Drug Interactions Prediction

Network Machine Learning – Final Project

Marie Biolková **Jiří Lhotka**

EPFL, Lausanne, Switzerland
{marie.biolkova,jiri.lhotka}@epfl.ch

Abstract

In this project, we investigated the link-prediction `ogbl-ddi` dataset. We first analysed the network and established that it is a small-world network with some scale-free properties. We then extracted custom features based on these findings and fit a logistic regression model, establishing a baseline performance of 10.7% Hits@20 on the test data. We then use the findings from this part to design a graph convolutional network (GCN) model, which achieves 62.9% Hits@20 on the test data, and would thus place in the top 10 models in the world-wide leaderboard without any fine-tuning.

1 Exploration

1.1 Dataset

The `ogbl-ddi` is a dataset representing drug interactions amongst FDA-approved drugs. These interactions can be represented in the form of an unweighted, undirected graph, where each node is a drug and each edge represents an interaction between the drugs[1]. An interaction is defined as the drugs influencing each other's effect, i.e. the effect of taking both drugs together being "considerably different from the expected effect" taking each one of the drugs on its own (Wishart et al., 2018; Hu et al., 2020).

In this section, we explore the data in order to identify an appropriate baseline, as well as to better understand the dataset and get an intuition for which models could perform well on it. The `ogbl-ddi` dataset comes pre-divided into training, validation and test edges. In this analysis, we will focus solely on the *training* part of the dataset.

The dataset is striking in that it does not contain any node attributes, which would likely be extremely helpful for the task (information such as drug intended effects or drug composition would likely help with edge predicting). Consequently, models designed for the task have to rely solely on the graph's structure, which will therefore be the sole focus of our exploration.

1.2 Initial Analysis

The graph is connected and has $4,267$ nodes and $1,067,911$ edges. This means it is unusually dense: most real networks are sparse, with number of edges $O(N)$ where N is the number of nodes (Frossard, 2022a). Indeed, the density of our graph is 0.12 and its average degree is 500.5, which is much higher than most real-world networks, whose average degrees rarely exceed higher tens (Frossard, 2022b).

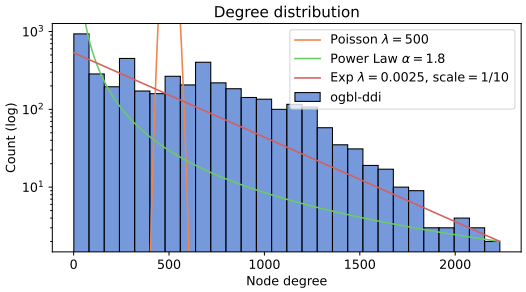

Figure 1: Degree distribution of `ogbl-ddi`.

The fact that our graph is unusually dense manifests itself also in its degree distribution, which is shown in Figure 1, with the nodes' degrees ranging from 1 to $2,234$. Unsurprisingly, the degree distribution does not match at all a Poisson distribution, which often works well for sparse random graphs. Perhaps more surprising is the relatively bad fit of a power law distribution to our data, since power law distributions are fairly common in real-world graphs and since our average distance (average shortest path) $\hat{d}$=2.1 $\sim \ln\ln N$=2.12, which

[1] See visualization: https://mariegold.github.io/link-prediction/index.html

is typical for scale-free networks with exponent $\alpha \in [2, 3]$. Furthermore, the second moment of our network is rather large, $176,801$, which is also a typical property of scale-free networks (Frossard, 2022c). Despite this similarity, Figure 1 clearly shows that the degrees of our network decrease too slowly when compared to a power law, even with the relatively low exponent of $\alpha = 1.8$. The best fit for our degree distribution therefore seems to be an exponential distribution, but even there the appearance may be deceiving – while on the logarithmic plot of Figure 1, the fit seems very close, for instance the average degree of the exponential plot plotted there is only ~ 95 (due to scaling – if we wanted to preserve the average, the fit is not tight at all), much lower than in the actual network.

Despite not fitting any of the theoretical distributions well, the graph presents many small-world properties: it has a high average clustering coefficient of 0.514 and a low diameter of 5, with an average distance $\bar{d}$ of 2.1. In small worlds, we typically observe that $\bar{d} \sim \frac{\ln N}{\ln \bar{k}}$ where N is the number of nodes in the network and k the average degree (Frossard, 2022b). In our case, this prediction forecasts a $\hat{d} \sim \frac{\ln N}{\ln \bar{k}} = \frac{\ln 4267}{\ln 500.5} = 1.34$, which is indeed close to the observed value of 2.1.

1.3 Comparison with Network Models

Model	N[1]	$\lvert E\rvert$ [2]	$\bar{k}$[3]	$\bar{d}$[4]	C[5]
`ogbl-ddi`	4267	1068k	500	2.1	0.51
BA	4267	1067k	500	3.94	0.02
WS (β=0)	4267	1068k	500	~4.3	0.75
ER (p=0.117)	4267	2134k	500	~1.3	0.12

Table 1: Comparison of `ogbl-ddi` with Barabási-Albert (BA), Watts-Strogatz (WS) and Erdős-Renyi (ER) models.
1 – # nodes, 2 – # edges, 3 – avg degree, 4 – avg distance, 5 – clustering coefficient

A quick comparison with the standard random network models in Table 1 shows none of them quite fit our graph – while Barabási-Albert and Watts-Strogatz fit the number of edges as well as the average degree quite well, they do not have the appropriate average distance or the average clustering coefficients. Furthermore, Barabási-Albert does not preserve the strong local connectivity we see in our graph (as also manifested by its small average clustering coefficient). Similarly, Watts-Strogatz is known not to generate very realistic degree distributions, and even though it was invented specifically to model small-world networks, we can see its average distance is still larger than that of our graph (Barabási and Pósfai, 2016). However, arguably, this is largely a consequence of our choice of β, and with some tweaking of this value, we could get an average distance as well as clustering coefficient more resembling that of our network. We can therefore conclude that out of the standard network models, `ogbl-ddi` most resembles the Watts-Strogatz model, which is another piece of evidence for the idea that our network exhibits small-worlds behaviour as we conjectured in §1.2. Finally, we can see the Erdős-Renyi model does not fit `ogbl-ddi` in any of the parameters except the average degree $\bar{k}$ and the number of nodes N, which were the values we used to calculate edge link probability p. This is not surprising, as the ER model does not generally represent real-world data very well (Frossard, 2022b).

1.4 Spectral Analysis

The fact that `ogbl-ddi` does not have any node or edge labelling and that the edges are unweighted limits the extent to which spectral analysis can be applied. However, we can still look at the eigen-decomposition of the adjacency matrix to contribute to our previous analysis. In particular, looking at the algebraic connectivity of 0.78 confirms our previous observations of the fact that the graph is very well connected.

1.5 Conclusion: Implications for link prediction

The initial link analysis has shown several interesting properties of the `ogbl-ddi` graph. Firstly, the graph is unusually dense and connected and, contrary to our expectations, shows less of a power-law tendency than anticipated. This means that while our initial intuition was that there would be a few drugs in the dataset that have conflicting impact with almost everything, the dataset is, in fact, much more balanced – it does exhibit some of the power-law behaviour in that there is a long tail in the degree distribution (c.f. Figure 1), but nodes around the average degree are much more common than in a scale-free network, in fact, the median degree (446) is close to the mean (500.5), which confirms the network is not exactly scale-free.

Further analyses show rather than being scale-free, the network exhibits many small-world properties, which can be a starting point for building a link-prediction model on top of it. In particular, note that the network has a very high average clustering coefficient: if we are asked to predict a link

between A and B, we know that if A and B share a neighbour, there is a $> 50\%$ chance that they are also connected (Barabási and Pósfai, 2016). While this might seem like barely higher than random at the first glance, this is actually a large improvement – since networks are typically sparse, generally the chance that any two nodes are connected is very low, much lower than 50%. In our case, since our network is unusually dense, the chance is $\sim 12\%$, the density of the graph (c.f. §1.2), which is much higher than in most graphs, but still much lower than 50%.

1.5.1 Neighbour Overlap Analysis

The finding of the previous paragraph leads to the following idea: since we know that in our network, many nodes have high degrees and the nodes are highly clustered, why only rely on the presence of a single common neighbour? We know that small-world systems tend to produce cliques, i.e. fully (or almost fully) connected subgraphs (Barabási and Pósfai, 2016). If two nodes are part of the same clique, the chance that they are connected is likely to be high, hence a useful feature could be to look at how many neighbours they have in common.

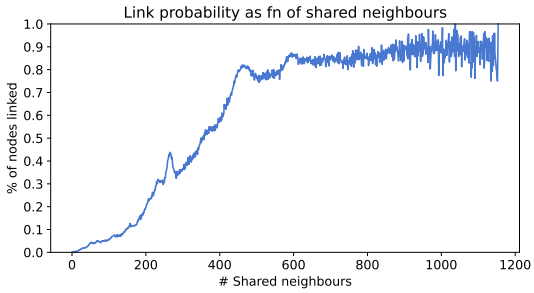

Figure 2: Link probability as function of neighbourhood overlap.

Such an analysis could be done either by looking at link probability as a function of the number of shared neighbours, or as a function of the number of shared neighbours weighted by the degrees of the nodes, the idea here being that for a node with fewer degrees, each of its links is more informative. The latter option could be measured by simply calculating the Jaccard distance between the neighbourhoods of each node, i.e. the intersection over the union of the neighbourhood sets.

We have performed both analyses and present the results of that of the mere number of shared neighbours in Figure 2 (the Jaccard distance analysis is not shown as it was very similar). The figure confirms our expectation: indeed, the more shared neighbours two nodes have, the higher their probability of being linked, and this relation is almost entirely monotone (notwithstanding the noise for nodes with large neighbourhoods, which is largely explained by the low number of such nodes).

The significance of this finding is limited by the fact that the number of nodes who share x neighbours declines exponentially as a function of x as shown in Figure 3. The number of shared neighbours is therefore not a silver bullet that completely solves the task at hand; however, it is nonetheless an important predictor that we can use to begin exploring potential models and define a baseline.

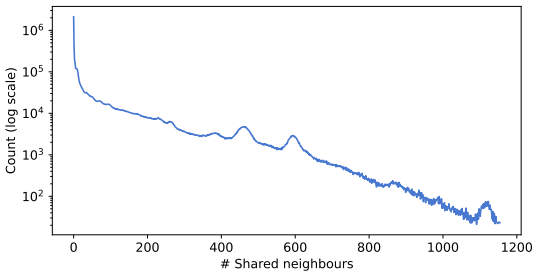

Figure 3: Histogram of shared neighbours.

2 Exploitation

Having explored the training graph, we can now start building our prediction model. The `ogbl-ddi` dataset comes conveniently pre-packaged into a set of training edges analysed earlier and a validation set of $133,489$ positive and $101,882$ negative edges. To train our model, we therefore sample edges with the sample probability to achieve roughly the same distribution and prevent negative edges being overrepresented.

The `ogbl-ddi` dataset is evaluated using the *Hits@20* metric, which is also the metric we will use in our evaluation. It represents the fraction of correct edges ranked above K by the model.

We present three models: a baseline logistic regression which only uses hand-crafted features (§2.1), a more advanced logistic regression which uses node embeddings from node2vec or Laplacian eigenmaps (§2.2), and finally a simple Graph Neural Network (§2.3). In this section, we only discuss the design choices made for each of the models, and we leave the discussion of their performance for the next section (§3).

To train all our models, we use all the existing edges in the training set and sample an equal number $(1,067,911)$ of negative edges from

the training graph using `torch_geometric`'s `negative sampling` method.

2.1 Baseline Model: Logistic regression with hand-crafted features

We start our work by defining a simple baseline, which we chose to be a logistic regression model with hand-crafted features. We opt for this model to get us a benchmark to compare more complicated models against, and also to exploit its high interpretability to get deeper insights into what features are important for our task.

Based on the analyses done in §1, we chose two groups of features for our logistic regression: *edge features* and *node features*. We define *edge features* as features that have a single value for each edge and are thus ordering invariant: these are the number of neighbours the nodes share (i.e. the number of paths of length 2 between the nodes), and the Jaccard distance of their neighbourhood sets of the nodes. The *node features* are the degree of each of the endpoints of the edge as well as their betweenness and eigenvector centralities, and local clustering coefficients. For each of the node features, we include their $\min$ and their $\max$ to alleviate the issue of ordering (i.e. the fact that edges (A, B) and (B, A) are the same edge, but the degree of the first and of the second node is different). The centrality measures are added to capture the relative importance of each node within the graph, which is likely to be useful for link prediction, as it captures the global structure of the graph.

2.2 Embeddings Model: Logistic regression with node embeddings

Hand-crafted features induce strong priors into the model and often generalize poorly. An alternative approach to learning node representations is using embeddings, i.e. low-dimensional representations of the nodes in a latent space that is designed to have a high a discriminative power for the task at hand. Different embedding-creating algorithms capture different properties in the embedding space. Here we are most interested in capturing close neighbourhoods and communities, as we have seen they are related to the link probability in §1.5.1 as well as when examining the results of the Logistic Regression model, which we present in §3.1.

To that end, we consider two embedding types: node2vec and Laplacian eigenmaps. Node2vec (Grover and Leskovec, 2016) is based on biased random walks. It is parametrized by p, q which determine whether the walk is biased towards a breadth-first-search to force similar embeddings for nodes within the same community, or a depth-first-search for capturing the role a node has in the graph. In our task, informed by the discoveries that our graph is a small world that we discussed earlier, we opt for the former, notably we set $p = 1$ and $q = 0.5$. The algorithm first generates a random sample of sequences, which are subsequently processed by a skip-gram model (Mikolov et al., 2013). The generation stage is a computationally expensive process, whose current implementations do not benefit from GPU acceleration. Consequently, due to limitations posed by the computational resources available to us, we limit our experiments to $1,000$ walks of length 5, although we expect a higher number of walks would be more appropriate considering the size of our graph. Our code uses `gensim` (Řehůřek and Sojka, 2010) for the implementation of the skip-gram model.

Laplacian eigenmaps (Belkin and Niyogi, 2003), as the name hints, work with the eigendecomposition of the graph Laplacian, L. Specifically, the embeddings are defined as the first D non-trivial eigenvectors of L. Since the computational overhead is not too high, we considered $D \in \{8, 16, 32, 64\}$.

Once we have obtained the embeddings, we use them to encode the nodes in our edge list. Each edge's features are then the Hadamard product of the embeddings of its endpoints, which is the standard approach for encoding edges (Thanou, 2022). These features are then fed into a logistic regression model.

2.3 GNN Model

Graph neural networks (GNNs), which have have received a lot of attention in recent years, and have been successfully used in various contexts, including drug discovery. Crucially, they offer desirable properties of permutation invariance and equivariance, meaning that the way we choose to label the nodes is not important, and permuting the input will lead to the same permutation in the output layer (Thanou, 2022).

Our architecture has 2 major components: *convolutional layers* and *fully connected layers*. The embedding matrix is trained simultaneously with the rest of the network, from randomly initialized weights (using Xavier normal initialization (Glorot

and Bengio, 2010)). The embedding dimension was set to 128, with the reasoning that it should be large enough to allow sufficient flexibility without overparametrizing the model. The convolutional layers are the crux of our network: they aggregate information from neighbours, and apply an activation function before passing the output to another convolutional layer. This so-called message-passing approach is a powerful tool for obtaining graph representations. In our implementation, we employ the `GCNConv` variant, whose propagation rule is based on a first-order approximation of spectral convolutions on graphs (Kipf and Welling, 2017). We use the ReLU activation function, so that our update rule from layer l to the next one is:

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

where W is the trainable weight matrix, $\tilde{A}$ is the adjacency matrix with self-loops added, and $\tilde{D}$ is a diagonal matrix of its row sums. For simplicity, we apply 3 convolutional layers of size 128 each.

Finally, the convolved embeddings are used by the predictor, f. Given a pair of nodes as input, the predictor first passes them through the convolutional blocks to encode them. Then, it processes their element-wise multiplication by 2 fully connected layers of size 128. The last one outputs a scalar which is transformed by a sigmoid to yield the final probability of a link between the nodes.

The loss function incorporates N_{neg} negative samples, ones that are not in the set of edges E:

$$\mathcal{L}(E) = -\frac{1}{|E|}\sum_{(u,v)\in E}\log(f(u,v)) - \frac{1}{N_{\text{neg}}}\sum_{(u,v)\notin E}\log(1-f(u,v)).$$

Unfortunately, the training of GNNs is computationally expensive. Consequently, given the computational budget at hand, we did not focus on any hyperparameter tuning. We thus consider our results to be a lower bound on the model's potential. Specifically, we use Adam (Kingma and Ba, 2015) with the default 0.001 learning rate to optimize for 100 epochs, without weight decay, and clip all gradients to 1 for better stability. We regularize via dropout with probability 0.5 after convolutional and fully connected layers (except for the last layer). A large batch size of 512 examples proved to be a good choice.

3 Results

3.1 Baseline Model

First, the baseline logistic regression model achieves a rather impressive Hits@20 score of 0.107 on the test set. This result is much better than we anticipated: indeed, the last model on the official `ogbl-ddi` leaderboard, i.e. 20$^{\text{th}}$ best model world-wide from 2020, which uses 1,224,193 parameters and was the result of a scientific paper, achieves only a slightly better result of ~ 0.1368.

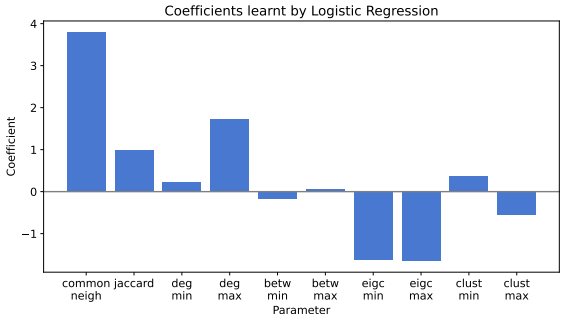

Figure 4: Feature importance for logistic regression (features rescaled to $\mu = 0, \sigma = 1$).

Figure 4 shows the resulting relative importance of rescaled features, which confirms our observations from the exploration part, namely from §1.5.1, that common neighbours are an important predictor – it turns out to have the highest coefficient and thus to be the most important feature. (Note that the features were scaled to zero mean and unit variance, which enables this conclusion.) Its importance was also confirmed by an *ablation analysis* – removing the *common neighbours* and *Jaccard distance* features causes a 1000-fold drop in Hits@20 to 0.00016.

Interestingly, local clustering coefficient is not as useful as we expected – this is likely partly due to the fact that the same properties it would capture are already (and better) captured by common neighbours and Jaccard distance.

Similarly, betweenness centrality was not a good predictor, probably because it best characterizes nodes that are in between communities, or at the peripheries.

3.2 Embeddings Model

The results of the embeddings model show that the embeddings we have created fail to capture the important notions that we have handcrafted in the baseline model. With node2vec, we only achieved 0.025% Hits@20. The reported score was obtained

with $D = 32$, and since varying the dimensionality did not improve the poor performance, we did not investigate node2vec any further. A likely explanation for such a low score is that the embeddings did not have enough data to learn from. Indeed, as we mentioned previously, with our graph size and density, we would need to run many more random walks to capture the graph structure sufficiently well. However, this was computationally unfeasible, which shows the limitations of this model for dense graphs with many edges such as ours – indeed, even the relatively small model shown here took several hours to train on our 2016 MacBook Pro.

Laplacian eigenmaps performed significantly better, with 5.9% test Hits@20, achieved with $D = 16$. This method is method was not only much faster than node2vec, it also does not have as many hyperparameters to tweak. However, it does not outperform our baseline.

3.3 GNN Model

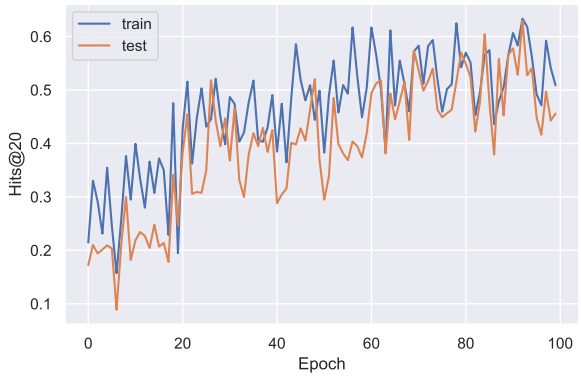

Figure 5: Evaluation of GNN at each epoch during training.

As expected, the best performance, 62.9% Hits@20 was achieved by the GNN. Without much tuning, it would place among the top 10 models on the official leaderboard, which is a testimony of the expressiveness of GCNs.

The evolution of Hits@20 throughout training is shown in Figure 5. Despite the noisiness of the data, we see that the model improves steadily on both the training and test data, demonstrating that it is learning without overfitting during the entire training process.

The superior performance of this model is most likely due to the convolutional blocks, which learn the structure of the local neighbourhood with great detail via message passing. This enables the network to capture local structures, similarly (but better) to what we did manually with hand-crafted features for the baseline model. Furthermore, the fact that the embedding matrix is learned from scratch complements the rest of the architecture, as the model can customize the embeddings to capture each node's features appropriately for the task. This hypothesis is also supported empirically – after the first several epochs, initializing the embeddings using the representations from §3.2 achieved vastly inferior results to initializing them randomly and letting the model learn them.

4 Conclusion

In this project, we analysed the link-prediction `ogbl-ddi` dataset. We compared it to standard network models and used metrics from network theory to classify the dataset as a small-world network with several scale-free network features. We use these findings to create a baseline logistic regression model, which uses only several hand-crafted features, but already performs on par with vastly more complicated models on the world-wide leaderboard, achieving a Hits@20 score of 10.7%. We also investigated the effect of exchanging these hand-crafted features for node2vec or Laplacian eigenmap embeddings, which led to inferior results. Finally, we explored the impact of last decade's findings in network science and created a GNN model which vastly outperformed the baseline and places comfortably in the top 10 in the worldwide leaderboard, achieving a Hits@20 of 62.9% without any tuning.

References

Albert-László Barabási and Márton Pósfai. 2016. *Network science*. Cambridge University Press, Cambridge.

Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396.

Pascal Frossard. 2022a. Graph theory lecture notes.

Pascal Frossard. 2022b. Random networks lecture notes.

Pascal Frossard. 2022c. Scale-free networks lecture notes.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural net-

works. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org.

Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA. Association for Computing Machinery.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

Dorina Thanou. 2022. Graph neural networks: Building blocks.

David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, et al. 2018. Drugbank 5.0: a major update to the drugbank database for 2018. *Nucleic acids research*, 46(D1):D1074–D1082.