Data Compression

Submitted by:

Avinash Anand (22053063)

Submitted to:

Prof. Vijay Kumar Meena

Transaction Data Compression

The aim of this project is to write an efficient code which compresses a dataset of transactions.

This project consists of three parts.

1)Part 1 – This code compresses the original dataset and also displays the mapping. It also tells the total no. od transactions, total number of unique items and top 10 most frequent items.

2)Part 2 – This part removes the mapping keys which are not used to represent any data in the compressed dataset.

3)Part 3 – This code runs both the 1st code and 2nd code together. It also tells us execution time that was taken to compress the data.

Part 1

A report on dataset compression and the FP-Growth algorithm.

1. Overview :- The FP-Growth algorithm implementation for frequent itemset mining is covered in this paper. Using transactional data the algorithm constructs an FP-Tree mines frequently occurring itemsets uses these itemsets to compress the dataset and stores the compressed data along with a mapping of itemsets to identifiers.
2. Function Descriptors:-
   2.1 FPTree Class: The FPTree class is used to represent the nodes in an FP-Tree. Each node in the class has references to its parent and child nodes as well as a value (item) and count (frequency). In order to enable easy access during mining it also maintains links for nodes that have the same item.
   2. 2: Including Transactions: Add new transaction to the FP-Tree recursively using add_new_transaction. The function updates the links in the header table by adding a new child node or increasing the count of an already-existing node.
   2.3: Using a Conditional Prefix : apply_conditional_prefix: Retraces paths from nodes containing an item back to the root in order to construct the conditional pattern base. Building conditional FP-Trees for additional mining is made easier by these patterns.

2. 4: Building the FP-Tree and Header Table: build_hd_table: Builds a header table by calculating item frequencies throughout the dataset. Only products that satisfy a minimal support requirement are listed. Creates an FP-Tree by filtering and sorting transactions based on frequency (create_fp_tree). Mining frequently occurring itemsets is primarily done with this tree.
2.5: Exploiting the FP-Tree: mine_fp_tree: Finds every frequent itemset by recursively mining the FP-Tree. It creates conditional FP-Trees expands the prefix path and mines them to uncover more patterns.
2.6: Condensing the dataset: compress_fp_growth: Manages the header table creation FP-Tree construction and frequent itemset mining for the FP-Growth process. The key elements of the algorithm are connected by this function.
2.7: Bind and save the compressed dataset : bind_cmp_map: Replaces itemsets in the original dataset with these identifiers to map each frequent itemset to a unique identifier (such as A, B or C). This process compresses the dataset. save_cmp_set: This function records how the dataset was reduced by saving the compressed dataset and the mapping of itemsets to identifiers in a text file.
2. 8: Loading and Examining the Information: load_raw_data: This function loads the dataset from a file where a transaction is represented as a set of items on each line. analyze_dataset: Gives the datasets fundamental statistics such as the quantity of transactions the number of unique items and the most common items. Prior to processing this analysis aids in understanding the dataset.
2.9: Main: Coordinates the dataset loading analysis FP-Growth algorithm execution dataset compression and result storage. 3. To sum up. FP-Growth is efficiently implemented by the provided code which also uses frequent itemsets to compress the dataset. Offering a workable solution for frequent itemset mining and dataset compression the functions are well-structured to handle every stage of the process from data loading to result storage.

Part 2

Report on Code for Removing unused mapping keys
1. Overview: By eliminating unnecessary or erroneous mappings between identifiers and itemsets the provided code is intended to

process a compressed dataset. The refined dataset is saved to a new file after the compressed dataset has been loaded and the mapping has been updated.

2. Function Descriptors.

2.1:rem_invalid_keys : This functions goal is to filter the mapping dictionary by eliminating any keys that are absent from the compressed dataset. It guarantees that only itemsets that are genuinely used in the transactions are included in the mapping.

2.2 : save_cmp_data. Goal: This function saves the updated mapping and the cleaned compressed dataset to an output file. It records every transaction in an easily readable format for humans along with the matching mapping of itemsets to identifiers.

2.3: load_cmp_map : This function loads the mapping and compressed dataset from a given file. The fileset-to-identifier mapping and dataset are rebuilt by parsing the file.

2.4: Main: The workflow is orchestrated by the primary function. Following the removal of invalid keys from the mapping it loads the compressed dataset and mapping from an existing file and saves the updated data to a new file. It outputs an error message if the input file cannot be located.

In summary. By eliminating unnecessary mappings and preserving the refined dataset this code efficiently updates a compressed dataset. By guaranteeing consistency between the mapping and the dataset it enhances both the compressed datas readability and storage.

Part 3

This code runs both the 1st code and 2nd code together. It also tells us execution time that was taken to compress the data.

After running the code we find out that the program took

Execution time: 0.8226470947265625

The compression ratio of the dataset is approximately 1.84. This means the original dataset is about 1.84 times larger than the compressed dataset.

Total Transactions: 3196

Total Unique Items: 75

Top 10 Most Frequent Items: [('58', 3195), ('52', 3185), ('29', 3181), ('40', 3170), ('60', 3149), ('36', 3099), ('7', 3076), ('62', 3060), ('34', 3040), ('56', 3021)]