Kernel Methods - Data Challenge Repport
Graph classification

Paule Grangette - Antonin Joly - Kaggle : *Paule + Antonin*

April 12, 2023

Abstract

The task in this Kaggle challenge was to classify graphs representing molecules into two classes representing a function that molecules bear or not. The goal was then to use graph kernels to get graph embeddings and then apply a classifier on them.

You will find our code here : `https://drive.google.com/drive/folders/1wwn-t6-zJVtzdVus2eyF8DaWYIgr6Rci?usp=sharing`.

1. Pipeline

Our pipeline consists in applying a kernel for graph to get graph embeddings, normalizing the features that we got, then applying a simple kernel such as linear or RBF kernel to them and finally applying a classifier. We implemented several kernels for graph : Weisfeiler-Lehman subtree/edge kernel, shortest path kernel, random walk kernel and graphlet kernel. To get the kernel of the dataset in classifiers, we applied either the linear kernel or the RBF kernel on the graph embedding given by the kernels for graph. Then we implemented four different classifiers : KernelSVM, Kernel Logistic Regression, Kernel Ridge Regression and GMM.

2. Kernels for graphs

There is three main types of kernels for graph : graphs based on walks and paths (e.g. random walk kernel, shortest path kernel), graph kernels based on limited-size subgraphs (e.g. sampling graphlet kernel) and graph kernels based on subtree patterns (e.g. Weisfeiler-Lehman kernels), as reported in [2]. We implemented at least one of each.

2.1. Subtree-based kernels

2.1.1 Weisfeiler-Lehman subtree kernel

The principle of this kernel is based on iterating a relabeling technique consisting in merging the label of a considered node with the labels of its neighbors and then relabeling the graph with new features. At each iteration, we count the number of each label present in the graph and merge it with counts from previous iterations. Figure 1 shows an example of the relabeling method on our data.

We set the number h of iterations of the algorithm to 2, which gives a feature vector of size 4080 (which corresponds to the initial feature vector and two iterations of the relabeling algorithm). With $h = 3$, the size is 20607 but computing the kernel of the training size made the RAM crash, so we kept $h = 2$. For the record, the results with the logistic regression with $h = 2$ and $h = 3$ were very similar. The computation time of the feature vectors of the whole dataset is remarkably fast, it only requires few seconds.

The implementation follows the article [2].

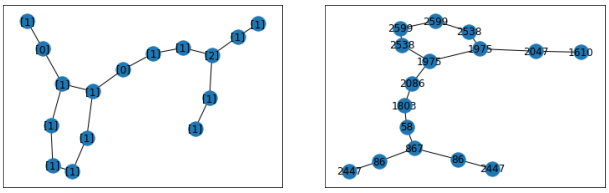

Figure 1. Graph from the training dataset (on the left) relabeled with the Weisfeiler-Lehman procedure (on the right) with $h = 2$

2.1.2 Weisfeiler-Lehman edge kernel

This kernel is a major update of the precedent kernel, as it does not only take into account the labels of nodes but also the label of edges. Indeed, we represent an edge label in the graph as (label_node_1, label_node_2, label_edge), where the first two values are the labels of ending nodes of edges and the last value the label of the edge. In a undirected graph, we could remove the last dimension. Besides the performance, this kernel is a real improvement as the size of feature vector is much smaller than the one of the subtree kernel for the same number of iterations. For $h = 2$, the feature vector size is 124, for $h = 3$, 134, and $h = 4$, 144. One other thing is that with the size reduction, we can compute until $h = 5$ iterations without RAM crashes.

2.2. Path-based kernel

2.2.1 Random walk kernel

The Random Walk kernel requires N^2 iterations where N is the number of graphs, (6000 in train). To have statistically

good result we would have to do at least 10 paths of length 5 to compute for each graph, for each node. This is quite long to compute. As it is less efficient than WL-kernel in most of the benchmark, we didn't try to optimize these codes with any compilation. To compute it faster we could implement a vectorization of each graphs thanks to random walks with algorithms such that Word2Vec.

2.2.2 Shortest path kernel

To implement the shortest path kernel, we used Dijkstra algorithm to get the shortest path between each node in a graph. In the case when there is isolated points, the distance given by the algorithm is set to infinity. We count the number of occurrences of each possible path length in the distance matrix to get feature embedding. Its results were pretty bad (as seen on Table 4), close to randomness. This is not surprising since this kernel does not take into account any label in its implementation. The implementation is inspired from the article [1].

2.3. Graphlet kernels

For the Graphlet kernel, we implemented two approaches : random Graphlets and all graphlets.

The random Graphlets consists in sampling n random graphlets for each graph of size m nodes. Then we can compare for the two graph the isomorphism between all their graphlets. Nevertheless this method is time consuming inasmuch as comparing the isomorphism of two graph is very long.

All graphlets consists in generating all the graphlets of size m nodes and then count for each graph how many graphlets of each type is present. It is quite useful to have a vectorization of each graph which lead to a faster computation. Nonetheless we were able only to count all the graphlets of size 3 inasmuch the number of graphlets of size m grows exponentiall with m.

3. Classifiers

We implemented four classifiers but only one worked well at the end which is the Kernel Logistic Regression. The metric used is the AUC score as the dataset was very unbalanced : the two-class repartition is : 90%-10%. Furthermore, we had isolated points in some graphs which caused some issues especially with the shortest path kernel. We implemented the Gaussian Mixture Model (GMM) classifier but did not work because feature vectors were sparse and let division by zero occur.

3.1. Kernel SVM

We used the optimization library *cvxopt* to implement this classifier, since with $scipy.optimizer$, the running time was extremely low. Unfortunately, we did not manage to make it work as we had each time issues with singular matrices or the optimal vector constant almost everywhere to the margin constant C. We tried to regularize on the kernel but it did not work too.

3.2. Kernel Logistic Regression

We tried two different versions of this classifier. The first was inspired from the courses slides using *solvingKRR*. The problem is that it is very dependant from its initialisation and did not give good results. The second version was based on implementing from scratch the Newton algorithm with gradient descent. Once more, we add issues with singular matrices (for both versions) so we add regularization in the resolution of the system with a lambda value of $1e-2$. However, when running on the training dataset, the RAM crashed after only one iteration of the Newton algorithm so, we reduced the algorithm to only one iteration and as the RAM crashed also when training on the whole dataset, we only used 80% of the training set to make our predictions on Kaggle.

4. Results

All our predictions made on Kaggle were got from 80% of the training dataset (due to memory issues).

Kernel	KLR (AUC)
Random Walk	Not computed(too long)
All Graphlets (size 3)	0.55 (on validation)
Shortest path kernel	0.53
WL-subtree (h=2)	0.63
WL-subtree (h=3)	0.63
WL-edges (h=2)	0.80
WL-edges (h=3)	0.79

Table 1. Results

5. Conclusion

During the project, we have really been challenged by the implementation of classifiers and singular matrices issues. The best result we got was when using the Weisfeiler-Lehman edge kernel combined with RBF kernel and the Logistic Regression classifier, with 0.80 of AUC score. The fact that the Weisfeiler-Lehman kernel gives the best result is consistent with the fact that this is the only which takes into account both node and edge labels.

References

[1] Linus Hermansson, Fredrik D. Johansson, and Osamu Watanabe. Generalized shortest path kernel on graphs, 2015. 2

[2] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12(null):2539–2561, nov 2011. 1