Kernel Methods for Machine Learning - Kaggle Challenge
Classification of molecular graphs

Team name: MarCar

Carlos Santos García
ENS Paris-Saclay
carlos.santos@student-cs.fr

Martin Van Waerebeke
ENS Paris-Saclay
martin.vw@student-cs.fr

Abstract

This document presents our different implementations of kernel-based methods for graph binary classification, as well as some conducted experiments, in the context of the MVA course Kernel Methods for Machine Learning. Our final best method achieved 0.864 AUC score on the public set. Our code and results are available here.

1. Problem Definition

This challenge consisted in performing binary classification on molecular graphs. Let $\mathcal{D} = (x_i, y_i)_{1 \leq i \leq N}$ be the training dataset available, composed of N labelled graphs (here $N = 6000$). Each molecule x_i was composed of labelled nodes (atoms) and labelled edges (chemical bonds), and had an associated binary label $y_i \in \{-1, 1\}$ representing some chemical property of the molecule. We relied on support vector machines (SVMs) and kernels for graphs to solve the classification problem. This consisted in finding the function $\hat{f} \in \mathcal{H}$ where $\mathcal{H}$ denotes the RKHS of a chosen kernel K solving the following problem:

$$\underset{f \in \mathcal{H}}{\text{minimize}} \ \frac{1}{2}\|f\|_{\mathcal{H}}^2 + C \sum_{i=1}^{N} \xi_i \quad \text{subject to} \begin{cases} y_i(f(x_i) + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \tag{1}$$

The parameter $C \in \mathbb{R}$ controls the regularity of the function $f \in \mathcal{H}$, and was systematically found through cross-validation. Solving this problem is equivalent to solving a finite-dimensional quadratic problem by the Representer Theorem. Our implementations relied on `cvxopt` Python module, which was several orders of magnitude faster than `scipy`'s general purpose optimizer.

As the available data was highly unbalanced, with a ratio of positive to negative labels of 1:10, we decided to penalize differently mistakes on each class. To do this, we transformed problem 1 by introducing constants C_1 and C_{-1}:

$$\underset{f \in \mathcal{H}}{\text{minimize}} \ \frac{1}{2}\|f\|_{\mathcal{H}}^2 + C_1 \sum_{y_i=1} \xi_i + C_{-1} \sum_{y_i=-1} \xi_i \quad \text{subject to} \begin{cases} y_i(f(x_i) + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \tag{2}$$

These constants are chosen inversely proportional to the size of each class, giving more importance to less-represented classes and significantly improving our results. The only preprocessing step we performed was to remove the 7 molecules that had no bonds in it (*i.e.* they consisted of individual atoms only). This left us with 5993 data points.

2. Graph Kernels

A key aspect of the problem was that it was difficult to know beforehand the most appropriate graph kernel to use for this problem. We therefore implemented the following kernels to test on our dataset.

2.1. RBF kernel on handcrafted-features

A simple but efficient kernel we tested was representing each graph G as a vector $u_G \in \mathbb{R}^K$ of features. We included node labels histogram, node degrees histogram, edge labels histograms, graph density and graph diameter. If two graphs G_1 and G_2 were represented by features u_{G_1} and u_{G_2}, the kernel value between the two graphs is:

$$K(G_1, G_2) = \exp\left(-\frac{1}{2\sigma^2}\|u_{G_1} - u_{G_2}\|^2\right) \tag{3}$$

The optimal value of σ was found performing cross-validation.

2.2. Walk Kernel

To compute the walk kernel, for two graphs G_1 and G_2 we computed their product graph $G_\times$ with adjacency matrix

$A_\times$. We then computed the walk kernel of order N as:

$$K(G_1, G_2) = 1^\top A_\times^N 1 \tag{4}$$

The parameter N controls the length of the walks used to assess the similarity of the graphs. This implementation however makes this kernel very costly in terms of computational cost, since it needs creating the product graph of each possible combination of graphs.

2.3. Shortest Path Kernel

Shortest path kernels were introduced by Borgwardt and Kriegel in [1], and measure the similarity between graphs by comparing the shortest paths between their nodes. Our implementation relies on the Floyd-Warshall transformation of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where each graph is densified and each edge between nodes gets as label the shortest path between those nodes in the original graph. We then compute:

$$K(G_1, G_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k_{\text{walk},1}(e_1, e_2). \tag{5}$$

Where k_{walk} is a kernel on edge walks of length one. More specifically :

$$k_{\text{walk},1}(e_1, e_2) = \begin{cases} 1 \text{ if } l(e_1) = l(e_2) \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

With l measuring the shortest path between the two nodes of a given edge. We also tested taking into account the labels of the endpoint atoms, as following:

$$k_{\text{walk},1}(e_1, e_2) = \begin{cases} 1 \text{ if } l(e_1) = l(e_2) \wedge u_1 = u_2 \wedge v_1 = v_2 \\ 0 \text{ otherwise} \end{cases} \tag{7}$$

Where $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$.

2.4. Weisfeiler Lehman Subtree Kernel

The Weisfeiler Lehman (WL) kernel was introduced by Shervashidze *et al.* in [2] and is based in a label enrichment technique, since consists in assigning labels to the nodes of each graph and iteratively updating those labels based on the labels of the nodes in their local neighbourhoods. Our initial labels correspond to the chemical atoms ones, and we iteratively update them by creating new labels up to a given depth D. When this procedure is finished, we can compare the representation of two graphs in terms of their WL features to compute their similarity. In our tests, we implemented both a linear and RBF kernel using the WL features.

3. Experiments

Solving problems 1 or 2 required appropriate tuning of the regularization parameter C, as well as other kernel-specific parameters (σ, D, N). When possible, the optimal parameters were found by performing a 5-Fold stratified cross-validation to preserve the original class-balance.

The computational complexity of our methods forced us to rely on pre-computed kernels that were then used to train the SVMs. Gram matrices took between 5 minutes (RBF, WL) and 20 hours (Walk Kernel) to compute.

Inspired by `Grakel`'s documentation, we tested normalization of our Gram matrices by dividing each $K(G_i, G_j)$ value by $\sqrt{K(G_i, G_i)K(G_j, G_j)}$. This was particularly useful for Shortest Path and Walk kernels, where kernel values could get relatively big. Normalizing the kernel matrix improved the overall results.

4. Results

Our results are shown in figures 1, 2, 3 and 4, and raw data is available here. Overall, feature-based kernels like RBF or WL outperformed other graph specific kernels in our setting. These kernels were fast to compute, only requiring a few minutes to obtain train and test matrices, and allowed more hyperparameter tuning than walk or shortest path kernels. Due to its complexity and seeing that the results were not as promising as with other kernels, the order N controlling length walks was not extensively tested.

Best results per kernel were: WL kernel ($\sigma = 5$, $D = 4$, $C = 1$, AUC $= 0.910$), Gaussian kernel ($\sigma = 5$, $C = 0.5$, AUC $= 0.872$), Shortest Path kernel (with normalization, $C = 10$, AUC $= 0.813$) and Walk Kernel ($N = 3$, with normalization, $C = 50$ AUC $= 0.769$). Our final submissions relied on averaging predictions after cross-validating.

5. Conclusion

The kernel trick appears as a powerful tool that allows solving graph classification problems with SVMs, achieving competitive results. These methods seem however difficult to scale, especially for graph-specific kernels, and the question of choosing the most appropriate kernel for a given task remains unanswered. More advanced techniques, combining for example WL label enrichment on top of walk kernels, or using weighted sums of kernels, could further be explored to improve our results.

References

[1] K.M. Borgwardt and H.P. Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–, 2005. 2

[2] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011. 2

A. Experimental results

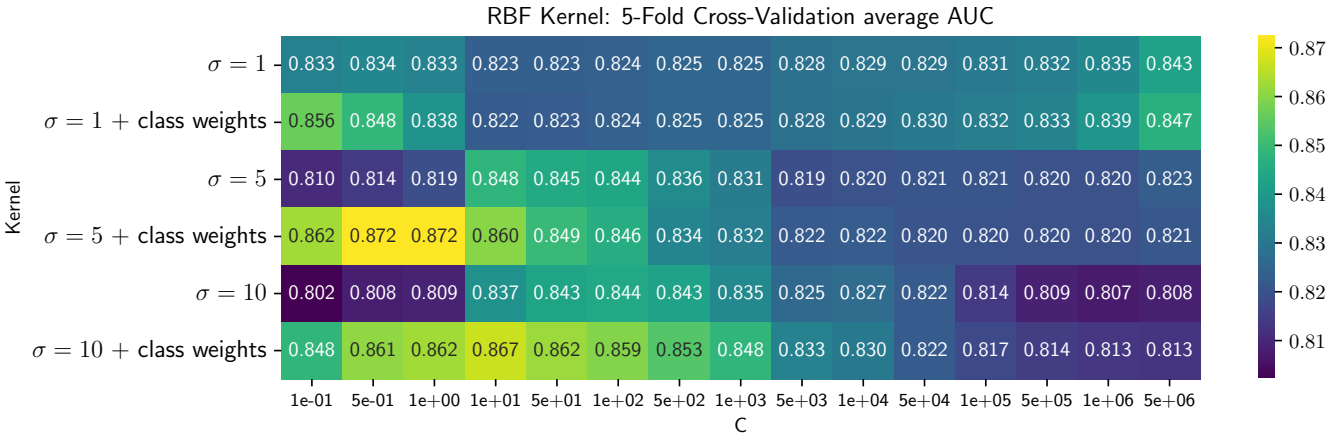

Figure 1. Cross-validation results obtained with RBF kernel and different values of σ. When using class weights, we used the framework presented in 2 with constants C_i inversely proportional to the size of each class $i \in \{-1, 1\}$.

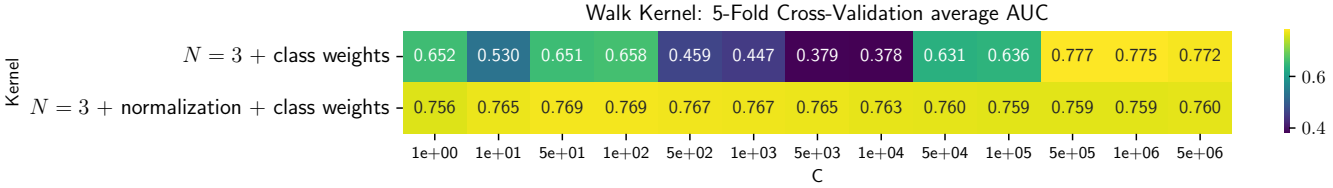

Figure 2. Cross-validation results obtained with our N^{th} order Walk Kernel. Due to the high computational cost of this kernel, taking more than 20 hours to compute the necessary matrices for the experiments, and seeing that other kernels provided better results, no further experimentation was made on the influence of N. When using class weights, we used the framework presented in equation 2 with constants C_i inversely proportional to the size of each class $i \in \{-1, 1\}$. As can be noted, normalization significantly improves the performance of this kernel. We considered here that lower values of C were more suitable for the challenge since regularization could reduce the risk of overfitting, and therefore selected $C = 50$ as our best value.

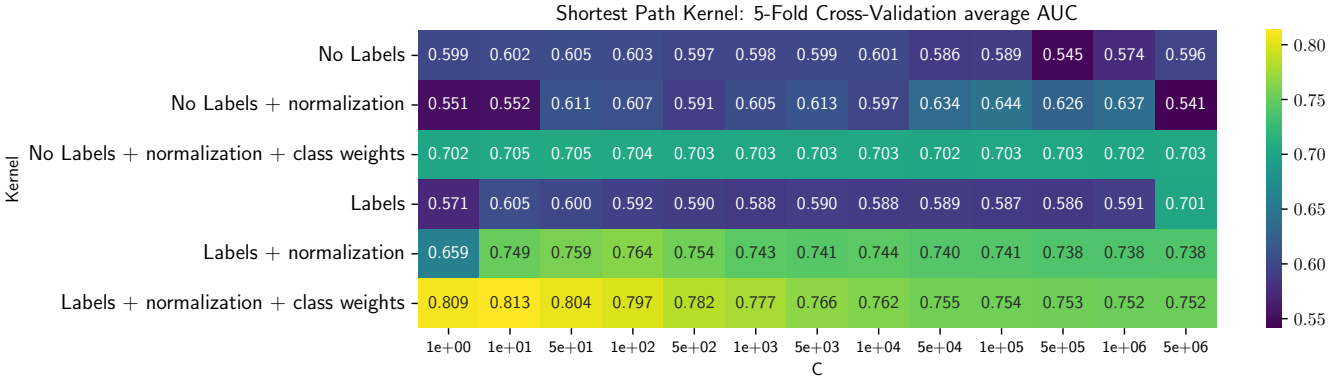

Figure 3. Cross-validation results obtained with two versions of our Shortest Path Kernel. As defined in equations 6 and 7, the labelled shortest path kernel compares paths where both endpoints have the same atom categories and having the same shortest path length. When using class weights, we used the framework presented in 2 with constants C_i inversely proportional to the size of each class $i \in \{-1, 1\}$. When using the labelled version of the shortest path kernel, normalization and class weights seem to highly improve the expressivity of the kernel.

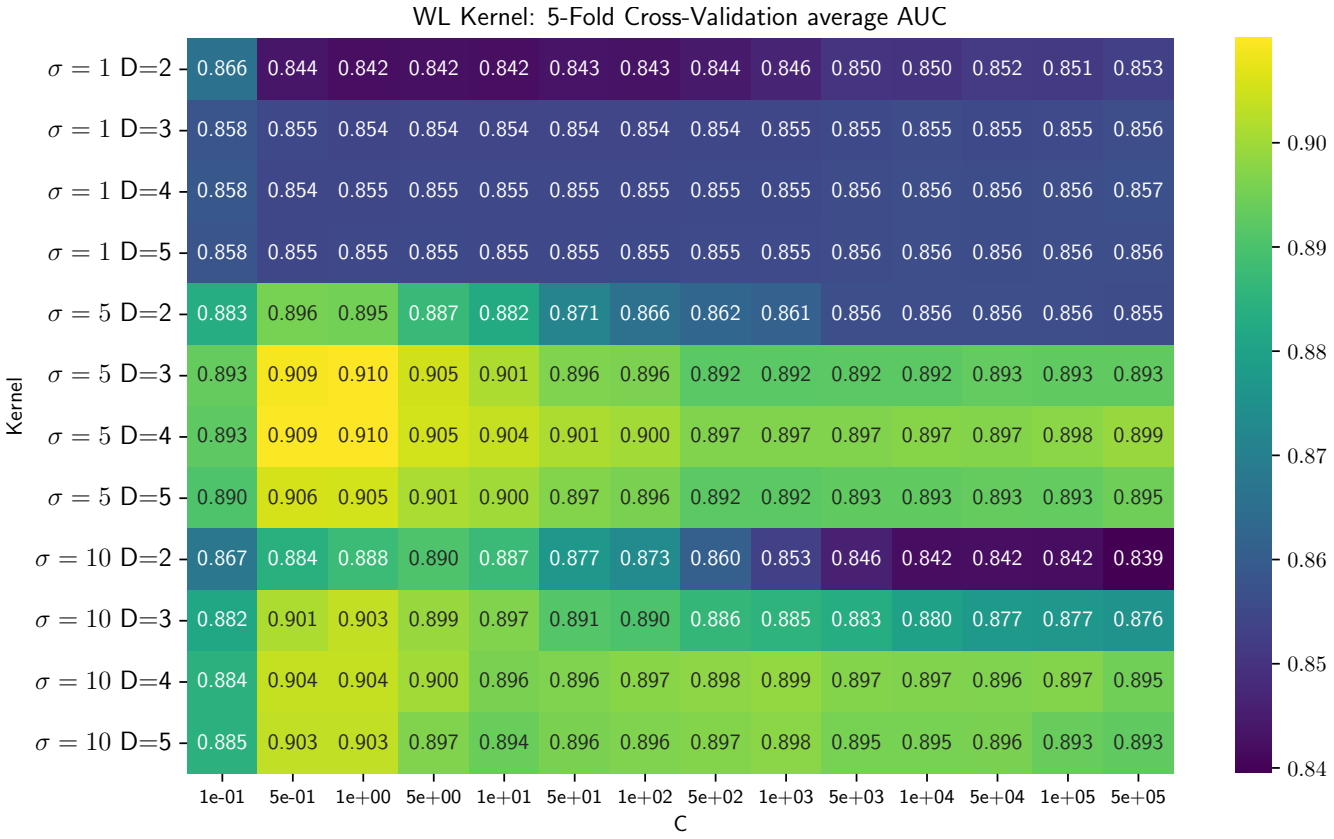

Figure 4. Cross-validation results obtained with our Weisfeiler Lehman (WL) Kernel. WL features are extracted from each graph and compared between two molecules with a Gaussian kernel parametrized by σ. The value of D accounts for the depth used in the WL kernel to iteratively update labels using local neighbourhoods: the higher the value, the more specific the labels. We systematically used class weights, as presented in 2 with constants C_i inversely proportional to the size of each class $i \in \{-1, 1\}$.