Data Challenge : Kernel Methods

Emma Bou Hanna, Sebastian Partarrieu

ABSTRACT

This report is written as part of the Kaggle Challenge of the Kernel Methods course of the MVA master's degree. The goal of the challenge is to classify molecules that have a certain property by applying kernel methods on graphs to perform this classification task.

1 DATA EXPLORATION

Our training dataset consists of 6000 molecules, which are represented by graphs, in which nodes are atoms that form bonds with other atoms (edges). A binary label is associated to each of these molecules, characterizing the presence of a certain property. Therefore, the problem we aim to solve is a binary classification task on graphs. There is a strong class imbalance present in the training dataset, as seen in Figure 1 so we need to take this into account in our experiments. The testing dataset is made up of 2000 molecules. The training and testing dataset graphs have both node and edge features which are integer values.

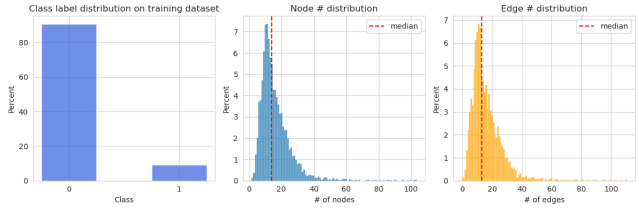

Figure 1: Visualizing the training data

2 IMPLEMENTING THE KERNEL

We started by comparing different kernel functions, and more precisely the associated representations $\Phi : G \in \mathcal{G} \rightarrow \Phi(G) \in \mathcal{H}$, where G is a certain graph in the molecule space $\mathcal{G}$. To do this, we first looked at [1], specifically Table 5, which compares the classification accuracy of a large number of different kernels (all with a downstream SVM) on 7 classification datasets with node-labeled graphs. Kernels with neighborhood aggregation mechanisms seemed to give very good performance without prohibitive complexity on these types of graphs and so we set out to implement the Weisfeiler-Lehman framework that we describe next.

2.1 Weisfeiler-Lehman algorithm

The Weisfeiler Lehman (WL) algorithm takes into account a node's neighborhood by operating an iterative relabeling procedure[2]. The time complexity for one graph is $O(|E|k)$ where k is the number of iterations of the algorithm and $|E|$ the number of edges, which makes the algorithm well suited to our dataset. The Weisfeiler Lehman algorithm proceeds as shown in Algorithm 1 :

Algorithm 1 Weisfeiler Lehman algorithm

Graph G, N nodes, H steps, f hash function
$C_n \leftarrow 1$
$i \leftarrow 1$
while $i \leq H$ **do**
  $L_n \leftarrow \{C_n, (C_m)_{m \in N_i}\}$ ▷ N_i = neighborhood
  $C_n \leftarrow f(L_n)$
  $i \leftarrow i + 1$
end while

C_n can be interpreted as a compressed label of node n, whereas L_n is a multi-set of compressed labels of each neighboring node. The hash function f should output a unique value for each representation, and can be chosen arbitrarily. We chose the BLAKE hashing function, which is widely used in cryptography.

2.2 Kernel Derivation

The general Weisfeiler Lehman framework introduced in [2] is as follows:

- Compute a sequence of h Weisfeiler Lehman graphs, thanks to the Weisfeiler Lehman algorithm with h iterations $G_1, ..., G_h$ and $G'_1, ...G'_h$ for two graphs G and G'
- Compute the Weisfeiler Lehman kernel

$$w_{kl}(G, G') = k(G_1, G'_1) + ... + k(G_h, G'_h)$$

where k can be any graph kernel. Note that each intermediate WL graph is compared thanks to the kernel function.

We chose a base kernel k that was first introduced in [3] as such :

$$k =< hist_G, hist_{G'} >$$

where $hist_G$ is the histogram of vertex labels, meaning the counts of compressed labels. More precisely, if

$$hist_G = \{"label1" : 4, "label2" : 10, "label4" : 1\}$$

and

$$hist_{G'} = \{"label1" : 4, "label3" : 10, "label4" : 1\}$$

then

$$k(G, G') = 4 * 4 + 10 * 0 + 0 * 10 + 1 * 1 = 17$$

We adapted this kernel for graphs with different numbers of "generalized nodes", by normalizing each histogram by the number of labels, giving the labels' frequencies.

3 IMPLEMENTING THE CLASSIFIER

Once we have chosen a kernel function and the associated representation, we set out to implement a few classification algorithms. Before testing out the classifier, it was important to create the right validation dataset as there is strong class imbalance in the training set. In our experiments, whether performing 5-fold cross validation or a simple train/val/test split, we used a balanced validation dataset with the same number of positive and negative examples. We then kept the model and the validation set for which we had the best performance.

3.1 Logistic Regression

The first seemingly obvious choice for a binary classification was to implement a logistic regression due to ease of implementation. We implemented an l1-regularized logistic regression using PyTorch. We tested a number of different hyperparameters (learning rate, batch size and the strength of the l1-regularization) and selected the best performing model on a validation set. This did not give us very high accuracy on the testing leaderboard, even when correcting the class imbalance by weighting our loss function, and so we chose to implement another classifier.

3.2 the Kernel Perceptron

The second classifier we tried out is the Kernel Perceptron, described in Algorithm 2:

Algorithm 2 Kernel Perceptron

Kernel $K_{ij} = K(G_i, G_j)$, training data $(G_i, y_i)_{1:N}$, $\alpha \in \mathbb{R}^N$, epochs M

$\alpha \leftarrow 0$
for $i = 1...M$ **do**
  Predict $\hat{y} = sign(\alpha^T K + \epsilon)$
  for $j = 1...N$ **do**
    if $\hat{y_j} \neq y_j$ **then**
      $\alpha_j \leftarrow \alpha_j + y_j$
    end if
  end for
end for
return α

Fixing $M = 300$, $\epsilon = 10^{-5}$, and performing a 5-fold CV with a balanced validation set gave us satisfactory performance on the leaderboard (see start.py script for reproducibility).

4 CONCLUSION

We implemented a WL-vertex histogram kernel along with a logistic regression and a kernel perceptron to tackle this graph classification challenge. Best results were obtained with the kernel perceptron. With more time and computational resources, another approach would have been to implement another WL-based kernel (such as WL-Optimal Assignment) and a few other classifiers before running a large gridsearch to find the best combination of kernel + classifier along with their best hyperparameters.

REFERENCES

[1] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. 2021. Graph Kernels: A Survey. *Journal of Artificial Intelligence Research* 72 (nov 2021), 943–1027. https://doi.org/10.1613/jair.1.13225

[2] Nino Shervashidze. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12.9 (2011).

[3] Mahito Sugiyama and Karsten Borgwardt. 2015. Halting in Random Walk Kernels. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/31b3b31a1c2f8a370206f111127c0dbd-Paper.pdf