M2 Mathématiques Vision et Applications

Kernel Methods

Report

Data Challenge : Graphs classification

Team name : Matias & Ludovic

Authors :
Ludovic De Matteïs
Matias Etcheverry

Version du

April 14, 2023

1 Preliminar study

1.1 Problem

As a final project for the course on Kernel Methods, we took part in a data challenge. The goal was to learn how to implement and understand kernel methods algorithms to perform a classification task on graph data. In the current report, we will present the main conclusions and implementations of our work. Our work is made publicly avalaible[1].

1.2 Dataset analysis

In order to have a grasp on the main difficulties of the problem, we conducted a dataset analysis. The main conclusions we can draw are the following :

- The training set is strongly unbalanced, with about 90.75% of samples labeled 0 and 9.25% of samples labeled 1.
- There are 50 types of atoms and 4 types of bonds
- Some samples of the training set have unconnected components (some samples even have no edges)

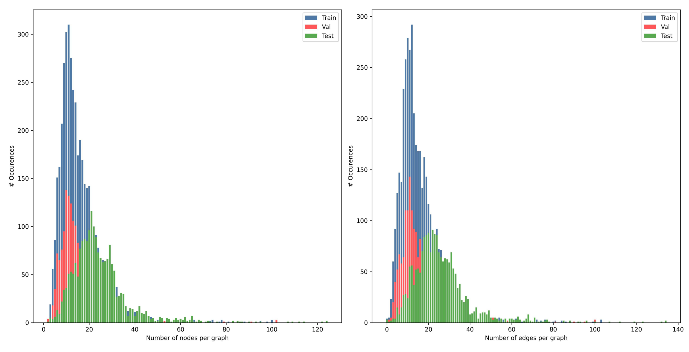

Figure 1: Comparison of training, validation and test dataset distributions in terms of graph nodes and edges.

We split the data in a train and validation set according to a 70-30% ratio. Fig. 1 shows the training, validation and test distributions of the nodes and edges in the datasets. For every split, we notice an asymmetric distribution with a heavy tail towards large number of nodes and edges per graph. Furthermore, the test dataset has a shift distribution compared to the initial data, which also introduces a shift distribution with the training and validation data. Solving this shift distribution problem through up or downsampling did not lead to better performance. We therefore left the distributions as shown in Fig. 1.

[1]`https://github.com/MatiasEtcheve/KM-graph-classification`

2 Classification algorithm

In order to perform the classification, we used a SVM classifier as we did in homework 2. However, in order to apply it with a large training dataset (4800 samples) with a large features space, we had to improve the optimisation step. To do so, we took advantage of the structure of the minimization problem (it is a QP problem) as *scipy.minimize* did not use this particular structure. We used *cvxpy* but other implementations are possible (for instance ProxQP, the current faster QP solver [1]). In order to fight the large class imbalance, the SVM is trained with specific weights on positive and negative samples. Specifically, the negative samples all share the same weights: $\frac{|\mathcal{D}|}{2|\mathcal{D}^-|}$, where $\mathcal{D}$ and $\mathcal{D}^-$ are respectively the whole dataset and the negative dataset. Similarly, the positive samples share the same weights: $\frac{|\mathcal{D}|}{2|\mathcal{D}^+|}$.

3 Kernels implementations

Based on the litterature on graph kernels [2], [3], including existing graph kernels implementations [4], [5], we tried to implement and to adapt to our problem several kernels. In this part, we will only present some of the kernels we consider as most important or yielding best results.

3.1 N-th order Random Walk Kernel

At first, we tried to implement a random walk kernel as we have seen in class. The adjacency matrix has been computed using the python library networkx. However, the Gram matrix obtained with our implementation is not positive semi-definite, which shows a implementation problem, preventing the use of kernel SVC. We suspect that this issue may come from some samples without edges that are ill-handled.

3.2 Node and edge histogram kernels

Another implementation we tried is vertex and edges histogram. It builds for each graph an histogram of the vertex (or edges) types and runs a RBF kernel on the obtained vectors. Even though this implementation gave correct results, it lacks understanding of the structure of the graph. Indeed, it cannot discriminate graphs with different structures if they yield the same histograms.

3.3 Weisfeiler-Lehman kernel

The final kernel we implemented is based on subgraph analysis. In this method, each of the nodes (or edges) are iteratively relabeled based on the neighbors labels. Therefore, nodes with similar neighborhood will get similar labels. An algorithm such as the histogram kernel presented above can then be applied to the relabeled graphs.
In the original presentation of this kernel, the authors proposed to use unique relabelling (only node with same labels and same neighborhood can have the same labels) but this is hard to implement in practice. Some implementations of this kernel use an hash key to label the nodes but this did not yield good results for our classification problem as it create huge differences in magnitude between the nodes labels.

4 Results

Scalability The large imbalance and the size of our training dataset invited us to compute feature vectors on a single graph basis compared to pairwise graph features. We were limited in times and in terms of computational resources. Therefore, we focused our work on the development of fast and scalable kernels.

Evaluation For each implemented kernel, we compute multiple evaluation metrics on the validation dataset: accuracy, recall and Area Under the Curve. The first 2 metrics report how a kernel behaves with a large class imbalance. The AUC metric is used to select the best kernel for the Kaggle submissions.

Hyperparameter search Each kernel is the source of an intense search for the optimal hyperparameter. On the one hand, we have solver hyperparameters such as regularisation, which in our case manifests itself by the maximum coefficient of a support vector. On the other hand, we have hyperparameters intervening in the kernel itself. As most kernels are wrapped around a RBF kernel, we have to choose the sigma allowing the best generalization. This hyperparameter search was conducted with the `Optuna` framework.

Results Table 1 shows the results obtained on different kernels. Our best result is obtained with a linear combination of two Weisfeiler-Lehman kernels. Specifically, an inital Weisfeiler-Lehman kernel is trained on the vertices of the graphs while another Weisfeiler-Lehman kernel is trained on the edges. The logits of both models are then linearly combined. It reaches a score of **0.86227** on the Kaggle leaderboard.

Kernel	Val AUC	Kaggle Score
Node Histogram	0.662	-
Edge Histogram	0.697	0.727
WL Edges	0.779	-
WL Nodes	0.787	0.801
Mix WL Edges - Nodes	**0.812**	**0.862**

Table 1: AUC scores on different kernels

Fig. 2 shows the average activation across the 2 trained Weisfeiler-Lehman kernels. The two kernels total 2811 support vectors out of 4200, which is a good sign of generalisation. Moreover, we notice that most support vectors have an activation coefficient around 29. This coefficient is the maximum coefficient available for negative classes as mentioned in Section 2. This figure shows that negative samples provide more classification information than positive samples.

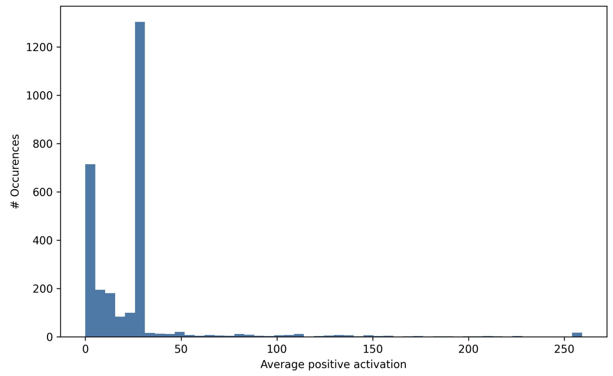

Figure 2: Average positive activation across the 2 trained Weisfeiler-Lehman kernels.

5 Conclusive remarks

Our work shows good generalization and performances. It would be interesting to study in depth the activation coefficient. Indeed, with such a large dataset, selecting the relevant graphs into a smaller dataset can lead to tremendous improved computational time. Removing least discriminative patterns in graphs can also highly improve the classification performances [6]. With more time involved, we would have looked for pairwise feature kernels, like optimal assignment. This special kind of kernel allows to create a mapping between 2 graphs, and is often a good sign of isomorphism. Moreover, it can be applied to graphs fed to Weisfeiler-Lehman algorithm.

References

[1] A. Bambade, S. El-Kazdadi, *et al.*, "PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond," in *RSS 2022 - Robotics: Science and Systems*, New York, United States, Jun. 2022. [Online]. Available: `https://hal.inria.fr/hal-03683733`.

[2] N. M. Kriege, F. D. Johansson, *et al.*, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, 2020.

[3] G. Nikolentzos, G. Siglidis, *et al.*, "Graph kernels: A survey," *Journal of Artificial Intelligence Research*, vol. 72, 2021.

[4] G. Siglidis, G. Nikolentzos, *et al.*, "Grakel: A graph kernel library in python," *The Journal of Machine Learning Research*, vol. 21, no. 1, 2020.

[5] M. Sugiyama, M. E. Ghisu, *et al.*, "Graphkernels: R and python packages for graph comparison," *Bioinformatics*, vol. 34, no. 3, 2018.

[6] T. Ma, W. Shao, *et al.*, "Graph classification based on graph set reconstruction and graph kernel feature reduction," *Neurocomputing*, vol. 296, 2018, ISSN: 0925-2312. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0925231218303217`.