CPSC4310/5310/7310 - Spring 2023
Data Mining and Deep Learning

Course Project

Due: TBA

In this course project, you are required to use C++ or Python to design, implement and improvethe Apriori algorithm that we have learned in class. A practical and efficient implementation of the Apriori algorithm would be too big to fit into a semester's work. Therefore, we limit it with reasonable assumptions and simplifications in our implementation

Part I

1. Introduction

This is a group project. Each group can have at most *four* students and each group must finish the project independently. All the members in a group will be awarded the same mark. So it is your group's responsibility to make sure that each member in your group contributes equally to the project.

The project consists of implementing and improving the Apriori algorithm for finding frequent itemsets in a transaction database.

In addition to the original Apriori algorithm, we have also discussed two other ideas to improve it.

Idea 1: when scanning the database, if we find two k-itemsets that are already frequent, we generate a candidate (k+1)-item from them (if possible) and start collecting its frequency when we continue scanning the database. This way, we hope to reduce the number of times scanning the database.

Idea 2: we divide the database into equal partitions and try to find those local frequent itemsets in each partition and then check them to see whether they are also global frequent for the whole database.

Please read the relevant Notes on the Moodle to see these two ideas.

For undergraduate students, you are required to design and implement the original Apriori and Idea 1.

For undergraduate students, you are required to design and implement the original Apriori, Idea 1 and Idea 2.

2. Project Details

2.1 Databases

Assume that there are only 100 items in the database (an over-simplified database) and they are already symbolized into a uniform format: i0, i1, i2, …, i99. (The lexicographic order of them is i0 < i1 < i2 < … < i99.)

You will create four databases using those items. The names of the four databases are D1K, D10K, D50K, D100K, and their sizes are 1000 transactions, 10000 transactions, 50000 transactions, and 100000 transactions, respectively. They are stored as four files.

The general process of creating a transaction is a random one. Let's call the function that does this GenTransactio*n*(int *num*), where the parameter *num* is a random integer between 5 and 15. The function will randomly pick *num* items from i0, i1, i2, …, i99 and add them to a transaction. The function returns a transaction using a data structure at your jurisdiction. For instance GenTransaction(6) would return {i9, i10, i40, i45, i80, i92}. For each database, you randomly generate the required number of transactions and add them to it. For instance, the following could be some portion of a transactional database you are going to generate. (You can use either ',' or ' ' to delimit the items in a transaction.)

…
i9, i10, i40, i45, i80, i92
i10, i31, i34, i45, i63, i89, i99
i3, i7, i12, i15, i16, i17, i29, i33, i45, i89, i98
…

The program to create the four transactional databases described as above is called GenDatabase.cc or GenDatabase.py. (Of course, you can have some auxiliary files, such as *.h files, if needed.)

2.2 Implementing Apriori (for both undergraduate and graduate students)

Read the Apriori algorithm carefully in the Notes on the Moodle and implement it. You can use library data structures to store items, transactions, itemsets, etc. The program to implement the Apriori algorithm is called apriori.cc or apriori.py. (Of course, you can have some other auxiliary files, such as *.h files, if needed.)

There is a function in apriori.cc or apriori.py called

itemsets apriori(db, float ms) // Students using Python may need to change it slightly.

The function generates and returns the frequent itemsets from the database *db* using the minimum support *ms*. The data types of the parameter *db* and the required returned result are not specified on purpose. This way, you can have your freedom to choose your own data structures to represent them.

Your program saves the frequent itemsets into a file whose name corresponds to the name of the database you have passed in. For instance, when you pass in the database D10K, the corresponding frequent itemsets should be saved into a file called D10K_Apriori_1.freq, which means that the frequent items are obtained from D10K using the Apriori algorithm under the minimum support 1%.

Your program also needs to print out the number of times scanning the database in order to generate the frequent itemsets.

Test your program using the four databases you have created in Section 2.1 using min_support = 1%.

2.3 Implementing Idea 1 (for both undergraduate and graduate students)

Read the idea carefully in the Notes on the Moodle and implement it. You can use library data structures in your implementation. The name of your source code is idea1.cc or idea1.py. (Of course, you can have some other auxiliary files, such as *.h files, if needed.)

There is a function in idea1.cc or idea.py called

itemsets idea1(db, float ms) // Students using Python may need to change it slightly.

The function generates and returns the frequent itemsets from the database *db* using the minimum support *ms*. The data types of the parameter *db* and the required returned result are not specified on purpose. This way, you can have your freedom to choose your own data structures to represent them.

Your program saves the frequent itemsets into a file whose name corresponds to the name of the database you have passed in. For instance, when you pass in the database D10K, the corresponding frequent itemsets should be saved into a file called D10K_Idea1_1.freq, which means that the frequent items are obtained from D10K using the Idea 1 under the minimum support 1%.

Your program also needs to print out the number of times scanning the database in order to generate the frequent itemsets.

Test your program using the four databases you have created in Section 2.1 using min_support = 1%.

2.4 Implementing Idea 2 (for only graduate students)

Read the idea carefully in the Notes on the Moodle and implement it. You can use library data structures in your implementation. The name of your source code is idea2.cc or

idea2.py. (Of course, you can have some other auxiliary files, such as *.h files, if needed.)

There is a function in idea2.cc or idea.py called

itemsets idea2(db, float ms, int n) // Students using Python may need to change it slightly.

The function, by dividing *db* into *n* equal partitions, generates and returns the frequent itemsets from the database *db* using the minimum support *ms*. The data types of the parameter *db* and the required returned result are not specified on purpose. This way, you can have your freedom to choose your own data structures to represent them.

Your program saves the frequent itemsets into a file whose name corresponds to the name of the database you have passed in. For instance, when you pass in the database D10K, the corresponding frequent itemsets should be saved into a file called D10K_Idea2_1.freq, which means that the frequent items are obtained from D10K using the Idea 2 under the minimum support 1%.

Your program also needs to print out the total number of times scanning the database in order to generate the frequent itemsets.

Test your program using the four databases you have created in Section 2.1 using min_support = 1%.

2.5 Hints

Vectors, queues, lists, etc. are good and efficient data structures you can take advantage of.

Compare the frequent itemsets from different approaches and see whether they match.

To be continued.

Part II: Experiments and Submission.