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*—In the age of big data, the ability to mine and extract valuable information from massive datasets can give the user an unparalleled edge against the competition. Therefore, this requirement made by the lecturer is designed to simulate one of the three most fundamental challenges in data mining. Through these series of tasks, we will explore some algorithm implementations and solve different problems as well as explore their trade-offs. Each task is a different algorithm to explore and implement with their corresponding datasets. Through these tasks, we will gain some practical insight and experience in working with these algorithms as well as a better understanding of their pros and cons to be able to cater to each dataset based on their characteristics.**

I. Introduction

This report is divided into three large sections corresponding to the first three tasks provided by the lecturer. We will explore and present our findings while putting the proposed algorithms into practice.

Task 1 proposes utilizing the A-Priori algorithm in a Hadoop MapReduce program to discover groups of customers shopping on the same date as well as interacting with Hadoop Distributes File System (HDFS) to store files. By applying these methods, we will be able to understand how to extract patterns from large datasets locally.

The second task focuses on implementing the Park-Chen-Yu (PCY) algorithm using Object-Oriented Programming (OOP) principles and PySpark DataFrame to identify frequent item pairs and generate association rules from customer purchase data stored in Google Drive. The implementation, while generating association rules, also has to follow object-oriented programming principles inspired by PySpark's Frequent-Pattern Growth (FPGrowth) class.

In the third task, we will implement and compare the MinHashLSH algorithm and an alternative of our choice - in this case, a manual method of calculating Jaccard distance. Both of these approaches should achieve the same goal of searching for similar pairs of dates where the Jaccard distance is above a predetermined threshold. After that, we will visualize their runtime with their threshold ranging from 0 to 1 with 0.1 increments to gauge their performance and outline some characteristics between both approaches. Through these implementations, we demonstrate practical applications of data mining techniques with a given dataset. With these findings, we highlight the trade-offs between various aspects across different algorithms within the given time and constraints.

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

A. Overview of A-priori Algorithm

1) What is the A-Priori Algorithm: The A-Priori algorithm is a classic algorithm in data mining used to identify frequent itemsets and derive association rules. In the context of this project, it is used to discover pairs of customers who frequently shop together.

- It is based on the principle that all subsets of a frequent itemset must also be frequent.
- It uses a level-wise, breadth-first search approach to count the frequency of itemsets.
- It is typically implemented in multiple passes: the first pass finds frequent individual items (1-itemsets), and subsequent passes find larger frequent itemsets (e.g., 2-itemsets, 3-itemsets, etc.).

2) How the A-Priori Algorithm Works (in this task):

- **First Pass – Frequent Individual Customers:** Count how many times each individual customer appears in the grouped transaction data. Customers with frequency above a defined threshold are considered frequent.
- **Second Pass – Frequent Customer Pairs:** For each transaction (i.e., group of customers on a given date), generate all possible customer pairs where both customers are frequent. Count the number of times each pair appears.
- **Support Threshold:** A predefined threshold used to filter out itemsets (customers or pairs) that do not appear frequently enough to be considered relevant.
- **Output:** The algorithm outputs customer pairs that occur together at or above the support threshold.

B. First subtask

In the first subtask, we are assigned to store the data on 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 the input data and emitting key-value pairs. Key aspects of its implementation include:

- Input Processing: The input is a CSV file where each line contains multiple fields, including `Member_number` (customer ID) and `Date` (transaction date).
- Filtering Headers: The Mapper skips header lines by checking if the first token equals `Member_number`.
- Emitting Key-Value Pairs: The key is the transaction date, and the value is the customer ID. This allows transactions to be grouped by date in the shuffle and sort phase.

Example Output from Mapper:

(01/01/2014, 11111)
(01/01/2014, 22222)
(02/01/2014, 11111)

2) The Reducer Class: CustomerGroupByDateReducer

The Reducer aggregates the values emitted by the Mapper for each unique date. Key implementation features include:

- Collecting Unique Customers: Customer IDs are added to a `HashSet` to remove duplicates.
- Joining Values: The set of unique customer IDs is converted to a comma-separated string.
- Emitting Results: The final output consists of the transaction date as the key, and a list of unique customer IDs as the value.

Example Output from Reducer:

(01/01/2014, 11111,22222)
(02/01/2014, 11111)

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

- Job Setup: A Hadoop job is created with the name `Customer Date Groups`, using `GroupMapReduce` as the main class.
- Mapper and Reducer Assignment: The appropriate Mapper and Reducer classes are set. The Reducer is also used as a Combiner.
- I/O Paths: The input and output paths are taken from command-line arguments.
- Job Execution: The job is submitted to Hadoop, and the program exits based on success or failure.

C. Second subtask

In the second subtask, we implement the A-Priori algorithm to identify frequent customer pairs. This is achieved using two MapReduce passes.

Note: The output of the first subtask (Customer Grouping by Date) will serve as the input for both passes of the second subtask (A-Priori Algorithm):

- First Subtask (Customer Grouping by Date):
  - `args[0]` – Input path to the raw transaction CSV file.
  - `args[1]` – Output path where grouped customer data by date will be written.
- Second Subtask (A-Priori Algorithm):
  - `args[0]` – Input path to the grouped customer data (output from the first subtask).
  - `args[1]` – Output path for the first pass (frequent individual customers).
  - `args[2]` – Output path for the second pass (frequent customer pairs).

This approach allows the workflow to seamlessly transition from the first subtask (grouping by date) to the second subtask (identifying frequent customers and pairs).

1) The First Pass: Identifying Frequent Individual Customers

- Mapper Class: *AprioriFirstPassMapper*
  - Function: Reads grouped customer data (output of first subtask), splits the customer list, and emits each customer ID with a value of 1.
  - Input Format: Each line is a tab-separated pair where the key is a date and the value is a comma-separated list of customers.
  - Filtering: Skips malformed lines where the customer list is missing.
  - Example Input:

    01/01/2014 12345,67890
  - Example Output from Mapper:

    (12345, 1)
    (67890, 1)
- Reducer Class: *AprioriReducer*
  - Function: Aggregates the counts for each key (customer).
  - Filtering: Emits only `<key, value>` pairs whose frequency is greater than or equal to the support threshold.
  - Example Output from Reducer (First Pass):

    (12345, 2)
    (67890, 1)

2) The Second Pass: Identifying Frequent Customer Pairs

- Mapper Class: *AprioriSecondPassMapper*
  - Setup: Loads the list of frequent customers from the first pass output using Hadoop's distributed cache.
  - Processing: For each transaction line, splits the list of customer IDs, filters only frequent customers, and generates all valid customer pairs.
  - Emitting: Outputs each pair of frequent customers with a count of 1.
  - Example Output from Mapper:

(12345,67890, 1)
(12345,54321, 1)

- Reducer Class: *AprioriReducer*
  - The same class as used in the First Pass.
  - Example Output from Reducer (Second Pass):

    (12345,67890, 3)

3) Driver Program (Main Method):

- First Pass Execution:
  - The first MapReduce job is run with `AprioriFirstPassMapper` to compute individual customer frequencies.
  - The output is saved and later loaded into memory for the second pass.
- Second Pass Execution:
  - The second job uses `AprioriSecondPassMapper`, which loads the frequent customers using Hadoop's cache mechanism.
  - It then computes the frequency of customer pairs and applies the support threshold.

4) Helper Method: createJob

A reusable helper method named `createJob` is implemented to reduce code repetition when setting up MapReduce jobs.

- Parameters:
  - `jobName`: A string representing the name of the job.
  - `mapperClass`: The class to be used as the Mapper.
  - `inputPath`, `outputPath`: Paths for input and output directories.
- Functionality:
  - Configures the job with the specified name and sets the `AprioriReducer` as both the Combiner and Reducer.
  - Assigns key and value output types and adds file paths.
  - Returns a configured `Job` instance ready for execution.

III. Second Task: PCY Algorithm for Frequent Items

A. Overview of PCY

Frequent itemsets 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:

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:

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 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

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

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

A. Approach

Firstly, we will go through the theoretical basis and its possible implementation in the context of our task, after which, we will see the algorithm in action.

B. Theoretical basis

The core of MinHashLSH algorithm is the utilization of two concepts: MinHash signatures and locality-sensitive hashing (LSH) to create and effective algorithm for detecting similar sets. The first concept describes the process of hashing the dataset into a more manageable "signature" for each set. While the latter describes how these "signatures" will be stored to achieve the expected result.

TABLE III
VALIDATED ASSOCIATION RULES

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

C. Jaccard distance and Shingling

Before performing any calculations to any data, we must convert the raw data into a distinguished vector before using any distance calculation between the two sets to check for their similarity. Shingling performs this task by breaking down text data into smaller units to create "shingles" before hashing them into their representation in the form of a binary vector.

The Jaccard distance describes the similarity between two different sets and is represented by the following formula:

$$d_J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} = 1 - J(A,B) \qquad (1)$$

With the result ranging from 0 to 1, we can determine whether the sets in question are related to each other or not for bucket assignment.

D. MinHash Signature

To determine if the pairs are similar or not, we need a way to convert the raw data into usable data for the algorithm to process. In this case,MinHashing is used to perform this task. The binary vector, created through a process of shingling, is converted into a signature vector. Note that if these sets 2 are similar, their signature vector will also have some similarities, and these can be utilized by the algorithm to sort these sets into their suitable buckets.

E. Locality Sensitive Hashing

The idea of LSH when dealing with the problem of finding similar pairs or sets is to maximize the probability of collision in a bucket due to that fact that the hash-code for these sets would be indifferent (if these sets are the similar) and therefore they should be in the same bucket. We can achieve this by breaking the hash-code down even more into subsequences of hash-code that has a higher chance of being similar, giving us a higher chance of finding similar pairs, improving the effectiveness of the algorithm at solving the problem.

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.