Kernel Methods for Machine Learning : Data Challenge Report

Binary classification of molecules using Graph Kernel Methods

Eugène Berta
Télécom Paris, IP Paris
MVA, ENS Paris-Saclay
eugene.berta@gmail.com

Yannis Cattan
Mines Paris, Université PSL
MVA, ENS Paris-Saclay
cattan.yannis@gmail.com

1 INTRODUCTION

This report presents our work on the Kaggle Challenge organized within the course 'Kernel Methods for Machine Learning' of MVA master program. The aim of the challenge is to perform binary classification on molecules using graph kernel methods.

This report is organized as follow: we first conduct a dataset analysis, then we explain our choice of methods based on the previous analysis. We finally explain our implementation (**Weisfeiler-Lehman subtree kernel** combined with a **Support Vector Classifier**) and present hyperparameter tuning steps that led to a significant validation score improvement.

Our code is publicly available here: https://github.com/eugeneberta/mva-kernel.

2 DATASET ANALYSIS

This analysis sums up a more thorough one which can be found in our dedicated notebook: *dataset_analysis.ipynb*.

The training and test sets respectively contain 6000 and 2000 molecules represented by graphs. Each molecule contains atoms (nodes) labelled with an integer from 0 to 49. Each bond (edge) is labelled with an integer from 0 to 3.

In the training set, we observe that the median length of molecules is 14 atoms. The distribution of atom labels is skewed : more than 94% of the atoms are labelled with 0, 1 or 2. Moreover, we observe a **heavy class imbalance in the training set**, with only 9.25% of positive-label graphs. Finally, the data contains a small proportion of **unconnected graphs** (see Figure 1). We made the assumption that the prediction problem concerned molecules (connected graphs only) and we decided to **clean the dataset** (*clean_dataset* function in *utils.py*). First, we removed isolated atoms (Figure 1, right) in both the training and test sets. Then, we discarded more complex unconnected graphs (Figure 1, left) from the training set only, resulting in a cleaned training set of size 5774. This means we use a different processing for the training and test sets. Still, we consider (and observe empirically based on validation score) that cleaning our data is worth this compromise.

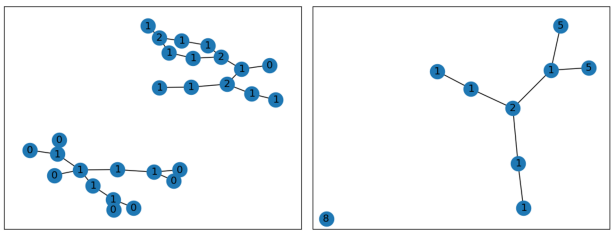

Figure 1: Two main types of unconnected graphs.

3 CHOICE OF METHOD

To make a relevant choice of method, we relied on a review on graph kernels published by Kriege et al. [1]. In this survey, the authors thoroughly define various graph kernels, conduct an in-depth performance comparison of these methods, and derive a practitioner's guide to kernel-based graph classification.

Using their practitioner's guide, we deduced that using explicit methods was critical given our large dataset (6000 molecules) and limited computing resources. Indeed, it is in general more computationally expensive to compute the gram matrix of a large dataset using an implicit method than with an explicit one.

Given the performances of the various explicit methods, we decided to move along with the **Weisfeiler-Lehman (WL) subtree kernel**. We refer the reader to the section 4.1 for an explanation of this kernel.

On top of this graph kernel, we decided to use a **Support Vector Classifier**. It is well known to be a top performing algorithm in kernel methods and machine learning in general. Moreover, we could rely on our implementation from the second homework of the kernel course.

4 IMPLEMENTATION

In this section, we introduce the methods we used to classify the molecules.

4.1 Weisfeiler-Lehman subtree kernel

Theory. The WL subtree kernel relies on the WL transform. This transform leverages the expressiveness of a graph by relabelling each node given its label and the ones of its neighborhood. By doing it in an iterative way, one can extract local structures within the graph. We refer the reader to the cited survey for a thorough explanation (with a nice figure) of this transform. The output of this transform is a vector $\phi(G)$ representing the graph G. Then, we can compute the WL subtree kernel between two graphs by computing the inner-product on a RKHS $\mathcal{H}$ between their feature vectors:

$$K_{WL}(G1, G2) = \langle \phi(G1), \phi(G2) \rangle_{\mathcal{H}}$$

We decided to use the RKHS associated to the Gaussian kernel:

$$\langle \phi(G1), \phi(G2) \rangle_{\mathcal{H}} = \exp\left(- \frac{||\phi(G1) - \phi(G2)||^2}{2\sigma^2} \right) \quad (1)$$

Using the Gaussian kernel is a common practice which gives, according to the survey cited above, the best performances. Moreover, it adds an additional parameter σ that, provided it is wisely tuned, makes the model more adaptive to the problem. We also tried the linear kernel which led to worse performances as expected. Yet, we did not try the Laplacian kernel which leverages the l_1-norm instead of the l_2-norm of the Gaussian kernel.

Implementation. We implemented our own version of the WL transform (see *WL_features* function in *utils.py*). Then, we used our function *WLK_l2_norm* to compute $||\phi(G1) - \phi(G2)||^2$. Finally, we

computed the desired quantity (1) by dividing by $-2\sigma^2$ and composing with the exponential. To check that we duly implemented WL subtree kernel, we first conducted by hand checks on several simple molecules, and then we plugged it into a simple k-nearest neighbors algorithm which achieved 0.68 AUC on a validation set, making us think that signal was indeed extracted from our kernel.

4.2 Support Vector Classifier

QP solver. We first experimented with the SVC that we implemented for the second homework of the class. However, we realised that the speed of the optimization procedure would be prohibitive if we wanted to try different kernels and tune the hyperparameters (several hours of computing to fit a single Gram matrix). We decided to implement our solver with **the open source library CVXOPT** [2] (see *KernelSVC* in *utils.py*). CVXOPT is clearly not the fastest solution out there [3] but it is easy to handle, it has a dedicated function to solve quadratic programs and we had already used it in previous courses in the MVA. After a few tries, we realised we could fit a 5000×5000 gram matrix in about one minute, which was more than enough to run our experiments, so we decided not to further optimize our solver.

SVC for imbalanced classes. Support Vector Classifiers are known to be sensible to class imbalance in the training data. Contrary to neural networks, we cannot simply build a balanced dataset by oversampling the under-represented class during the training procedure. However, a widely used method to handle this, introduced in [4], is to use a **custom penalty term (C value) for each point in the dataset**. Classically, in binary classification, a smaller weighing is used for the over-represented class ($C_{small} = w_{small} * C$) to apply a larger penalty on miss-classified examples, thus forcing the margin to be harder on the majority class and softer on the minority one. A simple way to tune this is to use a ratio $w = \frac{w_{large}}{w_{small}}$ equal to the proportions of the two classes in the dataset. We decided to go further and to allow for more flexible C values. In our experiments, we set $w_{small} = 1$ for the negative samples (majority class) and a **custom** $w = w_{large} > 1$ **for the positive samples** (minority class). This gave us an additional parameter w to tune for our model.

4.3 Hyperparameters tuning.

Cross validation. To validate our experiments in a robust way, we implemented a **k-fold cross validation** on our 5774 training samples. Our function *stratified_cross_val* is optimal in the sense that the full Gram matrix on the training set needs to be computed only once. It uses indexing directly on the matrix to split between validation and training. We use a 6-fold cross validation as we consider that ~ 1000 points is enough for validation. This procedure resulted in high score variability on the different folds. We realised that the high imbalance of our training data makes it important to use **stratified cross validation**. To this end, we first split the data between 0-labeled points and 1-labeled points and we apply a 6-fold split on these two sets separately before re-building balanced folds by merging again. This ensures that the proportion of positive values in the training set is preserved within each validation set. We observed much more stable results with this new safeguard.

Grid search. The strength of our approach is that we managed to have many degrees of freedom in the kernel and the SVC (C value, variance of the RBF kernel σ, positive class weight w to apply to C, depth of the WL subtree h). This way, we could find the optimal set of parameters for the problem at hand with a grid search. To reduce the dimensionality of the parameter space, we first tried a few models by hand to identify a reasonable range for our grid search. Among other things, we realised that taking $h > 2$ in our WL subtree led to rapid over-fitting so we decided to keep $h = 2$ for all our experiments. Figure 2 illustrates one of our grid searches, showing 6-fold cross validation ROC AUC score for models trained on the **cleaned dataset** with different sets of parameters.

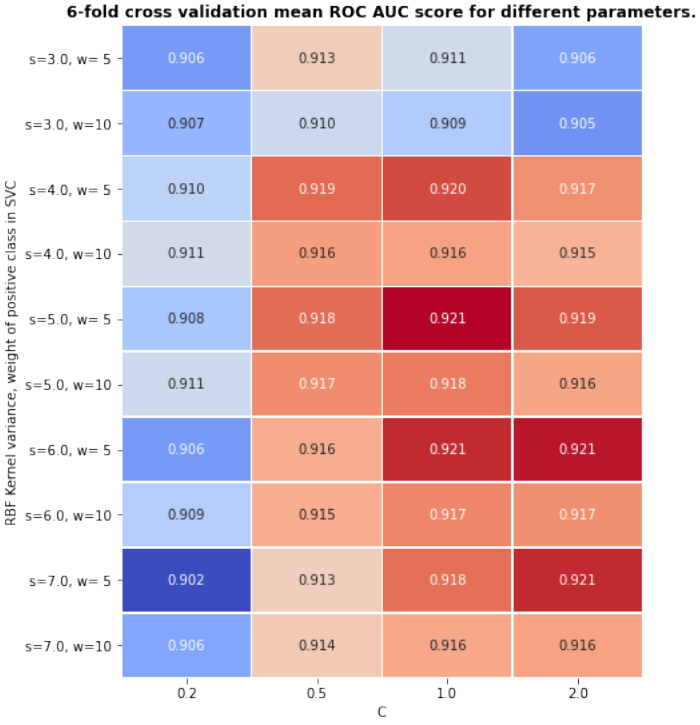

Figure 2: Grid search on model hyperparameters. The redder the better.

5 ALTERNATIVE METHODS

This short report is not appropriate for an in-depth discussion about every method that we tried during this project. Still, among other things we also experimented:

- Using implicit kernels (n-th order walk, ...) which revealed very long to compute on our dataset, this confirmed our intuition that explicit feature computation would prove an asset.
- Using Kernel Ridge Regression or even K-Nearest Neighbors to fasten the fitting of our classifiers. As expected the performances of these two methods were not competitive.

6 SUBMISSIONS

For our final submissions, we trusted our grid search results in Figure 2 and we used $\sigma = 5.0, w = 5, C = 1.0$. However, we observed better generalization results on the public test set with the slightly sub-optimal $\sigma = 4.0, w = 5, C = 0.5$, which can be explained by the higher regularization and a potential data drift on the test set. We decided to select this model as our second choice. We obtained our best scores both on public and private set with this second model: 0.866 AUC and 0.844 AUC respectively.

REFERENCES

[1] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1), jan 2020.
[2] Joachim Dahl Martin S. Andersen and Lieven Vandenberghe. Cvxopt.
[3] Stéphane Caron. qpsolvers benchmark.
[4] XULEI YANG, QING SONG, and YUE WANG. A weighted support vector machine for data classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(05):961–976, 2007.