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

Siegfried Nijssen, Charles Thomas

Due November 20, 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 create an implementation of PrefixSpan or SPADE that is able to find closed sequential patterns under a *gap constraint*. You will evaluate the impact of this constraint on the quality of the results that you find. 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 labeled with one class label and those labeled 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].

Patterns in sequential databases consist of symbols as well. What is not clear, is whether it makes sense to map the symbols of a pattern far away from each other in a sequence in the data. After all, words that are far away from each other in a paragraph may or may not be related to each other any more.

Your task in this project will be to determine whether a change in a *gap* parameter yields different results on both these datasets.

1.2 Algorithm

Your task is to implement an algorithm that is capable of finding sequential patterns under the following constraints:

- Supervision: **This concept will be seen during the next course**. 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 maximize the *Wracc* score. The k best patterns are thus the sequences that have the highest *Wracc*. Note that if multiple patterns obtain the same score as the k-th pattern, you should output all these patterns – hence, in some cases, the output could be larger than k.

- Closedness: all patterns that are returned need to be closed: 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.

- Gap: when a pattern is matched to the data, it should be possible to specify a *maximum gap* constraint. For instance, the following pattern matches the data with a gap of 2:

  BCB $\Rightarrow$ A<u>B</u>AA<u>C</u>A<u>B</u>A,

  A gap of 2 means that the largest number of symbols that need to be skipped between symbols of the pattern in order to match the pattern is 2.

Your prime objective is to create an algorithm that is correct. Efficiency is a secondary objective. Please consider the following:

- You are free to build on the ideas of the SPADE or the PrefixSpan algorithm. Note however the following problem that can occur with gaps:

  BCB $\Rightarrow$ ABAACAAA<u>BCB</u>A

  In the traditional PrefixSpan algorithm, it would be assumed that the first occurrence of BC (with a gap of 2) can always be expanded to an occurrence of BCB as BCB is present in the string. However, this is not the case when using gaps. SPADE is easier to modify to deal with gaps.

- The easiest solution to deal with the closedness constraint is to check its requirements only when inserting a pattern in the set of $k-$best patterns, and to check it by maintaining for these k patterns what their covers are.

2 Directives

- The project will be done by groups of at most two students.

- Your implementation must output all top-k closed sequential patterns in a file given in parameter. Each pattern should be written on a single line followed by its support in both classes and it's score according to the following format:

  [<symbol 1>,<symbol 2>, ... <symbol k>] <support in 1st class> <support in second class> <score>

  For example, 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

- For this project, you are free to use the language of your choice. However, your program must be launched from the command line and accept the following arguments:

  – both paths of the input files;

  - the path of the output file;
  - a parameter k;
  - the maximum gap allowed.

- 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 experimental comparison. In this comparison you discuss:

  - for which gap setting of the parameter k you obtain patterns with a better score for both datasets;
  - an illustration of the patterns you find for gap values;
  - the effect of the parameters k and gap on the runtime of your implementation.

  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 of your implementations.
  - The optimisation(s) that you implemented and the performances of your most advanced solution.
  - The quality, structure and readability of your code.
  - The quality and relevance of your report, justifications and performance analysis.

- The deadline for this project is **Monday 20th of November at 23:55**. Your submission should be uploaded in a single ZIP archive on moodle. The archive will contain all the source files of your program and your report in pdf format. You can also include a readme file explaining how to compile and/or launch your program.

3 Tips

- Even if the concepts of supervised mining and *Weighted relative accuracy* have not yet been seen during the course at the start of the project, you can already work on a first version of the algorithm that returns the k most frequent patterns in both classes (for example, use the sum of the supports in both classes as score).
- The easiest way to check the closeness constraint is to search for and remove eventual sub-patterns with the same supports when adding a pattern to the best patterns.
- We provide you a basic utility class in Java to read a dataset file and extract its transactions or its items. You are free to use it or modify it at you convenience.
- We also provide you a small dataset file (named `test_dataset`) along with the expected output for a k value of 15 or more and a gap of 1. Note that the order in which the patterns found appear in the output file does not matter as long as they are all present. You might also want to round a bit the Wracc 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.
- 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.