Project Plan Description

Introduction

Abstract

In class, we were introduced to the problem of frequent itemset mining. In simplest form, the goal of frequent itemset mining is to discover *sets of items* (itemsets) *that appear frequently* (defined by support). One of the biggest limitations of frequent itemset mining is that it assumes that all itemsets are equally valuable. For instance, suppose that in a dataset of grocery items, we find that the itemset $\{milk, bread\}$ is very frequent. This transaction, while frequent, does not really have any value to us since these are both lower cost items in comparison to, for instance, $\{champagne, caviar\}$.

This limitation called for a generalization of the frequent itemsets problem to **high utility itemset mining.** The fundamental difference between frequent itemset mining and high utility itemset mining is that when we are performing high utility itemset mining, every item has an associated weight describing how profitable the item is. High utility itemset mining also considers the quantity of the items in the database.

In this project, we plan to provide a preliminary introduction to the field of high utility itemset mining by implementing and evaluating the performance of two separate high utility itemset mining algorithms.

Definitions

Before continuing with our algorithms, we define terminology that is essential to understand our algorithms and their performance.

1. **Transaction Database:** A set of records (transactions) indicating the items purchased by customers at different times. Typically, transactions are represented as rows in a dataset.
2. **Quantitative Transaction Database:** A transaction database that includes the quantities of items in transactions and weights indicating the relative importance of each item to the user.
3. **Support measure:** The support of an itemset X in a transaction database D is defined as $sup(X) = |\{T \mid X \subseteq T \wedge T \in D\}|$, or the number of transactions that contain X.
4. **Frequent itemset:** An itemset is frequent if its support $sup(X)$ is no less than the *minsup* threshold.
5. **Utility:** The utility of an item i in a transaction T_c is denoted as $u(i, T_c)$ and is defined as $u(i, T_c) = p(i) \times q(i, T_c)$, where $p(i)$ denotes the external utility, or weight, of the item, and $q(i, T_c)$ denotes the internal utility, or the quantity of i in the transaction T_c.
6. **High-utility itemset:** An itemset X is a *high-utility itemset* if its utility $u(X)$ is no less than a user-specified minimum utility threshold *minutil* set

by the user.

7. **The TWU measure:** The *transaction utility* (TU) of a transaction T_c is the sum of the utilities of all the items in T_c. The *transaction-weighted utilization* (TWU) of an itemset X is defined as the sum of the transaction utilities of transactions containing X.

Our **problem definition** for high-utility itemset mining is to discover all high utility itemsets, where an itemset X is a *high-utility* itemset if its utility $U(X)$ is no less than a user-specified minimum utility threshold *minutil*. We otherwise classify X as a *low-utility* itemset.

Algorithms

Abstract: Short Description of the Two Algorithms

Our baseline is the Two-Phase algorithm, and we are comparing it to the Fast High-Utility Miner (FHM) algorithm. The Two-Phase algorithm first generates a set of candidate high utility itemsets by overestimating their utilities in phase 1 using a breadth-first search approach. The algorithm overestimates the utilities of the itemset using the TWU of an itemset. Again, the TWU of an itemset is the sum of the transaction utilities of an itemset. Thus, using the property that the TWU is a monotone upper-bound on the utility measure, the Two-Phase algorithm can prune itemsets in the search space since the supersets of an infrequent itemset are infrequent. Therefore, only the combinations of high TWU itemsets are added into the candidate set at each level during the level-wise search. In phase 2, the algorithm performs only one extra database scan to filter the overestimated itemsets.

The FHM algorithm uses a depth-first search approach to prune the search space and eliminate low utility itemsets. The algorithm first scans the database to calculate the TWU of each item. Then, the algorithm identifies a set of all items having a TWU greater than the minimum utility threshold specified by the user. A second database scan is done to build a utility list and to build an Estimated Utility Co-Occurrence Structure (a set of triples of the form $(a, b, c) \in I^* \times I^* \times \mathbb{R}$ such that $TWU(a, b) = c$). After the utility list construction, a depth-first search is done to calculate each itemset's utility and explore its extensions and prune the search space using the Estimated Utility Co-Occurrence Structure (EUCS). The pruning condition is that if there is no tuple (a, b, c) in the EUCS such that $c \geq$ *minutil*, then the itemset and all its supersets are low utility itemsets and do not need to be explored.

Two-Phase

The Two-Phase algorithm is a high-utility itemset mining algorithm that works similarly to the Apriori algorithm. To mine frequent itemsets, the Apriori algorithm prunes the supersets of infrequent itemsets. The Two-Phase algorithm prunes supersets of itemsets whose *transaction-weighted utility* is less than *minutil* using the following properties: $TWU(X) \geq U(X)$ and $TWU(X) \geq U(Y) \forall Y \supset X$. If the algorithm takes in a quantitative transaction database and a *minutil* threshold as input. The first phase of this algorithm calculates the TWU of the necessary itemsets, pruning itemsets that can't be high-utility itemsets, and the second phase calculates the exact utility of the candidate high-utility itemsets.

The first phase of the algorithm starts by calculating the TWU of all itemsets with length 1. It then performs a breadth first search to find larger itemsets that could have high utility. To get candidates at level P_k it generates the possible itemsets from the itemsets in P_{k-1} that are candidate high-utility itemsets. Then, any itemset in P_k that is a superset of a low utility itemset from P_{k-1} is removed. The TWU of the remaining itemsets are calculated and itemsets with $TWU \geq minutil$ are candidate high-utility itemsets. For example, if for a dataset, the 1-itemsets with $TWU \geq minutil$ are $\{a\}$, $\{b\}$, and $\{d\}$, the TWU of the following 2-itemsets will be checked: $\{a, b\}$, $\{a, d\}$, and $\{d, b\}$. If $\{a, d\}$ is a low-utility item it will not be used to generate 3-itemsets, however the 3-itemsets generated, $\{a, b, d\}$ contains $\{a, d\}$, therefore $\{a, b, d\}$ is removed, and the TWU is not checked, as $\{a, b, d\}$ can't be a high-utility itemset. The second phase scans the database and calculates the *utility* of all candidate high-utility itemsets that are generated from phase one.

The Two-Phase algorithm has a few drawbacks to consider. One limitation to consider is that the bound provided by using the TWU of itemsets is a very loose upper bound on the utility of itemsets. This means the algorithm could have to store many itemsets and calculate the utility of many itemsets with low-utility. Another drawback of the Two-Phase algorithm is it generates possible itemsets without looking at the database. This means it could be calculating the TWU of itemsets that do not appear in any transactions.

FHM

FHM is a one-phase algorithm for high-utility itemset mining. The main algorithm takes a quantitative transaction database and the *minutil* threshold as input. Then, FHM scans the database to calculate the TWU of each item and creates the set I^*, which contains all items having a TWU no less than *minutil*. We define a total order $\succ$ as the order of ascending TWU values. Another database scan is performed, where items in transactions are reordered according to $\succ$, the utility-list of each item in I^* is built, and a structure named EUCS (Estimated Utility Co-occurrence Structure) is created.

A utility-list for an itemset is a set of tuples for each transaction. Each tuple is of the form $(tid, iutil, rutil)$, where *tid* is the transaction ID, *iutil* is the utility of the itemset in the transaction, and *rutil* is the total utility of all the items in the transaction that have a TWU greater than those in the itemset. As we will seen, utility-lists allow us to quickly calculate the utility of an itemset and upper-bounds on the utility of its supersets. Additionally, utility-lists for itemsets with size greater than 1 can be quickly created by joining utility-lists of smaller itemsets. The EUCS is defined as a set of triples of the form $(a, b, c) \in I^* \times I^* \times \mathbb{R}$ such that $\text{TWU}(\{a, b\}) = c$. The EUCS is the main novelty in FHM which allows for the pruning mechanism named EUCP (Estimated Utility Co-occurrence pruning) that will be mentioned later.

After the EUCS is created, a recursive depth-first search of the itemsets is performed. The search algorithm, called *FHMSearch*, takes as input (1) an itemset P, (2) a set of extensions of P with the form Pz where Pz was created by appending item z to P, (3) *minutil*, and (4) the EUCS. The first call to *FHMSearch* gives an empty set for the itemset, I^* for the set of extensions of the itemset, *minutil*, and the EUCS. *FHMSearch* executes as follows. For each extension Px of P, if the sum of the *iutil* values for Px's utility-list (which is equal to the utility of Px) is no less than *minutil*, then Px is a high-utility itemset and it is output. We do not consider extensions of P that have utilities less than *minutil* because these extensions are by definition low-utility itemsets. Next, if the sum of *iutil* and *rutil* values in Px's utility-list are no less than *minutil*, then the extensions of Px are explored. We do not explore extensions of Px if the sum of *iutil* and *rutil* values in Px's utility-list is less than *minutil* because the extensions of Px and their supersets are low-utility itemsets. We then explore the extensions of Px by considering all extensions Py of P such that $y \succ x$. If there exists (x, y, c) in the EUCS (i.e., $\text{TWU}(\{x, y\}) = c$) such that $c \geq \mathit{minutil}$, then we merge Px with Py to generate extensions of the form $Pxy = P \cup \{x, y\}$. We do not generate Pxy if c is less than *minutil* because this would mean that Pxy and all its supersets are low-utility itemsets. This step is key to EUCP as it avoids the costly join operation to calculate the utility-list of an itemset that is detailed below.

To construct the utility-list of Pxy the *Construct* algorithm is called to join the utility-lists of P, Px, and Py. *Construct* takes P, Px, and Py as input an executes as follows. The utility-list of Pxy is initialized as an empty set. Next, for each tuple ex in the utility-list of Px, if there exists a tuple ey in the utility-list of Py such that $ex.tid = ey.tid$, and the utility-list of P is empty, then exy − the tuple for Pxy's utility-list − is formed as $(ex.tid, ex.iutil + ey.iutil, ey.rutil)$. Note that P being empty implies that $Px = \{x\}$ and $Py = \{y\}$. If the utility-list of P is non-empty, then we search for the tuple e in the utility-list of P such that $e.tid = ex.tid$ and create exy as $(ex.tid, ex.iutil + ey.iutil - e.iutil, ey.rutil)$. After exy is created, it is appended to the utility-list for Pxy. Once we have considered all the tuples in the utility-list of Px, we return the utility-list of Pxy, which terminates the execution of *Construct*.

After we have created the utility-list of Pxy, a recursive call to *FHMSearch* with Pxy is done to calculate its utility and explore its extension(s). Starting with single items, *FHMSearch* recursively explores the search space of itemsets by appending single items until all high-utility itemsets are discovered.

Experiments

The basis for our experiments came from previous research literature on both Two-Phase and FHM specifically. Concretely, we plan to test the following:

1. **Execution time:** Obviously, one of the most important criteria to test would be execution time of the algorithms. Especially as the data scales larger, it is important that our algorithms perform as quickly as possible. To do this, we will implement a similar timing procedure to HW1 on Frequent Items Mining using `timit()`.
2. **Pruning effectiveness:** One of the major differences between FHM and Two-Phase is the implementation of pruning through a mechanism named EUCP (Estimated Utility Co-occurrence Pruning), which relies on a structure called the EUCS, as previously mentioned. To do this, we will measure the percentage of candidates pruned for each dataset, and attempt to see if a relationship exists between pruning and runtime.
3. **Memory overhead:** We should also be concerned about memory for both the Two-Phase and FHM algorithms. We can do this by monitoring the memory footprint of each algorithm. The library `psutil` can be used to show the memory footprint of a particular program.
4. **Scalability:** We also want to make sure that our algorithms perform well as the number of transactions increases. To do this, we have chosen datasets of intentionally different sizes. We also intend to vary the number of transactions for each dataset while setting the minimum utility parameter to observe the influence of the number of transactions on execution time.

Datasets

We plan to evaluate the performance of our algorithms using three different datasets, each of varying size and complexity, and each of which are datasets that we have not encountered to date. The source for all datasets can be found by clicking *here*.

The three datasets are:

1. **foodsmart:** This dataset represents customer transactions from a retail store. The dataset has 4,141 transactions, 1,559 items, and an average item count of 4.42 per transaction.
2. **chainstore:** This dataset represents customer transactions from a major grocery store chain in California, USA. The dataset has 1,112,949 transactions, 46,086 items, and an average item count of 7.23 per transaction.
3. **Fruithut:** This is a dataset of customer transactions from a US retail store focusing on selling fruits. The dataset contains 181,970 transactions, 1,265 items, and an average item count of 3.58 per transaction.

Work

- Write code for the Two-Phase algorithm.
- Write code for the FHM algorithm.
- Write experiment scripts to evaluate both algorithms.
- Write content for the final video.
- Produce the final video.

Plan

Submit by phase 3:

- Code for the Two-Phase algorithm.
- Code for the FHM algorithm.

Submit by phase 4:

- Code for the Two-Phase algorithm.
- Code for the FHM algorithm.
- Code for the experiment scripts.
- Results of experiments.
- Final video.

We will submit our materials via a compressed file for phase 3 and 4.

Logistics

We have a GitHub repository that can be found by clicking *here*. The repository is where we share code and data among the members of the group.

We plan to split up the work as follows. The first member's main responsibility is to write the code for the Two-Phase algorithm. Similarly, the second member's main responsibility is to write the code for the FHM algorithm. The third and fourth members' main responsibilities are to write the experiments that evaluate the two algorithms. All members are also responsible for producing the final video.

The timeline for our project is described below:

- April 19 (end of phase 3): finish writing the Two-Phase and FHM algorithms.
- April 30: finish writing experiment scripts and evaluating both algorithms.
- May 7: finish writing content for the final video.
- May 21: finish making the final video.