LINGI2364 - Mining Patterns
Project 3 - Classifying Graphs

Group 10
de Voghel Brieuc - 5910 1600
Mersch-Mersch Severine - 3298 1600

Friday 15th May, 2020

1 Introduction

In this project we used the gSpan graph mining algorithm to construct pattern-based classifiers for molecular data. We will first discuss their implementation and then we will analyse their performance and compare their results.

2 Phase 1 : Top-k mining with gSpan

2.1 Frequent Positive And Negative Graphs

In order to use the gSpan algorithm to find subgraphs among positive and negative classified graphs, we had to implement a PatternGraphs class that finds the k most confident patterns.
The confidence is computed as the maximum between the positive and negative confidence[1] :

$$confidence = max(\frac{p}{p+n}, \frac{n}{p+n}) \tag{1}$$

Based on the previous project and the given template, we had to modify three methods to accommodate for confidence for the gSpan algorithm: init, store and prune.

2.1.1 Init method

5 attributes allow us to keep track of the k most confident and frequent patterns : `most_confident` (a Heap containing the patterns), `min_support`, `min_confidence`, `subsets` and k (representing the number of patterns to find).

As a reminder of the last project, a Heap is a data structure easing the use of a min- or max-priority queue. It stores items that are tuples with the first element being the key to sort on, and the second element being the value.

2.1.2 Store method

The store method calls the add_sequence)() method created during the last project. It updates the `most_confident` Heap by adding the pattern found if its confidence and support are better than what has already been found.

2.1.3 Prune method

The prune method serves as a heuristic for pruning the search tree. It checks if the support is higher than the `min_support`. If it is the case, it returns False, otherwise it returns True : we have to prune since the support is not big enough.

[1] In this first phase, only the positive confidence had to be taken into account

2.2 Finding subgraphs

To sum up, in order to find the k most confident subgraphs with a specified minimum support, we first create a graph database and read the positive and negative files to add the graphs in the database. The graph ids are used as subsets.

Then we create a `FrequentPositiveAndNegativeGraphs` task and run the gSpan algorithm on it. This permits to fill the heap with the k most confident patterns having a support larger than the min support.

The final step is to print the output in the good format.

3 Phase 2 : Training a basic model

During this phase, the goal was to train a model based on patterns extracted from the previous phase.

K-fold cross-validation is used to train and test the model on different subsets of the database. For every fold, we call the `train_and_evaluate()` method. It uses the implementation of the previous task to find the k most confident patterns in a certain subset and then trains the model on them. Then, the model is evaluated against test examples of another subset. This allows to avoid optimistic bias in the computation of its accuracy.

The model that is used is a decision tree classifier from the Skikit-learn library.

4 Phase 3 : Sequential Covering for Rule Learning

During this phase, the goal was to implement a sequential covering algorithm to learn a rule list. Like the last phase, a K-fold cross-validation was used to train our model.

For every fold, the `train_and_evaluate()` is run. This method has been modified compared to last phase : instead of running once the gSpan algorithm (to get the k most confident patterns of a subset), we run it k times to only get the one most confident (still with a specified minimum support) pattern every time. This pattern is called a rule. It is the pattern that, lonely, classifies patterns the best.

By iteratively filtering the database with the graphs that can be classified with this rule, we can classify the whole database with a high confidence. Before running gSpan to find the next rule, it is thus important to remove the graphs that can already be classified with the rules previously found.

Finding these rules can be considered as the training. The training goes until there are k rules or until there are no transactions remaining. What is left to classify, is given a default value, according to the most represented class.

To classify new examples (to test the accuracy in our k-fold CV for example), the rules are checked one by one, starting with the first rule found. If no rule can classify the transaction, it is given the default value.

5 Phase 4 : Another classifier

For this last phase, we had to build a model that has a better accuracy than the previous two (decision tree from Scikit-learn and the sequential covering). The model we will work with is a support vector machine (SVM) from the Scikit-learn library. This model has been chosen after a comparison made with the random forest (RF) model. Since the SVM, when correctly tuned on its hyper-parameters, always out-performed the RF, we decided to concentrate on the comparison of the SVM model.

The data extraction works in the same way as the second phase : we run the gSpan algorithm and find all the k most confident patterns. Several parameters can thus influence the model's accuracy : the threshold of confidence we want for our patterns, and the number of patterns we want to train our model with. The last one has an important role in the overfitting of our classifier. A more in-depth analysis is made in the next section.

6 Comparison

6.1 Time and memory performance analysis

Time and memory performances analysis were made on the small dataset in order to have a benchmark performance comparison. The small dataset was chosen for the fact it would have taken way to much time (for still reasonable parameters) if a bigger dataset was used. The values are a mean over 5 runs. (Time indication

was highly dependant on the used profiler so it may not be most precise but serves as comparison.)

As we can see for the first phase (finding patters by gSpan, Table 1), k has not a big impact on time and memory usage. It has an effect though on the number of patterns found, but this effect is diminishing as min-support goes down, or when the number of patters gets saturated because of a too high min-support compared to the number of transaction.

Memory consumption isn't varying too much and time decreases exponentially as the minimum support goes up. This shows the effect of pruning the search tree of gSpan thanks to our min-support heuristic.

k	5	10	5	5	5	5	5	10	5	10	5	10
min-support	3	3	4	5	6	7	8	8	15	15	30	30
#patterns found	906	1060	154	166	80	59	37	54	10	21	4	4
time [s]	15.45	14.01	4.23	3.01	2.07	1.68	1.31	1.59	0.97	0.99	0.89	0.91
max used memory [MB]	84.16	84.80	84.32	82.57	82.32	82.09	81.82	81.85	80.98	81.01	79.71	79.80

Table 1: Performance analysis of phase 1 (small dataset) : finding subgraphs with gSpan

As phase 2 and phase 4 both also make use of gSpan with similar k and min-support parameters, we can expect a similar behaviour. The main differences lie in the fact n-fold cross validation is used. Thus, gSpan will be computed n times on the n different subsets. Every run will take a comparable amount of time, proportional to n as the whole dataset is split into n train and test sets.

As an example, for 5-fold cross validation, gSpan will be run 5 times. It will train on 80% of the data and will be validated against 20% of the data. As the validation (testing) takes a negligible amount of time, only data parsing (finding the patterns with gSpan) and model training will be time-consuming.

6.2 Accuracy performance analysis

The accuracy of a model depends on its implementation and its parameters. The parameters we can tune in phase 2, 3 and 4 to improve this accuracy is k (for the k most confident patterns or for the k rules to find) and the minimum support.

Doing a n-fold cross validation only serves to estimate the accuracy of a model. Changing the number of folds does not impact on the model's accuracy, although it may lead to computing a different estimate for the accuracy. Doing too many folds leads to too few validation points (leading the model to overfitting), and to few folds leads to too few training points (leading to a bad trained model). This is why cross validation is often done on 5 to 10 folds. This is why the comparisons are not dependent on the number of folds (5 seemed to be a good compromise of time and precision).

k	5	10	10	50	100
min-support (#transactions : 4335)	2000	2000	1000	1000	500
folds	5	5	5	5	5
mean of #patterns found	5	7	10	37	118
estimated accuracy	0.62	0.63	0.60	0.66	0.71
time per fold [s]	17.6	20.0	28.7	32.2	96.7

Table 2: Accuracy analysis of phase 2 (full dataset) : decision tree model training with gSpan

From Table 2 we can deduce that the number of patterns found by gSpan, and used for the training, has a large impact on model precision. Again, as long as the minimum support is the limiting factor, k has not a big impact.

The time explodes when more patterns are searched, but more patterns seem to be the key to a higher accuracy, task 3 has been analysed with the medium dataset. Also, k is now the number of rules found.

k	10	10	10	50	50	50
min-support (#transactions : 386)	20	50	100	20	50	100
folds	5	5	5	5	5	5
estimated accuracy	0.64	0.66	0.61	0.69	0.67	0.63

Table 3: Accuracy analysis of phase 3 (medium dataset) : sequential rule model

Here again, the number of patterns (rules) used for the model seems to be the key of accuracy. Although a decline in accuracy is observed when the min-support gets too high. No rule with such a high minimum support has a good confidence : its classification accuracy diminishes.

With the previous observations in mind and knowing that a high min-support limits the searching time, but that setting a too high min-support limits the number of available patterns, we tried to build a new model that could out-perform the previous models.

The SVM classifier was chosen for the accuracy it yielded despite the (sometimes) few training examples it was given. The patterns it is trained on are patterns immediately taken from gSpan (just like phase 2).

Also note that k and min-sup were now given as a fraction of the total number of transaction (Table 4). This allows to seamlessly scale to other datasets.

k	0.2	0.3	0.4	0.2	0.3	0.4
min-sup	0.1	0.1	0.1	0.2	0.2	0.2
folds	5	5	5	5	5	5
estimated accuracy	0.72	0.77	0.72	0.63	0.62	0.65

Table 4: Accuracy analysis of phase 4 (medium dataset) : SVM model

When run on the big dataset, our model performs with similar results, but taking way more time. This proves its scalability but limits the performance if the model wants to be run fast on large datasets.

7 Difficulties encountered

The biggest difficulty encountered was during phase 3 : after finding the k patterns used for the rules, we created a decision tree with the Scikit-learn library to classify the test. This lead us to almost the same solution as needed. We thus took a long time to finally being told that we shouldn't use the decision tree, but that we should apply the rules by ourselves. In fact, the rules are indeed used as a tree classifier, but a tree with only right or left branches. Using the rules to train a decision tree on is basically overfitting.

8 Conclusion

After understanding frequent graph mining algorithms and implementing multiple techniques, some in order to classify, we compared these different techniques.

We applied k-fold cross validation in order to compare different models on patterns found by the gSpan algorithm. We were able to build a model that can classify a dataset based on a number of confident patterns.