Kernel methods - Kaggle challenge

Ambroise Odonnat
Master MVA
ENS Paris-Saclay
ambroise.odonnat@eleves.enpc.fr

Roman Plaud
Master MVA
ENS Paris-Saclay
roman.plaud@eleves.enpc.fr

Abstract

The purpose of the challenge is to classify graphs into 2 *different categories and achieve the highest ROC AUC score on the test set provided with* 6000 *training graphs. We heavily studied the data to choose an adequate kernel and developed a fast classifier. Our implementation is available at https://github.com/AmbroiseOdonnat/Kaggle-Graph-Classification and here.*

1. Introduction

This report aims at giving an overview of the work done for this assignment. We will not detail the choice of hyperparameters but rather give some insights into our method and reasoning.

2. Dataset

The dataset we were given is extracted from an unknown source, therefore we cannot include any prior knowledge of the data except that we deal with graphs representation of proteins. Therefore, we have adopted a fully data-driven method.

2.1. Statistics

Each dataset of the training set in a graph$\mathcal{G}$ where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ are the nodes and $\mathcal{E}$ are the edges which connect two nodes of a graph. Each node (resp. edge) comes with a label ranging from 0 to 49 (resp. 0 to 3). Each graph has a binary label (0 or 1)

Split	$\#\lvert\mathcal{G}\rvert$	Avg. $\lvert\mathcal{V}\rvert$	Avg. $\lvert\mathcal{E}\rvert$
Train	6000	16.14	24.45
Test	2000	16.26	25.75

Table 1. Dataset Statistics

We observe a high imbalance in the distribution of node labels as shown in Figure 2 and also of edge labels. Indeed, 71% of nodes in the dataset have label 1, and 62% of edges have label 0 while the other possible labels are underrepresented. We can thus intuit that the information lies in the structure of the graph itself rather that in its attributes. We display in Figure 1 a representation of graphs extracted from the dataset.

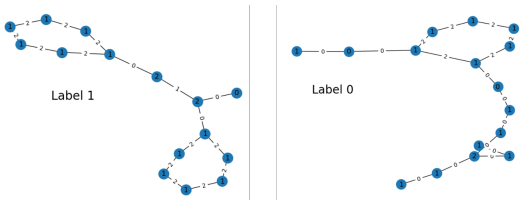

Figure 1. Example of two graphs extracted from the training set

2.2. Imbalanced Classification

A very important thing to note is that the training set is highly unbalanced with class 0 being largely overrepresented, as we can see in Table 2.

Split	# of label 0	# of label 1
Train	90.75%	9.25%

Table 2. Class Repartition in the Training Set

2.3. Data Preprocessing

To prepare our data we perform two preprocessing steps.

- Make graphs undirected
- Get rid of graphs with no edges

The first step is empirical: we noticed that the original graphs were directed and that adding the mirror edge for each edge of each graph boosted the performance. Making the graphs undirected was a fundamental step to reaching our results. The second step is a technical condition to make our algorithm work. Very few graphs (7) were concerned with that condition. Moreover, in what follows we do not use any edge labels.

3. Evaluation

Due to the imbalance between classes, the evaluation is performed with the ROC AUC metric.

4. Kernels

In this section, we list the kernels we used to then perform classification.

- Random Walk Kernel
- n-th Order Kernel
- Shortest Path Kernel
- Vertex Histogram Kernel

Those choices are mainly motivated by the course and the type of data we deal with. We do not detail their specificity as it was already done in the course. The Shortest Path and the Random Walk Kernels were not competitive so we will not present them in our final results. It should be noted that the computation time to obtain the kernel matrix is very high except for the Shortest Path Kernel.

5. Label enrichment

As in the course, we performed the Weisfeiler-Lehman (WL) procedure to enrich the node labels. This procedure was combined with the vertex histogram and the shortest path kernels. Considering the computation time of the methods, we could not use cross-validation for each hyperparameter. We set the number of iterations of the WL procedure to 5.

6. Classifiers

In this section, we list the classifiers we used.

- Support Vector Classifier (SVC)
- Kernel Logistic Regression (KLR)
- Multiple kernel learning (MKL)

One of the main difficulties of this challenge was implementing efficient classifiers. In particular, we had to reformulate the SVC optimization problem. The SVC and the KLR provided very similar results while the MKL showed slightly better results but at a very high computational cost. In the end, we focused on the SVC.

7. Training methodology

We split our training data in a training and a validation set with a split of 90%. We have three main limitations:

- Class Imbalance: prediction will be biased towards the over-represented class
- Time complexity to compute the matrix of kernel quadratic with the number of data
- Time complexity to perform classification

To contravene these three drawbacks, a solution is to implement an aggregation strategy. For a given kernel K and classifier clf, we train n models with N randomly chosen training data for each as follows:

1. for the model $i \in [0, n-1]$, randomly select $N/2$ examples from each class, compute the kernel matrix over these N data, and then fit the classifier. Evaluate on the validation split and predict on the test set.
2. We have n predictions for each test graph, we aggregate these predictions with an aggregation function to obtain a final prediction for each test graph.

We implemented 5 aggregation functions:

- $\mathrm{mean}(x_1, \ldots, x_p) = \frac{1}{p} \sum_{i=1}^{p} x_i$
- $\mathrm{median}(x_1, \ldots, x_p)$
- $\mathrm{sign\_mean}(x_1, \ldots, x_p) = \mathrm{sign}(\frac{1}{p} \sum_{i=1}^{p} x_i)$
- $\mathrm{mean\_sign}(x_1, \ldots, x_p) = \frac{1}{p} \sum_{i=1}^{p} \mathrm{sign}(x_i)$
- $\mathrm{voting}(x_1, \ldots, x_p) = \mathrm{sign}(\frac{1}{p} \sum_{i=1}^{p} \mathrm{sign}(x_i))$

8. Results

In this section we show the results obtained on the public test score. Before submitting on Kaggle, we ran several trials with the classic train/validation framework. It appears that the Weisfeiler-Lehman procedure combined with the Vertex Histogram Kernel was the best trade-off between computation time and performance. We focused on this model and evaluated which aggregation function was the best one. We display in Table 3 our results. Our highest ROC AUC score is 0.878 and placed us 9^{th} on the public leaderboard.

W-L	Kernel	Agg	Classif.	ROC AUC
No	N-th order	mean	SVM	0.620
Yes	Vertex	voting	SVM	0.835
Yes	Vertex	mean_sign	SVM	0.868
Yes	**Vertex**	**mean**	**SVM**	**0.878**
Yes	Vertex	sign_median	SVM	0.773
Yes	Vertex	median	SVM	0.859

Table 3. ROC AUC on Kaggle public test set

A. Addition Figures

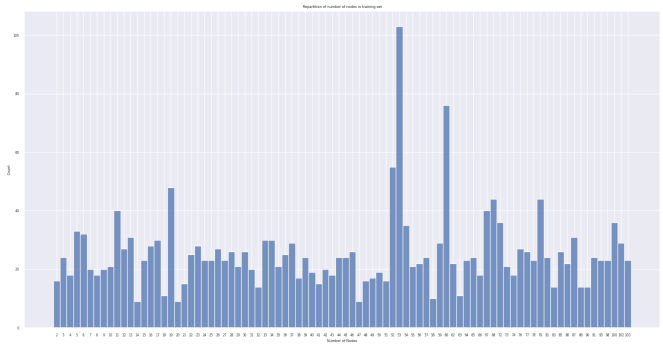

Figure 2. Distribution of Node Labels in the Training Set