori algorithm using the MapReduce model in Java, run it via HDFS, and validate the results by checking the threshold and frequent item pairs

- Task 2: Use PySpark DataFrames to implement the PCY algorithm to identify frequent pairs and association rules, and check whether the results meet the specified threshold and confidence requirements.

## Acknowledgment

I would like to express our sincere gratitude to MSc.Nguyen Thanh An for his thorough guidance and support, which enabled our group to complete four task of this report.

## References

[1] J. Ullman, "Association Rules and Frequent Itemsets," Stanford University. [Online]. Available: http://infolab.stanford.edu/~ullman/mining/assocrules.pdf

[2] Q. Zhang, "Research on Association Rules Algorithm for Big Data," Ph.D. dissertation, Xi'an Univ. Sci. Technol., Xi'an, China, 2017.

[3] J. Zhou, J. Liu, and C. Yu, "Research and Application of Data Mining Based on Web Log," *Science, Technology and Engineering*, vol. 10, pp. 2762–2766, 2010.

[4] K. Pandey and D. Shukla, "Mining Relationships in Big Data Era Using Improved Apriori Algorithm with MapReduce Approach," in *Proc. IEEE Conf.*, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8933674

[5] Y. Zhu, "Research on the Improvement of Apriori Algorithm Based on Array," Ph.D. dissertation, Harbin Normal Univ., Harbin, China, 2014.