Homework 4 - Graph Learning on Biological Data

Olga Sorokoletova: 1937430

December 23, 2021

I. Introduction

The overall goal of the project was to compare two machine-learning models: the one that relies on **Node2Vec** embeddings as features vectors (Approach 1) and a **graph neural network** (Approach 2) given the task of node classification based on biological data from the Biogrid dataset.

An Approach 1 consists of 2-step model training: first, **Node2Vec** model, that learns node embeddings, and then, the classifier, which takes an input, featured with learnt embeddings, is trained. Both architecture of the Node2Vec model and the choice of classifier matter. As for the latter, two classifiers were employed: **SVM** and **MLP**.

Meanwhile, Approach 2 is presented by the Graph Convolution Network (**GCN**).

II. Data Inspection and Preparation

i. Datasets

First, both PPI and GDA datasets are loaded, preliminary basic modifications on them are preformed and reduced version of PPI dataset, corresponding to only first $10K$ **interactions**, is created. The reported statistics correspond to this reduced version (PPI-reduced), however, execution of the pipeline finishes successfully also for the full version although with a larger execution time. From the PPI-reduced a list of the (4263) unique genes of interest is obtained. Then iterating over filtered GDA dataframe and counting number of associated genes per disease, the **disease with the largest number of associated genes** is detected (Table 1) and only gene names, associated with this disease, are kept to assign labels. About 0.62% of the genes of interest are associated with the detected disease, which means that dataset, corresponding to the chosen disease, **is balanced**.

Id	C0027651
Name	Neoplasms
Semantic Type	Neoplastic Process
Description	Give rise to tumor cells
Associated Genes N	2621

Table 1: *Disease of interest.*

ii. Graph

ii.1 Networkx Graph

Data now is prepared to construct the Networkx graph. It was done within 3 steps:

1. Create the dictionary mapping gene **names to indices**;
2. Use the dictionary to vectorize gene names and PPI mapping and create the **data object**, describing a homogeneous graph with gene indices, binary 0/1 node-level ground-truth labels and index pairs graph connectivity;
3. Covert the data object into Networkx **graph as undirected and with self-loops removal**, since edge directions and self-loops are out of the scope of the considered graph application.

The resultant full graph can be then plotted with Networkx (see Figure 1).

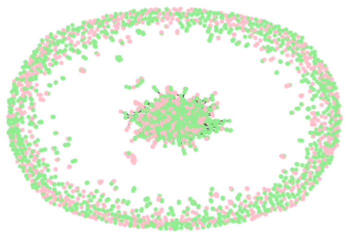

Figure 1: *The global (full) graph plotted using Networkx. Colors correspond to different classes.*

This representation has several important drawbacks. First, visualization of the graph with 4263 nodes and 4310 edges takes around 1.5 minutes and would take even longer for the full PPI dataset, meanwhile for the effective model training having of a more data is helpful. And then, the representation **does not allow visual inspection** of the graph, since the edges are not visible and Networkx drawing utilities do not provide interactive exploration. Typical solution for such a problem is to work on subgraphs, but then the question of how to extract the meaningful subgraphs arises.

ii.2 Cytoscape

To approach the problem of meaningful subgraphs extraction and study the graph properties, Cytoscape software platform for visualizing large networks has been chosen amongst alternatives as it is a suggested utility for various kinds of problem domains, including bioinformatics.

This platform supports **interactive exploration** of the network data as well as **filtering and analysing** tools for the computation of the numerical measures/statistic (e.g. **centralities**) with a subsequent plotting of the output results (e.g. histogram over betweenness centrality), which is quite convenient.

In the Figure 2 one of the possible choices for the full graph representation with Cytoscape is demonstrated.

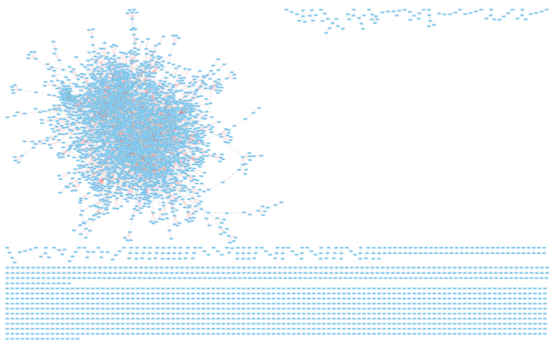

Figure 2: *The full graph plotted using Cytoscape.*

In the Figure 3 a subgraph with only nodes of degree, higher than 10, is selected as an example of Cytoscape subgraph extraction capabilities.

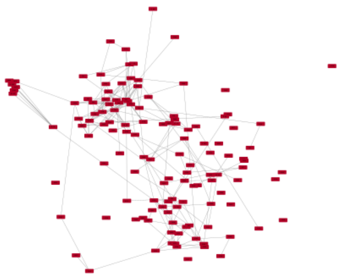

Figure 3: *The subgraph of nodes with high degree centrality plotted using Cytoscape.*

ii.3 Subgraph Extraction

The list of nodes or/and edges, corresponding to any subgraph of interest, found with Cytoscape, can be then saved in a `csv` and exported for the integration with a problem pipeline. The special utilities that create and plot a Networkx subgraph are implemented.

An example of a Networkx subgraph, corresponding to the Cytoscape subgraph in the Figure 3, as undirected and after self-loops and isolated nodes removal is shown in the Figure 4.

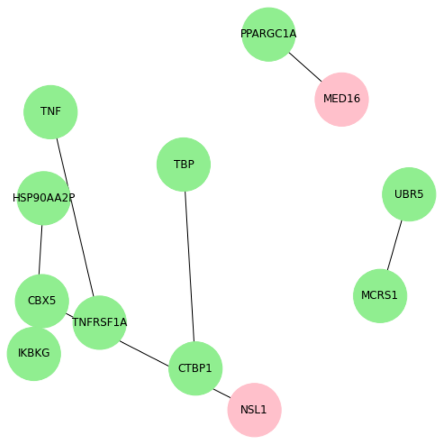

Figure 4: *The sampled subgraph of nodes with high degree centrality plotted using Networkx.*

Comparing Figure 1 and Figure 4, it could be seen, that the desired goal of **splitting the full graph into analysable components has been achieved**.

Note: by default implemented utilities extract and plot different subgraphs at the different execution trials due to the randomization components of the sampling algorithm.

iii. Feature Vectors

Using both Cytoscape and custom Networkx subgraph extraction utilities, effective detection of the most **representative features** can be done. The following 5 **node centralities** have been explored and chosen as the features:

1. Degree centrality;
2. Betweenness centrality;
3. Load centrality;
4. Eigenvector centrality (for the graph);
5. Closeness centrality.

Each of the features has been **normalized** to train and predict in a common scale. While computing the degree centrality, for multigraphs or graphs with self-loops the maximum degree might be higher than $n - 1$ and values greater than 1 are possible. Similar statement holds also for the other centralities. Therefore, it is correct that self-loops were deleted and graph was created as not a multigraph.

There was an **issue regarding node betweenness centrality** which takes too long time to be computed (at least with Networkx). There exist 2 common ways to approach this issue: use parallel execution or execute on the subgraphs, however, neither each of them, nor they both at the time give significant improvement. For the **PPI-reduced execution time is feasible**, and therefore, experiments are done on this dataset. Two available alternatives are to perform experiments on the full PPI, but without betweenness centrality or to search for the better performing library implementation.

Finally, data processing procedure completes with the creation of two dataset classes `InputDataset` and `ClfDataset`, the former of which is used for the Node2Vec and GCN model training, the latter – for the SVM and MLP.

III. Models

The two model training pipelines that rely on the different approaches to the modeling have been implemented: one relies on the Node2Vec embeddings (Approach 1) and another one – on the graph neural network (Approach 2).

i. Approach 1

i.1 Node2Vec (N2V)

First of all, Node2Vec embeddings have to be obtained, and for that **Node2Vec model has to be trained**. The model has been imported from the `torch_geometric.nn`.

Train-test split in a proportion 80/20% has been performed (2984 training nodes).

The set of the **hyperparameters** used for the best performing model training is reported in the Table 2, and the resultant embedding vectors found with this model and projected into $2D$ with PCA are plotted in the Figure 5.

Hyperparameter	**Value**
Epochs	100
Batch Size	128
Latent Dim	64
Context Size	10
Optimizer	SparseAdam
Learning Rate	10^{-2}
Walk Length	20
Walks per Node	10
Number of Negative Samples	1
p	0.5
q	2

Table 2: *Node2Vec hyperparameters.*

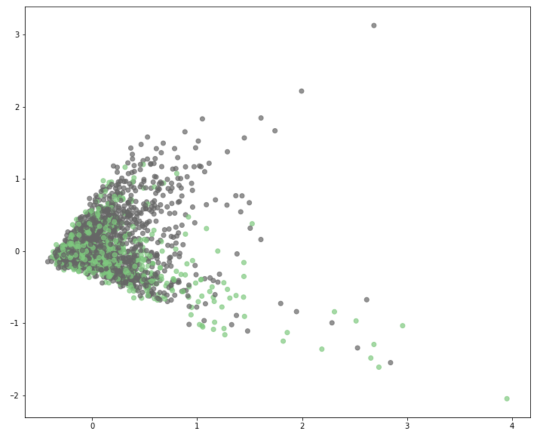

Figure 5: *N2V embeddings, projected to 2D with PCA.*

i.2 Classifiers

Once embedding has been obtained, it can be fed into classification model of choice. Since it was not obvious which classification model is appropriate for a given data, **both SVM and**

MLP models were implemented and the results were compared (refer to the following section to see the results).

Hyperparameters for both models (Table 3 and Table 4) have been found through the **grid search**, performed on the validation set with a preliminary partitioning into train, test and validation sets in a 60/20/20% proportion (2557 training nodes).

Hyperparameter	Value
Regularization Parameter	1
Kernel Type	Polynomial
Polynomial Degree	2

Table 3: *SVM hyperparameters.*

Hyperparameter	Value
Regularization Parameter	10^{-3}
Limit of Iterations	10000
Number of Hidden Layers	2
Hidden Activation	Sigmoid
Hidden Layers Sizes	(64, 64)

Table 4: *MLP hyperparameters.*

ii. Approach 2

For the GNN approach **Graph Convolutional Network (GCN)** model was chosen as the most suitable for the semi-supervised learning and implemented manually.

The best found set of **hyperparameters** can be found in the Table 5. Note, that even though network architecture is rather simple, its more complicated modifications are not helpful to boost the performance.

Hyperparameter	Value
Epochs	1000
Number of Layers	3
Hidden Activation	ReLU
Hidden Dim	16
Dropout Rate	0.2
Optimizer	Adam
Learning Rate	10^{-4}

Table 5: *GCN hyperparameters.*

IV. Experiments

The classification performances of the three classifiers **(SVM, MLP, GCN)** are evaluated on the **test data** and **compared** in terms of **(accuracy, recall, precision, and $F1$-score)**, the corresponding **confusion matrices** are then plotted. Note: the **Node2Vec best accuracy is** 0.62 and consequently limits the performance of the node classification of the entire Approach 1.

The results are **averaged** over 10 independent runs.

The **loss monitoring** has been performed during the training.

The comparative table of the average quantitative results of the models is given by the Table 6. The first observation is that **neither of the models significantly overcomes in the results two others**.

Model	Accuracy	Precision	Recall	F1-score
SVM	0.64	0.65	0.89	0.75
MLP	0.61	0.63	0.88	0.74
GCN	0.60	0.65	0.81	0.72

Table 6: *Node classification performances on the test set.*

Within a more precise exploration, it is seen, that also training history of all the models behave similarly. For example, the training history of the GCN model is drawn in the Figure 6 and Figure 7.

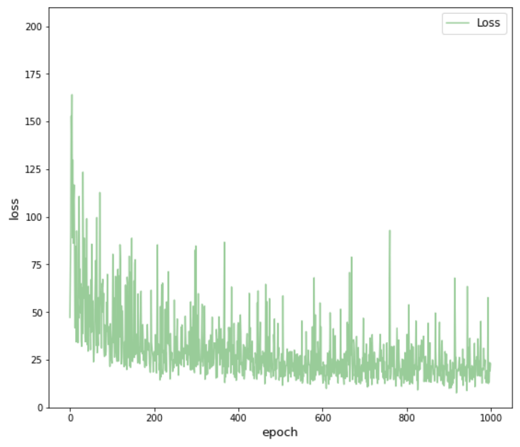

Figure 6: *GCN training loss.*

It is clear, that **the model stops to train and gets stabilized around the first few hundred of epochs**. The most noticeable decrease in loss curve and increase in the training curves of the

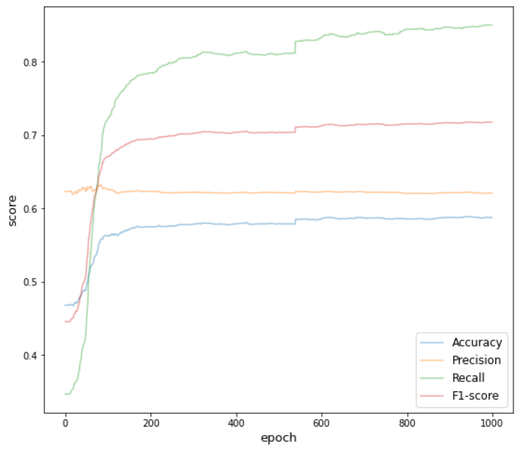

Figure 7: *GCN training metrics: accuracy (blue), precision (yellow), recall (green) and F1-score (red).*

performance measures is seen within the first 100 epochs time interval.

The most **rapid increase corresponds to the recall**, meanwhile **precision** curve almost (**does not improve**), which is also confirmed by the high value of the former and a low one – of the latter – in the Table 6. Precision-recall is a useful measure of success of prediction when the classes are very **imbalanced**. Therefore, **correctness and completeness of the system need to be checked** in a more detailed way. For that confusion matrices can be consulted. These matrices (normalized over all the population and not normalized), obtained on the tests for the GCN, SVM, MLP, respectively, are demonstrated in the Figure 8, Figure 9, Figure 10.

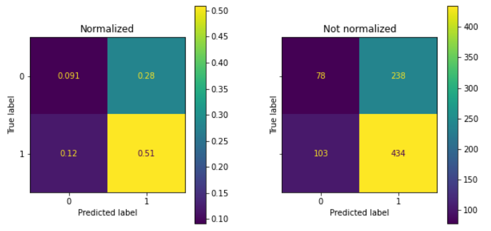

Figure 8: *GCN confusion matrices (random run).*

Plots confirm the hypothesis: the majority of all positive results are returned, but the results are not accurate (low precision). This happens when **the model is not training**. Indeed, it can be easily detected that all **the models are biased** and tend to predict all node labels as 1.

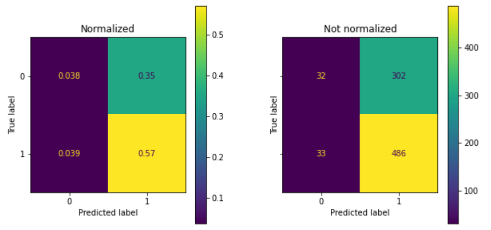

Figure 9: *SVM confusion matrices (random run).*

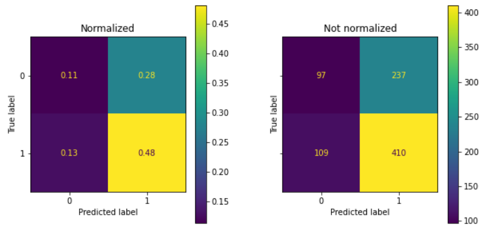

Figure 10: *MLP confusion matrices (random run).*

In general, for the biased models accuracy metric does not bring any relevant information, because for a model that does not learn anything and always predicts one of the classes, given the dataset, where the most of the samples correspond to this class, it will have a high value. The similar reason discards recall as a representative metric. And since $F1$-measure is in fact the weighted average between the precision and recall, the only really useful performance **measure to be improved at the current stage is the precision**.

Regarding the choice of the best performing classifier in the Approach 1, Table 6 could have created an impression about the SVM classifier to be slightly better, but this result seems to be misleading, because MLP model outputs confusion matrices, which characterize the **better ability to classify correctly "the minor" class**.

Finally, the execution time of the problematic components of the pipeline is reported in the table Table 7.

Component	**Time**
Networkx graph plotting	86.6
Feature computation	127.3
SVM with grid search	47.9
MLP with grid search	537.8

Table 7: *Execution time on Colab GPU (in seconds).*

V. Explainability

To be able to better treat the reasons behind the graph model predictions and develop the further improvement of it, the **GNNExplainer** is one of the possible choices of a tool.

It is reasonably desired to explore the classification result for **the nodes, belonging to the "minor class"** – the class, for which the model mismatches in the most of the cases. For example, the node with index, equal to 8, is one of such nodes and it is described in the Table 8.

Name	RBM42
Index	8
Predicted Class	0
True Class	0

Table 8: *Node to explain: 8.*

Node was explained by the GNNExplainer as represented in the Figure 11 and Figure 12, where both results are obtained deriving an **average edge mask** through the 10 independent runs and in the latter case the number of hopes is set to be equal to 2.

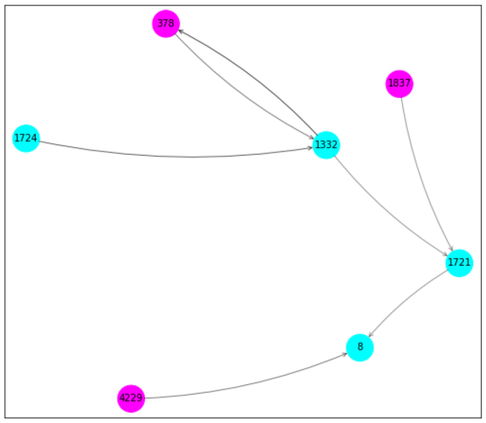

Figure 11: *Explain node 8, num_hops = default.*

As it can be seen, probably the most important node (through the information aggregation, depending on the number of hops) for the explanation of the node 8 is the node 1332.

Then the process can be continued.

VI. Conclusion

The Graph Learning on the Biological Data is a challenging task, and based on the obtained

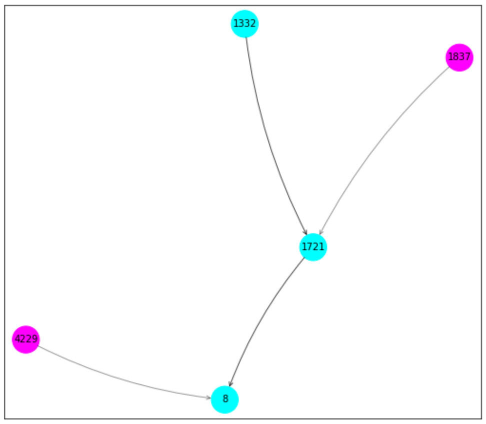

Figure 12: *Explain node 8, num_hops = 2.*

results, **it is not immediately clear, what is the right answer on the main posed question: which technique was able to better characterize the graph**. Although the various papers suggest that GCN is a more promising approach than traditional methods, such as one of the non-graph classifiers, combined with Node2Vec embeddings, the currently achieved results do not really demonstrate that. In fact, on average the models, analogous to the ones, used for the Approach 1, can outperform the graph-based models in terms of predictability. Then also, **both modeling pipelines can be improved**, e.g. more features or more important features could have been extracted on the step of data processing within a more detailed exploration of the Cytoscape capabilities, upgrading of the Node2Vec model architecture could have been performed to learn better node embeddings, more complex GCN (or another suitable for the semi-supervised learning application (Attentive FP looks interesting)) architecture could have been used, etc.

Additionally to be said, **it is also not obvious if the problem of a low precision is actually the problem of "unbalancing"** as it looks like, because in fact the dataset, corresponding to the disease of interest, has been chosen as a balanced one. But if the hypothesis that this is the indeed the case could have been confirmed, then numerous methods to adapt model training pipelines could have been tried, e.g. weighting of the graph edges, specific loss function, re-prioritizing of the "under-predicted" class, etc.

So, **the conclusion is: the further developments are needed**.