Ecole Polytechnique de Louvain

LINGI2364: Mining Patterns in Data

Project 3: ClassifyingGraphs

Group 11

Alessandra Rossaro (01211800), Matteo Salvadore (01731800)

Version 1.0 - December 20, 2018

AY 2018-2019

1 Implementations

The implementations of our algorithms are performed in Python language (version 3.6). The source code and all the resources used to perform this project are available in the following GitHub repository: `https://github.com/JustSalva/ProjectsOfMiningPatternsInData`.

1.1 Finding sub-graphs

In order to perform the search of the top k most confident frequent subgraphs on the positive class we have extended the provided template class *PatternGraphs* with our own class, called: *K_MostConfidentAndFrequentPositiveSubGraphs*.
The main structure of our top k mining algorithm is more or less the same of the structure implemented in the previous assignment, but we had to adapt its implementation in Python language since the previous project were coded in Java. As in the previous assignment in the store method we update the structure (a dictionary containing as key a confidence and as value a set of frequencies of the patterns related to that confidence) but here the main difference is that we must update both confidence and frequency value and order each set of frequencies, for each singular confidence, in crescent order. We also keep a ordered list containing all the confidence values, in order to be able to know which confidence is the minimum one, this is useful when the search algorithm finds a bigger confidence value and so we must shift the structure, deleting the item with the smallest pair of confidence and frequency; to do so we need a second ordered list, that contains the frequency values related to the smallest confidence, this list is updated every time a pair of values must be deleted and when the min confidence value changes it is repopulated with the frequency values of the new min confidence value.
In the prune method we just check that the minimum frequency is respected if the confidence is less than the minimum one, or if the frequency of the checked pattern is less than the minimum frequency related to the minimum confidence. Obviously until our structures do not contain k distinct pairs (confidence, frequency) our algorithm continues to fill the structure, pruning only if the frequency is less than the threshold provided as input. In our implementation we compute both positive and negative confidence and we consider only the biggest one; the negative value is computed only if a flag, that signals that we must consider also negative examples, is set.

1.2 Training a basic model

To implement this second task we have followed the specification provided in the assignment and we have adapted the provided template (*example2.py*) using as classifier the *tree.DecisionTreeClassifier* class of *SciKit-learn* python package and using our task class, implemented for the previous task.

1.3 Sequential covering for rule learning

Also for solving this task we have implemented a cross-validation that follows the guidelines provided as template and for each fold of the k fold cross-validation, we iteratively call k times (k is the parameter provided as input) the gSpan algorithm with our implementation of the task to be ru,n but this time with max number of topK equal to one and with the flag that enables the task to consider also negative examples set to True, in order to find all the patterns with maximum score. At each iteration we update the subsets structure provided to the task (*gid_subsets* that is composed by four different lists containing respectively the positive training set, the positive test set, the negative training set and the negative test set) in order to exclude all the patterns that are classified by the last selected rule. As specified in the assignment when the gSpan algorithms returns more than one value the first in lexicographic order is selected. When our algorithm finds k rules, or when there are no more train examples, our algorithm performs the classification phase on the test set in order ot evaluate the performances. The rules found are stored in order into a list and so our classification algorithm just traverse the list of rules and check for each test example if there is a match. As described in the assignment we set the default class as equal to positive class if the maximum number of rules are positive or if positive class if both classes are equally present or if there is no transaction remaining, and negative only if the majority of the rules are negative. A pattern is classified with the default label only no rule matches the transaction.

1.4 Our classifier

To implement our classifier we have decided to adapt the previous task's implementation but introducing some improvements:

- the minimum frequency is computed according to the size of the data-set present in the path file specified in input, in particular we have decided, after having performed some different experiments, to set it to the 12,5 % of the total number of transactions present in either positive and negative data sets. We have decided to use exactly this percentage because we noticed that the performances were better, without affecting the accuracy of the results.
- the evaluation function is changed into the *Laplace rule evaluation metric* that is defined as: $Laplace = \frac{n_c+1}{n+k}$ where n_c is the number of examples in the training data-set classified with the right class by the pattern, n is the number of total occurrences of the pattern in the training set and k is the number of classes, that in this case is equal to 2 because we have only two types of labels: positive and negative. We have decided to choose this rule evaluation metric after trying three different approaches:
  - information gain evaluation metric;
  - our personal evaluation function: $f := n_c * N_c/n * N$ where N_c is the cardinality of the class associated to the rule in the selected data-set and where N is the total number of transaction in the selected data-set
  - laplace evaluation metric;

  After several attempts, also using different types of patterns ties management, we have observed that the best performances were achieved with the Laplace metric.
- We have decided also to use as size of *topK* a parametric value that depends on the size of the number of transactions, in particular the value is set to the 12,5 % of the total number of transactions; notice that this is a maximum value and sometimes our implementation covers the entire training data-sets with a smaller number of rules.

2 Experimental comparison of the obtained results

3 Notes

The biggest difficulty that we have encountered was that we have used Java in the other projects and not Python so we have had to adapt our solution of the previous project in another programming language. In addition, using a new library is always difficult because you have to spend time to understand the structures used, the methods already implemented and how to use them. We have wasted time to understand some pieces of the template provided because of the lack of comments in the code and in the second task, at the beginning, it was not clear which classifier to use: we tried with different classifiers, but each time we didn't obtain the same results of the example provided. Only after the clarifications of the assistant we have found the correct one with the correct parameters. The last issue that we have had to manage it was the general structure of the templates: having only lists we had to scan them each time and it was difficult to understand the structure used and so how to obtain the right fields. It took much time, we would have preferred to work with objects.