LINGI2355: Mining Patterns in Data
Project 2: Implementing Sequence Mining

Due April 16 2021, 23:55

1 Context

The aim of this project is to discover closed frequent sequential patterns in a supervised dataset with sequences. To this aim, you will implement a sequence mining algorithm such as PrefixSpan or SPADE. You will then adapt your algorithm in order to use it in a supervised context. Finally you will have to adapt your algorithm to return only the closed sequences and compare different scoring functions. Below, we will first describe the two datasets provided, followed by the details on the implementation we expect you to make.

1.1 Datasets

We provide you two sequence datasets that were obtained from this URL: `http://adrem.ua.ac.be/scii`. More precisely, a Protein dataset and a Reuters dataset. Each dataset is a supervised dataset with two class labels. We are interested in finding patterns that allow us to understand the differences between the examples labelled with one class label and those labelled with another class label. Each class corresponds to a separate file. In the files, each transaction is formatted as follow:

```
<Symbol 1> <Position 1>
<Symbol 2> <Position 2>
⋮
<Symbol n> <Position n>
<empty line>
```

After the empty line a new transaction starts. The sequences in the Reuters dataset consist of words; the sequences in the Protein dataset consist of nucleotides.

For the Reuters dataset, the negative class will be the file `acq.txt` and the positive class will be `earn.txt`. For the protein dataset, the negative class will be the file `PKA_group15.txt` and the positive class will be `SRC1521.txt`.

Note that for this task we are not interested by sequences of itemsets but by sequences of symbols. We only keep the symbols in a sequence by order of appearance. That means that the transaction [{A,B},{A}] will be considered as [A,B,A].

1.2 Tasks

Your task is to implement different algorithms capable of finding sequential patterns:

1. Frequent sequence mining: The first algorithm that you will have to implement is frequent sequence miner based on SPADE or PrefixSpan. Your algorithm will have to find the k most frequent patterns in both classes of the dataset. In order to do so, you will have to consider the sum of the support in both classes. For example, a sequence which has a support of 3 in the positive class and 4 in the negative class will have a total support of $3+4=7$. Note that if multiple sequences obtain the same total support, you should output all these sequences and count it as only one increment in k – hence, in some cases, the output could be larger than k.

2. Supervised sequence mining: In this project, we are interested by the sequences that will help us distinguish between the two classes of the dataset. In order to find them, your algorithm needs to be able to find the k best patterns according to the *Weighted relative accuracy* score.

   The *Weighted relative accuracy* (or *Wracc* for short) is defined for a pattern x as:

   $$Wracc(x) = (\frac{P}{P+N} * \frac{N}{P+N}) * (\frac{p(x)}{P} - \frac{n(x)}{N})$$

   Where P and N are the number of transaction in the Positive and Negative classes and $p(x)$ and $n(x)$ are the supports of x in the positive and negative classes. It computes the balanced support of a pattern in both classes. A pattern which has a high support in the positive class but a low support in the negative class has a high positive *Wracc* score. Inversely, a pattern with a high negative support and a low positive support has a high negative *Wracc* score. A pattern which has a similar support in both classes has a *Wracc* score close to 0.

   We are interested to find the patterns that are highly present in the positive class but not in the negative class. The k best patterns are thus the sequences that have the highest *Wracc*. Similarly to the frequent sequence mining algorithm, if multiple sequences obtain the same total support, you should output all these sequences and count it as only one increment in k.

3. Supervised closed sequence mining: You will then have to adapt your supervised sequence mining algorithm in order to return only closed patterns: i.e., for none of the returned patterns there is a supersequence that has the same support in both the positive and the negative classes. When considering a pattern, if the support is different for at least one class in each super-pattern of the pattern, then the closeness constraint is respected. For example if the pattern AB with supports of 3 and 4 has one super-pattern ABC with supports 3 and 3, it is closed.

4. Alternative scoring functions: Finally, you will have to adapt your closed supervised sequence mining algorithm to use alternative scoring functions: The first adaptation will be to consider the absolute value of the *Wracc* score. That implies that the k best patterns will also include patterns strongly present in the negative class. You will also need to use the *Information Gain* scoring function instead of the *Weighted relative accuracy* and compare the patterns obtained using these different scoring function.

Your prime objective is to create algorithms that are correct. Efficiency is a secondary objective.

2 Directives

- The project will be done by groups of at most two students.
- As in the first project, the project will be done in Python.
- Your algorithms will have to follow the specifications defined on INGInious in order to be correctly evaluated. You will have to submit a single `.py` file that contains a main method that receives the following command line arguments:
  - the path to the positive dataset file;
  - the path of the negative dataset file;
  - a parameter k;

  and outputs the top k sequential patterns in the format defined hereafter.
- Your implementation must output all top-k sequential patterns on the standard output. Each pattern should be written **on a single line** followed by its support in both classes and it's total support (for the first algorithm) or score (for the others) according to the following format:

  [<symbol 1>, <symbol 2>, ... <symbol k>] <support in positive class> <support in negative class> <score>

  For example, the pattern A with supports of 3 and 4 in both classes and thus a total support of 7 should be displayed as: [A] 3 4 7 by the first algorithm. The pattern BCB with supports of 4 and 6 in both classes and a *Wracc* score of 3.5 should be written as: [B, C, B] 4 6 3.5 by the second or third algorithm.

- In order to avoid imprecisions due to the computation of floating point values, the correction script rounds the Wracc and other scores values to 5 decimals. You should thus round the result of the scoring function to 5 decimals when computing the score of a pattern (this should be done directly after the computation as it can influence the topk itemsets found).
- Your report must not exceed 4 pages. It has to contain a short description of your implementation. You have to explain and justify your implementation choices. The report must also contain an analysis of the patterns found using the different scoring functions. How many patterns in common were found by the different scoring functions? Were there strong differences?

  Optionally, you can 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 in your explanations. Do not hesitate to use tables, schemas or graphics to illustrate.
- You will be graded based on several criteria:
  - The correctness and performance 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 16th of April 2021 at 23:55**. Your submission should be uploaded on INGInious according to the modalities specified for each task.

3 Tips

- The easiest solution to deal with the closeness constraint is to check its requirements only when inserting a pattern in the set of $k-$best patterns by searching for and removing eventual sub-patterns with the same supports. The check can be done efficiently by maintaining for the k patterns what their covers are.
- You can reuse the utility Dataset class provided for the first project to read the dataset files and access their transactions. You might want to adapt it for the algorithms that you intend to implement.
- We will also provide you a small dataset file (named `test_dataset`) along with the expected output for a k value of 15 or more for each algorithm. Note that the order in which the patterns found appear in the output does not matter as long as they are all present (however the order of the items inside the patterns does matter!). You might also want to round a bit the score values when comparing with your implementation. The dataset is small enough for you to check manually other inputs. We encourage you to make other tests by yourselves.
- For larger datasets (such as reuters), having an efficient algorithm is mandatory to find the top k itemsets in a normal amount of time. Computing bounds is one way to improve your algorithm: Once you have found k itemsets during your search, you can compute bounds based on the minimum score among the k scores found. For example, with the sum of supports, based on your minimum score (minscore) at some point in the search, you can compute lower bounds for the positive (p) and negative (n) support as: $lb(p) = minscore - cn$ and $lb(n) = minscore - cp$ where cp and cn are the current measurements of positive and negative support in your depth first search.
- Another way of improving your algorithm is using a heuristic: In order to quickly find itemsets with high scores and thus compute efficient bounds, you can alter the order in which you explore the different symbols using a heuristic. For example, if you branch first on items having a higher score, you augment the chances of finding itemsets having a higher score. This will give you a higher minimum score and thus allow you to better prune your search tree by computing better bounds.
- The sequence mining algorithm might take a lot of time on some of the datasets. When testing your implementations, always start with a low value for k, or consider adding an additional threshold on Wracc.
- 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.