ALTEGRAD: CHALLENGE

Tom Salembien, Ohm_Tom
Master 2 Data science IPP
ALTEGRAD
`salembien.tom@gmail.com`
GitHub Tom Salembien

Abstract

This document provides a basic comparison betIen several algorithm of classification. I run several experiments with 5 classifiers, to find the classifier with better performance for the covertype dataset.

1 Introduction

The goal of this project is to study and apply machine learning/artificial intelligence techniques to a classification problem from the field of bio-informatics. In this challenge, you are given the sequence of 6, 111 proteins along with the graph representation of their structure. The nodes of these graphs represent the amino acids and two nodes are connected by an edge based on the Euclidean distance between these residues, their order in the sequence, and the different chemical interactions between them. The goal of this challenge is to use the sequence and structure of those proteins and classify them into 18 different classes, each representing a characteristic of the location where the protein performs its function obtained from the Cellular Component ontology.

2 Our Dataset

G (train/test)	Nodes	Edges	Ftrs(node/edge)
4888/1223	1572264	15213222	86/5

Table 1: Dimensions and lenght of dataset

3 Litterature

Protein graph classification is a challenging task in bioinformatics that involves identifying the function or structure of a protein based on its graph representation. Proteins are complex biomolecules that perform a wide range of functions in living organisms, and their structure and function are closely related. Protein graph classification can be used to predict the function of a protein, understand its interactions with other molecules, or analyze its evolutionary history.

The literature on protein graph classification is vast and covers a wide range of methods and approaches. One of the most popular approaches is to use machine learning algorithms, such as decision trees, Random Forest, neural networks, and support vector machines (SVMs). These algorithms can be trained on a dataset of protein graphs and their known functions to predict the function of new protein graphs. Another popular approach is to use graph kernels, which compare the topological similarity of protein graphs. There are also methods based on graph convolutional networks (GCNs) which have been used effectively in this task.

In recent years, deep learning methods have become increasingly popular for protein graph classification. These methods, such as graph convolutional networks (GCNs), have been shown to be effective in capturing the complex relationships between proteins and their functions. Additionally, there has been a growing interest in using unsupervised learning methods, such as graph autoencoders and variational autoencoders, to learn representations of protein graphs that can be used for downstream tasks, such as classification and clustering.

The main task of this challenge was to use as many features as possible to get the best classification. Indeed, we had access to nodes and edges features with also the sequence of amino acid of each protein. The state of the art algorithm on these type of task mainly use the Protein Data Bank (PDB) file format is a textual file format describing the three-dimensional structures of molecules held in the Protein Data Bank like.

4 HGP-SL Architecture

The architecture of the Hierarchical Graph Pooling with Structural Learning model is pretty recent, it learns a low-dimensional representation for the entire graph. The advantage with this method is the capability of the model to preserve the essential graph structure information, which will facilitate the message passing procedure. Indeed the idea is to use node embedding, the embedding vectors are learned by an unsupervised learning method and we build the adjacency matrix with the relations between the nodes of the graph.(**?**)

Then, we have the hierarchical pooling, which is performed on the adjacency matrix to obtain subgraphs of reduced size. This pooling is performed using a hierarchical clustering algorithm to group similar nodes to learn a refined graph structure that can best preserve the essential topological information.

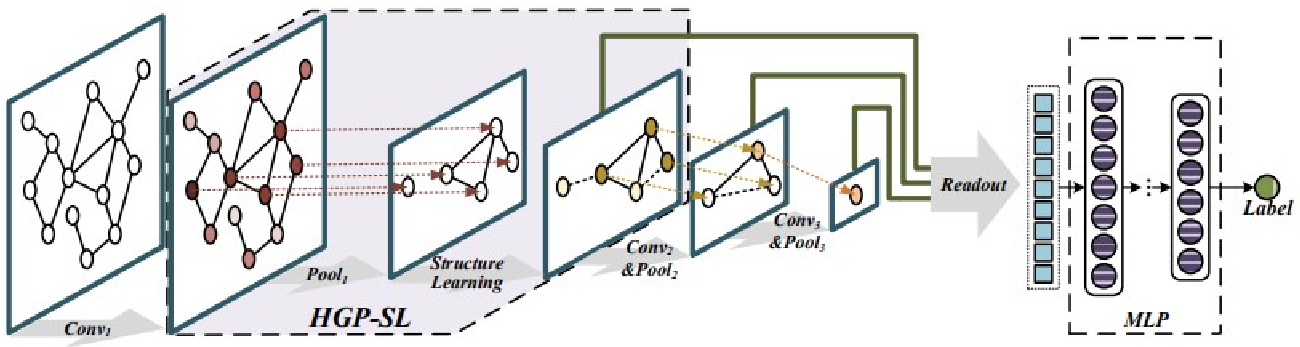


Figure 1: HGP-SL

Finally, we extract the graph structure which is learned from the subgraphs obtained in the previous step, using a supervised classification method. And we classify the new graphs with 2 fully connected layers. The figure 3, represent the global architecture of the model.

4.1 Graph Pooling

The article propose a new way of down-sampling on graph data. Indeed, the proposed operation is non-parametric and utilizes both node features and graph structure information. The key element of the proposed operation is the use of a criterion to guide the node selection procedure, called the "node information score". This score evaluates the information contained in each node by considering the node's neighborhood. The score is defined as the Manhattan distance between the node representation and the one constructed from its neighbors.

$$p = ||(\mathbf{I}_i^k - (\mathbf{D}_i^k)^{-1}\mathbf{A}_i^k)\mathbf{H}_i^k||_1$$

With D the diagonal degree matrix, A the adjacency matrix and H the node representation matrix (I is the identity), i is the ith node and k is the kth neighbor.

If a node's representation can be reconstructed by its neighborhood representations, it can be deleted in the pooled graph with almost no information loss. Then we preserve the nodes with relative larger node information score in the construction of the pooled graph, as they can provide more information. We will order the nodes regarding their score infromation and then extract the top k nodes that will represent the node features and structure for the next layer.

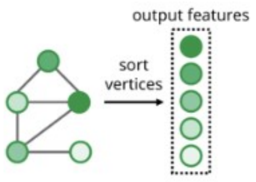


Figure 2: Top k score information nodes

4.2 Structure Learning

The problem is how can we classify the refined graph structures of the the pooling part correctly ? Indeed, as the pooling operation down-sample the graph data, the graphs might result in highly related nodes being disconnected, thus we loose pretty much structure informations and it can hinders the message passing procedure. In order to obtain optimal graph structures for the learning task in GNN, the article propose a new method based on (Li et al. 2018) that learns sparse graph structure through sparse attention mechanism (Martins and Astudillo 2016). We get the hidden representation ($\mathbf{H}_i^k$) and the structure information ($\mathbf{A}_i^k$) from the the kth pooled subgraph of the ith graph written $\mathbf{G}_i^k$. To build the pairwise relationship between each pair of nodes (p, q), we use a similarity score written,

$$\mathbf{E}_i^k(p,q) = \sigma(\mathbf{a}[\mathbf{H}_i^k(p,:)||\mathbf{H}_i^k(q,:)]^T + \lambda * \mathbf{A}_i^k(p,q)$$

With $\mathbf{A}_i^k(p,q)$=0 if the nodes p and q are not directly connected, then $\mathbf{A}_i^k$ encodes the induced subgraph structure information.

Finally, we need to normalize the scores as we want to compare them after. We could use a softmax function however, this introduce noise as a softmax function always return non zero values. That is why we use a sparsemax function that return the Euclidean projection of input onto the probability simplex,

5 Results

Epochs	lr	nhid	pool/drop-ratio	# Conv/Pool/Linear
111	0.001	256	0.8/0.2	3/2/3

Table 2: Best HGP$_S Lmodelparameters$

The training of this model takes a lot of time, indeed the calculations of the similarities between each pair of nodes when the model is learning the structure with huge graphs is time consuming. To reduce this time we can reduce the pooling ratio in order the reduce the number of nodes and edges and the new graphs, however the accuracy will also drop. With this model I could only reach about 1.7 in the final loss, but the structure is really interesting.

6 Other models Tested

6.1 CountVectorizer and Logistic Regression

I used count vectorizer to convert the collection of sequences into a numerical feature matrix, where each row represents a sequence and each column represents a letter from the vocabulary (for ngram=(1, 1)). The value in each cell of the matrix is the number of times the amino acid in the corresponding column appears in the protein represented by the corresponding row. For the experience, I used ngram=(3,3) to consider the groups of three leters instead of only one letter.

Finally, I used the Logistic Regression to classify the protein. The method is a supervised learning algorithm that tries to model the relationship between the input features (in my case, the group of 3 letters counts from the count vectorized sequence) and the class groundtruth (one of the 18 classes). The logistic regression algorithm will output a probability for each sequence.

I also tried to classify the sequences with a tree based model. I tried Random Forest because the main advantage of this model over the single decision tree is that it helps to reduce overfitting as it creates multiple decision trees during training, each one is trained on a random subset of the data. Thus, this reduces the chance of a decision tree overfitting to the training data. Additionally, the randomness in the feature selection process during training also helps to reduce overfitting. Even if it takes the average or majority vote of the predictions made by each tree to make the final prediction, this method was less accurate than a simple logistic regression.

6.2 Fusion model

In this method, I wanted to used both informations from the sequence of the proteins and there structures with the graphs and the features. To do so, I used a simple GNN to extract features from the structures and a simple MLP to extract informations from the embedding (TFIDF or Count Vectorizer) of the sequences. Then on concatenate the features extracted from both model and I passed them into 2 fully connected layers.

6.3 CNN and LSTM

Based on the same idea as before I tried to build a fusion of CNN and LSTM models to extract featuers from the sequences of the proteins and only this file. I created a vocabulary dictionnary and I embedded the sequence based on nn.Embedding function which take as inputs a tensor of integers, where each integer represents an element in the input data. The output of this layer is a tensor of float values, where each value represents an embedding for the corresponding input element. Finally I concatenate the output of CNN and 1D max pooling layers with the output of the LSTM and classify them with two fully connected layers.

6.4 Other methods

I hadn't time to entirely finish my tests, but I tried to extract features from the structure of the graphs with a VGAE and I would have liked to classify the extracted featurees with a SVM or a logistic regression.

I also tried to extract features from the sequence with a pretrained BERT model from RostLab in hugging Face called ProtBert which in my opinion could have potentially great results but I didn't finished my tests for now.

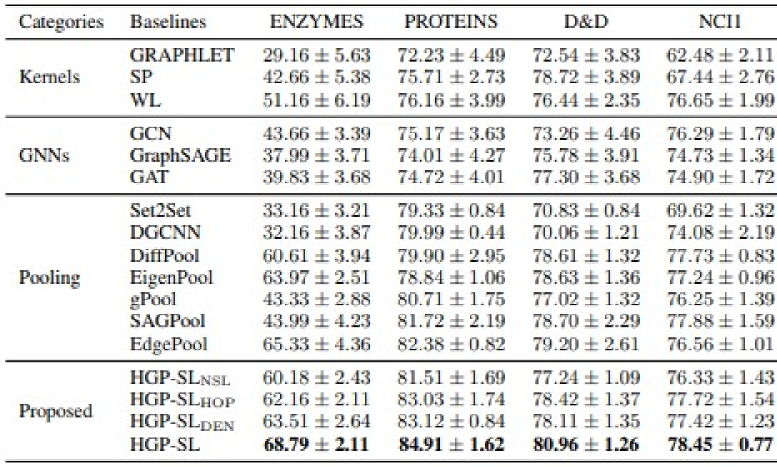
An other interesting method to try is node2vec embedding of the structure and make a classification with a SVM model on the embeddings.

7 Results and Conclusion

Classifier	Loss Test Set	Time run
Emb + LogisticRegssion	1.2	Fast
Emb + MLP	1.38	Pretty fast
CNN-BiLSTM	1.6	Slow
HGP-SL	1.8	Really Slow

Table 3: Loss and time of execution of main algorithms tested

In this challenge, I tried a lot of possible method of classification. I was astonished by the fact that the simplest methods had the best accuracies. Indeed, the model HGP-SL had great results on others dataset, for example,



Categories	Baselines	ENZYMES	PROTEINS	D&D	NCI1
Kernels	GRAPHLET	29.16 ± 5.63	72.23 ± 4.49	72.54 ± 3.83	62.48 ± 2.11
	SP	42.66 ± 5.38	75.71 ± 2.73	78.72 ± 3.89	67.44 ± 2.76
	WL	51.16 ± 6.19	76.16 ± 3.99	76.44 ± 2.35	76.65 ± 1.99
GNNs	GCN	43.66 ± 3.39	75.17 ± 3.63	73.26 ± 4.46	76.29 ± 1.79
	GraphSAGE	37.99 ± 3.71	74.01 ± 4.27	75.78 ± 3.91	74.73 ± 1.34
	GAT	39.83 ± 3.68	74.72 ± 4.01	77.30 ± 3.68	74.90 ± 1.72
Pooling	Set2Set	33.16 ± 3.21	79.33 ± 0.84	70.83 ± 0.84	69.62 ± 1.32
	DGCNN	32.16 ± 3.87	79.99 ± 0.44	70.06 ± 1.21	74.08 ± 2.19
	DiffPool	60.61 ± 3.94	79.90 ± 2.95	78.61 ± 1.32	77.73 ± 0.83
	EigenPool	63.97 ± 2.51	78.84 ± 1.06	78.63 ± 1.36	77.24 ± 0.96
	gPool	43.33 ± 2.88	80.71 ± 1.75	77.02 ± 1.32	76.25 ± 1.39
	SAGPool	43.99 ± 4.23	81.72 ± 2.19	78.70 ± 2.29	77.88 ± 1.59
	EdgePool	65.33 ± 4.36	82.38 ± 0.82	79.20 ± 2.61	76.56 ± 1.01
Proposed	HGP-SL$_{\mathrm{NSL}}$	60.18 ± 2.43	81.51 ± 1.69	77.24 ± 1.09	76.33 ± 1.43
	HGP-SL$_{\mathrm{HOP}}$	62.16 ± 2.11	83.03 ± 1.74	78.42 ± 1.37	77.72 ± 1.54
	HGP-SL$_{\mathrm{DEN}}$	63.51 ± 2.64	83.12 ± 0.84	78.11 ± 1.35	77.42 ± 1.23
	HGP-SL	**68.79 ± 2.11**	**84.91 ± 1.62**	**80.96 ± 1.26**	**78.45 ± 0.77**

Figure 3: HGP-SL: Graph classification in terms of accuracy with standard deviation (in percentage).

However, the results obtained for me were not that great. The problem with this architecture is that it is time consuming, the training takes a lot of time and in our case especially we had large graphs. Thus, it might not be the optimal solution to use in out case, moreover it should have been great to combine a similar model with the sequences such as for the fusion model in order to capture much more informations.

8 References

(I had an issue with the official references latex format so I wrote them here)

[Kipf and Welling 2017] Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. ICLR.

[Lee, Lee, and Kang 2019] Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. In ICML, 3734–3743.

[Li et al. 2018] Li, R.; Wang, S.; Zhu, F.; and Huang, J. 2018. Adaptive graph convolutional neural networks. In AAAI.

[Ma et al. 2019] Ma, Y.; Wang, S.; Aggarwal, C. C.; and Tang, J. 2019. Graph convolutional networks with eigenpooling. In SIGKDD.

[Martins and Astudillo 2016] Martins, A., and Astudillo, R. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification. In ICML, 1614–1623.

[Hongyang Gao2 February 2021] Graph Neural Networks: A Feature and Structure Learning Approach

[Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, Can Wang, 2019] Hierarchical Graph Pooling with Structure Learning