UCLouvain-EPL

Mining Patterns in Data

Project 1: Implementing Sequence Mining

Teacher:
Siegfried Nijssen
Course assistants:
Thomas Charles
Guillaume Derval

Authors:
Nima Farnoodian - 68372000
nima.farnoodian@student.uclouvain.be

César Liesens - 39161400
cesar.liesens@student.uclouvain.be

Group 53

April 16, 2021

1 Introduction

In this project, for the Sequence Mining task, we were asked to implement either *Spade* [1] or *PrefixSpan* [2] *Algorithm*, and adapt them for the following tasks: ***Top-k Frequent Sequence Mining***, ***Top-k Supervised Sequence Mining*** with `Wracc` scoring function, and ***Top-k Supervised Closed Sequence mining*** with respect to `Wracc` and `Absolute Wracc` as well as `Information Gain` scoring function. In total, this project revolves around five sequence miners and the evaluation of their extracted patterns. To this end, we implemented *the Spade algorithm* and one of its variations called *f2-Spade algorithm* to overcome the aforesaid tasks. Moreover, as the second goal of this project was to compare the patterns detected using the said tasks, we compared the detected sequence sets (e.g, how many sequences they share) and analyzed the performance of detected patterns in the supervised prediction task by training classifiers. Indeed, we tried to figure out how well the patterns could lead to accurate supervised prediction. It is remarkable that, you can find our source codes and all our experiments on ***our github repository***,

In what follows, in section 2, we will discuss the algorithms and the strategies we applied for responding to the asked tasks. In section 3, we will present our experimental results in-depth. Finally, we will conclude this paper in section 4.

2 Algorithms and Strategies

In this section, before embarking on the details regarding the problems and the strategies, we will briefly converse about the chosen algorithms. Next, we will explain each problem along with its solution using the chosen algorithms. It is worth mentioning that, in this project, the sequence mining task is accomplished by sequences of symbols not sequences of itemsets.

2.1 Spade and F2-Spade Algorithms

Spade algorithm [1] mainly borrows its concept from *ECLAT algorithm* that is one of the famous Itemset mining algorithms. *The spade algorithm* benefits from the vertical representation of the dataset. Although there are two implantation approaches for this algorithm (DFS and BFS), we only concentrated on its DFS implementation as it was found to be faster in different literature. In the DFS implementation of this algorithm, after computing the vertical representation of the dataset, all frequent symbols are first detected and deemed as the prefixes from which new sequences may be found. Then, from each prefix, let's say A, a depth first search begins to detect super-sequences with the prefix A. During each recursive call for a frequent prefix $A...B$, a frequent sequence is found by adding a symbol C to the prefix as long as the support for the sequence $A...BC$ is above a user-specified threshold x. If the sequence is frequent, the search will then continue in-depth until no frequent sequences are observed. Computing supports is done using a linear search through the transaction id-lists of the prefix and the candidate symbol in the vertical representation of the dataset. During our experiments on *the spade algorithm*, we realized that support computation dominates the running time of the whole algorithm. Therefore, if we use a weak data structure for storing the vertical dataset and treat the support computation problem naively, we may impose a huge computation overhead, possibly precluding solving the frequent sequence mining task for large datasets. For this reason, we made use of a two-dimensional python dictionary (e.g., Vertical[u][v]) where keys u are the sequences, v are the transaction ids (tid) related to the sequences, and values are an ordered list of the positions of the sequences in the transactions. For example, Vertical[(ABCD)][20]=[1,6] indicates that the positions of the sequence $ABCD$ in tid=20 are 1 and 6. Such a data structure could speed up *the spade algorithm* since the search space for computing the covers and supports dramatically decreased. To clarify this, we should remark that the search is started on the prefix's tids. As the patterns grow, the tid-list of the prefixes shrink quickly.

F2-Spade algorithm [1] is a faster version of the original algorithm for handling larger transaction datasets. In *f2-Spade algorithm*, not only are the frequent symbols computed but also the frequent sequences of the size of 2 are computed before DFS searches. Although this change might seem vain at first glance, this could make the frequent mining tasks faster, specifically, for large datasets. The reason is that 2-frequent

sequences can be much more quickly computed using a horizontal dataset obtained from the vertical dataset, or before computing the vertical dataset. Suppose, for transaction i, the vertical dataset holds the symbols A as a pair *(A, u)* where u is the location of the symbol A in the transaction. Then, the computation of 2-frequent sequences is straightforward. We form a list of all 2-sequences in the list for each transaction, and update counts in a 2-dimensional array indexed by the frequent symbols. *f2-Spade algorithm* helps avoid computing 2-frequent sequences during recursive calls, which are computationally heavy due to cover computation.

2.2 Sequence Mining Tasks

In this section, we will discuss about the sequence mining tasks that we could handle by modifying *the Spade algorithm* and *the F2-Spade algorithm.*

2.2.1 Top-k frequent sequence mining

```
class k_selector:
    def initialization (k):
        initiate a data structure named "data" of size of k to hold top k sequences
    def append(score,sequence):
        if data has free space:
            append sequence with its score to data
        else:
            if score> minimum score in data:
                append sequence with its score to data
                remove the sequences with the score equal to the minimum
            if score== minimum score in data:
                append sequence with its score to data
        return True if the sequence is appended, otherwise False

def topk_spade(data_positive,data_negative, k):
    # <> indicates a sequence
    vertical_pos_data= compute vertical dataset of data_positive
    vertical_neg_data= compute vertical dataset of data_negative
    selector=k_selector(k) # initialize a data structure
    symbols=get_symbols(vertical_pos_data) Union get_symbols(vertical_neg_data)
    for symbol in symbols:
        total_supp=positive_supp(symbol)+ negative_supp(symbol)
        selector.append(total_supp,<symbol>)
    for symbol in selector: # in the descending order of score
        DFS(symbol,vertical_pos_data[symbol],vertical_neg_data[symbol])
    for sequence in selector:
        print(sequence)

def DFS(sequence,vertical_pos_seq, vertical_neg_seq):
    for symbol in symbols:
        vertical_pos_seq_new= join(vertical_pos_seq,vertical_pos_data[symbol])
        vertical_neg_seq_new= join(vertical_neg_seq,vertical_neg_data[symbol])
        total_supp=Compute the support of new sequence <sequence+symbol>
        if total_supp>0: # To ensure not to discover zero-frequent sequences
            if selector.append(total_supp, <sequence+symbol>)==True:
                DFS(<sequence+symbol>,vertical_pose_seq_new,vertical_neg_seq_new)
```

In the top-k frequent sequence mining problem, the task is to select k frequent sequences whose total supports (support for positive dataset + support for negative dataset) are higher than the rest. As defined in our project, the top-k selection is accomplished according to the observed scores, not the observed sequences. For example, if there are four sequences with the same score, they are counted only once.

As the interest in this task is the total support, we could adapt *the spade algorithm* to extract the sequences from both positive and negative datasets simultaneously. To do so, we just needed to add a few lines of code so as to compute the total support of a found sequence. A DFS search for a prefix continues if the total support of the prefix is above zero. Fortunately, the top-k frequent problem benefits from the anti-monotonicity property, enabling the algorithm to prune the search tree efficiently. The idea is that if a sequence cannot be appended to the list of top-k sequences, then all its super-sequences discovered by it cannot be top-k sequences since $support(sequence) >= support(super-sequence)$. To this end, we suggested a class with a specific data structure that could hold all top-k frequent sequences. This class was enhanced by a method named `"append"`, which appends a sequence to the top-k list if its score is greater than or equal to the minimum score that has been seen. The data structure is regularly updated after each appending in order to ensure that it contains all top-k frequent sequences. It is worth noting that, the `"append"` method returns `True` if it could add the sequence to the top-k list, otherwise `False`. Therefore, if after adding a sequence, the `"append"` method returns `True`, then the DFS search continues, otherwise the search stops and pruning occurs. The above pythonic pseudo-codes clearly represent our solution using *the spade algorithm* for this problem.

2.2.2 Top-k Wracc supervised sequence mining a.k.a "Top-k Wracc"

In this problem, the goal is to rank the sequences based on their `Wracc` scoring function and select only the top-k high-ranked sequences. Unfortunately, we could not find any specific property such as `anti-monotonicity` to prune the search space like Top-k frequent sequence mining problem. Therefore, we chose post-processing approach using *F2-Spade algorithm*. In this approach, the algorithm first detects the sequences in the positive and negative datasets sequentially, and then aggregates them in one set. After extracting both sequence sets, the Wracc for sequences are computed and they are ranked from high to low, accordingly. In the end, only those sequences whose Wracc scores are above the top-k scores are returned. We should remark that we used a heuristic for this problem to handle the large dataset. It is obvious that Wracc favors the highly frequent sequences in the positive dataset while it opposes the highly frequent sequences in the negative dataset. Thereupon, we heuristically set the minimum support threshold for the positive dataset equal to 0.2, and for the negative dataset equal to 0.05 just to make sure that it will not discover zero-frequent sequences during the DFS search.

2.2.3 Top-k supervised Closed-sequence mining: top-k-Closed-wracc, top-k-Closed-abswracc, top-k-Closed-infogain

A closed sequence is a sequence that one cannot add any symbol to it and get the same cover for the result. In this problem, the aim is to detect the closed sequences and then select only top-k closed sequences with respect to the following scoring functions: Wracc, Absolute Wracc, and Information Gain. Like Top-k Wracc supervised sequence mining, to tackle this problem, we used the post-processing technique using *F2-Spade algorithm*, but with different minimum supports. For the positive datasets, we set the minimum support to 0.02 since there might be closed sequences with low supports in the positive datasets, and set the minimum support to 0.05 for the negative datasets. To obtain closed sequences, we exactly followed the tips in the directive of the project. In other words, after detecting all positive and negative sequences, *"the algorithm checks closeness constraint only when inserting a sequence in the set of the k-top sequence by searching for and removing eventual sub-patterns with the same supports"*.

3 Performance Evaluation

In this section, we initially evaluate the performance of the algorithms including Top-k frequent sequence mining, Top-k Wracc, top-k-Closed-wracc, top-k-Closed-abswracc, and top-k-Closed-infogain in terms of runtime and sequence set analogy. Afterward, we show the impact of the outcome of the supervised sequence miners on the accuracy of classifiers. It is worth noting that we chose Logistic Regression as our baseline for evaluating the performance of our supervised miners because it does not require any parameter tuning.

The transaction datasets are listed in Table1 in ascending order of magnitude. The implementations, measurements, and comparisons were carried out in Python 3.8.3 under Windows 10 on a Dell all-in-one PC powered by Intel Core i5-4590S CPU @ 3.00GHz and 8 GB of main memory.

	Number of Transactions in positive dataset	Number of Transactions in negative dataset
Protein	314	381
Reuters	2838	1594

Table 1: Sample Transaction Datasets

3.1 Run-Time Evaluation

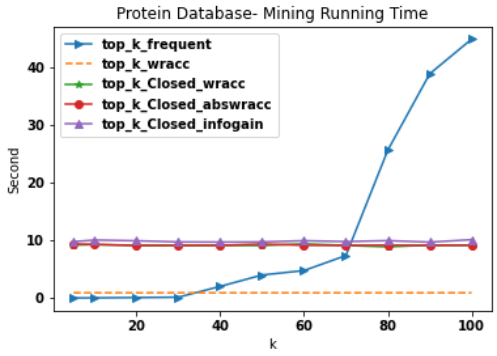

Figure 1: Algorithms' run-time on Protein

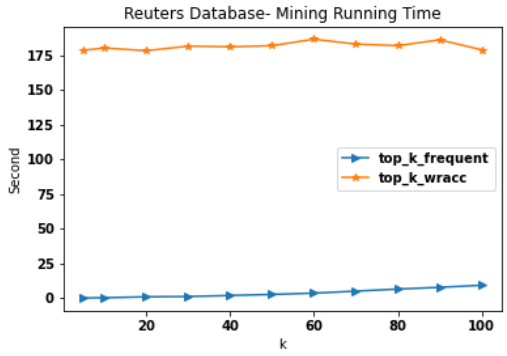

Figure 2: Algorithms' run-time on Reuters

First and foremost, a cursory look at Figures 1 and 2 reveals that all our algorithms could handle middle-size dataset "Protein" in a reasonable time while only top-k-frequent and top-k-wracc algorithms could detect the patterns in the large dataset. It is interesting to say that the required time for the top-k-frequent algorithm using the spade algorithm is very low so that it could unveil the top-100 patterns in less than 25 seconds in the large dataset. It attests to the practicability of our specific data structure and the pruning technique we applied for this task. One might observe that all post-processing algorithms that utilize the f2-spade algorithm always have the same run-time for even large k. The reason is that the f2-spade algorithm dominates the run-time and the post-processing activities such as finding closed patterns does not incur computational overhead —we tried to make the post-processing as efficient as possible.

3.2 Pattern Set Resemblance

One of the task that we were asked to accomplish was to analogize the patterns detected using different miners that we implemented. To this end, we detected the patterns using each miner for different $k =< 100$, and then calculate the resemblance between the resulting patterns of the miners. We define resemblance as the percentage of the patterns that are shared between two pattern sets. In the end, we calculated the mean of all observations along with the standard deviation. Figure 3 illustrates the resemblance of the extracted patterns on the Protein dataset. We avoid adding the result regarding the Reuters dataset since we could only run top-k-frequent and top-k-wracc algorithms on this dataset as earlier said.

To be precise about the result, we should remark that, we have only seen a relative resemblance between top-k-Closed-infogain and other miners except for the top-k-frequent miner. This observation is intersting as we could find out that there might be a correlation between infogain scoring function with Wracc and Absolute Wracc. However, since we did not have other datasets to run our experiments on, we cannot be 100 percent sure concerning our observation.

Baseline	top_k_frequent	top_k_wracc	top_k_Closed_wracc	top_k_Closed_abswracc	top_k_Closed_infogain
top_k_frequent	1.0 +/- 0.0	0.134 +/- 0.084	0.134 +/- 0.084	0.233 +/- 0.081	0.187 +/- 0.034
top_k_wracc	0.134 +/- 0.084	1.0 +/- 0.0	1.0 +/- 0.0	0.555 +/- 0.085	0.696 +/- 0.136
top_k_Closed_wracc	0.134 +/- 0.084	1.0 +/- 0.0	1.0 +/- 0.0	0.555 +/- 0.085	0.696 +/- 0.136
top_k_Closed_abswracc	0.233 +/- 0.081	0.555 +/- 0.085	0.555 +/- 0.085	1.0 +/- 0.0	0.739 +/- 0.097
top_k_Closed_infogain	0.187 +/- 0.034	0.696 +/- 0.136	0.696 +/- 0.136	0.739 +/- 0.097	1.0 +/- 0.0

Figure 3: Pattern Set Resemblance on Protein Dataset. +/- indicates standard deviation

3.3 Label Prediction based on detected patterns

We understand we exceeded the page limit, but honestly, we were really interested in understanding how sequence patterns may help train accurate classifiers. So please, do not reduce mark.

To see how different patterns may lead to a supervised classifying model, we followed the Pattern-based machine learning models, discussed in Lecture 8. The approach is straightforward. We first detected the top-k patterns using each miner where $k =< 100$. Then, we created a tabular data frame whose attributes are the detected patterns, and the observations are the transactions. In this tabular data frame, attributes (patterns) are 1 if they exist in the related transactions otherwise they are assigned 0. Then, using these data frames, we could train different logistic regression classifiers to compare the performance of the miners. Figure 4 illustrates the test and training accuracy of the algorithms on the Protein dataset, and Figure 5 shows only the performance of top-k-frequent and top-k-wracc algorithms on the Reuters. An eyeball look at these two figures indicates that all algorithms, for $k > 10$, almost have reasonable performance although top-k-closed-infogain miner outperforms others in the Protein dataset. Nevertheless, it should be kept in mind that it is a costly algorithm in terms of run-time compared to top-k-frequent that has fairly good accuracy.

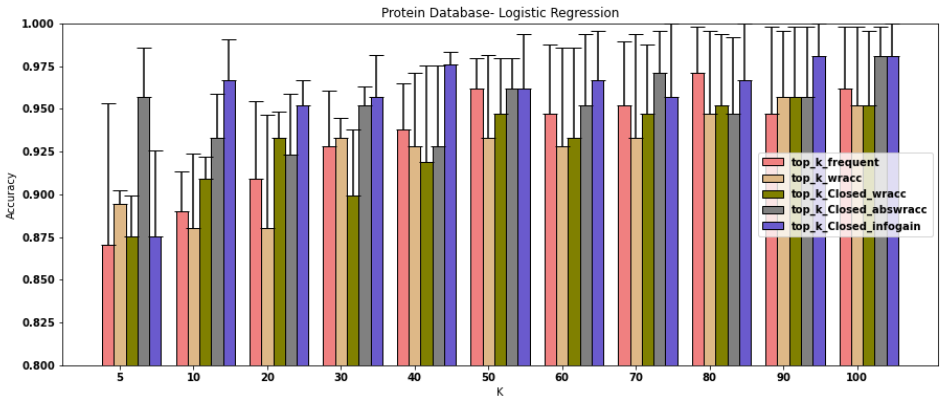

Figure 4: Protein-Test and Train Accuracy. The error bar on bar-tip indicates training accuracy

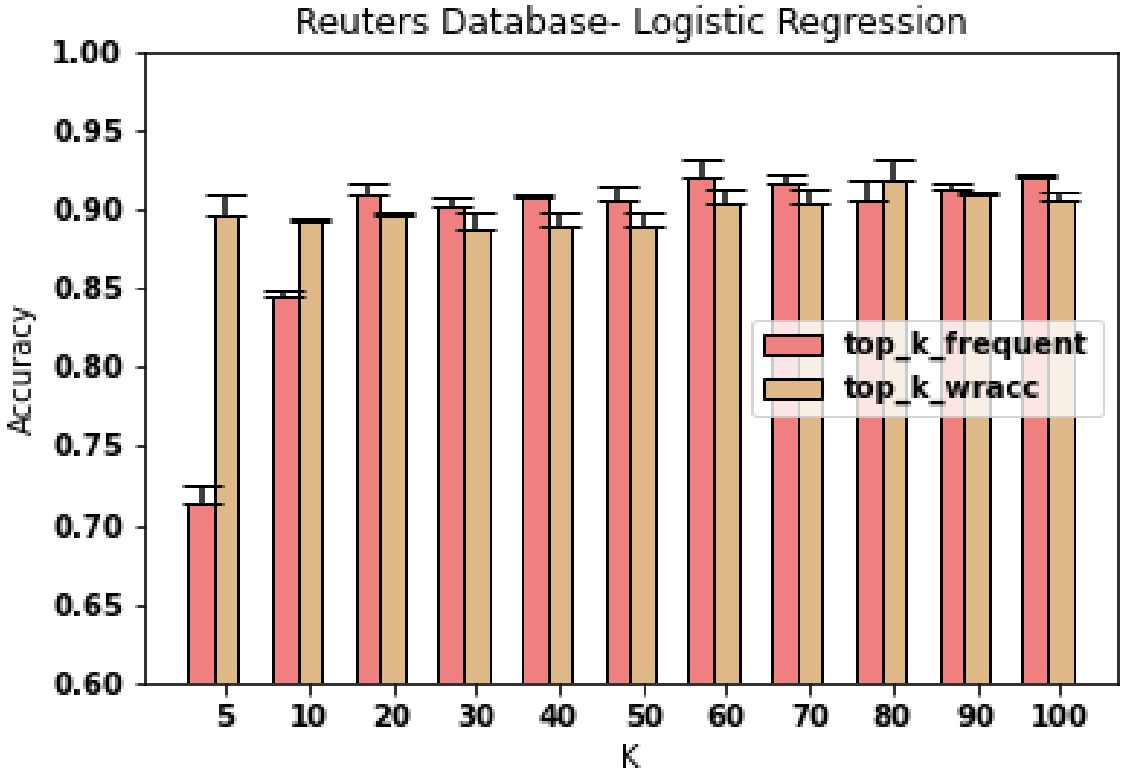

Figure 5: Reuters-Test and Train Accuracy. The error bar on bar-tip indicates training accuracy

References

[1] Mohammad J. Zaki. *SPADE: An Efficient Algorithm for Mining Frequent Sequences*. Machine Learning, 42, 31–60, 2001,°c 2001 Kluwer Academic Publishers.

[2] Jian Pei, Jiawei Ha, Behzad Mortazavi-As, Helen Pinto, Qiming Che, Umeshwar Dayal, Mei-Chun Hs. *PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth*. Link.