Introduction

In this work, we are asked to do binary classification on proteins (the term gene is also used interchangeably) if it is related to a specific disease or not and compare two models as will be explained below further. The assumption is that genes associated with the same or similar diseases tend to accumulate in the same neighborhood of the molecular network. Therefore, a key step is to measure the distance between candidate genes and known disease genes in the protein-protein interaction (PPI) network.

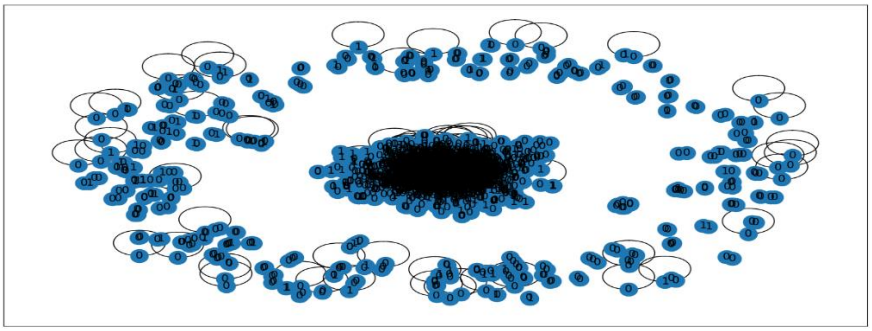
1. Data preparation:

I. before going deep into models we need to prepare our data by the use of two files provided. The first one is the Protein-protein interaction file and the other one is a file with information about different diseases and more importantly the gene causing the disease. For example, in this file which is called gene_disease_association, there are different diseases, which in this work I chose Diabetes in particular. In this table, only disease name and geneSymbol columns are important for me and by some lines of codes I create a new lighter data frame to be used later and it can be shown that there are 2359 genes causing diabetes (Note: at the end of my work I realized diabetes was not a good disease maybe because not too many genes are causing it so I changed the disease to Neoplasms(df.diseaseName.value_counts() shows the disease with the maximum number of genes)

II. On the other hand in the PPI file each line shows the interaction between genes/protein. At first glance, it is obvious that this file can be also represented as a graph where each node is a gene and simply there is an edge between two nodes if they can be seen in the same line in the PPI file. I will create the graph G by the use of the networkx library

III. Another step of our data preparation(which is done alongside step 2) is to give values/labels to each protein. in other words for each node in the graph obtained from PPI, I check if it is causing diabetes/Neoplasms By comparing the node with nodeSymbol in df obtained in step1. If so I associate the label True/1 to it and false otherwise

Now our data is somehow ready to be fed into our machine learning models. For example, the following figure is the PPI network with 01 labels for each gene if they are causing diabetes or not. This figure is obtained on a reduced version of the PPI file and the name of genes are removed from the graph so the figure is more understandable



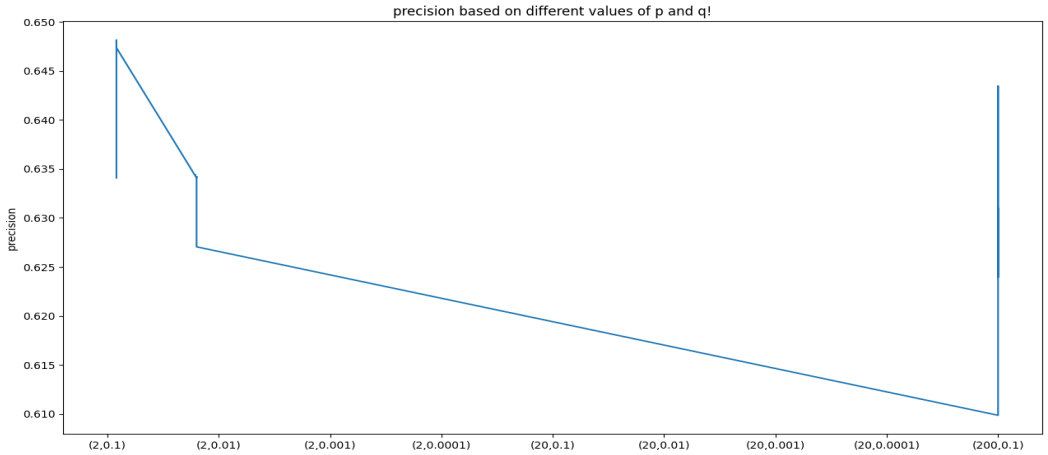
2. Model Training

a. Node2Vec embedding model: node2vec is based on word2vec models where here graph paths are seen as sentences, So by node2vec, our graph can be embedded into vectors.

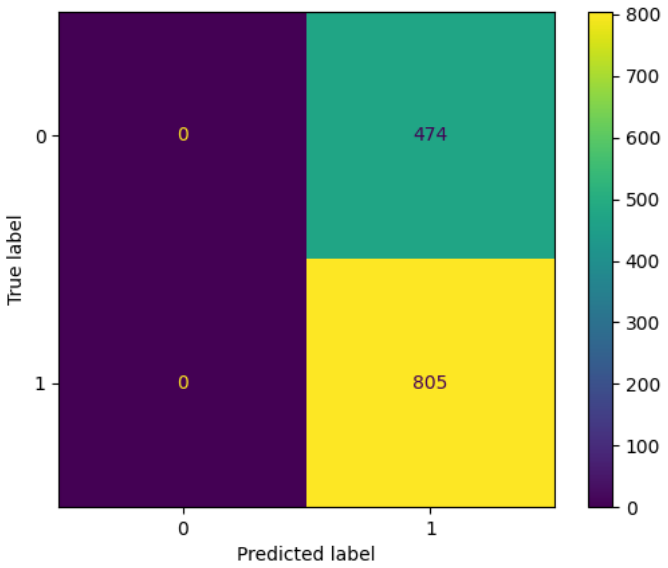
Node2vec defines a random walk with two parameters p and q. The parameter p controls the possibility of revisiting a node during the random walk. When the value of p is high, the nodes that have been visited will rarely be sampled. This strategy encourages moderate exploration and avoids 2-hop redundancy in sampling. On the other hand, if the value of p is low, it would lead the walk to backtrack a step and this would keep the walk “local” close to the starting node u. Parameter q allows the search to differentiate between “local” and “global” nodes. if q > 1, the random walk has a greater probability of sampling the nodes around the node v. Such walks can get a local view of the underlying graph. BFS samples nodes within a small locality. In contrast, if q < 1, the random walk will go farther away from v, which can get more global features information. Therefore, the distance between the sampling node and the given source node u is not strictly increased. But in turn, the measurement benefits from the superior sampling efficiency of preprocessing and random walk. the performance of the algorithm is robust to the parameters p and q. and must be evaluated
To do the actual embedding node2vec takes a graph and returns vectors for each node. If we take parameter dimension=128, for example, it simply means that each node is defined with 128 different features and each node has a value of either zero or one. hence, this can be fed to our ML model which here I have chosen SVM,
The metrics for this disease for parameters dimensions=128, workers=1,p=20,q=0.01 is as follows:

Accuracy: 0.6364347146207975 **Precision: 0.6364347146207975**
Recall: 1.0 **F1-Score: 0.7778308647873865**

However, I should check to see if these metrics are related to these values or not to do so I decided to evaluate the precision metric(since I supposed it to be more important in this context) at this point for different values of p and q : vals=['(2,0.1)','(2,0.01)','(2,0.001)','(2,0.0001)', '(20,0.1)','(20,0.01)','(20,0.001)','(20,0.0001)', '(200,0.1)','(200,0.01)','(200,0.001)','(200,0.0001)'] (from this article) on features with 32 bits dimensions. the following is the result where the best value for p and q can be chosen that are (**p,q)=(2, 0.1) (this value is unstable but let’s consider it as a good value)**



Additionally, the confusion matrix for these values can be shown as follows and other metrics are as follows:



Based on this it can be said that no disease gene is predicted as not being a disease gene, however some normal genes are predicted to cause disease

b. GNN:

in GNN, the input is a graph with nodes, and the output of it would be a graph with nodes with vector representation(how the node belongs in the context of the graph by some knowledge about node and neighbors iteratively in many steps to expand this knowledge). The neural network here has the responsibility to obtain this knowledge by updating weights in the network. So the only challenge in this part would be understanding what is appropriate input for neural networks, and the rest of the load of the work would be leveraged on the power of the neural network. For this purpose, same as the previous part the graph G must be created based on the protein-protein interaction file and edges must be added from each node. Additionally, I decided to consider an edge from a node to itself. (in code I needed to consider one more consideration after creating the graph and it is actually a papping index of nodes instead of their names) After creating the graph I need to transform it into a simple representation to be understandable for my code and nothing could be better than an adjacency matrix(not considering space cost by dense matrix) which will give information about the connection of nodes to the others and additionally they correspond to their label (y).
now having input and output, the neural network can be created, which decided to have 2 hidden layers and one output layer. The convolutional layers for the graph is implemented by GCNConv by torch_geometric library, there are some design choices in implementation by NN which worth further evaluation like number of layers, number of output layer in the first layer, learning rate, and other famous parameters in every ML algorithm but what is interesting here is that it is much faster than the previous model and can be reached to more or less same accuracy in a very short amount of time with the following result:

```
Train Accuracy: 0.6404155495978552
Test Accuracy: 0.6270523846755277
```

2. Comparison

The graph Conv. NN seems mathematically more difficult to understand concerning the node2vec model but it seems faster since classification, and embedding is all done by NN and the message passing by the use of convolutional NN. I believe GNN is less dependent on parameters such as p and q that we have in node2vec. Consequently, maybe node2vec might not be appropriate for very large data with many features, GNN is interestingly faster and it can be extended to be used for explainers by the use of GNNExplainer for example the following is the output of GNNexplainer on node 10