Link Prediction in Disease Gene Association Network

Proietti Lorenzo
Perrella Stefano
Montano Francesco

Motivation and Data

The information about genes and variants involved in human diseases can be used for different research purposes, including the investigation of molecular mechanisms of species diseases and their comorbidities, the analysis of the properties of disease genes, the generation of hypotheses on drug therapeutic action and drug adverse effects, the validation of computationally predicted disease genes, and the evaluation of text-mining methods performance.

Our project focuses on the task of link prediction on disease-genes association network, which are such that nodes are genes and diseases and edges represent associations between them.

The dataset we used is the disease-genes association network you can find at dataset.

The dataset

This dataset is quite large, it contains 23484 nodes, of which 5663 represent diseases and 17821 represent genes. In particular it is highly connected, having 15509618 edges, that are a lot given that the graph is bipartite (a gene can be connected only to a disease, and viceversa).

Nodes	23484
Disease nodes	5663
Gene nodes	17821
Edges	15509618

Features

Genes and Diseases Features

We considered as Features of both genes and diseases their respective similarity. In particular, the genes feature matrix contains gene-gene similarities and the diseases feature matrix contains disease-disease similarities.

Each gene is typically associated to a set of GO (Gene Ontology) terms; each of these terms refers to some specific characteristic of genes. GO terms are organized in DAGs (Directed Acyclic Graphs) which define their ontology; we can have three types of ontologies: Molecular Function, Biological Process and Cellular Component. We picked Molecular Function's DAG and computed the similarity between every pair of genes that are present in our dataset considering them as sets of GO terms and implementing a similarity between such sets.

Also the diseases can be represented as sets of terms. In this case, each set contains DO (Disease Ontology) terms that represent the causes of the disease. The similarity measure we used is the same for GO terms and DO terms and is in the following slide.

Genes and Diseases Features

This formula has been written referring to terms, but the same goes for the diseases.

$$\mathrm{Sim}(G1,G2)=\frac{\sum_{1\le i\le m}\mathrm{Sim}(go_{1i},GO_2)+\sum_{1\le j\le n}\mathrm{Sim}(go_{2j},GO_1)}{m+n}$$

Where go_{ki} represents the *i-th* GO term of gene *K*. *m* and *n* are respectively the number of terms of the two genes we are comparing. In each sum there is instead the similarity between a single term and a set of terms and is defined as below:

$$\mathrm{Sim}(go,GO)=\max_{1\le i\le k}(S_{GO}(go,go_i))$$

The similarity $S_{GO}(go, go_i)$ has been instead defined starting from the DAG structure and we haven't reported it here for brevity reasons. All specific formulas can be found at [1].

Methods

- Weisfeiler-Lehman Link Prediction
- DeepWalk
- Graph Convolutional Networks

Weisfeiler-Lehman Link Prediction

Description and Model Structure

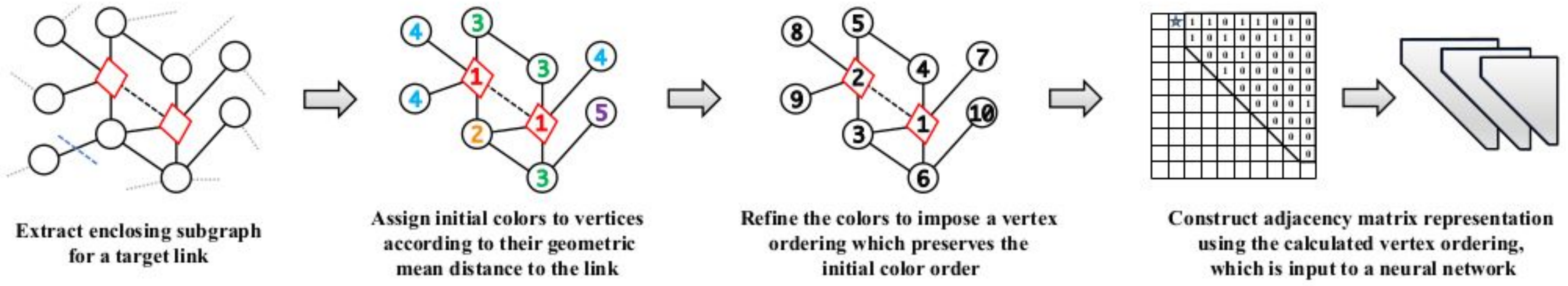
This model consists in extracting various subgraphs from the original graph and use them to predict the existence of links. In particular, here we don’t use any of the previously described features, which will be employed only in the method that makes use of Graph Convolutional Networks.

The task is to predict the existence (or not) of a link in the graph. In order to make such prediction we

- compute an embedding for each link,
- predict if a link is present or not.

Link Embeddings

To compute the embedding of a link we compute the subgraph associated to it: each subgraph is the flattened vector associated to the strictly upper triangular adjacency matrix relative to the subgraph (after having removed the entry that codifies the presence of the link of interest). Moreover, in order to get vectors that are meaningful and and coherent with each other, we determined the number of nodes for each subgraph to be equal to 10 and sorted all the nodes that appear in each subgraph according to the same metric: vertices receive similar rankings if their relative positions and structural roles within their respective subgraphs are also similar.


Extract enclosing subgraph for a target link
Assign initial colors to vertices according to their geometric mean distance to the link
Refine the colors to impose a vertex ordering which preserves the initial color order
Construct adjacency matrix representation using the calculated vertex ordering, which is input to a neural network

Link Predictions

In order to predict the presence (or not) of a link we use a three layer feedforward fully connected neural network.

The input to the network is the embedding of a link, which is a vector of 44 entries. The network is structured in the following way:

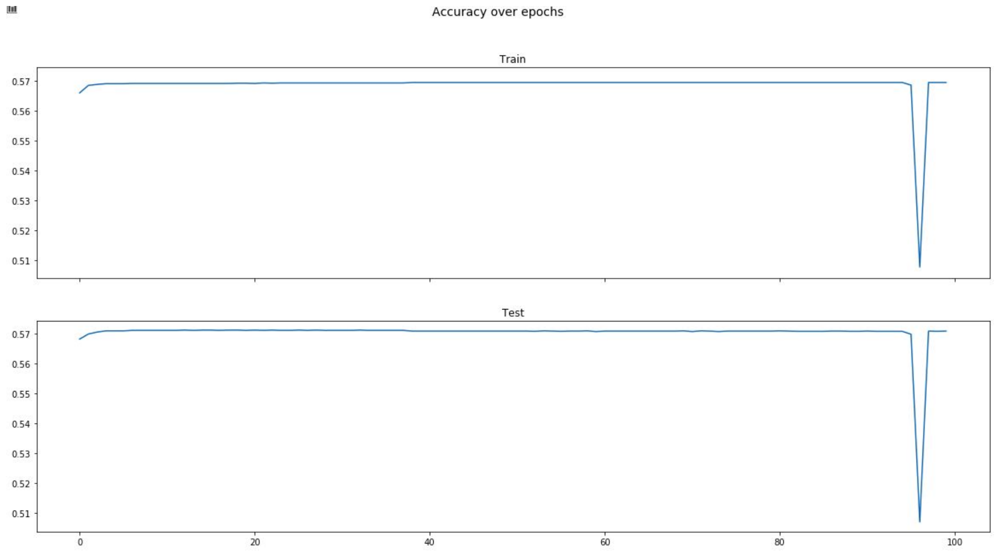
- a 32 nodes Dense layer,
- a ReLU activation function,
- a 32 nodes Dense layer,
- a ReLU activation function,
- a 16 nodes Dense layer,
- a ReLU activation function,
- a 1 node Dense layer,
- a Sigmoid activation function

The output of the network is a value that represents a degree of confidence with which the input link it’s believed to exist. This value is rounded to the nearest integer representing existence (1) or non existence (0).

Results

As we can see, the results are terrible. This is probably due to the high connectivity of the graph: given to the ordering of nodes in the subgraphs, the vector of most subgraphs embeddings contain the exact same values, resulting in almost every input to the neural network being the same.

The final accuracy score on both train and validation sets is 0.57.



DeepWalk

Description and Model Structure

With this approach we first build the embedding of the nodes of the graph by using DeepWalk technique and then use them to predict the existence of links. Also in this case we don't use the features of genes and diseases that we have previously computed.

The task is to predict the existence (or not) of a link in the graph. In order to make such prediction we

- compute an embedding for each node,
- compute the embedding of each link,
- predict if a link is present or not.

Node Embedding

We used DeepWalk in order to build the embedding of each node of the network. This method is divided in two phases:

- generate truncated random walks
- use word2vec in order to generate the embedding

The truncated random walks are represented as a sequence of nodes. These sequences can be thought of short sentences in a special language, so we can apply word2vec (the skipgram version) to build the embedding of each node (term of the sentence). SkipGram is a language model that maximizes the co-occurrence probability among the words that appear within a window w around the target word. By considering $\phi(v)$ as the embedding of the node v we maximize

$$P(u|\phi(v))$$

for each node u that appears within the window w of v.
In this implementation I decided to use 50 walks starting from each node, each one of length 30. In word2vec I used a window size of 3 and the resulting embedding of the nodes is of size 100.

Link Prediction

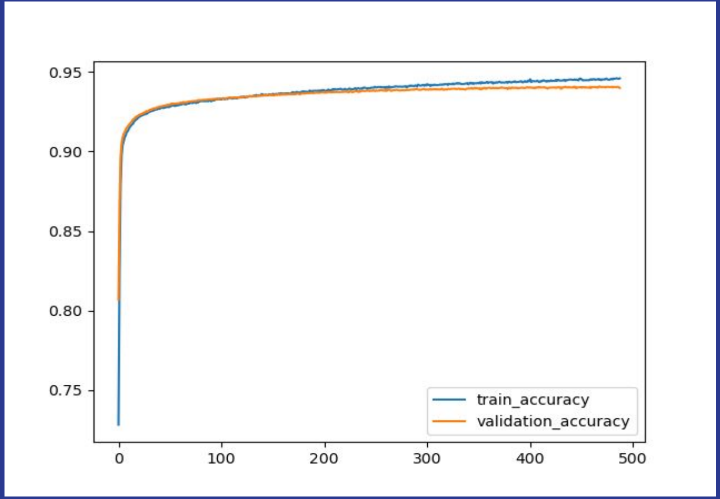
In order to perform the task of link prediction, we compute a link embedding. Each link embedding will be defined as the concatenation of the embeddings of the two nodes which participate to the link (the first is the gene, the second is the disease). So we represent each link as a vector of size 200.

In order to predict the presence of a link we use a simple fully connected neural network of two layers, whose structure is the following:

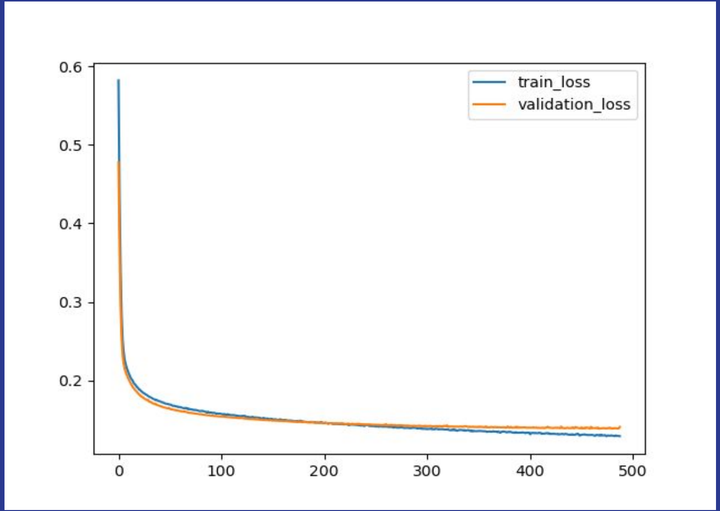
- a Dense layer with 300 nodes,
- a Dropout layer with 0.5 drop rate,
- a ReLU activation function,
- a Dense layer with 1 node,
- a sigmoid activation function.

DeepWalk Results

We trained the model for almost 500 epochs with an early stopping of 20. The final accuracy score on the test set is of 0.9393.



accuracy



loss

Graph Convolutional Neural Networks

Description

With this approach we use a Graph Convolutional Neural Network (GCN) to build the embedding of each node. Differently from other methods, we never build the actual embedding of a link.
The embeddings of the nodes are then used to predict the existence or not of the link to which the nodes would participate.

In order to predict the existence of a link we train the gcn in an end to end manner such that the embeddings are generated directly depending on the fact that a link is present or not, and not based on some other technique or heuristic (like enclosing subgraphs in our first method, or DeepWalk in our second method).

The Convolution

When it comes to develop convolutional neural network having graph structures as input, the convolution is computed as follows:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Where $\tilde{A}$ is the adjacency matrix after self loops on every node have been added. $\tilde{D}$ is the degree matrix, which is a diagonal matrix whose element at row *i* is the degree of node *i*. $H^{(l)}$ is the output of layer *l-1* (in the first layer this matrix is represented by the features of nodes) and $W^{(l)}$ is the weight matrix at layer *l.*

The Model

In order to define the graph neural network model, we need first to define a graph convolutional block: in this block the operation previously described has been implemented. Moreover, we implemented two versions of this block in order to be able to deal with sparse matrices.

The neural network has the following structure:

- a graph convolutional block with 64 nodes,
- a ReLU activation function,
- a graph convolutional block with 64 nodes.

The output of the last layer is a matrix with num_nodes rows and 64 columns, such that row *i* is the embedding of node *i*.

Link Prediction

The link prediction task is performed in the following way:

- compute the scalar product of the embedding of two nodes (a gene and a disease),
- apply sigmoid activation function to the obtained value,
- round the value to the nearest integer (0 represents non existence while 1 represents existence).

We took inspiration for the structure of the model from [2], implemented in tensorflow 1.0, but we had to reimplement it from scratch in tensorflow 2.0 in order to be able to use such model with a graph as large as ours.

Results

We performed different trainings of the same model. In particular, we can differentiate between two categories of trainings

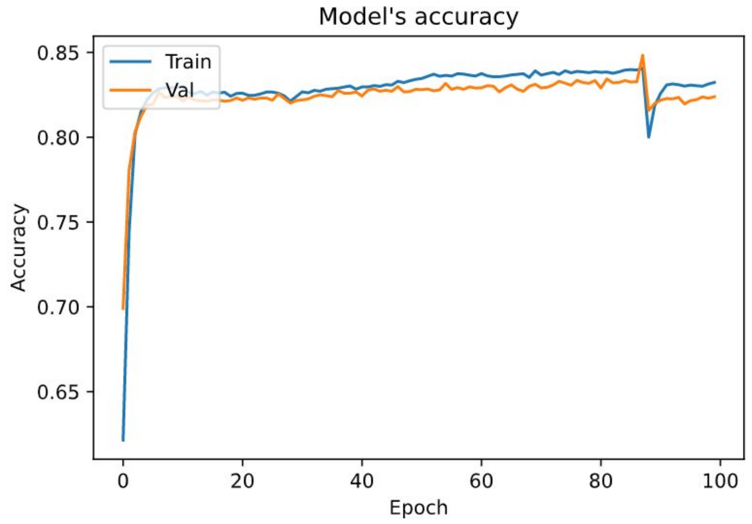
- using features of nodes,
- not using features of nodes.

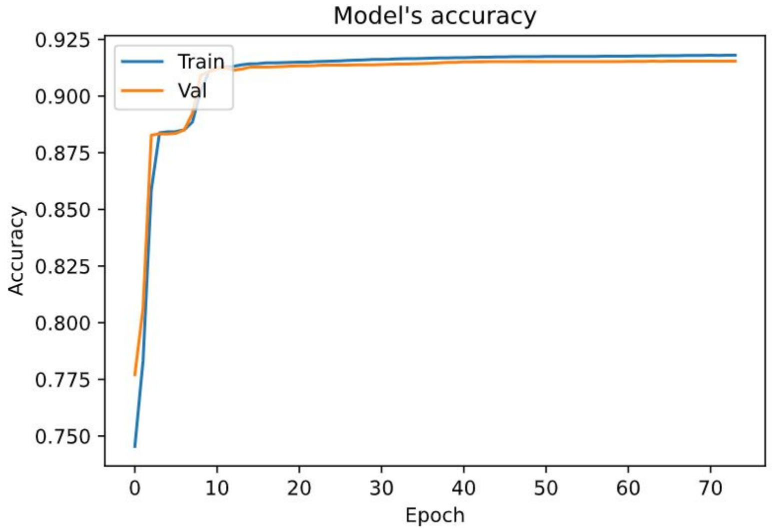
Moreover, when using features the model was impossible to train because of its size, so we trained a 0.25-percentage sampled version of it. In order to compare with the featureless version, we trained a sampled version of it too.

In addition, we tried to train a model that uses only the upper triangular part of the adjacency matrix of the graph, as if it was directed.

Results - features vs featureless

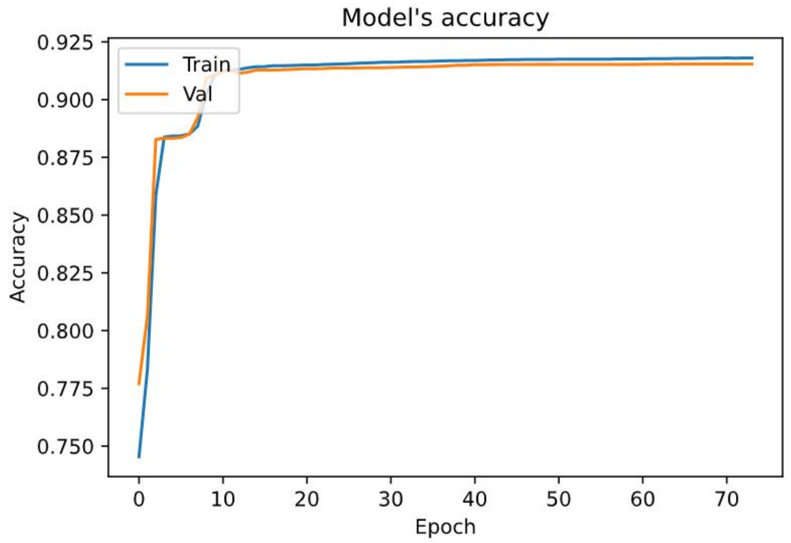
On the left we have the accuracy score of the model which uses the features of nodes as input, while on the right we have a model whose input is constituted by an identity matrix, as if the only feature of a node is the node itself. The featureless model reaches a way higher accuracy.

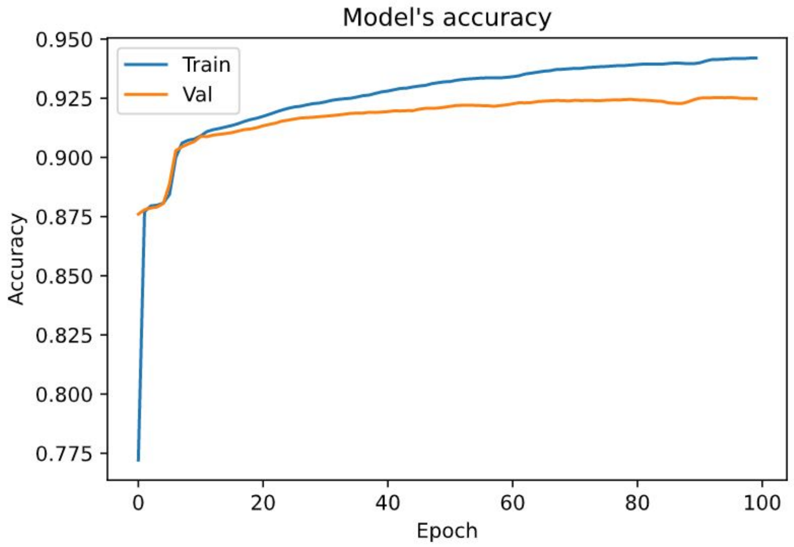




Results - complete adj vs upper triangular adj

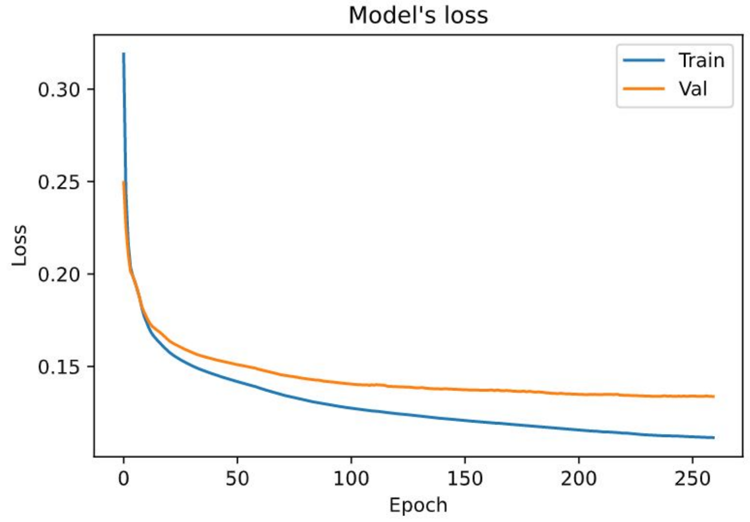
The model on the left uses the entire adjacency matrix and its accuracy on the validation set is around 0.915. The other model uses instead only the upper triangular version, and its accuracy is around 0.925, which is slightly better. At the same time, the model on the right manages also to reach a train accuracy of almost 0.95, showing to be more capable to learn the underlying structure of data.

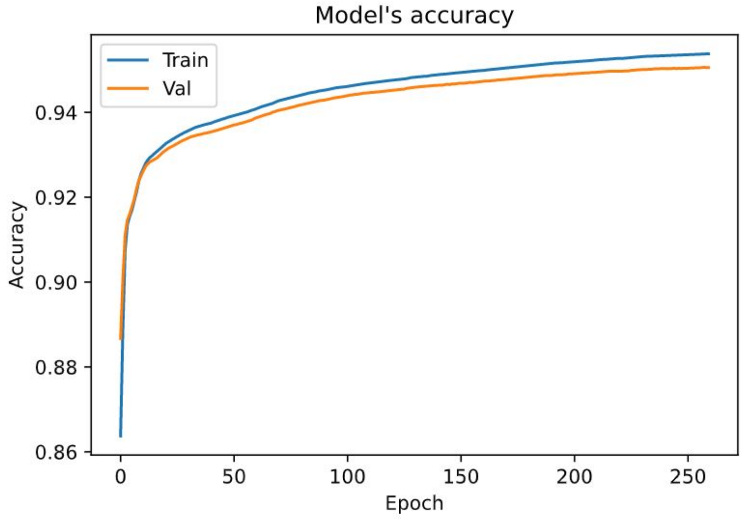




Results on the entire graph

We decided to train the entire model using only the upper triangular version of the adjacency matrix, considering as features the identity matrix. These decisions have been made based on the results of the sampled versions. We can see how this model, even after 200 epochs, continues to slowly improve reaching the highest accuracy of 95%.





Conclusions

We trained different models using different techniques. Some of these models were not able to train on the entire data and so we sampled it. We can only compare models which where trained on the same amount of data.

Anyway, both using sampled data and the entire data, according to accuracy metric the technique which has proven to be the best is the one employing Graph Convolutional Networks, closely followed by the one which uses DeepWalk embeddings. The memory and time constraints we encountered when training the gcn, though, have been stricter: we were not able to train it using a GPU because of high memory usage and as a result each epoch took something like 12 minutes, resulting in almost 50 hours of training. The DeepWalk based method, instead, required 6-7 hours to build the embeddings and just one hour to train, with less strict memory requirements.

References

[1] A new method to measure the semantic similarity of GO terms, Wang et al., 2007.

[2] Graph Convolutional Prediction of Protein Interactions in Yeast.