LINGI2364: Mining Patterns in Data
Project 1: Implementing Apriori

Siegfried Nijssen, Charles Thomas

Due March 13, 23:55

1 Context

This project is focused on the Apriori algorithm which aims to find the frequent itemsets in a dataset given a fixed minimum support. The Apriori algorithm is the most basic *join-based* algorithm for frequent itemset mining. It exploits the *anti-monotonicity* property and uses a *level-wise* approach. Many optimisations have been proposed to improve the performances of the basic implementation. In this project, you will have to implement a variation of the algorithm and compare its performances with another frequent itemset miner.

2 Directives

The project will be done by groups of two students and graded with INGInious. Please join the course on INGInious (`https://inginious.info.ucl.ac.be/course/LINGI2364`) and register in a group as soon as possible.

For this project, you will have to implement **at least two** different frequent itemset miner algorithms in Python. One of them **must** be a version of the Aprioi algorithm. The other must be either a different version of the Apriori algorithm with one or more optimisations or an implementation of a Depth First Search algorithm such as ECLAT. You will then have to compare your frequent itemset miners on several datasets with different support values in order to determine the impact of your implementation choices and optimisation(s) on the performance of the algorithms. Additionally, you will have to write a short report explaining the key aspects of your algorithms and containing your performance analysis.

The project will be done in Python. Use the corresponding task on INGInious. More information regarding the submission will be provided in the task directives. A template is provided (see Section 4.1 for more detail). You are free to modify this template as long as you respect the submission directives. The methods calling your frequent itemset miners should be called `apriori` for your implementation of the apriori algorithm and `alternative_miner` for your second implementation (Depth First or Apriori variation). These methods must have the following signatures:

```
def apriori(filepath, minFrequency)
def alternative_miner(filepath, minFrequency)
```

- The argument `filepath` is a string corresponding to the path to a dataset file. The format of the dataset files is detailed in Section 4.2.
- The argument `minFrequency` corresponds to the minimum frequency that an itemset must have in order to be considered as frequent. The frequency of an itemset is represented by a double and is defined as its support (the number of transactions containing an itemset) divided by the total number of transactions in the dataset.

Each method must output every frequent itemset found by your implementation of the algorithm on the standard output. Each itemset must be printed on a single line following this format: "[<item 1>, <item 2>, ... <item k>] (<frequency>)". For example, an itemset containing the items 1, 2, 3 and having a frequency of 0.92 will be printed as: "[1, 2, 3] (0.92)". Note that the items inside an itemset should be ordered according to the lexicographical order but the order in which the itemsets are printed does not matter. Make sure that you print only the itemsets and no other text in your INGInious submission.

You will be graded based on several criteria:

- The correctness and performances of your implementations (12/20).
- The quality and relevance of your report, justifications and performance analysis (8/20).

The deadline for this project is **Friday 13th of March at 11:55 pm**. Your submission should be uploaded following the modalities given on INGInious.

3 Report

Your report must not exceed 4 pages. It has to contain short descriptions of your different implementations as well as the optimisation(s) that you added. You have to explain and justify your implementation choices. The report must also contain an experimental comparison of the performance in terms of time (and optionally memory) on several datasets with different support values. You can also briefly discuss the difficulties that you encountered during the project.

Your report must contain the number of your group as well as the names and NOMAs of each member. It should be written in correct English. Be precise and concise. Do not hesitate to use tables or graphics to depict the results of your comparison of the variations of the algorithm.

4 Resources

4.1 Provided template

We provide a small template that can serve as starting point of your implementation. It is available on moodle (in the `Template.zip` archive). It contains a class named `Dataset` which is a utility class that reads a dataset file and allows to access its transactions and items through dedicated methods. Note that the whole dataset content is loaded in memory for faster access during the computation of your algorithm. This will take some place in your RAM depending on the dataset used but shouldn't cause any problem with the files given for the project. For the same reason, it may also take some time to initialize the object. Take this into account in your evaluations of the performance of your algorithms. You are not obligated to use this class and can modify it in any way you want.

4.2 Datasets

Six different datasets are available on moodle in the `datasets.zip` archive. Each dataset file has an extension in .dat. The files have the following format: Each line corresponds to a transaction. A transaction is represented by a series of integers separated by single spaces and sorted in ascending order which represent the different items present in the transaction. No item is included more than once in a transaction.

The `toy.dat` dataset is a small dataset that can be used to track manually the execution of your algorithms. The other datasets come from this site: `http://fimi.ua.ac.be/data/`.

We provided three files to check if your implementation:

- `toy_itemsets0125.txt` contains all the itemsets that your algorithm should find in the `toy` dataset for a minimum frequency of 0.125.
- `chess_itemsets09.txt` contains all the itemsets that your algorithm should find in the `chess` dataset for a minimum frequency of 0.9.
- `accidents_itemsets08.txt` contains all the itemsets that your algorithm should find in the `accidents` dataset for a minimum frequency of 0.8.

5 Important tips

- The algorithms might take a lot of time on some of the datasets for low frequency values. When testing your implementations, always start with a high frequency and decrease it until your algorithm takes too much time.

- One of the files used in the tests has a large number of different items. Depending on how you implemented your algorithms, this can have a huge impact on your performances. Think about what you can do to deal with this problem.
- If you have other performance issues, try to identify which are the particularities in the datasets that make your algorithm inefficient and which are the parts of your code that cause this issue. Also, be careful with the structures and operations that are used in your code.
- The performance analysis of your algorithms is the part of the report that is worth the most points. It will require some time to perform an appropriate number of experiments. Do not underestimate this part of the project.
- Test your code locally before submitting on INGInious. This tool is used by several courses and its resources limited. INGInious is a tool to grade your code, not debug it!
- Follow carefully the directives on INGInious when submitting your code.
- Make full use of the time allocated. Do not start the project just before the deadline!
- Do not forget to comment your code.
- Plagiarism is forbidden and will be checked against! Do not share code between groups. If you use online resources, cite them.