Kernel Methods : Kaggle Challenge - MVA 2022/2023

Vincent Garot vincent.garot@ens-lyon.fr
Julien Gaubil julien.gaubil@ens-paris-saclay.fr

April 15, 2023

1 Introduction

The provided dataset was composed of 6000 training graphs and 2000 test graphs with binary labels. They represent molecules: nodes for atoms (labeled by a number from 0 to 49) and edges for bonds within the molecule between atoms (labeled by an integer from 0 to 3). The task is a binary classification task on the graphs. Our implementation is available at `https://github.com/JulienGaubil/MVA/tree/main/Kernel-Methods-ML` and will be released when the final results on the private leaderbord will be available.

2 Approaches developed

2.1 Data visualization

We first visualized the data and noticed that this task is an imbalanced classification task: most of the training graphs had a negative label (0), which is likely to lead to low recall issues. We also noticed that most graphs were composed of approximately 10 atoms or less. We did not exclude any graph nor applied any pre-processing procedure to the kernels before using them in learning algorithms.

2.2 Kernel ML algorithms

We first describe the kernel Machine Learning algorithms developed to solve this task. The metric used to evaluate the submissions is the Area Under Curve (AUC), we therefore focused on algorithms that regress a score and predict a binary label based on the sign of this score. We developed the Kernel Ridge Regression, the kernel C-SVM and the Kernel Logistic Regression to solve these tasks. As for the implementation details, we fixed the penalization parameter λ for the Kernel Ridge Regression, using 5-Fold cross-validation for robustness. For the Kernel Logistic Regression, we also select this penalization parameter by cross validation, and fix a maximal number of iterations for Newton's method. The C parameter of the C-SVM is selected by 5-fold validation.

2.3 Kernels used

2.3.1 Random walk kernels

We then describe the kernels used to embed the graphs in a RKHS. We first tried two kernels presented in the course, namely Geometric and N-th order Random Walk Kernel, that have similar implementations. For both of them, a pre-processing stage is required to extract the adjacency matrix of the direct product of labelled graph to evaluate the kernels. By profiling our first approaches that used NetworkX methods to extract this adjacency matrix, we found out that most of the computation time was dedicated to this step. We therefore decided to re-implement it by ourselves in order to be able to scale to the computation of the kernel on the 6000 training samples. For the Geometric Kernel, we used the fast implementation proposed in [2] that leverages from Conjugate Gradient on sparse matrices. For the N-th order kernel, we leverage available GPUs to accelerate the pre-processing step with PyTorch to extract the adjacency matrix of the product graph.

For the Geometric kernel, we tried several values for the geometric parameter λ and observed that too high values ($\lambda > 0.1$) led to Nans in the resulting kernel, the series being non-convergent. We also chose the N hyperparameter of the N-th order Random Walk kernel using cross-validation.

These approaches resulted in low precision-recall AUC (typically less than 0.5) whatever learning algorithm they were coupled with. They presented very low recall by never predicting positive samples, which is a major struggle in this task with imbalanced classes, most training samples being negative. These low results can also be explained by the fact that these approaches do not make use of all the available data: they only take into account nodes labels to create the Graph product, and not the edge labels, which is a clear limitation. Using the SVM with the Geometric kernels obtained has been impossible due to non-singular matrix errors. For more details on the best scores obtained on 5-fold cross validation after tuning the parameters, please refer to the GitHub.

2.3.2 WL kernel

To further improve this performance, we then implemented the Weisfeiler-Lehman subtree graph kernel (WL), following [1] method.

This algorithm is based on the relabelling of the graph. We can replace each node by the subgraph of its neighborhood. Then we use an hashing function to compress this label and finally we count the number of this label appearing in the graph and store this data in a feature vector. We can reproduce the same algorithm on the graph we obtained and hence add more features. Finally to obtain the WL kernel we just can compute an inner product between two features.

Moreover, we decided to use edge relabelling as well as nodes relabelling since we believe that using only node information was an important limitation of our previous experiments. To do so, we added to the relabelling the labels of edges to the neighbors. However, we believe that we could make it more powerful if labels from the edges are not the same type as the node labels (the both of them are integers, but we could change and use a character for edge labels). Finally the hashing function we have isn't bijective but it was easy to implement and gave satisfying results.

This improved the results previously obtained by a significant margin. We therefore submitted these experiments on the Kaggle and tried two of the settings that yield our best results on 5-fold validation. The best results on the public leaderboard were obtained with the following settings: $C = 0.1$, $h = 2$ and tol $= 10^{-8}$ with edge relabelling. This led to a 0.86647 AUC on the public test set, ranked 16/53.

3 Possible improvements

Due to the constraint of implementing everything from scratch, we restricted ourselves to simple learning algorithms as well as simple kernels and tried to carefully implement them and tune them to achieve good results. This approach enabled us to rank 16th on the public dataset. However, more advanced kernels methods could have been tried, typically Graph Convolutional Kernel Networks. Using ensemble algorithms to get more robustness could also improve the performance on the prediction of the rare positive samples.

References

[1] Nino Shervashidze et al. "Weisfeiler-Lehman Graph Kernels". In: *Journal of Machine Learning Research* 12.77 (2011), pp. 2539–2561. URL: http://jmlr.org/papers/v12/shervashidze11a.html.

[2] S.V.N. Vishwanathan et al. "Graph Kernels". In: *Journal of Machine Learning Research* 11.40 (2010), pp. 1201–1242. URL: http://jmlr.org/papers/v11/vishwanathan10a.html.