MLNS Project Report

Alexis-Raja Brachet, Louis Chirol, Sophia Chirrane and Tanguy Olympie

CentraleSupélec

ABSTRACT

This project aims to benchmark various machine learning and deep learning algorithms for graph classification. Graph classification is a fundamental problem in graph analysis, with numerous applications in various domains, including bioinformatics, social networks, and computer vision. The goal of this project is to evaluate the performance of state-of-the-art ML and deep learning algorithms for graph classification on the COLLAB dataset. The project will involve the implementation of various algorithms, including traditional ML algorithms such as Random Forest and Support Vector Machines, and deep learning algorithms such as Graph Convolutional Networks and Graph Attention Networks.

ACM Reference Format:

Alexis-Raja Brachet, Louis Chirol, Sophia Chirrane and Tanguy Olympie. 2023. MLNS Project Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION / MOTIVATION

Graph classification is a fundamental task in machine learning that has a wide range of applications in various domains such as bioinformatics, chemistry, social network analysis, and recommendation systems. Graph classification involves assigning labels to a set of graphs based on their topological structures and properties. It is a **challenging task** since graphs can have different sizes, shapes, and connectivity patterns, making it difficult to extract meaningful features and patterns.

The use of graph classification algorithms can provide several benefits, including the ability to identify similarities and differences among graphs, cluster similar graphs together, and predict the properties of new graphs based on their structural features. In bioinformatics, graph classification is used to predict protein functions and analyze gene expression data, while in chemistry, it is used to predict the properties of molecules and design new drugs.

In recent years, **Graph Neural Networks** (GNNs) have emerged as a powerful tool for graph classification tasks. GNNs are neural networks that operate on graph-structured data, and they can capture complex dependencies and interactions between graph elements. GNNs have been shown to outperform traditional machine learning algorithms on several graph classification tasks, including molecule classification, social network analysis, and citation network classification.

GNNs have been introduced almost 20 years ago. Based on the success on Convolutional Neural Networks (CNN) on images, which are grid graphs, it was quite natural to implement such techniques on more complex graph structures such as proteins. Such algorithms on graphs are based on Recurrent Neural Networks (RNN) framework called Message passing. Each layer corresponds to the information coming from the neighbors of each node. By putting k layers, we consider messages coming from k hops neighbors. One limitation of GNNs is clear : adding more and more layers will reduce the performances of the model as considering too far neighbors is averaging the messages passing through each node. Recently, newer techniques arose to tackle generalisation and computational issues such as the degree variation between nodes or large scale graph. To do so, GraphSAGE was proposed which consists in a fixed-size sampling neighbors and perform aggregation when updating the node embedding. Then, **Graph Attention Networks** (GAT) were known as being good for dealing with variable sized inputs, focusing on the most relevant parts of the input [8].

For this project, we will compare the performance of a deep learning approach using GNNs with classical machine learning algorithms on the COLLAB dataset.

2 PROBLEM DEFINITION

The COLLAB contains graphs representing scientific collaborations between authors of papers in the arXiv repository. The task is to classify each graph into one of three categories based on the subject area of the papers. The COLLAB graph dataset is a collection of scientific collaboration networks. Each graph represents the collaborations between authors on a paper in a specific research area. The nodes in the graph represent authors, and the edges between nodes represent co-authorship on a paper. The dataset consists of 5,000 graphs with a variable number of nodes and edges, depending on the number of authors and their collaboration patterns. The COLLAB dataset poses several challenges for graph classification, including varying graph sizes, the sparsity of the graphs, and the presence of noise and outliers.

Despite these challenges, the COLLAB dataset is interesting for graph classification tasks because it reflects real-world collaboration patterns in scientific research. The dataset can be used to explore various research questions, such as how collaboration patterns differ across research areas or how to predict the success of a research collaboration based on the co-authorship network. Additionally, the COLLAB dataset provides an opportunity to evaluate the performance of graph classification algorithms in real-world scenarios. The dataset has been used in various studies in the field of graph classification, and the results have led to insights on the effectiveness of different classification algorithms and features for

.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

https://doi.org/10.1145/nnnnnnn.nnnnnnn

this task.

Let's denote $\mathcal{G} = (G_i)_{i \leq 5000}$ the graph collection where each graph is associated to certain label, denoted as $l(G)$.

The goal of this project is to learn a labelling function $\hat{l}$ such as :

$$\hat{l} : G \in \mathcal{G} \mapsto \hat{l}(G) \in \{0; 1; 2\}$$

We aim to maximize the accuracy on the test set of the labelling function :

$$\text{accuracy} = \frac{\#(\hat{l}(G) = l(G))_{G \in \mathcal{G}_{test}}}{\#\mathcal{G}_{test}}$$

3 RELATED WORK

The COLLAB graph dataset has been used extensively in the literature as a benchmark for evaluating graph classification algorithms. The state-of-the-art performance on this dataset is constantly evolving, as new methods are proposed and new features are extracted see Table 1.

Table 1: Accuracy results on the COLLAB dataset using different methods

Method	Accuracy (%)
GCN [4]	81.5
GAT [7]	83.0
AGNN [10]	83.0
SVM [11]	73.9
RWR [9]	75.1
GraRep [4]	78.9

Our project is at the same time replicating, different and complementary to these works. Indeed, we will base our architectures on the findings of both ML based and GNN based architecture. Mixing the two approaches is complementary to these related works, taking benefit from the best of each world. In addition, distance based approach has been explored, which is a different approach from the related works.

4 METHODOLOGY

To compare the performance of GNNs and classical machine learning algorithms, we would follow the following methodology:

4.1 Data collection process

The COLLAB dataset is convenient to work with. Indeed, a simple line of code downloads the entire dataset from the TUDataset collection [6].

General Data Preprocessing. We preprocess the COLLAB dataset by converting it into a format that is compatible with both GNN and classical machine learning algorithms.

- For Machine Learning we would work with the Networkx Python library
- For Graph Neural Networks we would use the DGL (Deep Graph Library) and the Pytorch Python libraries

Some additional processings have been performed for each methods such as feature extraction for ML-based

4.2 Data exploration

It is always interesting to spend some time on data exploration to better understand our data. When it comes to graph classification, useful information can be the number of nodes, edges, the average node degree, and the class distribution for all the graphs and for each attribute. We summarize this information in figure 1. This figure teaches us that a lot of graphs have few nodes and edges, but a significant number of them has a lot more edges, leaving a whole in the distribution of edges. The average node degree is 74 which is quite high, with some graphs having a much higher average degree. It is interesting to note that for a majority of graphs, the diameter is only two. Finally, the classes are quite imbalanced, so that it is interesting to examine as well the former information, combined with a label information, which is done in figure 2. The number of nodes boxplot is not very informative as all the boxes are severely overlapping, but the number of edges one shows that graphs in class one have significantly less edges than the other classes, and the graphs of class 2 can have more. The boxes in terms of average node degree are also well distinguished. Graphs of class one never have a diameter of 1, while only a few graphs of class 0 have. As a whole, this short analysis highlights that basic features are quite informative about the classes, yet overlapping too much to draw arbitrary boundaries between the classes. Consequently, feeding such features and more advanced ones into classifiers could well produce a good classification, in order to exploit this between-class variance. Another noteworthy element is the diameter of the graphs being less than 2, which means that the graphs is highly connected.

4.3 Machine Learning approach

4.3.1 ***Features extraction.*** In order to use classical ML techniques, we are going to extract features from the graph such as degree centrality, betweenness centrality, and clustering coefficient. We would also perform Graph Kernel techniques as seen during the lectures in order to perform graph similarity in an higher dimension Hilbert space.

4.3.2 ***Distance based method.*** Another approach we could perform is to use distance based classification method such as adapted KNN or K-means. These classic algorithms can perform a classification task by comparing the samples with respect to a define distance (often euclidean distance on the feature spaces). We could therefore think of defining a handcrafted distance based on the graphs of the dataset properties.

Finding an appropriate distance between graphs is not a trivial question. It is often solved using expert knowledge i.e. defining a specific distance for a specific case of application using the information represented by the graph [2]. This method does not necessarily take into account the whole graph information but sometimes some specific properties : the number of cliques for communities search, the number of leaves for trees, the node degrees … Another approach used in [5] could be to define a distance based an optimal transport framework to previous further analysis define a distance

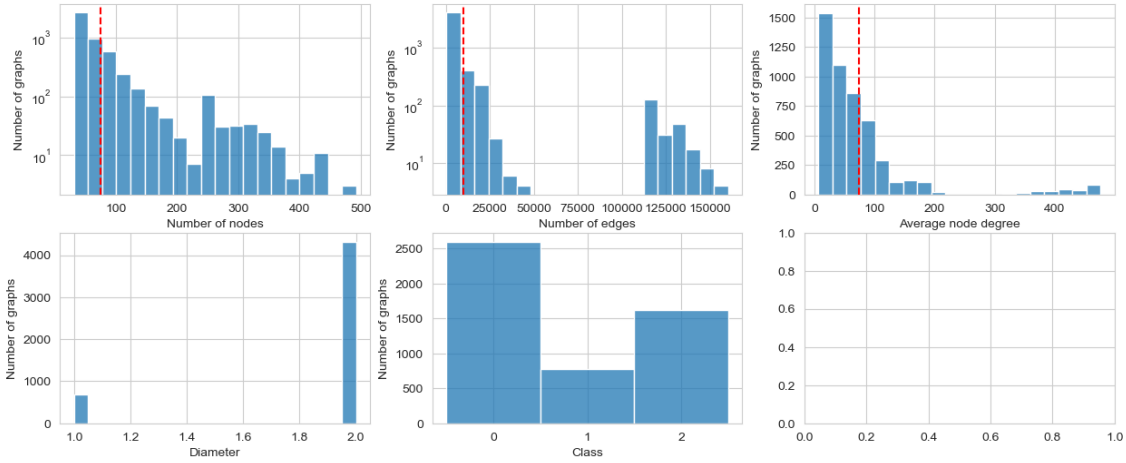

Figure 1: Number of nodes and edges distribution and label distribution of the COLLAB dataset. Red dashed lines indicate the average.

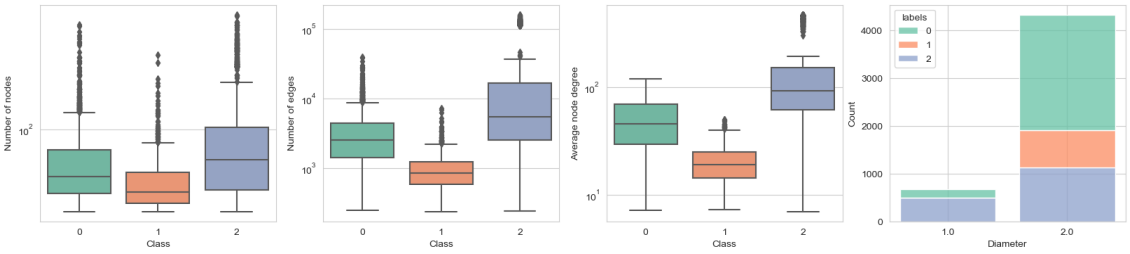

Figure 2: Label-wise distributions for the number of nodes, edges, the average node degree and the diameter of the COLLAB dataset.

based on a graph alignment problem.

We studied different classic distances to compare graph :

- Random Walks Distance (RWD) (Equation 1): RWD is a distance measure based on random walks on graphs. The RWD between two graphs is defined as the sum of the similarities of their random walk distributions. Random walk distributions can be computed using graph Laplacians, and the similarity between distributions can be measured using various metrics, such as cosine similarity or KL divergence.
- Shortest path distance (SPD) (Equation 2): The shortest path distance measures the distance between two nodes in a graph as the length of the shortest path connecting them. This distance can be used as a measure of similarity between graphs, by computing the average shortest path distance between nodes in each graph.
- Diameter distance (Equation 3) : The diameter of a graph is the maximum distance between any two nodes in the graph. In other words, it is the length of the longest shortest path in the graph. It can be computed by finding the shortest path between all pairs of nodes in the graph and then taking the maximum over all pairs.

$$RWD(G_1, G_2) = \sum_i sim(p_i, q_i) \tag{1}$$

where G_1 and G_2 are the two graphs being compared, n is the number of nodes in the graphs, p_i and q_i are the random walk distributions for node i in each graph, and $\text{sim}(p_i, q_i)$ is a similarity measure between the two distributions, such as cosine similarity or KL divergence.

The shortest path distance between two nodes in a graph can be defined as the length of the shortest path connecting them, denoted by $\delta(u, v)$ for nodes u and v. The average shortest path distance between all pairs of nodes in a graph G can be computed as:

$$SPD(G) = \frac{1}{n(n-1)} \sum_{u \in V} \sum_{v \in V, v \neq u} \delta(u, v) \tag{2}$$

where V is the set of nodes in the graph, n is the number of nodes, and $\delta(u, v)$ is the length of the shortest path between nodes u and v.

The formula for computing the diameter of an undirected graph G with n nodes is:

$$diameter(G) = max_{u,v \in V} d(u, v) \tag{3}$$

where V is the set of nodes in G, and $d(u, v)$ is the length of the shortest path between nodes u and v in G.

The K-nearest neighbors (KNN) algorithm is a simple and effective method for classification. When using a graph distance for classification, the KNN algorithm works as follows:

- For each training graph, compute its distance to the test graph using the chosen graph distance measure.
- Select the K training graphs with the smallest distances to the test graph.
- Assign the class label of the test graph as the majority class label among the K nearest neighbors.

The value of K is a hyperparameter that can be tuned to optimize the classification performance. Additionally, the choice of the graph distance measure can also have a significant impact on the classification performance.

4.4 Graph Neural Network approach

For the GNN implementation, we use the DGL library to build our networks. The data is processed in order to be in a dataloader with batches. The goal here is to implement a classifier fully based on neural network, from the graph representation to the classification.

In order to build the representation, we implement :

- GraphSAGE architecture
- GAT architecture

to produce the nodes embeddings. Then we average them and feed the resulting embedding into a Multi layer perceptron.

GraphSAGE has been introduced in 2017 [3]. At each step k, the node embedding of v denoted as h_v^k is computed as it follows :

- $\mathcal{N}(v)$ neighboors are sampled from all the neighboors of v. Then, an aggregation function is applied on this subset giving
$$h_{\mathcal{N}(v)}^k = \text{AGGREGATE}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$
- then the previous embedding h_v^{k-1} and $h_{\mathcal{N}(v)}^k$ are fed into a fully connected layer in order to compute h_v^k
$$h_v^k = \sigma(W^k \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k))$$

This approach allows GraphSAGE to efficiently scale to large graphs while maintaining the ability to capture complex structural patterns in the data.

GAT has been introduced in 2018 [8]. It takes advantage from the attention mechanism developed in Natural Language Processing introduced in 2015 [1]. At each time step, the embedding h_i' is the concatenation of each attention head. An attention head computes the weighted sum of each neighbor of i.

For an attention head, we have :

$$h_i' = \sigma(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} h_j)$$

where

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}h_i || \mathbf{W}h_j]))}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}h_i || \mathbf{W}h_k]))}$$

where $\mathbf{a}$ and $\mathbf{W}$ are learnable parameters. Finally, each head is concatenated in order to produce the updated embedding of the node i :

$$h_i' = \text{CONCAT}_k(\sigma(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k W^k h_j))$$

GAT architecture is nowadays the state-of-the-art for most GNN problems due to the use of attention mechanism which is so powerful.

One may know that a deep network in GNN framework is dangerous as it will average node representations, which is called oversmoothing. Indeed it is annoying for node classification or link prediction. However, in our case, we want to capture graph level properties so computing a deep network would be beneficial. This method hasn't been explored.

The dataset has been splitted in 3 parts :

- a train set on which the model will be trained representing 64% of the dataset
- a validation set to control overfitting representing 16% of the dataset
- a test set on which the performance is evaluated representing 20 % of the dataset

All the datalaoders have been weighted in order to tackle the imbalance issue of the dataset.

The final hyperparameters we chose for each architecture are the following :

for GraphSAGE. , a 2 SAGE layer embedding model with 128 neurons in each and batch-norm layers in between has been chosen and a learning rate of 10^{-3}. Each SAGE layer use the *pooling* aggregation function.

for GAT. , a 2 GAT layer with 4 attention heads, 128 neurons in each and a learning rate of 10^{-3}.

For both architecture, the input embedding for every node was at first a vector with the graph level features defined in section 5.1.

They are then concatenated with the output of the GNN in the 3 layer MLP for classification. The number of neurons in each layer is divided by 2 from one layer to another. A softmax is then applied in order to compute the probability over the 3 possible classes.

At the end of the project, we started to consider as input the embeddings given by the Node2Vec algorithm. Sadly, we didn't have time to finish our computation due to time limit, non optimized function and no GPUs remaining. The first the obtained results were much higher and usable with this initialization so we would have followed this path.

The Node2Vec algorithm is based on the NLP domain. Multiple random walks are performed on each node of the graph. Then, an embedding is computed such as neighbors, ie nodes occurring in the same walk, have similar embeddings. As a consequence, it can capture the structure of the graph and is much more expressive than a random initialization.

Results and discussion are reported in section 5.3.

4.5 Hybrid approach

Finally, we could design a new method that includes both machine learning and deep learning for a classification task problem. The deep learning approach would be used to automatize the process of representation learning in order to select a relevant graph representation. Then, this learned representation could be used as the input of a machine learning approach. The GNN would be trained with a specific loss based on the performance of the ML classifier.

We also thought about trying to learn graph representations with a GNN before feeding a ML classifier through self-supervised learning, which is a hot topic in the modern literature and would have been fun to experiment with in this project. But our lack of knowledge and the diverse design choices to make (self-supervised task definition, dataset generation, amount of data) led us to abandon this approach.

5 EVALUATION

5.1 ML method

The exact list of features used is :

- diameter
- clustering coefficient
- mean number of edges
- total number of edges
- number of nodes
- mean of the nodes average degree connectivity
- standard deviations of the nodes average degree connectivity
- average shortest path length

3 classical machine learning classifiers were trained : a logistic regression classifier and a Random Forests algorithm and a Support Vector Machine algorithm.

Table 2: Balanced Accuracy in (%) results on the COLLAB dataset using different Machine Learning algorithms

Classifier	Accuracy
Logistic regression	69
Random Forests	77
SVC	68

The best performing of the 3 is the Random forest algorithm with an accuracy of .77. We can extract the features importances :

The importance of the feature seems to be rather well distributed, as a most have a non negligible impact on the result. The most important feature is the average number of edges while the least important are the number of nodes, diameter and density.
These simple algorithms based on feature extraction can be used as a simple baseline for further models. Even if the classical models take very little time to be trained, the feature extraction in itself is quite costly. The results are slightly below those of the

Table 3: Importance of the graph features for the Random Forests decisions

Feature	Importance
number of edges	.14
number of nodes	.056
average number of edges	.211
clustering coeficient	.120
density	.099
diameter	.001
maximum number of connected components	.058
average shortest path lenght	.106
mean of the nodes average degree connectivity	.101
standard deviation of the nodes average degree connectivity	.101

5.2 Distance based method

As the sizes of the graph of the dataset differ and can be quite big the distance choice has a strong influence on the computation time. The 3 studied distances were chosen due to they rather fast computation.

To evaluate the distance based approach we decided to only study a sub part of the whole dataset due to computational constraints. We so created sub dataset taking x, the number of sample per class.

Table 4: Balanced Accuracy results in (%) on the COLLAB dataset using different distance based methods

Metric	k NN	50 graphs/class	100 graphs/class
RWD	k=3	26.8	29.8
SPD	k=3	33.3	33.3
Diameter	k=3	**40.7**	**45.10**
RWD	k=10	36.1	30.68
SPD	k=10	33.3	33.33
Diameter	k=10	**37.96**	**35.27**

The following table (table 4 summarizes the obtained results. We can see with those results that the choice of the distance is crucial and lead to significantly different results. The distance that compared the diameter of the graphs performs better than the other for every set up. However this distance seems to have a maximum accuracy with respect to the number of neighbors used for classification when we add some data to the training. For instance, when the number of neighbor is equal to 3 we have an increase of the balanced accuracy dealing from 50 to 100 graphs per class. However, when increasing to reach 200 graphs/class the accuracy decreases to 42.16 (Figure 3). The same phenomenon is observed with 10 neighbors.

Furthermore, this distance is the only one for which the performances are not increased when the number of neighbor increases. The SPD distance does not perform better than a random affectation. We can take a closer look of what is happening when using this distance.

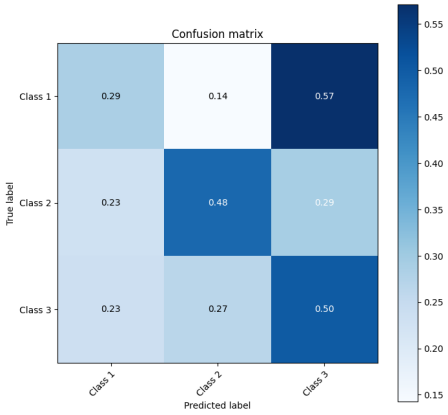

Figure 3: Confusion matrix when using the diameter distance for a dataset containing 200 graphs per class and 3 neighbors

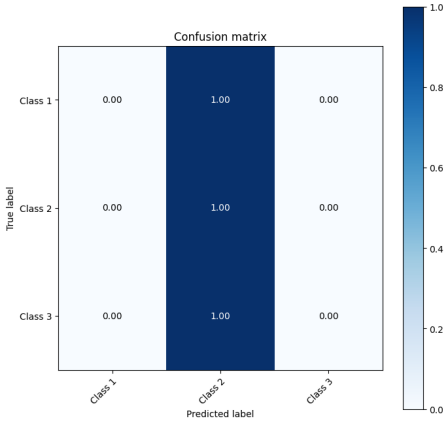

Figure 4: Result using the SPD distance

On figure 2 we then see that every graph is classified as belonging to the second class.

Overall, even this approach is highly interpretable it does not lead to very good results. Moreover, the choice of the distance is crucial both in term of performances but also in term of computation complexity.

The diameter in the Collab dataset seems to play a key role in order to classify the graphs.

5.3 GNN method

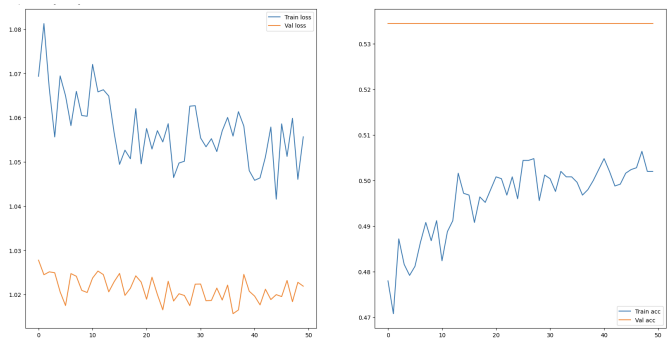

Figure 5: Learning curves of the GraphSAGE architecture over 50 epochs showing the prediction issue

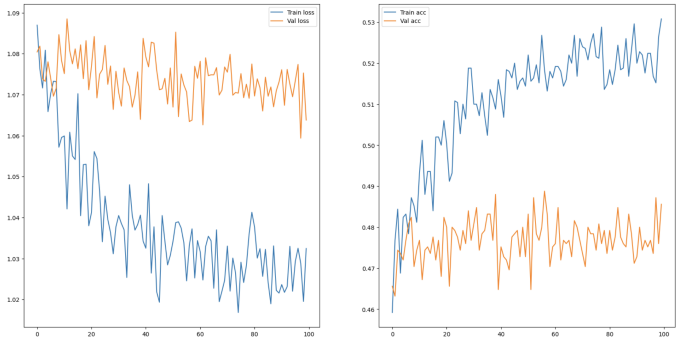

Figure 6: Learning curves of the GAT architecture for 1 run showing the prediction issue and a different local minimum

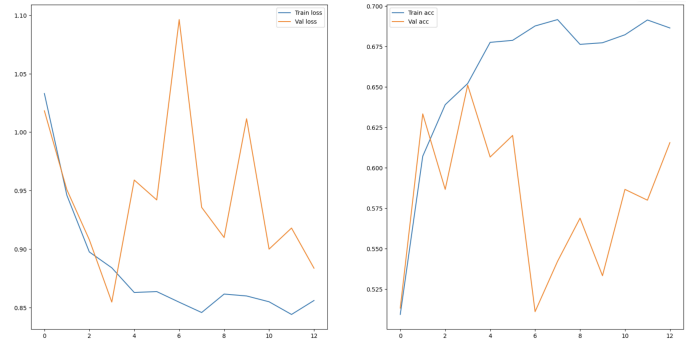

Figure 7: Learning curves of the SAGE architecture for 1 run showing with Node2Vec initialization along 12 epochs showing promising performances

Despite playing on the hyperparameters of each architecture, we found that the validation accuracy would most of the time get stuck in a local minimimum, corresponding to predict the same label for every graph. The label chosen is the most represented in the dataset. Implementing a learning scheduler didn't solve the problem. We coult think that the model is stuck into a large local minimum.

Table 5: Best accuracy in (%) results on the COLLAB dataset using different GNN architecture and initialization

Method	features initialization	Train acc	Val acc	Test acc
GraphSAGE	graph	67	53	53
GAT	graph	67	53	53
GraphSAGE	Node2Vec	**69**	**62**	...
GAT	Node2Vec	65	**62**	...

However, setting from small (10^{-6}) to large (10^{-1}) learning rates didn't change anything.

As the accuracy on the train set is increasing, the model is learning something.

The figure 5 illustrates quite well these words. The accuracy on the validation is constant along the 50 training epochs while the one of the training set is increasing.

At the end of our work, we started to implement the Node2Vec embedding method in order to initialize the embeddings fed into the GNNs. The predictions were no longer constant and the accuracy was significantly higher.

GraphSAGE with Node2Vec initialization get the best results during training.

This way is promising and we should explore it in depth. *The parameters of the code were the ones producing the scores of the Node2Vec initialization in table 5.*

5.4 Hybrid method

The hybrid method consists in training a GNN to learn relevant graph features, and then to use these features with a different type of classifier, for instance a Machine Learning classifier. The choice of ML classifier is broad: we could think of a Random Forest, a Gradient Boosting model like xgboost, an SVM model. It would also be easy to concatenate more classical graph features to the GNN features, like the ones used in the fully ML part above.

. We were unfortunately unable to test this approach, even though it was implemented. As we wanted to use it on top of our best GNN, it took some time before getting an interesting model, and we did not get enough time to exploit it. The original idea was to compare the results of a Random Forest, an xgboost and an SVC classifiers on top of the GNN, on the basis of their F1-score and their confusion matrix.

6 CONCLUSION

In this work, we performed an exhaustive study on the graph classification task, over the COLLAB dataset.

We first experimented with classical Network Science, with distance based methods suited for this task. They proved computationally expensive, and inefficient (45% accuracy at most) as the results were worse than a constant prediction based on the most populated class ($\simeq$ 53% of class 0).

Thus, we moved toward Machine Learning methods, relying on a supervised learning of the labels based on classical graph features. This attempts gave better results, with a 77% accuracy with a Random Forest classifier.

Finally, we tried to explore state of the art method through Graph Neural Networks. This approach required a lot a tuning and tricks to finally yield interesting results, despite the low number of classes in the dataset, only three. Yet, the method is once again slow and could not be extended to other approaches, such as a hybrid GNN/ML classifier, or a self-supervised training of the GNN for the learning of feature extraction.

It appeared to us that despite looking similar to other classification tasks, graph classification may prove tricky, as it requires to exploit a lot of network science knowledge, and a lot of efforts to extract relevant information from a graph without prior node or edge features in order to classify it.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] C. Colijn and G. Plazzotta. A Metric on Phylogenetic Tree Shapes. *Systematic Biology*, 67(1):113–126, 05 2017.

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[4] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *Neural Networks*, 61:147–160, 2017.

[5] Hermina Petric Maretic, Mireille El Gheche, Matthias Minder, Giovanni Chierchia, and Pascal Frossard. Wasserstein-based graph alignment. *CoRR*, abs/2003.06048, 2020.

[6] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *CoRR*, abs/2007.08663, 2020.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[9] Ulrike von Luxburg. A comparison of spectral clustering algorithms. In *Advances in Neural Information Processing Systems*, pages 857–864, 2007.

[10] Yufei Zhao, Leman Akoglu, and Hanghang Tong. Attributed graph neural networks for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4054–4061, 2019.

[11] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with local and global consistency. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 2*, volume 2, pages 321–328. IEEE, 2004.