Mining Massive Data Sets Midterm Report

1st 522H0036 - Luong Canh Phong
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0036@student.tdtu.edu.com

2nd 522H0092 - Cao Nguyen Thai Thuan
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0092@student.tdtu.edu.com

3rd 522H0075 - Tang Minh Thien An
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0075@student.tdtu.edu.com

4th 522H0167 - Truong Tri Phong
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0167@student.tdtu.edu.com

5th Instructor: Nguyen Thanh An
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
nguyenthanhan@tdtu.edu.com

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

I am not writing this part.

II. First Task: A-Priori Algorithm for Frequent Customers

A. Overview of MapReduce

1) What is MapReduce: MapReduce is a yarn-based system commonly used for processing massive dataset:

- Performs concurrent processing by dividing the dataset into multiple chunks on the Hadoop commodity servers.
- Instead of sending the data to the machine with the logic to execute, we send the logic to the data to execute, specifically, the server.

2) How MapReduce works: MapReduce is executed in the following order:

- Split: Divides the dataset into multiple data batches.
- Map: Maps every element within each data batch to a $< key, value >$ pair.
- Await Completion: Wait for all data batches to finish mapping the pairs.
- Combine: Generates $< key, value >$ pairs in the form of a list (e.g., $[[A, 1], [A, 1]]$)
- Partition: Determines which reducer should handle each key. It uses a hash function (e.g., hash(key) %num_reducers) to distribute keys evenly.
- Reduce: Processes every data assigned to it and return the output.

B. First subtask

In the first subtask, we are assigned to store the data on Hadoop Distributes File System (HDFS). After which we will implement a Hadoop MapReduce program in Java to discover groups of customers going shopping at the same date.

1) The Mapper Class: CustomerGroupByDateMapper

The Mapper class is responsible for reading input data and emitting key-value pairs. Key aspects of its implementation include:

- Input Processing: The input data is a CSV file with seven columns including Member_number (customer ID) and Date (transaction date).
- Filtering Headers: The Mapper ignores lines where Member_number is a header.
- Emitting Key-Value Pairs: The transaction date is used as the key, and the customer ID is used as the value. This allows all customer transactions on a given date to be grouped together during the shuffle and sort phase.

Example Output from Mapper:

(01/01/2014, 12345)
(01/01/2014, 67890)
(03/01/2014, 54321)

2) The Reducer Class: CustomerGroupByDateReducer

The Reducer class is responsible for aggregating the values emitted by the Mapper for each unique key. Key aspects of its implementation include:

- Collecting Unique Customer IDs: The reducer stores customer IDs in a HashSet to ensure uniqueness.
- Joining Values: The unique customer IDs are converted into a comma-separated string.
- Emitting Results: The final output consists of the transaction date as the key and the list of unique customer IDs as the value.

Example Output from Reducer:

(01/01/2014, 12345,67890)
(03/01/2014, 54321)

3) Driver Program (Main Method): The driver program configures and executes the MapReduce job. It performs the following tasks:

- Setting up the Job: The job is named "Customer Date Groups" and configured to use GroupMapReduce as the main class.
- Setting Mapper and Reducer: The Mapper and Reducer classes are assigned appropriately
- Defining Input and Output: The input and output paths are provided as command-line arguments
- Job Execution: The job is submitted to Hadoop for execution, and the program exits based on its success or failure.

C. Second subtask

In the second subtask, we are assigned to implement the A-Priori algorithm to identify frequent customer pairs in the form of 02 Hadoop MapReduce programs, each corresponding to a pass.

1) The First Pass: Identifying Frequent Individual Customers

- Mapper Class: AprioriFirstPassMapper
  - Function: Reads transaction data and emits each customer ID as a key with a value of 1.
  - Filtering: Skips header lines and ensures valid data is processed.
  - Example Output from Mapper:

    (12345, 1)
    (67890, 1)
    (12345, 1)
- Reducer Class: AprioriFirstPassReducer
  - Function: Aggregates the occurrences of each customer ID.
  - Filtering: Only customers meeting the support threshold (minimum occurrences) are retained.
  - Example Output from Reducer:

    (12345, 2)
    (67890, 1)

2) The Second Pass: Identifying Frequent Customer Pairs

- Mapper Class: AprioriSecondPassMapper
  - Setup: Loads frequent customers from the first pass output using Hadoop's distributed cache.
  - Processing: Reads transactions and filters out customers that did not meet the first pass threshold.
  - Pair Generation: Creates all possible pairs of frequent customers.
  - Example Output from Mapper:

    (12345, 67890, 1)
    (12345, 54321, 1)
- Reducer Class: AprioriSecondPassReducer
  - Function: Aggregates occurrences of customer pairs and filters based on the support threshold.
  - Example Output from Reducer:

    (12345, 67890, 3)

3) Driver Program (Main Method):

- First Pass Execution:
  - Runs the first MapReduce job to determine frequent individual customers.
  - Saves the output for use in the second pass.
- Second Pass Execution:
  - Loads the first pass results as cached data.
  - Runs the second MapReduce job to find frequent customer pairs.

III. Second Task: Park-Chen-Yu (PCY) Algorithm for Frequent Items

A. Overview of PCY

Frequent itemset mining is essential for discovering item associations in transactional data, such as market basket analysis. The PCY algorithm improves efficiency by using hash buckets to reduce the computational cost of finding frequent item pairs. This project applies the PCY algorithm to mine frequent itemsets and generate association rules based on support and confidence thresholds, using PySpark for scalable data processing.

The PCY algorithm is based on two key passes through the data. In the first pass, frequent individual items are identified and counted. In the second pass, frequent item pairs are counted, and hash buckets are used to prune less frequent pairs. The hash function maps item pairs to buckets, and only pairs that have a sufficient bucket count are considered frequent. This approach significantly reduces the number of pair comparisons and improves algorithm efficiency.

- Step 1: Count individual items using the support threshold.
- Step 2: Count pairs of frequent items and hash them into buckets.
- Step 3: Prune item pairs that are not frequent based on the bucket counts.
- Step 4: Generate association rules using the confidence threshold.

B. Implementation Details

1) Data Loading and Preprocessing: Data is loaded using PySpark's `read.csv` function. Each transaction is represented as a basket, and the data is grouped by customer and date. The `collect_set` function is used to create a list of items bought together in each transaction.

2) First Pass: Counting Frequent Items: In the first pass, each item's frequency is counted, and only those items that meet the support threshold are considered frequent. The item counts are stored in a dictionary, sorted in descending order.

3) Second Pass: Counting Frequent Pairs: During the second pass, the algorithm generates pairs from frequent items and counts their occurrences. Hashing is applied to map pairs into buckets, and the bucket counts are used to prune pairs that do not meet the minimum support threshold.

4) Association Rule Generation: For each frequent pair, confidence is calculated as the ratio of the pair's count to the individual item count. Association rules are generated if the confidence meets the given threshold. The rules are sorted by confidence.

C. Experimental Results

The PCY algorithm was executed on a transactional dataset of retail transactions, where each transaction (basket) represented a set of items purchased by a customer. The algorithm was applied with the following parameters:

- Support threshold: 2 (minimum count for items to be considered frequent).
- Confidence threshold: 0.5 (minimum confidence for association rules).

1) Frequent Items: The first pass of the algorithm counted the occurrence of each item in the transactions. The following are the top 30 frequent items identified based on the support threshold:

TABLE I
MOST FREQUENT ITEMS WITH THEIR COUNTS

Item	Count
Whole milk	2363
Other vegetables	1827
Rolls/buns	1646
Soda	1453
Yogurt	1285
Root vegetables	1041
Tropical fruit	1014
Bottled water	908
Sausage	903
Citrus fruit	795
Pastry	774
Pip fruit	734

2) Frequent Item Pairs: In the second pass, the algorithm counted pairs of frequent items across all transactions. The top 30 frequent item pairs, sorted by frequency, are as follows:

TABLE II
FREQUENT ITEM PAIRS WITH THEIR COUNTS

Pair	Count
('Whole milk', 'Other vegetables')	222
('Whole milk', 'Rolls/buns')	209
('Whole milk', 'Soda')	174
('Whole milk', 'Yogurt')	167
('Rolls/buns', 'Other vegetables')	158
('Soda', 'Other vegetables')	145
('Whole milk', 'Sausage')	134
('Whole milk', 'Tropical fruit')	123
('Yogurt', 'Other vegetables')	121
('Rolls/buns', 'Soda')	121
('Yogurt', 'Rolls/buns')	117
('Whole milk', 'Root vegetables')	113

3) Association Rules: Once frequent item pairs were identified, association rules were generated based on the confidence threshold of 0.5. The confidence for each rule was computed by dividing the pair count by the count of the antecedent item. The top 30 association rules, sorted by confidence, are presented below:

TABLE III
VALIDATED ASSOCIATION RULES

Rule	Confidence
Preservation products → Soups	1.00
Kitchen utensil → Pasta	1.00
Kitchen utensil → Bottled water	1.00
Kitchen utensil → Rolls/buns	1.00
Bags → Yogurt	0.50

4) Performance Evaluation: The PCY algorithm effectively identified frequent items, pairs, and association rules with reasonable execution time and memory usage. Given the dataset size, the distributed nature of Spark ensured that the computation was scalable.

- Time Complexity: The use of hashing significantly reduces the complexity of item pair generation, making the PCY algorithm faster than traditional algorithms such as the Apriori algorithm.
- Memory Usage: Memory usage was managed well by leveraging the distributed processing capabilities of Spark.

IV. THIRD TASK: MINHASHLSH FOR SIMILAR DATES

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.