LINFO2364: Mining Patterns in Data
Project 2: Sequence Mining & Classification

March 17, 2023

1 Context

In this project, you will use the *Spade* mining algorithm to construct pattern-based classifiers for sequential data. You will first write an implementation of the Spade algorithm and will subsequently use this algorithm to build a pattern-based classifier. Two sequential datasets are provided that you will need to use to evaluate your appproach. Below, we will first describe the two datasets provided, followed by the details on the implementations 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 building a pattern-based classifier that allow us to predict to which of the two classes a given sequence belongs. 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 approaches for learning a pattern-based classifier for this sequential data. We expect you to do so in the following steps:

1. Frequent sequence mining (INGInious): The first algorithm that you will have to implement is frequent sequence miner based on Spade. 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 (INGInious): 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 first 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. Subsequently, you will use the *Scikit-learn* library to train a model based on the top-k patterns found using the sequence mining algorithms developed in the previous 2 steps. Here, we will use decision trees such as implemented in Scikit-learn. You will use cross-validation to evaluate the quality of your model. In order to do this, you will need to split your data in to a training set and a test set. *Scikit-learn* allows to tune the search tree using multiple parameters. Please leave these parameters to their default value and set the random seed of the tree to 1: `classifier = tree.DecisionTreeClassifier(random_state=1)`. Repeat this exercise for the top-k frequent patterns and the top-k patterns with the highest Wracc. Your model has to be able to classify new sequences in one of the two classes. You will have to analyse the accuracy of your model. You are encouraged to use cross-validation to do so. Be wary of overfitting. The aim is to have a model as accurate as possible.

4. Finally, implement one alternative algorithm in which the pattern mining algorithm is used to iteratively find a top-1 pattern that is put in a classifier. This could be a sequential covering algorithm, a decision tree learning algorithm, or any other algorithm you could imagine. You are free to use any scoring function in this top-k mining algorithm, such as information gain, gini-index, or a diversity score.

Your prime objective for the pattern mining algorithms is to create algorithms that are correct. Efficiency is a secondary objective.

After implementing the different algorithms, you are expected to write a report summarizing your experiments and the algorithmic choices you have made.

Feature Set

In order for your top-k itemsets to be used to train a model, you need to build a representation of the data in terms of the patterns present in each of the transactions. The following illustrates this representation:

	Pattern 1	Pattern 2	$\cdots$
Transaction 1	1	0	$\cdots$
Transaction 2	1	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	

Here the bits indicate the presence or absence of the patterns. This format is required by the models of *Scikit-learn* to train them.

SciKit-learn

Scikit-learn is a python package providing multiple tools for data mining and machine learning tasks. In this project, it will be used to train classification models. Its documentation can be found at the following address: `https://scikit-learn.org/stable/index.html`. Note that *scikit-learn* requires `NumPy`, `SciPy` and `matplotlib` to work. It can be installed independently or as part of Anaconda (`https://www.anaconda.com/`).

Iterative Algorithms: Example for Sequential Covering for Learning Rule Lists

You are expected to implement one iterative algorithm that builds a classifier by repeatedly executing a pattern mining algorithm.

Here we provide one example of such an algorithm: the sequential covering approach to identify a rule-based classifier, consisting of a set of rules.

In this algorithm, repeatedly the pattern is added that obtains the highest score on the remaining examples, for instance, using a confidence score in combination with minimum support; hence, the approach is iterative: it searches repeatedly for patterns and does not post-process a set of patterns that was found earlier. The transactions that contain the pattern found are then removed and the next iteration searches a new pattern on the remaining ones.

To develop this algorithm, you can use the implementation that you used in the second step to find supervised subsequences. The resulting algorithm is expected to be applied k times (with a top-k size of 1) to find a new pattern to add to the set. If several patterns with the same confidence and frequency are found, you should take the lowest in the lexicographical order. After having k patterns in your pattern set, you should define a default class that will be assigned if none of the patterns in the pattern set is able to classify a transaction. This default class will be the one the most present in the remaining transactions or the positive class if both classes are equally present or if there is no transaction remaining.

The resulting list of rules can be used as a classifier without using SciKitLearn.

Cross-Validation

Cross-validated accuracy can be used as an evaluation criterion for the accuracy of a model. k-fold cross-validation assumes that the data is split into k folds; repeatedly one of these folds is removed, while the learning algorithm is applied on the remaining data. The resulting model is evaluated on the data that was left out.

In a pattern set mining context this means that the pattern mining algorithm needs to be applied k times to find a set of patterns. For each of these iterations, a model needs to be constructed from the patterns, and the resulting model needs to be applied on the remaining examples to evaluate the quality of the model.

You are encouraged to use this method in your performance evaluation. We limit the number of folds to 4 to reduce the run time of your experiments.

Implementing cross-validation requires that the occurrences of sequence patterns are also determined in the test data. While we could implement a separate algorithm for sequence matching to accomplish this, we recommend a different approach in this project: for every pattern that your sequence mining algorithm generates, we immediately determine the occurrence in the test data as well.

Note however that for a reliable evaluation of a machine learning algorithm, it is absolutely necessary that occurrences calculates on the test data are **not** used while building the pattern-based model; it should only be used for evaluating the quality of the model.

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 performances of your implementations (8/20).
  - The quality and relevance of your report, justifications and performance analysis. The quality and relevance of your report, justifications and performance analysis. Your source code will also be checked and graded here (e.g. hard-coding the solutions will give a grade of 0) (12/20).

- As with the previous projects your code will be graded using INGInious. The INGInious tasks will be made available later and will contain more details about the submission modalities. **Please make sure that your program works by testing it on your machine before uploading it**. INGInious is a grading tool, not a debugger, and its resources are limited. Make sure that your program works without bugs and provides the expected output for multiple test cases. You will be provided with several outputs to compare with.

- The deadline for this project is **Monday 17th of April 2023 at 23:55**. Your submission should be uploaded on INGInious according to the modalities specified for each task.

3 Tips

- 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.